Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Дипломна робота
бакалавра

з теми: «РОЗРОБКА ВЕБ-ЗАСТОСУНКУ "ІНФОРМАЦІЙНИЙ ПАКЕТ СПЕЦІАЛЬНОСТІ"»

Виконав: студент 4 курсу,
групи KN1-B18,
спеціальності 122 Комп’ютерні науки
Рисюк Аспазій Вадимович

Керівник: Іванюк В.А.,
завідувач, доцент кафедри
комп’ютерних наук, доктор технічних
наук, доцент

Рецензент: Поведа Т.П.,
кандидат педагогічних наук, доцент,
доцент кафедри фізики

Кам’янець-Подільський – 2022

ЗМІСТ

ВСТУП

Актуальність роботи. Веб-застосунками, або іноді також веб-системами, називаються на сьогоднішній день різні програмні продукти, доступ до яких здійснюється через веб-інтерфейс. **Розробка веб-застосунків** потрібна найчастіше різним комерційним організаціям, тому функціонал веб-застосунків зазвичай включає потужні бізнес-орієнтовані інструменти.

Створення веб-застосунків дозволяє **вирішувати найрізноманітніші завдання комерційних компаній**. Так, наприклад, за допомогою веб-застосунків можна вести облік часу роботи всіх співробітників, навіть якщо компанія має мережу віддалених філій, здійснювати облік вантажних або пасажирських перевезень, здійснювати моніторинг діяльності компанії, керувати роботою персоналу, а також нараховувати заробітну плату.

На базі веб-застосунків можна створити навіть цілу **ERP-веб-систему**, яка за своїм функціоналом не поступатиметься звичайним «внутрішнім» ERP-системам. У таких системах через веб-інтерфейс можна здійснити автоматизацію різних бізнес-процесів у компанії: фінансового та складського обліку, логістики, постачання, контролю якості тощо.

Цікаво, що веб-застосунки можуть бути корисними не тільки для бізнесу. Некомерційні організації теж часто потребують веб-застосунки. Так, популярні останнім часом **електронні щоденники — яскравий приклад веб-застосунків** некомерційного характеру. За допомогою електронних щоденників взаємодія між школою, кожним окремо взятим учнем і його батьками стає більш прозорою.

Метою – створення веб-застосунку для подальшого користування його в інших сферах.

Об’єктом дослідження – технології створення **веб-застосунку**.

Предмет дослідження – **Intellij IDEA, Visual Studio Code** для створення веб-застосунку.

Структура роботи. Дипломна робота складається зі вступу, двох розділів, списку використаних джерел та додатків.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД

1.1 Веб-застосунок

Розподілений застосунок, в якому клієнтом виступає браузер, а сервером — вебсервер. Браузер може бути реалізацією так званих тонких клієнтів — логіка застосунку зосереджується на сервері, а функція браузера полягає переважно у зображенні інформації, завантаженої мережею з сервера, і передачі назад даних користувача. Однією з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому веб-застосунки є міжплатформовими сервісами. Унаслідок цієї універсальності й відносної простоти розробки веб-застосунки стали широко популярними в кінці 1990-х — початку 2000-х років.

1.2 Технічні особливості

Істотною перевагою побудови веб-застосунків для підтримки стандартних функцій браузера є те, що функції повинні виконуватися незалежно від операційної системи клієнта. Замість того, щоб писати різні версії для Microsoft Windows, Mac OS X, GNU/Linux й інших операційних систем, застосунок створюється один раз для довільно обраної платформи та на ній розгортається. Проте різна реалізація HTML, CSS, DOM й інших специфікацій в браузерах може викликати проблеми при розробці веб-застосунків і подальшої підтримки. Крім того, можливість користувача налаштовувати багато параметрів браузера (наприклад, розмір шрифту, кольори, відключення підтримки сценаріїв) може перешкоджати коректній роботі застосунку.

На початку 2000-х років був популярним інший (менш універсальний) підхід з використанням Adobe Flash або Java-аплетів для повної або часткової реалізації призначеного для користувача інтерфейсу. Ці технології надавали програмістові більший контроль над інтерфейсом, і були здатні обходити

багато несумісностей у конфігураціях браузерів (хоча несумісність між Java або Flash реалізаціями клієнта спричиняла різні ускладнення). Станом на 2020 рік Java-аплети та Flash-технологія практично вийшли з ужитку.

Через архітектурну схожість з традиційними клієнт-серверними застосунками, певним чином «товстими» клієнтами, існують суперечки щодо коректності зарахування подібних систем до веб-застосунків; альтернативний термін «Насичений інтернет-застосунок».

1.3 Типи мобільних веб-застосунків

Існує декілька підходів для створення вебзастосунків для мобільних пристроїв:

- Адаптивний вебдизайн може бути використаний для створення як від звичайного вебсайту, так й для односторінкового застосунку, який зручний у використанні на пристроях з невеликими екранами.
- Прогресивний вебзастосунок поєднання звичайних вебсторінок із мобільним застосунком.
- Мобільний застосунок або «рідний застосунок» запускається на виконання безпосередньо на мобільному пристрою без веббраузера й зазвичай не потребують наявності інтернет з'єднання. Типово пишуться мовою Java для Android-пристроїв або на Objective-C чи Swift для iOS. Останнім часом, такі програмні каркаси, як React Native, Flutter, Xamarin дозволяють розробляти мобільні застосунки відразу для декількох мобільних платформ використовуючи одну мову програмування (як правило, одну із поширеніших, на зразок JavaScript), замість стандартних для мобільних застосунків.
- *Гібридні застосунки* вбудовують вебсайт всередину мобільного застосунку. Можуть бути побудовані за допомогою гібридних програмних каркасів (framework) таких як: Apache Cordova, Ionic або Appcelerator Titanium. Цей підхід дозволяє розробникам використовувати сучасні

вебтехнології, разом зі збереженням певних переваг саме мобільних застосунків: застосування апаратного прискорення, офлайн операції, доступ до магазину застосунків тощо.

1.4 Архітектура веб-застосунків

Архітектура Веб-застосунків включає компоненти (рис. 1.1), які відповідають за клієнтську та серверну сторони:

- компоненти клієнтської сторони створюються за допомогою JavaScript, HTML, CSS та відповідають за користувацький інтерфейс: інформаційні панелі, сповіщення, журнали активності та інші елементи. Веб-браузер запускає код та перетворює його в інтерфейс. Налаштування операційної системи або інших складових пристрою користувача не потрібні;

- серверні компоненти створюються за допомогою Java, .Net, NodeJs, Python та інших мов програмування. Сервер складається з двох частин: логіки застосунку та бази даних. Логіка - це операційний центр веб-застосунку. База даних містить всю необхідну для додатку інформацію (наприклад, облікові дані для входу користувачів в систему).

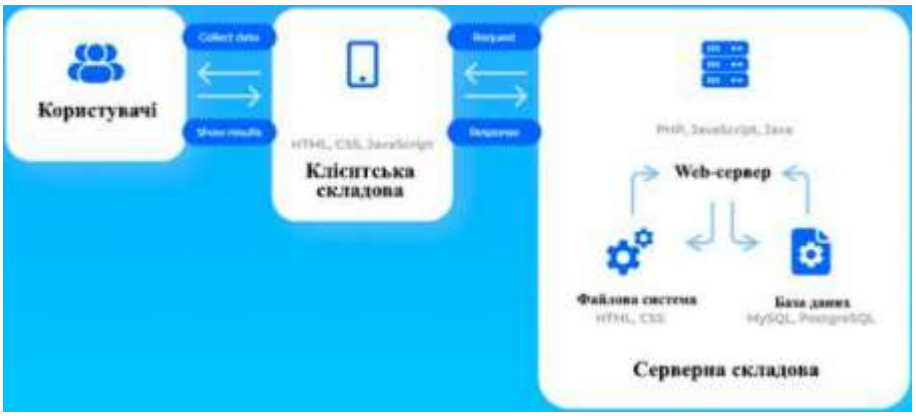


Рис. 1.1. Складові компоненти архітектури веб-застосунків

При інтеграції всіх компонентів та формуванні веб-застосунку відбувається створення наступних 4 рівнів:

- рівня представлення даних;

- рівня обслуговування даних;

- рівня бізнес логіки;

- рівня доступу до даних.

Рівень представлення даних відображає інтерфейс для користувачів та робить взаємодію з компонентами більш простою. Цей рівень архітектури веб-застосунків містить компоненти інтерфейсу, які обробляють та відображають дані додатку, а також компоненти, які встановлюють та обробляють дії користувача з елементами сторінки.

Основна мета рівня представлення - отримати вхідні дані, обробити запити користувачів, надіслати їх до рівня обслуговування даних та відобразити відповідний результат на сторінці .

Даний рівень доступний через браузер та компоненти інтерфейсу, які взаємодіють з іншими рівнями. Основні технології, які використовуються на даному рівні: JavaScript, HTML, CSS. HTML скрипти відповідають за наповнення та тему додатку, CSS відповідає за його зовнішній вигляд. Відповідь на запити користувачів при взаємодії з елементами інтерфейсу та компоненти наступних рівнів генеруються за допомогою інструментів мови програмування JavaScript та пов’язаних з нею фреймворків.

Рівень обслуговування отримує дані від попереднього рівня, передає на рівень бізнес-логіки для обробки та отриманий результат повертає до рівня презентації. Передаючи дані, якими оперує рівень бізнес-логіки, рівень обслуговування захищає інформацію веб-застосунку, оскільки він ізолює бізнес-логіку від клієнтської сторони.

Рівень бізнес-логіки відповідає за правильний обмін даними. Цей рівень визначає логіку бізнес-операцій і правил та відповідає за завершення обробки запитів клієнта з браузера, визначення шляхів доступу до даних додатку .

Прикладом роботи рівня бізнес-логіки є процес обробки входу в систему веб-застосунку. Напрямки, за допомогою яких серверна частина отримує дані та клієнтські запити формуються саме на рівні бізнес-логіки.

У веб-застосунку для управління послугами в сфері інформаційних технологій даний рівень визначає послідовність дій, які будуть виконувати компанії або IT-фахівці для реєстрації та авторизації в системі а також для

розміщення або надсилання запиту на виконання певного виду послуг.

