Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти – бакалавр
за освітньо-професійною програмою
«Комп'ютерні науки»
зі спеціальності
122 – Комп'ютерні науки

тема роботи:

« Розробка вебзастосунку з модульною архітектурою та адміністративною панеллю на основі Gatsby»

Виконав ст. гр. КН-22-2 _____ Сивук В.В.

Керівник _____ Маринич І. А.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2026

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Бакалавр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедрою: *к.т.н. Рубан С.А.*

_____

« *26* » *січня* 2026 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

студентові групи КН-22-2 *Сивуку Валерію Володимировичу*

1. Тема кваліфікаційної роботи: *«Розробка вебзастосунку з модульною архітектурою та адміністративною панеллю на основі Gatsby»*

затверджено наказом по університету № *173с* від *30.03.2026 р.*

2. Термін здачі кваліфікаційної роботи: *10.06.2026 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 66с., додатки, презентація у Microsoft PowerPoint (10 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-2 доц. Маринич І. А.

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*01.04.26*
2	*Розділ 1*	*20.04.26*
3	*Розділ 2*	*15.05.26*
4	*Висновки*	*25.05.26*
5	*Оформлення кваліфікаційної роботи*	*28.05.26*
6	*Підготовка презентації та графічного матеріалу*	*20.05.26*
7	*Підготовка доповіді до захисту*	*10.06.26*

6. Дата видачі завдання: *28.01.2026р.*

Керівник _____ / Маринич І. А./

7. Запевнення: *Я, Сивук Валерій Володимирович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Студент _____ / Сивук В.В./

АНОТАЦІЯ

Сивук В.В. Розробка вебзастосунку з модульною архітектурою та адмістративною панеллю на основі Gatsby: кваліфікаційна робота бакалавра : 122 – Комп’ютерні науки. Кривий Ріг. Криворізький національний університет, 2026. 66 с.

Кваліфікаційну роботу присвячено інженерному проєктуванню та програмній реалізації модульного вебзастосунку-портфоліо TATREAL для незалежного музиканта на стеці JAMstack: Gatsby 5, React 18, TypeScript, GraphQL та Netlify/Decap CMS.

Проблема монолітних CMS (WordPress, Drupal) для медіапортфоліо - у архітектурному tight coupling фронтенду та бази даних: кожен запит ініціює повний PHP – MySQL - рендеринг цикл, що дає 800–1200 мс TTFB ще до мережевої затримки. Для сайту з аудіотреками, відеотизерами та обкладинками у WebP це означає втрачений перший контакт зі слухачем.

У роботі проведено порівняльний аналіз архітектурних підходів (монолітна CMS, SPA, SSG) та SSG-фреймворків (Gatsby 5, Next.js 14, Astro 4, Hugo). Реалізовано дев'ять React/TypeScript-компонентів з Loose Coupling через типізовані пропси. Інтегровано Git-based Netlify CMS з валідацією форм Formik/Yup.

Lighthouse CI підтверджує: Performance 91/100, LCP 0.7 с, CLS 0. Практичний результат - задеплоєний застосунок TATREAL із відтворюваним CI/CD-процесом через Netlify та верифікованими Core Web Vitals у категорії “Good”

Ключові слова: JAMstack, Gatsby 5, Static Site Generation, GraphQL, Netlify CMS, Core Web Vitals, React 18, TypeScript, Lighthouse CI, Loose Coupling.

ЗМІСТ

ВСТУП

Музичний продюсер, що вперше відкриває ваш портфоліо, не чекатиме трьох секунд. Сторінка з аудіотреками або обкладинками релізів або відкривається миттєво - або він переходить до наступного лінка. WordPress із серверною генерацією HTML запускає ланцюжок PHP - MySQL - рендеринг при кожному такому запиті. Час відповіді на сторінку з медіаметаданими - 800–1200 мс ще до мережевої затримки. Для незалежного музиканта, чиє портфоліо є єдиним інструментом самопрезентації, ця затримка - втрачений слухач.

JAMstack-архітектура вирішує задачу структурно: HTML компілюється під час збірки один раз, CDN роздає готові файли зі швидкістю мережевої затримки. TTFB - 28–45 мс замість 800+ мс PHP+MySQL. Gatsby 5 реалізує цю