Рівень доступу до даних пропонує спрощений варіант отримання інформації, що зберігається в таких сховищах, як двійкові або XML файли. Даний рівень також відповідає за управління операціями CRUD: створення, читання, оновлення, видалення даних і має назву рівня збереження або сховища, який об’єднується з рівнем бізнес-логіки. Таким чином, рівень бізнес-логіки отримує інформацію про те, до якої бази даних необхідно застосовувати зміни та як оптимізувати отриману інформацію.

Правильний розподіл основного функціоналу веб-застосунку між відповідними рівнями дозволяє призначити кожному компоненту відповідальність за виконання окремого сервісу, що покращує інтегрованість та полегшує пошук помилок при розробці та використанні додатку .

1.5 Види веб-застосунків, принцип роботи

Веб-застосунки зазвичай розробляються за допомогою мов програмування, які підтримуються браузерами, такими як JavaScript та HTML. Деякі сторінки Веб-застосунків є динамічними і потребують обробки на стороні сервера. Інші повністю статичні і не вимагають обробки на сервері.

Веб-програма вимагає, щоб веб-сервер керував запитами клієнта, сервер додатків виконував функціональну логіку відповідно до запиту користувача, включаючи обробку та збереження інформації в базі даних.

Типовий процес роботи веб-застосунків виглядає наступним чином:

1. Користувач генерує запит до веб-сервера через веб-браузер, або через інтерфейс користувача програми;

2. Веб сервер пересилає цей запит на відповідний сервер веб-додатків;

3. Сервер веб-додатків виконує необхідну логіку відповідно до отриманого запиту – наприклад, виконує запит до бази даних або обробку даних, потім генерує результат роботи;

4. Сервер веб-додатків надсилає результати на веб-сервер із необхідною інформацією або обробленими даними;

5. Веб-сервер надає відповідь клієнту із потрібною інформацією, яка потім з'являється в інтерфейсі користувача.

Відповідно до функціональності та способу представлення інформації веб-застосунки розділені на декілька категорій, кожна з яких має свої особливості роботи. Основними видами є статичні, динамічні, комерційні, портальні веб-додатки та системи управління наповненням сайту .

Статичні веб-застосунки – це найперший вид, який з’явився в мережі Інтернет. Як правило, він не має ніякої взаємодії між користувачем і сервером, розробляється за допомогою HTML і CSS для відображення лише відповідного контенту та даних.

Прості веб-сайти зазвичай включають статичні сторінки, створені для інформаційних потреб .

В деяких випадках статичні веб-застосунки містять GIF-файли або відео

для залучення відвідувачів. Прикладом статичного веб-застосунка може бути веб-сайт портфоліо або веб-сайт компанії, представлений у вигляді візитної картки.

Схема функціонування статичного веб-застосунку представлена на рис. 1.2.

Статичний веб-сайт містить набір відповідних HTML-сторінок та файлів, розміщених на комп'ютері, на якому встановлений веб-сервер.

Веб-сервер – це програмне забезпечення, яке надає веб-сторінки у відповідь на запити веб-браузерів. Зазвичай запит на сторінку створюється під час натискання посилання на веб-сторінці, вибору закладки в браузера або вводу URL-адреси в адресному рядку браузера.

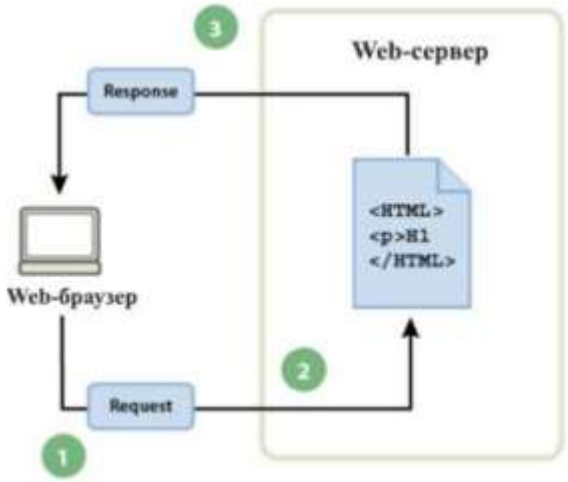


Рис. 1.2. Схема функціонування статичного веб-застосунку

Кінцевий вміст статичної веб-сторінки визначається розробником і залишається незмінним.

Весь HTML-код створюється розробником до того моменту, коли сторінка буде розміщена на сервері. Коли веб-сервер отримує запит на видачу статичної сторінки, то після аналізу запиту сервер знаходить необхідну сторінку і надсилає її браузеру.

Динамічні веб-застосунки – це оновлені статичні, оскільки їх складніше розробити з технічної точки зору. Їх основна мета – взаємодія з клієнтом. Вони мають різні інтерактивні елементи та методи залучення клієнтів до послуг або

продуктів, які пропонує веб-застосунок.

З появою Інтернету, особливо після популяризації блогів і соціальних мереж, потужність медіа-інтерактивності різко зросла і на сьогодні більшість організацій намагається залучати до участі користувачів на веб-сторінках .

Такі веб-застосунки використовують бази даних для збереження приватних та загальнодоступних даних, які відображаються на веб-сторінці. Для управління частинами сервера та інтерфейсу такі програми зазвичай мають панель адміністратора, де адміністратори можуть змінювати вміст та додавати нові інтерактивні елементи.

Схема функціонування динамічного веб-застосунку представлена на рис. 1.3. Коли веб-сервер отримує запит на видачу статичної сторінки, він надсилає сторінку безпосередньо браузеру. Проте, коли виконується запит на динамічну сторінку, дії веб-серверу неоднозначні. Сервер передає сторінку спеціальній програмі, яка формує кінцеве наповнення. Така програма називається сервером застосунків .

Сервер застосунків виконує читання коду, який знаходиться на сторінці, формує кінцевий результат відповідно до прочитаного коду, а потім видаляє його зі сторінки. В результаті цих операцій отримується статична сторінка, яка передається серверу, який в свою чергу надсилає її клієнтському браузеру.

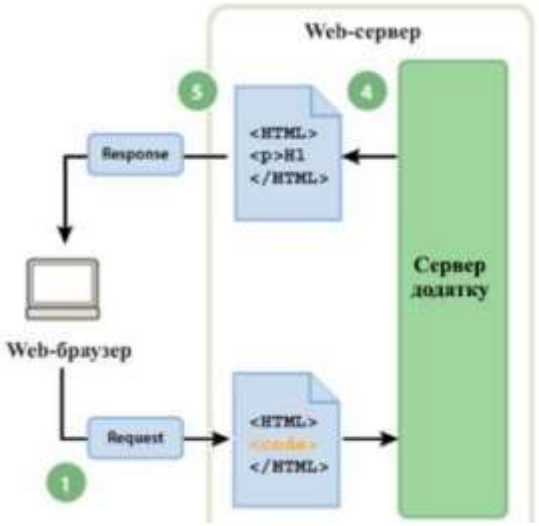


Рис. 1.3. Схема функціонування динамічного веб-застосунку

Для розробки надійного динамічного веб-застосунка використовується

багато мов програмування. Найпоширеніші – PHP та ASP.NET. Прикладом такого додатка може бути практично будь-який веб-застосунок, подібний до типового веб-сайту цифрового агентства, де користувачу потрібно залишити запит .

Сервер динамічного веб-застосунку також дозволяє працювати з серверними ресурсами, таким як бази даних. Наприклад, динамічна сторінка може виконати запит на сервер для отримання інформації з бази даних та виведення її на HTML сторінку.

Використання сховища бази даних дозволяє відокремити дизайн сайту від його наповнення. Замість того, щоб створювати окремі HTML-сторінки для кожного окремного контенту, розробник створює один шаблон, який може обробляти різні дані, що надходять зі сховища. При кожному запиті клієнта відбувається оновлення контенту в шаблоні та відображення актуальної інформації.

Схема функціонування динамічного веб-сайту з використанням бази даних представлена на рис. 1.4.

Веб-браузер виконує запит на динамічну сторінку, яку знаходить веб-сервер та надсилає до серверу додатку. Сервер додатку сканує сторінку та виявляє можливі запити, які надсилає до драйверу бази даних.

При виконанні певного запиту база даних повертає відповідну інформацію, яка додається до наповнення динамічної сторінки та відображається в браузері користувача.

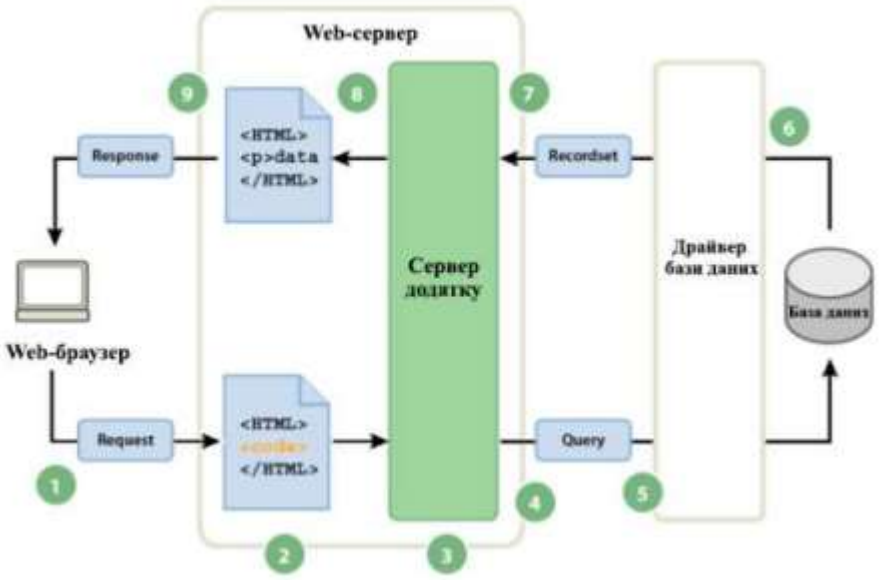


Рис. 1.4. Схема функціонування динамічного веб-застосунку з використанням бази даних

Якщо веб-застосунок безпосередньо рекламує продукти або послуги потенційним клієнтам, то він називається інтернет-магазин або веб-застосунок електронної комерції .

Під електронною комерцією мається на увазі процес купівлі або продажу товарів чи послуг через Інтернет. Інтернет-магазини стають все більш популярними через швидкість і простоту використання для клієнтів.

Продажі в Інтернеті можуть допомогти бізнесу вийти на нові ринки та збільшити продажі і прибуток. Якщо організація зацікавлена в продажі товарів чи послуг іншим підприємствам, вона може використовувати Інтернет, щоб знайти потенційних клієнтів, оголосити тендери та пропонувати продукти для продажу (чи через власний веб-сайт, або через сайт електронного ринку) .

Порівняно з динамічним веб-додатком, даний тип вимагає більше функцій, оскільки клієнти повинні виконати певні дії, щоб придбати та отримати товари чи послуги.

Функції даного виду веб-застосунків включають системи електронних платежів, панель управління для адміністратора (переважно для заповнення переліку продуктів, його оновлення або видалення товарів, а також для управління замовленнями клієнтів та обробки платежів) та особистий кабінет для користувача.

Веб-додаток порталу відноситься до застосунків, які містять багато різних розділів або категорій, доступ до яких реалізований на домашній сторінці. Після входу користувача в систему функціонал порталу також дозволяє постачальнику послуг відстежувати активність користувачів на веб-сайті. Прикладами такого типу додатків можуть бути форуми або чати.

Системи для управління контентом – це тип веб-застосунка, де користувач не потребує допомоги технічної команди, оскільки він може самостійно змінювати наповнення веб-сайту, не вивчаючи жодних мов програмування. Загальні особливості систем управління наповненням сайтом включають:

- створення контенту дозволяє користувачам легко додавати та форматувати вміст;

- зберігання інформації в одному місці, дотримуючись послідовності додавання даних користувачем;

- наявність багатьох робочих процесів та призначення ролей для керування вмістом, таких як автори, редактори та адміністратори;

- забезпечення публікації, упорядкування та розміщення контенту в реальному часі.

Прикладами систем управління контентом є WordPress, Joomla та Drupal.

На основі аналізу принципу роботи веб-застосунків можна виділити наступні переваги їх використання:

- Забезпечення доступу з будь-якого пристрою: з веб-застосунком можна працювати з будь-якої точки світу з комп’ютера, планшета або смартфона, підключеного до мережі Інтернет;

- Економія: веб-застосунки працюють на всіх платформах і виключають необхідність розробки додатку окремо для різних операційних систем чи платформ;

- Адаптивність: якщо для більшості додатків необхідні визначені операційні системи, то для роботи з веб-застосунком підходить будь-яка операційна система та будь-який браузер;

- Відсутність клієнтського програмного забезпечення: інсталяція,

обслуговування та модернізація значно дешевші та швидкі, оновлення до останньої версії додатку відбувається при черговому завантаженні сторінки;

- Мережева безпека: веб-система має єдину точку входу, яку можна централізовано налаштувати та забезпечити надійну безпеку;

- Масштабованість: зі збільшенням навантаження на систему не треба збільшувати потужність клієнтських місць, веб-застосунок дозволяє обробляти більшу кількість даних, як правило, лише силами апаратних ресурсів, без необхідності зміни коду та архітектури;

- Захист від втрати даних: дані користувачів зберігаються в хмарному сховищі або в базі даних, за цілісність яких відповідають провайдери. Дані захищені від втрати при будь-яких пошкодженнях апаратних складових пристрою користувача.

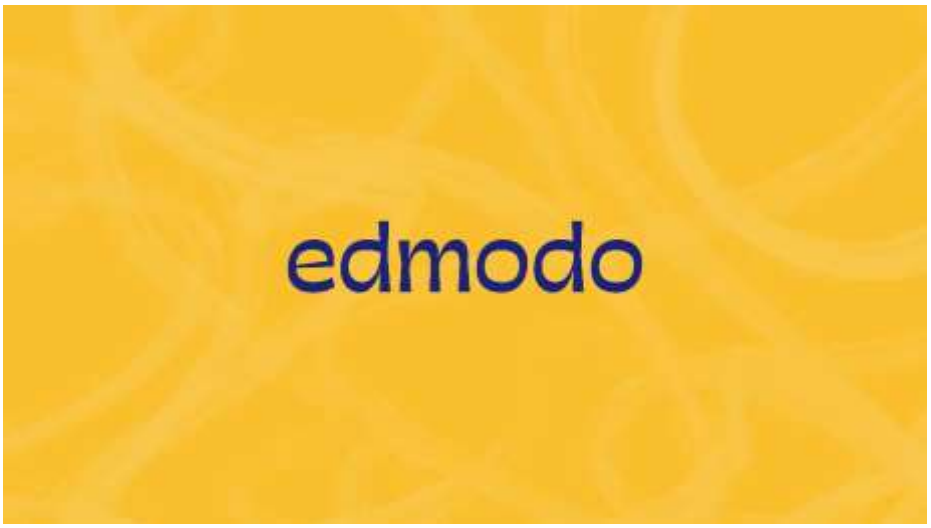
1.6. Приклади веб-застосунків

LightSail

Це додаток для читання електронних книг з бібліотекою з 80 000 текстів. Студенти можуть бачити свій прогрес читання. Викладачі можуть ставити задачі для своїх груп і призначати тести, а також бачити, хто не читав останнім часом.

Edmodo

Багато викладачів вже використовують Edmodo для спілкування зі студентами та виконання завдань. Його можна охарактеризувати як «Фейсбук для групи». Студенти можуть бачити контент, отримувати і відправляти завдання, а також взаємодіяти з викладачем та одногрупниками, брати участь в опитуваннях, обмінюватися повідомленнями, документами, зображеннями тощо.



SpiderScribe

SpiderScribe спрощує мозковий штурм, дозволяючи студентам візуально пов'язувати і організовувати ідеї, файли, події і зображення. Якщо у студентів з цим труднощі, SpiderScribe дає візуальне уявлення, щоб краще зрозуміти процес. Кілька людей можуть отримати доступ до одного облікового запису, тому члени команди і одногрупники можуть легко співпрацювати.

Socrative

Якщо ви хоча б частково зацікавлені в дистанційному навчанні або хочете отримати миттєвий зворотний зв'язок від своїх студентів, то Socrative може бути вам корисний. Це інструмент, який дозволяє викладачам ставити запитання студентам в реальному часі, а потім агрегує і візуалізує результати. Так ви можете швидко отримати відгуки чи опитати своїх студентів. Це дозволяє бачити рівень знань і розуміння всієї групи і відповідно адаптувати їх навчання. Це також змушує студентів відчувати себе залученими і зацікавленими.

Project Gutenberg

Якщо ви хочете впровадити електронні підручники, Project Gutenberg – хороший старт, особливо для викладачів англійської. Проект пропонує більш 54 000 електронних книг, доступних для скачування, безкоштовних для студентів і викладачів.

Kahoot!

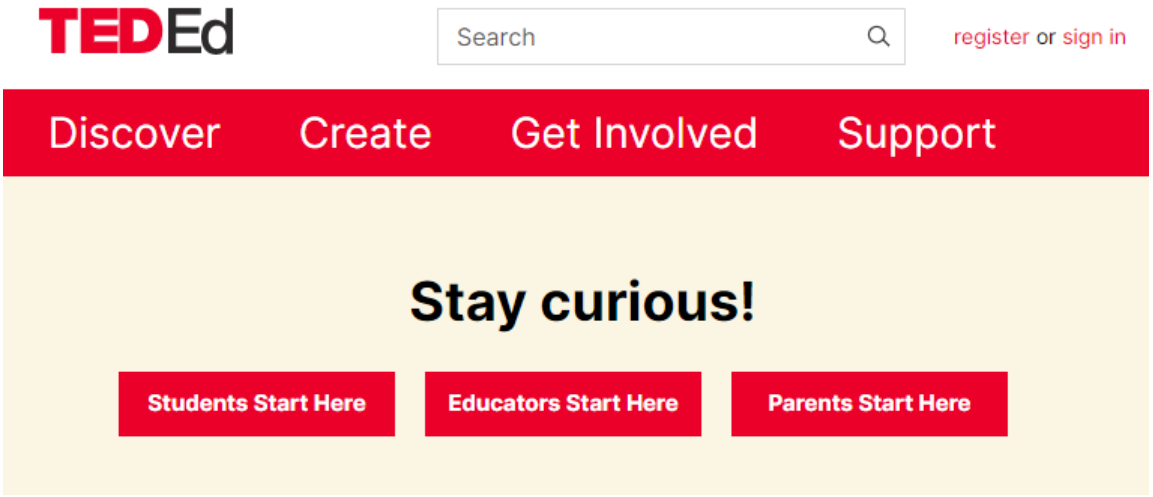
Kahoot є зручним інструментом для розробки запитальників і вікторин для групи. Користувач розробляє вікторину, опитування або анкету для перевірки знань своєї аудиторії. Студенти можуть відповідати на запитання з різних пристроїв. Вікторини та анкети сприяють створенню ігрової атмосфери в навчальному середовищі.

Prezi

Перетворіть звичайні презентації в захоплюючі. Використовуючи Prezi, ви побачите, що презентації так само цікаві, як і їх створення. Prezi дозволяє користувачам об'єднувати всі елементи інтерактивно, що створює історію, яка захопить будь-яку аудиторію.

TedEd

Ймовірно, ви чули про Ted Talks. Тепер ви можете поділитися своїми знаннями з допомогою Ted Ed. Це онлайн-інструмент, який дозволяє користувачеві створювати урок навколо будь-якого Ted Talk або відео на YouTube. Існує два типи уроків TED-Ed. Перші створені експертами-педагогами, сценаристами і аніматорами. Другий тип може створити будь-який відвідувач сайту і додати запитання, теми для обговорення та інші додаткові матеріали. Обидва типи занять TED-Ed можна регулярно використовувати в університеті.



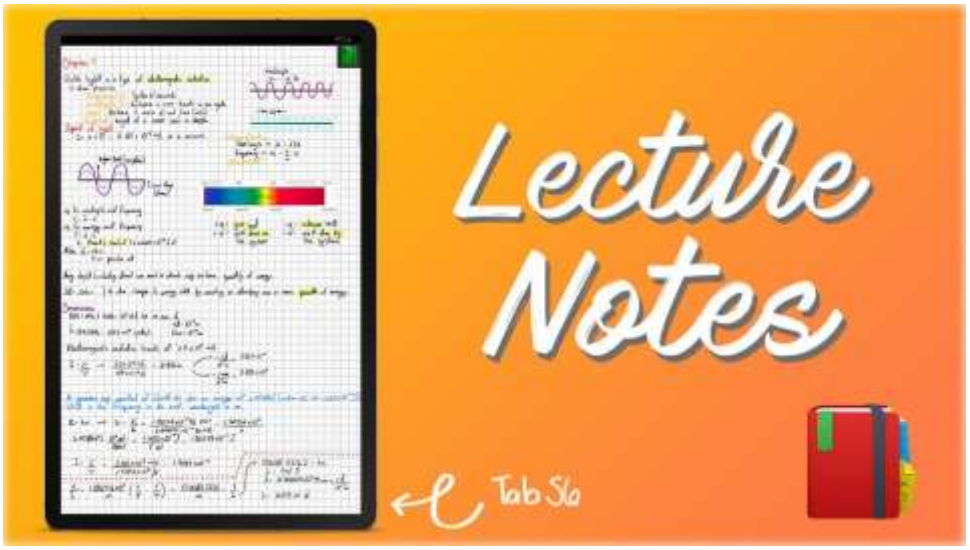
BloomBoard

Цікавий приклад перетину відгуків викладачів і професійного розвитку. Це сховище, в якому адміністратори можуть писати свої спостереження про групу, а потім підключати викладачів, щоб допомогти поліпшити викладання. Викладачі можуть створювати свої власні групи і вести чати в реальному часі про свою роботу.

LectureNotes

Додаток, призначений для нотаток, які можна і друкувати, і писати стилусом або ж пальцями.

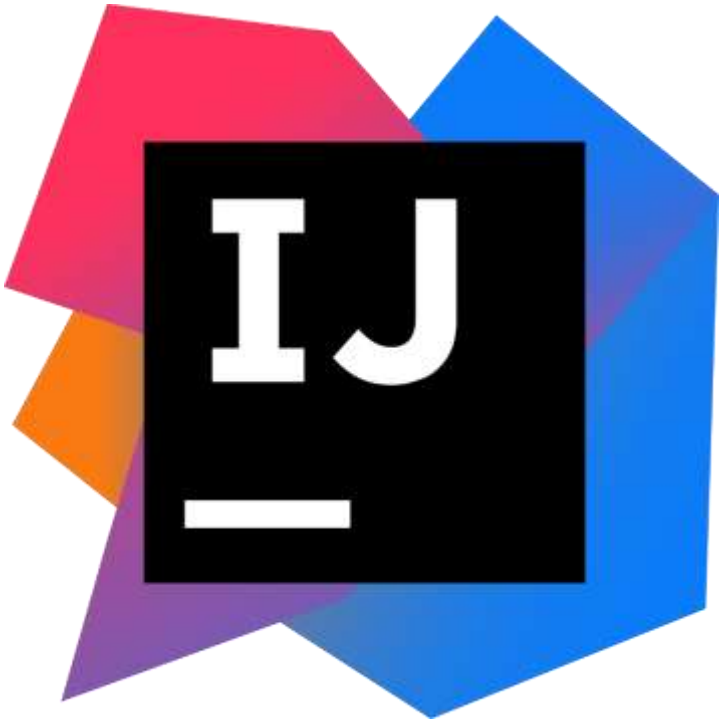
Його розробили для лекторів для підготовки до лекції або навіть для фактичного навчання (написавши на пристрої і проектуючи вміст екрану для аудиторії) або студентів для нотаток під час занять. Проте додаток знайшов багато користувачів за межами університету, наприклад, бізнесменів.



1.7. Фреймворк Intellij IDEA

Програмне забезпечення JetBrains IntelliJ IDEA є провідним середовищем швидкого розвитку мови Java. IntelliJ IDEA - це високотехнологічний комплекс тісно інтегрованих інструментів програмування, включаючи інтелектуальний редактор джерел з передовими інструментами автоматизації, потужні інструменти рефакторингу коду, вбудовану підтримку технологій J2EE, інтеграційні механізми з середовищем тестування Ant/ JUnit та системами контролю версій, унікальний інструмент оптимізації та перевірки перевірки коду, а також інноваційний візуальний графічний дизайнер Інтерфейси.

Унікальні особливості JetBrains IntelliJ IDEA знімають з програміста тягар рутинної роботи, допомагають своєчасно усунути помилки і поліпшити якість коду, піднявши продуктивність розробника на нову висоту.



Мінуси і плюси

Мінуси:

Не стільки мінус, скільки інформація до уваги. Не всі мови однаково добре підтримуються у IDEA. Достовірно відомо, що добре підтримуються Java, Kotlin та Scala. Erlang, наскільки я пригадую, теж добре підтримувався. Для більшості інших мов, які я пробував, всякі там Rust, Go, Haskell та інші дивні нікому не потрібні мови, або реалізовано лише підсвічування синтаксису, без нормального автокомпліту, переходу до визначення, рефакторингів і так далі, ну або вони реалізовані, але працюють аби як. Слід зазначити, що у таких випадках IDEA не гірше Vim як мінімум;

Винятково рідко, але буває, що натрапляєш на баги. Натиснув Shift-Shift, знайшов потрібний клас, а діалог пошуку не закривається. Або пишеш такий код, і в якийсь момент, пропав курсор. На зовсім. У першому випадку вдається якось викрутитись, у другому IDE доводиться перезапускати. Але ще раз зазначу, що трапляється таке не часто. Можна тижнями спокійно працювати і знати бід;

Підтримка деяких речей бажає кращого. Налаштував ти прапори в build.sbt, що скрізь -Xfatal-warnings, в одному підпроекті -deprecation:true, а в іншому -deprecation:false. Сам sbt збирає все без проблем, а IDEA не збирає, тому що в тебе є deprecation warning в підпроекті, де -deprecation:false. А потім, через якийсь час — раз, і вже все збирається, хоч ти нічого не змінював. Також підтримка Play Framework мене засмутила як у безкоштовній, так і платній версії IDEA. Половина коду підкреслюється червоним, автокомплит працює через раз — такі проблеми;

Плюси:

IDEA справді є найпотужнішим редактором вихідного коду. Тут є не тільки класичне підсвічування синтаксису, таби, фолдинг та кілька буферів обміну. Переходи до визначення, автокомплит з урахуванням потрібного в контексті типу, автоматичне додавання потрібних імпортів, а також оптимізація імпортів з видаленням непотрібних, вбудоване відображення документації та типів, підказка за аргументами, перейменування аргументів, змінних та модулів буквально за секунду, швидка вказівка типу, що повертається у методу, всякі підказки в стилі «чувак, а ти ось тут написав filter(...).size, давай замінимо на count(...)» або «ой, а ти тут не заімплементив ще два методи потрібних ось цьому трейту давай їх додамо». Тиснеш там, де підсвітило жовтим кольором, Alt + Enter, знову Enter, і код моментально виправлений за тебе. Багато з названого є в Scala Console, що робить її набагато зручнішою за звичайну sbt console. Все це дійсно дає дикий приріст до продуктивності. Спробувавши одного разу, ви більше ніколи не захочете повертатися ні на які Vim;

Є ще просто дофіга того, чим я особисто не користуюся або дуже рідко користуюся. Інтеграція з Git та іншими системами контролю версій, вбудований найпотужніший відладчик, вбудоване визначення ступеня покриття коду тестами, побудова діаграм класів та переходи по ієрархіях вгору та вниз, вбудований декомпілятор байткоду Java. Підозрюю, що є ще чимало такого, про що я ще й не здогадуюсь;

Вибагливість до ресурсів і гальмування IDE, так само як і мови Scala, на якій я зараз більшу частину часу пишу, або там JVM, виявилася дуже перебільшеною. Як уже зазначалося, навіть на далеко не найновішому і не найкрутішому ультрабуку чудово пишеться цілком великий і реальний проект, і вільних ресурсів при цьому залишається вагон. Дивно, але на цьому втрабуку компіляція проекту з нуля займає навіть на кілька секунд менше часу, ніж на

моєму основному комп'ютері, вже сам не знаю, чому. Крім того, IDE можна чудово користуватися на 13-дюймовому моніторі, я сам довгий час так писав;

Вперше на моїй практиці в команді немає срачів через code style або таке, що хтось ставить для оступів таби, а хтось три пробіли. Усі сидять під IDEA та код форматується автоматично. Більше того, ніхто не лається по пів дня через річ начебто більш правильно назвати метод і не докопується під час code review до дрібниць на кшталт однобуквенних змінних. Все тому, що перейменувати клас, методи або змінну займає реально одну секунду і назвати їх у будь-який момент можна абсолютно як завгодно. Коли в команді користуються вимами або саблаймами, такі речі постійно викликають суперечки, тому що потім виправити відступи або перейменувати функцію вимагатиме досить багато часу. Як результат, завдяки використанню IDE продуктивність команди значно вища;

1.8. Фреймворк Visual Studio Code

Visual Studio Code (VS Code) – редактор вихідного коду, розроблений Microsoft для Windows, Linux та macOS. Позиціонується як «легкий» редактор коду для кросплатформної розробки веб- та хмарних програм. Включає в себе відладчик, інструменти для роботи з Git, підсвічування синтаксису, IntelliSense та засоби для рефакторингу. Має широкі можливості для кастомізації: теми користувача, поєднання клавіш і файли конфігурації. Розповсюджується безкоштовно, розробляється як програмне забезпечення з відкритим вихідним кодом, але готові зборки розповсюджуються під ліцензією проприетарної.

Visual Studio Code заснований на Electron та реалізується через веб-редактор Monaco, розроблений для Visual Studio Online.

РОЗДІЛ 2. РОЗРОБКА ВЕБ ЗАСТОСУНКУ «Інформаційний пакет спеціальності»

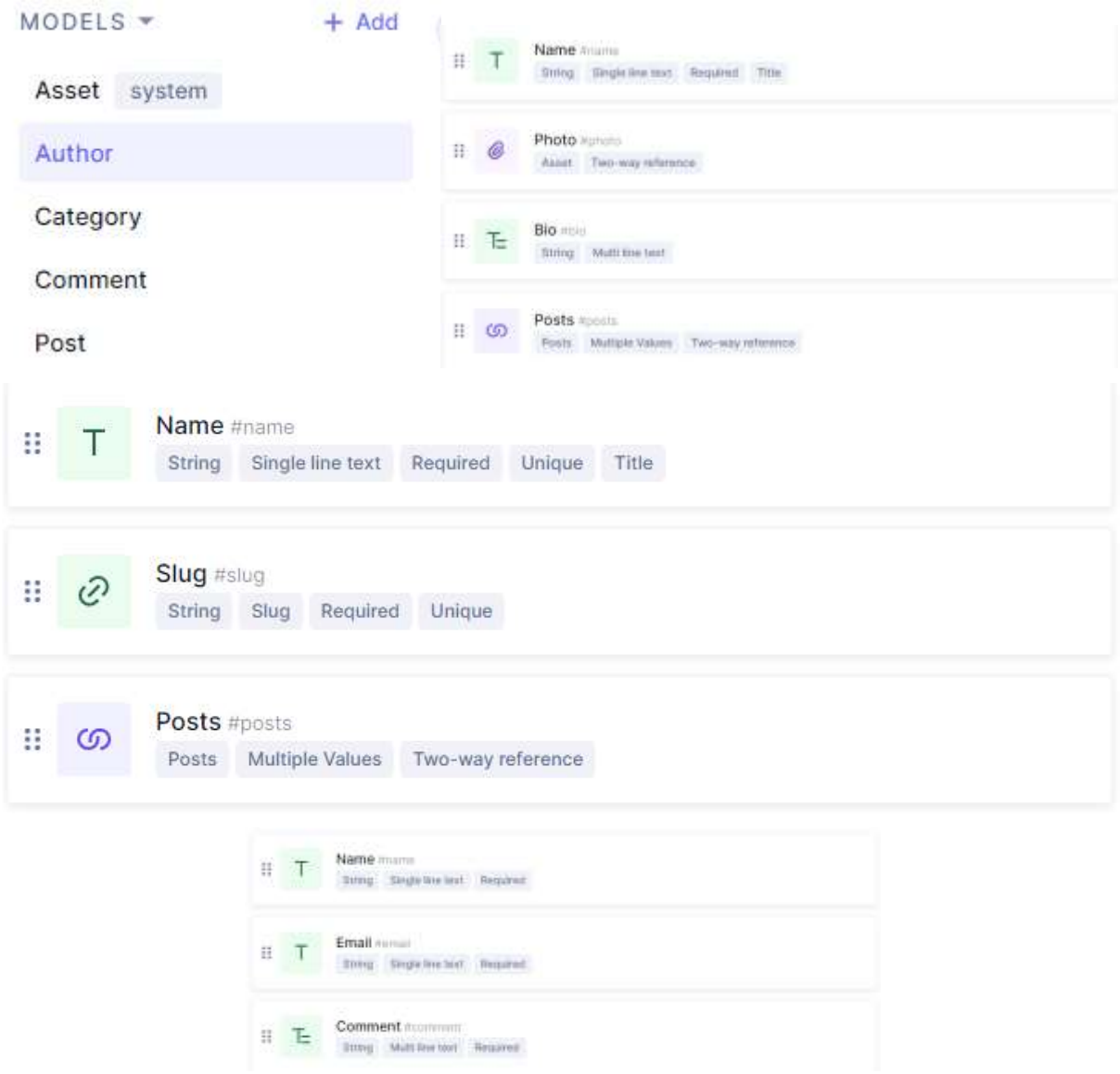
2.1. Постановка задачі

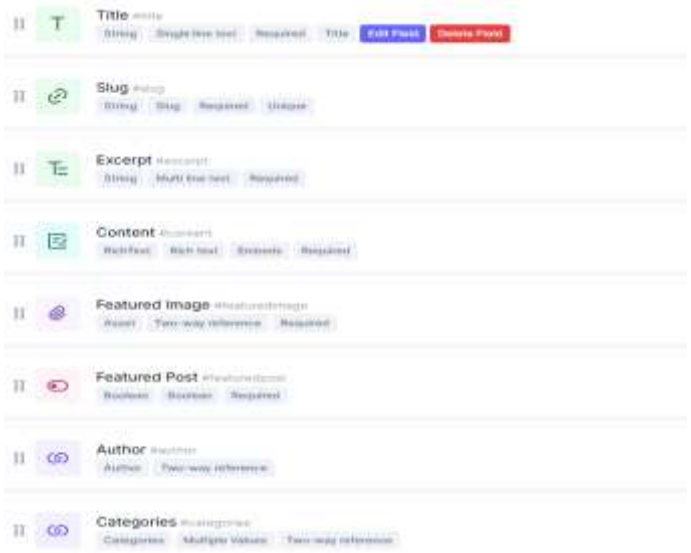
Поставлене завдання: «Розробка веб-застосунку "Інформаційний пакет спеціальності"» Завдяки певному сервісу, котрий допоміг мені включитися в роботу, я зміг скласти план роботи:

1) Розробка схеми;
2) Створення коду;
3) З’єднання через GraphQL;
4) Налаштування віджетів;

2.2. Створення схеми

Вся схема розроблялася на веб-застосунку Graphcms. Завдяки чому можна бистро і легко, створити моделі і на лаштувати їх по своєму бажанню.




Title
Slug
Excerpt
Content
Featured Image
Featured Post
Author
Categories

2.3. Написання коду

Для початку потрібно розібратися, що таке взагалі GraphQl? GraphQL — це мова запитів і маніпуляції даними з відкритим кодом для API і середовище виконання для обслуговування запитів з наявних даних. Після пояснення можна приступати до коду. Для початку потрібно створити пару файлів, щоби було зручно і для зменшення кількості коду. Тому дальше будуть такі файли: index.js, Postcard.jsx, PostDetail.jsx, PostWidget.jsx, index.jsx, Header.jsx, global.scss, app.js, Layout.jsx, Categories,jsx.

Вміст файлу index.js

```
import { FeaturedPosts } from '../sections/index';
import { PostCard, Categories, PostWidget } from '../components';
import { getPosts } from '../services';

export default function Home({ posts }) {
  return (
    <div className="container mx-auto px-10 mb-8">
      <FeaturedPosts />
      <div className="grid grid-cols-1 lg:grid-cols-12 gap-12">
        <div className="lg:col-span-8 col-span-1">
          {posts.map((post, index) => (
            <PostCard key={index} post={post.node} />
          ))}
        </div>
        <div className="lg:col-span-4 col-span-1">
          <div className="lg:sticky relative top-8">
            <PostWidget />
            <Categories />
          </div>
        </div>
      </div>
    </div>
  );
}

// Fetch data at build time
export async function getStaticProps() {
  const posts = (await getPosts()) || [];
  return {
    props: { posts },
  };
}
```

Вміст файлу Postcard.jsx

```
import React from 'react';
import Image from 'next/image';
import moment from 'moment';
import Link from 'next/link';

import { grpahCMSImageLoader } from '../util';

const PostCard = ({ post }) => (
  <div className="bg-white shadow-lg rounded-lg p-0 lg:p-8 pb-12 mb-8">
    <div className="relative overflow-hidden shadow-md pb-80 mb-6">
      <img src={post.featuredImage.url} alt="" className="object-top absolute h-80 w-full object-cover  shadow-lg rounded-t-lg lg:rounded-lg" />
    </div>

    <h1 className="transition duration-700 text-center mb-8 cursor-pointer hover:text-pink-600 text-3xl font-semibold">
      <Link href={`/post/${post.slug}`}>{post.title}</Link>
    </h1>
    <div className="block lg:flex text-center items-center justify-center mb-8 w-full">
      <div className="flex items-center justify-center mb-4 lg:mb-0 w-full lg:w-auto mr-8 items-center">
        <Image
          unoptimized
          loader={grpahCMSImageLoader}
          alt={post.author.name}
          height="30px"
          width="30px"
          className="align-middle rounded-full"
          src={post.author.photo.url}
        />
        <p className="inline align-middle text-gray-700 ml-2 font-medium text-lg">{post.author.name}</p>
      </div>
      <div className="font-medium text-gray-700">
        <svg xmlns="http://www.w3.org/2000/svg" className="h-6 w-6 inline mr-2 text-pink-500" fill="none" viewBox="0 0 24 24" stroke="currentColor">
          <path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2" d="M8 7V3m8 4V3m-9 8h10M5 21h14a2 2 0 002-2V7a2 2 0 00-2-2H5a2 2 0 00-2 2v12a2 2 0 002 2z" />
        </svg>
        <span className="align-middle">{moment(post.createdAt).format('MMM DD, YYYY')}</span>
      </div>
    </div>
    <p className="text-center text-lg text-gray-700 font-normal px-4 lg:px-20 mb-8">
      {post.excerpt}
```

```
    </p>
    <div className="text-center">
      <Link href={`/post/${post.slug}`}>
        <span className="transition duration-500 ease transform hover:-translate-
y-1 inline-block bg-pink-600 text-lg font-medium rounded-full text-white px-8 py-
3 cursor-pointer">Continue Reading</span>
      </Link>
    </div>
  </div>
);
```

Вміст файлу PostDetail.jsx

```
import React from 'react';

import moment from 'moment';

const PostDetail = ({ post }) => {
  const getContentFragment = (index, text, obj, type) => {
    let modifiedText = text;

    if (obj) {
      if (obj.bold) {
        modifiedText = (<b key={index}>{text}</b>);
      }

      if (obj.italic) {
        modifiedText = (<em key={index}>{text}</em>);
      }

      if (obj.underline) {
        modifiedText = (<u key={index}>{text}</u>);
      }
    }

    switch (type) {
      case 'heading-three':
        return <h3 key={index} className="text-xl font-semibold mb-
4">{modifiedText.map((item, i) => <React.Fragment
key={i}>{item}</React.Fragment>)}</h3>;
      case 'paragraph':
        return <p key={index} className="mb-8">{modifiedText.map((item, i) =>
<React.Fragment key={i}>{item}</React.Fragment>)}</p>;
      case 'heading-four':
        return <h4 key={index} className="text-md font-semibold mb-
4">{modifiedText.map((item, i) => <React.Fragment
key={i}>{item}</React.Fragment>)}</h4>;
      case 'image':
        return (
          <img
```

```
            key={index}
            alt={obj.title}
            height={obj.height}
            width={obj.width}
            src={obj.src}
          />
        );
      default:
        return modifiedText;
    }
  };

  return (
    <>
      <div className="bg-white shadow-lg rounded-lg lg:p-8 pb-12 mb-8">
        <div className="relative overflow-hidden shadow-md mb-6">
          <img src={post.featuredImage.url} alt="" className="object-top h-full
w-full object-cover  shadow-lg rounded-t-lg lg:rounded-lg" />
        </div>
        <div className="px-4 lg:px-0">
          <div className="flex items-center mb-8 w-full">
            <div className="hidden md:flex items-center justify-center lg:mb-0
lg:w-auto mr-8 items-center">
              <img
                alt={post.author.name}
                height="30px"
                width="30px"
                className="align-middle rounded-full"
                src={post.author.photo.url}
              />
              <p className="inline align-middle text-gray-700 ml-2 font-medium
text-lg">{post.author.name}</p>
            </div>
            <div className="font-medium text-gray-700">
              <svg xmlns="http://www.w3.org/2000/svg" className="h-6 w-6 inline
mr-2 text-pink-500" fill="none" viewBox="0 0 24 24" stroke="currentColor">
                <path strokeLinecap="round" strokeLinejoin="round"
strokeWidth="2" d="M8 7V3m8 4V3m-9 8h10M5 21h14a2 2 0 002-2V7a2 2 0 00-2-2H5a2 2
0 00-2 2v12a2 2 0 002 2z" />
              </svg>
              <span className="align-middle">{moment(post.createdAt).format('MMM
DD, YYYY')}</span>
            </div>
          </div>
          <h1 className="mb-8 text-3xl font-semibold">{post.title}</h1>
          {post.content.raw.children.map((typeObj, index) => {
            const children = typeObj.children.map((item, itemindex) =>
getContentFragment(itemindex, item.text, item));

            return getContentFragment(index, children, typeObj, typeObj.type);
          })}
        </div>
```

```
      </div>

    </>
  );
};
```

Вміст файлу PostWidget.jsx

```
import React, { useState, useEffect } from 'react';
import Image from 'next/image';
import moment from 'moment';
import Link from 'next/link';

import { grpahCMSImageLoader } from '../util';
import { getSimilarPosts, getRecentPosts } from '../services';

const PostWidget = ({ categories, slug }) => {
  const [relatedPosts, setRelatedPosts] = useState([]);

  useEffect(() => {
    if (slug) {
      getSimilarPosts(categories, slug).then((result) => {
        setRelatedPosts(result);
      });
    } else {
      getRecentPosts().then((result) => {
        setRelatedPosts(result);
      });
    }
  }, [slug]);

  return (
    <div className="bg-white shadow-lg rounded-lg p-8 pb-12 mb-8">
      <h3 className="text-xl mb-8 font-semibold border-b pb-4">{slug ? 'Related
Posts' : 'Recent Posts'}</h3>
      {relatedPosts.map((post, index) => (
        <div key={index} className="flex items-center w-full mb-4">
          <div className="w-16 flex-none">
            <Image
              loader={grpahCMSImageLoader}
              alt={post.title}
              height="60px"
              width="60px"
              unoptimized
              className="align-middle rounded-full"
              src={post.featuredImage.url}
            />
          </div>
          <div className="flex-grow ml-4">
            <p className="text-gray-500 font-
xs">{moment(post.createdAt).format('MMM DD, YYYY')}</p>
```

```
            <Link href={`/post/${post.slug}`} className="text-md"
key={index}>{post.title}</Link>
          </div>
        </div>
      ))}
    </div>
  );
};
```

Вміст файлу index.jsx

```
export { default as PostCard } from './PostCard';
export { default as PostDetail } from './PostDetail';
export { default as Layout } from './Layout';
export { default as Categories } from './Categories';
export { default as Author } from './Author';
export { default as PostWidget } from './PostWidget';
export { default as AdjacentPostCard } from './AdjacentPostCard';
export { default as FeaturedPostCard } from './FeaturedPostCard';
export { default as Comments } from './Comments';
export { default as CommentsForm } from './CommentsForm';
export { default as Loader } from './Loader';
```

Вміст файлу Header.jsx

```
import React, { useState, useEffect } from 'react';

import Link from 'next/link';
import { getCategories } from '../services';

const Header = () => {
  const [categories, setCategories] = useState([]);

  useEffect(() => {
    getCategories().then((newCategories) => {
      setCategories(newCategories);
    });
  }, []);

  return (
    <div className="container mx-auto px-10 mb-8">
      <div className="border-b w-full inline-block border-blue-400 py-8">
        <div className="md:float-left block">
          <Link href="/">
            <span className="cursor-pointer font-bold text-4xl text-white">Graph
CMS</span>
          </Link>
        </div>
        <div className="hidden md:float-left md:contents">
          {categories.map((category, index) => (
```

```
            <Link key={index} href={`/category/${category.slug}`}><span
className="md:float-right mt-2 align-middle text-white ml-4 font-semibold cursor-
pointer">{category.name}</span></Link>
          ))}
        </div>
      </div>
    </div>
  );
};
```

Вміст файлу global.scss

```
html,
body {
  padding: 0;
  margin: 0;
  font-family: 'Montserrat', sans-serif;
  &:before{
    content:'';
    content: "";
    width: 100%;
    height: 100vh;
    background-image: url("../public/bg.jpg");
    position: fixed;
    left: 0;
    top: 0;
    z-index: -1;
    background-position: 50% 50%;
    background-repeat: no-repeat;
    background-size: cover;
  }
}

.text-shadow{
  text-shadow: 0px 2px 0px rgb(0 0 0 / 30%);
}

.adjacent-post{
  & .arrow-btn{
    transition: width 300ms ease;
    width: 50px;
  }
  &:hover{
    & .arrow-btn{
      width: 60px;
    }
  }
}

.react-multi-carousel-list {
```

```
  & .arrow-btn{
    transition: width 300ms ease;
    width: 50px;
    &:hover{
      width: 60px;
    }
  }

}

a {
  color: inherit;
  text-decoration: none;
}

* {
  box-sizing: border-box;
}
```

Вміст файлу app.js

```
import React from 'react';

import '../styles/globals.scss';
import { Layout } from '../components';

function MyApp({ Component, pageProps }) {
  return (
    <Layout>
      <Component {...pageProps} />
    </Layout>
  );
}
```

Вміст файлу Layout.jsx

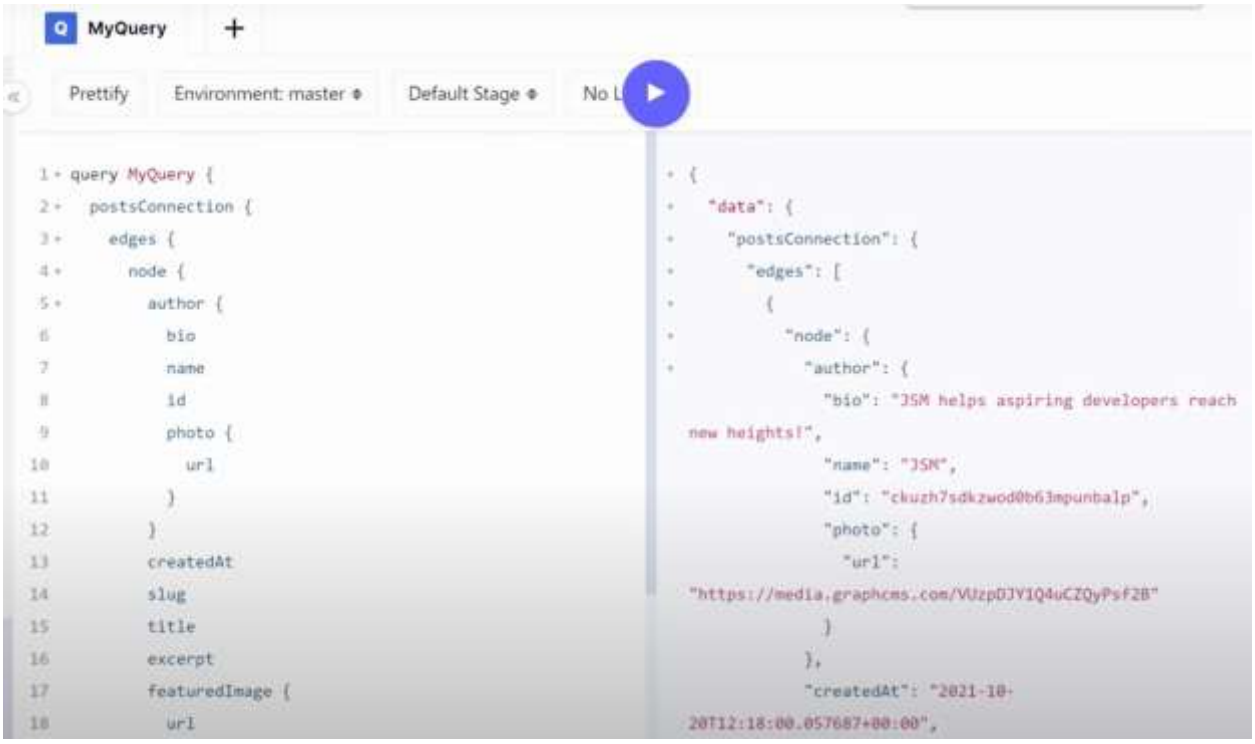
```
import React from 'react';
import Header from './Header';

const Layout = ({ children }) => (
  <>
    <Header />
    {children}
  </>
);
```

Вміст файлу Categories.jsx

```
import React, { useState, useEffect } from 'react';
import Link from 'next/link';

import { getCategories } from '../services';

const Categories = () => {
  const [categories, setCategories] = useState([]);

  useEffect(() => {
    getCategories().then((newCategories) => {
      setCategories(newCategories);
    });
  }, []);

  return (
    <div className="bg-white shadow-lg rounded-lg p-8 pb-12 mb-8">
      <h3 className="text-xl mb-8 font-semibold border-b pb-4">Categories</h3>
      {categories.map((category, index) => (
        <Link key={index} href={`/category/${category.slug}`}>
          <span className={`cursor-pointer block ${(index === categories.length -
1) ? 'border-b-0' : 'border-b'} pb-3 mb-3`}>{category.name}</span>
        </Link>
      ))}
    </div>
  );
};
```

2.4. З’єднання GrapQL

Потрібно зайти у веб-застосунок і подивитися, як правильно виконати і налаштувати з’єднання. Далі ми створюємо файли і налаштовуємо саме з’єднання.



Вміст файлу index.js

```
import { request, gql } from 'graphql-request';

const graphqlAPI = process.env.NEXT_PUBLIC_GRAPHCMS_ENDPOINT;

export const getPosts = async () => {
  const query = gql`
    query MyQuery {
      postsConnection {
        edges {
          cursor
          node {
            author {
              bio
              name
              id
              photo {
                url
              }
            }
            createdAt
            slug
            title
```

```
            excerpt
            featuredImage {
              url
            }
            categories {
              name
              slug
            }
          }
        }
      }
    }
  `;

  const result = await request(graphqlAPI, query);

  return result.postsConnection.edges;
};

export const getCategories = async () => {
  const query = gql`
    query GetGategories {
        categories {
          name
          slug
        }
    }
  `;

  const result = await request(graphqlAPI, query);

  return result.categories;
};

export const getPostDetails = async (slug) => {
  const query = gql`
    query GetPostDetails($slug : String!) {
      post(where: {slug: $slug}) {
        title
        excerpt
        featuredImage {
          url
        }
        author{
          name
          bio
          photo {
            url
          }
        }
        createdAt
        slug
```

```
        content {
          raw
        }
        categories {
          name
          slug
        }
      }
    }
  `;

  const result = await request(graphqlAPI, query, { slug });

  return result.post;
};

export const getSimilarPosts = async (categories, slug) => {
  const query = gql`
    query GetPostDetails($slug: String!, $categories: [String!]) {
      posts(
        where: {slug_not: $slug, AND: {categories_some: {slug_in: $categories}}}
        last: 3
      ) {
        title
        featuredImage {
          url
        }
        createdAt
        slug
      }
    }
  `;
  const result = await request(graphqlAPI, query, { slug, categories });

  return result.posts;
};

export const getAdjacentPosts = async (createdAt, slug) => {
  const query = gql`
    query GetAdjacentPosts($createdAt: DateTime!,$slug:String!) {
      next:posts(
        first: 1
        orderBy: createdAt_ASC
        where: {slug_not: $slug, AND: {createdAt_gte: $createdAt}}
      ) {
        title
        featuredImage {
          url
        }
        createdAt
        slug
      }
```

```
      previous:posts(
        first: 1
        orderBy: createdAt_DESC
        where: {slug_not: $slug, AND: {createdAt_lte: $createdAt}}
      ) {
        title
        featuredImage {
          url
        }
        createdAt
        slug
      }
    }
  `;

  const result = await request(graphqlAPI, query, { slug, createdAt });

  return { next: result.next[0], previous: result.previous[0] };
};

export const getCategoryPost = async (slug) => {
  const query = gql`
    query GetCategoryPost($slug: String!) {
      postsConnection(where: {categories_some: {slug: $slug}}) {
        edges {
          cursor
          node {
            author {
              bio
              name
              id
              photo {
                url
              }
            }
            createdAt
            slug
            title
            excerpt
            featuredImage {
              url
            }
            categories {
              name
              slug
            }
          }
        }
      }
    }
  `;
```

```
  const result = await request(graphqlAPI, query, { slug });

  return result.postsConnection.edges;
};

export const getFeaturedPosts = async () => {
  const query = gql`
    query GetCategoryPost() {
      posts(where: {featuredPost: true}) {
        author {
          name
          photo {
            url
          }
        }
        featuredImage {
          url
        }
        title
        slug
        createdAt
      }
    }
  `;

  const result = await request(graphqlAPI, query);

  return result.posts;
};

export const submitComment = async (obj) => {
  const result = await fetch('/api/comments', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(obj),
  });

  return result.json();
};

export const getComments = async (slug) => {
  const query = gql`
    query GetComments($slug:String!) {
      comments(where: {post: {slug:$slug}}){
        name
        createdAt
        comment
      }
    }
  `;
```

```
  const result = await request(graphqlAPI, query, { slug });

  return result.comments;
};

export const getRecentPosts = async () => {
  const query = gql`
    query GetPostDetails() {
      posts(
        orderBy: createdAt_ASC
        last: 3
      ) {
        title
        featuredImage {
          url
        }
        createdAt
        slug
      }
    }
  `;
  const result = await request(graphqlAPI, query);

  return result.posts;
};
```

Вміст файлу .env

ESLINT_NO_DEV_ERRORS=true

NEXT_PUBLIC_GRAPHCMS_ENDPOINT='https://api-eu-central-1.graphcms.com/v2/cku56f92114s901yz0ce9ah3f/master'

GRAPHCMS_TOKEN='eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6ImdjbXMtbWFpbi1wcm9kdWN0aW9uIn0.eyJ2ZXJzaW9uIjozLCJpYXQiOjE2MzI5MzEwNTIsImF1ZCI6WyJodHRwczovL2FwaS1ldS1jZW50cmFsLTEuZ3JhcGhjbXMuY29tL3YyL2NrdTU2ZjkyMTE0czkwMXl6MGNlOWFoM2YvbWFzdGVyIiwiaHR0cHM6Ly9tYW5hZ2VtZW50LW5leHQuZ3JhcGhjbXMuY29tIl0sImlzcyI6Imh0dHBzOi8vbWFuYWdlbWVudC5ncmFwaGNtcy5jb20vIiwic3ViIjoiODQyMTFmOWQtODk5My00ODQ2LWExZDItOWI2ZmUwNjVlMGI3IiwianRpIjoiY2t1NW96anh2MXRnYzAxeG5nbDNyM3F0MyJ9.Kg-yIyRma2sHtDUsDGWV-wa7DFQow98Yea7qkVIH5YmTg2C0XpwS8XSvnPuB64z09l1Jd0IKBHes_Sxv8EwMk-XlTjzqgxx3u96xBTlv5t-

UA94zlCv2E1GEGWtsCWqxHBxISXB5wHwigqS_pZYCHWjG0WwjIj8aQ2z_SxwiZErxwFCuG9l1f12_Wfs2IDmkQMA8mFsbQvOSy-MqxuMt-5o82oM9i-Usi69j2vm4veBQcKss9TWGkK-ZcVkifl_-JDrJ41qXu9G66WxnZzI2TQF9BcanwuUmsB0N_fhnkcX4BFN5Xq0OmDfSOKNKUQ6zZCy1PbU6vX4sQJ5eiRgFPXmkXtnTcsVgVsrbO2sP8SKj0KLA4diI3X5UsxduzDRCXP833_8z3AfQQmp0zUg4caDtVjHTBzBRMLNeX10PFaFQ6toeFvrFjwcNHO7jUmgdUR6kYhL8cNu1VLNuwOyZN0Lefc6kpiMTM7bYbmGwQq3FaICbyxl2hB_OQopXWlBcDGopGCrO3ZRnqGdagPUxVFcIMcQfB9kKeO78P8LbUjRinoTqBBSIclKxWNjkGU-joHAA4T5-FE3hknwYQWwffhEoFn2HNyEFLDPVFd7cDGZciCjd4app3a4YtJbFx9D18gjORam5XNI988iwo56FsbTAX-VjmYM-9M5LQ0xYsD8'

2.5. Налаштування віджетів

Для того щоби віджети виглядали нормально і правильно були поставленні, потрібно створити ще пару файлів і вписати певний код. Створені нові файли: [slug].js, Author.jsx, Comments.jsx, CommentsForm.jsx, FeaturedPostcard.jsx, AdjacentPostCard.jsx.

Вміст файлу [slug].js

```
import React from 'react';
import { useRouter } from 'next/router';

import { getCategories, getCategoryPost } from '../../services';
import { PostCard, Categories, Loader } from '../../components';

const CategoryPost = ({ posts }) => {
  const router = useRouter();

  if (router.isFallback) {
    return <Loader />;
  }

  return (
    <div className="container mx-auto px-10 mb-8">
      <div className="grid grid-cols-1 lg:grid-cols-12 gap-12">
        <div className="col-span-1 lg:col-span-8">
          {posts.map((post, index) => (
            <PostCard key={index} post={post.node} />
          ))}
        </div>
        <div className="col-span-1 lg:col-span-4">
          <div className="relative lg:sticky top-8">
            <Categories />
          </div>
        </div>
      </div>
    </div>
  );
};
export default CategoryPost;

export async function getStaticProps({ params }) {
  const posts = await getCategoryPost(params.slug);

  return {
    props: { posts },
  };
```

```
}

  const categories = await getCategories();
  return {
    paths: categories.map(({ slug }) => ({ params: { slug } })),
    fallback: true,
  };
}
```

Вміст файлу Author.jsx

```
import React from 'react';
import Image from 'next/image';

import { grpahCMSImageLoader } from '../util';

const Author = ({ author }) => (
  <div className="text-center mt-20 mb-8 p-12 relative rounded-lg bg-black bg-
opacity-20">
    <div className="absolute left-0 right-0 -top-14">
      <Image
        unoptimized
        loader={grpahCMSImageLoader}
        alt={author.name}
        height="100px"
        width="100px"
        className="align-middle rounded-full"
        src={author.photo.url}
      />
    </div>
    <h3 className="text-white mt-4 mb-4 text-xl font-bold">{author.name}</h3>
    <p className="text-white text-ls">{author.bio}</p>
  </div>
);

export default Author;
```

Вміст файлу Comments.jsx

```
import React, { useEffect, useState } from 'react';
import moment from 'moment';
import parse from 'html-react-parser';

import { getComments } from '../services';

const Comments = ({ slug }) => {
  const [comments, setComments] = useState([]);

  useEffect(() => {
```

```
    getComments(slug).then((result) => {
      setComments(result);
    });
  }, []);

  return (
    <>
      {comments.length > 0 && (
        <div className="bg-white shadow-lg rounded-lg p-8 pb-12 mb-8">
          <h3 className="text-xl mb-8 font-semibold border-b pb-4">
            {comments.length}
            {' '}
            Comments
          </h3>
            {comments.map((comment, index) => (
              <div key={index} className="border-b border-gray-100 mb-4 pb-4">
                <p className="mb-4">
                  <span className="font-semibold">{comment.name}</span>
                  {' '}
                  on
                  {' '}
                  {moment(comment.createdAt).format('MMM DD, YYYY')}
                </p>
                <p className="whitespace-pre-line text-gray-600 w-
full">{parse(comment.comment)}</p>
              </div>
            ))}
        </div>
      )}
    </>
  );
};
```

Вміст файлу CommentsForm.jsx

```
import React, { useState, useEffect } from 'react';
import { submitComment } from '../services';

const CommentsForm = ({ slug }) => {
  const [error, setError] = useState(false);
  const [localStorage, setLocalStorage] = useState(null);
  const [showSuccessMessage, setShowSuccessMessage] = useState(false);
  const [formData, setFormData] = useState({ name: null, email: null, comment:
null, storeData: false });

  useEffect(() => {
    setLocalStorage(window.localStorage);
    const initalFormData = {
      name: window.localStorage.getItem('name'),
      email: window.localStorage.getItem('email'),
```

```
      storeData: window.localStorage.getItem('name') ||
window.localStorage.getItem('email'),
    };
    setFormData(initalFormData);
  }, []);

  const onInputChange = (e) => {
    const { target } = e;
    if (target.type === 'checkbox') {
      setFormData((prevState) => ({
        ...prevState,
        [target.name]: target.checked,
      }));
    } else {
      setFormData((prevState) => ({
        ...prevState,
        [target.name]: target.value,
      }));
    }
  };

  const handlePostSubmission = () => {
    setError(false);
    const { name, email, comment, storeData } = formData;
    if (!name || !email || !comment) {
      setError(true);
      return;
    }
    const commentObj = {
      name,
      email,
      comment,
      slug,
    };

    if (storeData) {
      localStorage.setItem('name', name);
      localStorage.setItem('email', email);
    } else {
      localStorage.removeItem('name');
      localStorage.removeItem('email');
    }

    submitComment(commentObj)
      .then((res) => {
        if (res.createComment) {
          if (!storeData) {
            formData.name = '';
            formData.email = '';
          }
          formData.comment = '';
          setFormData((prevState) => ({
```

```
            ...prevState,
            ...formData,
          }));
          setShowSuccessMessage(true);
          setTimeout(() => {
            setShowSuccessMessage(false);
          }, 3000);
        }
      });
  };

  return (
    <div className="bg-white shadow-lg rounded-lg p-8 pb-12 mb-8">
      <h3 className="text-xl mb-8 font-semibold border-b pb-4">Leave a Reply</h3>
      <div className="grid grid-cols-1 gap-4 mb-4">
        <textarea value={formData.comment} onChange={onInputChange} className="p-
4 outline-none w-full rounded-lg h-40 focus:ring-2 focus:ring-gray-200 bg-gray-
100 text-gray-700" name="comment" placeholder="Comment" />
      </div>
      <div className="grid grid-cols-1 lg:grid-cols-2 gap-4 mb-4">
        <input type="text" value={formData.name} onChange={onInputChange}
className="py-2 px-4 outline-none w-full rounded-lg focus:ring-2 focus:ring-gray-
200 bg-gray-100 text-gray-700" placeholder="Name" name="name" />
        <input type="email" value={formData.email} onChange={onInputChange}
className="py-2 px-4 outline-none w-full rounded-lg focus:ring-2 focus:ring-gray-
200 bg-gray-100 text-gray-700" placeholder="Email" name="email" />
      </div>
      <div className="grid grid-cols-1 gap-4 mb-4">
        <div>
          <input checked={formData.storeData} onChange={onInputChange}
type="checkbox" id="storeData" name="storeData" value="true" />
          <label className="text-gray-500 cursor-pointer" htmlFor="storeData">
Save my name, email in this browser for the next time I comment.</label>
        </div>
      </div>
      {error && <p className="text-xs text-red-500">All fields are mandatory</p>}
      <div className="mt-8">
        <button type="button" onClick={handlePostSubmission}
className="transition duration-500 ease hover:bg-indigo-900 inline-block bg-pink-
600 text-lg font-medium rounded-full text-white px-8 py-3 cursor-pointer">Post
Comment</button>
        {showSuccessMessage && <span className="text-xl float-right font-semibold
mt-3 text-green-500">Comment submitted for review</span>}
      </div>
    </div>
  );
};
```

Вміст файлу FeaturedPostCard.jsx

```
import React from 'react';
import moment from 'moment';
import Image from 'next/image';
import Link from 'next/link';

const FeaturedPostCard = ({ post }) => (
  <div className="relative h-72">
    <div className="absolute rounded-lg bg-center bg-no-repeat bg-cover shadow-md
inline-block w-full h-72" style={{ backgroundImage:
`url('${post.featuredImage.url}')` }} />
    <div className="absolute rounded-lg bg-center bg-gradient-to-b opacity-50
from-gray-400 via-gray-700 to-black w-full h-72" />
    <div className="flex flex-col rounded-lg p-4 items-center justify-center
absolute w-full h-full">
      <p className="text-white mb-4 text-shadow font-semibold text-
xs">{moment(post.createdAt).format('MMM DD, YYYY')}</p>
      <p className="text-white mb-4 text-shadow font-semibold text-2xl text-
center">{post.title}</p>
      <div className="flex items-center absolute bottom-5 w-full justify-center">
        <Image
          unoptimized
          alt={post.author.name}
          height="30px"
          width="30px"
          className="align-middle drop-shadow-lg rounded-full"
          src={post.author.photo.url}
        />
        <p className="inline align-middle text-white text-shadow ml-2 font-
medium">{post.author.name}</p>
      </div>
    </div>
    <Link href={`/post/${post.slug}`}><span className="cursor-pointer absolute w-
full h-full" /></Link>
  </div>
);
```

Вміст файлу AdjacentPostCard.jsx

```
import React from 'react';
import moment from 'moment';
import Link from 'next/link';

const AdjacentPostCard = ({ post, position }) => (
  <>
    <div className="absolute rounded-lg bg-center bg-no-repeat bg-cover shadow-md
inline-block w-full h-72" style={{ backgroundImage:
`url('${post.featuredImage.url}')` }} />
    <div className="absolute rounded-lg bg-center bg-gradient-to-b opacity-50
from-gray-400 via-gray-700 to-black w-full h-72" />
```

```
    <div className="flex flex-col rounded-lg p-4 items-center justify-center
absolute w-full h-full">
      <p className="text-white text-shadow font-semibold text-
xs">{moment(post.createdAt).format('MMM DD, YYYY')}</p>
      <p className="text-white text-shadow font-semibold text-2xl text-
center">{post.title}</p>
    </div>
    <Link href={`/post/${post.slug}`}><span className="z-10 cursor-pointer
absolute w-full h-full" /></Link>
    {position === 'LEFT' && (
      <div className="absolute arrow-btn bottom-5 text-center py-3 cursor-pointer
bg-pink-600 left-4 rounded-full">
        <svg xmlns="http://www.w3.org/2000/svg" className="h-6 w-6 text-white w-
full" fill="none" viewBox="0 0 24 24" stroke="currentColor">
          <path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2"
d="M10 19l-7-7m0 0l7-7m-7 7h18" />
        </svg>
      </div>
    )}
    {position === 'RIGHT' && (
      <div className="absolute arrow-btn bottom-5 text-center py-3 cursor-pointer
bg-pink-600 right-4 rounded-full">
        <svg xmlns="http://www.w3.org/2000/svg" className="h-6 w-6 text-white w-
full" fill="none" viewBox="0 0 24 24" stroke="currentColor">
          <path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2"
d="M14 5l7 7m0 0l-7 7m7-7H3" />
        </svg>
      </div>
    )}
  </>
);
```

ВИСНОВКИ

Завдяки цій дипломній роботі було потрачено чимало зусиль, на те щоби її виконати. В загальному ця робота дала зрозуміти, що насправді все є не так просто. Звісно і були свої плюси, наприклад: нові навички, нове бачення, цікавий процес роботи і нові знання. Особливо більше зацікавило те, що чим більше ти будеш мати навичок, знань то перед тобою може багато, що нового відкритися, особливо можливостей. Тому бажано ніколи не стояти на місці а завжди рухатися вперед. Відкривати для себе все більше і більше нового. Ця робота змогла доказати те, що навіть проста людина багато чому може добитися і тому ніколи не потрібно роботи шагу назад або взагалі здаватися. Потрібно завжди іти вперед.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ І ЛІТЕРАТУРИ

1. Сучасний підручник JavaScript (Електронний ресурс). Режим доступу: https://uk.javascript.info
2. React (Електронний ресурс). Режим досступу: https://ru.reactjs.org
3. Керівництво по GraphQL (Електронний ресурс). Режим доступу: https://dou.ua/lenta/articles/working-with-qraphql/
4. Next.js:детальний посібник (Електронний ресурс). Режим доступу: https://habr.com/ru/company/timeweb/blog/588498/
5. Tailwind CSS як впровадити його на сайт або React-додаток? (Електронний ресурс). Режим доступу: https://medium.com/nuances-of-programming/что-такое-tailwind-css-и-как-внедрить-его-на-сайт-или-в-react-приложение-cfff5f8168f4
6. Документація Node.js (Електронний ресурс). Режим доступу: https://nodejs.org/uk/docs/
7. Програмування веб-додатків (фронт-енд та бек-енд): навч.посіб. - Львівська політехніка / Мельник Р.А., 2018 – 248 с.

ДОДАТКИ

Додаток А

Graphcms

Додаток В

Веб-застосунок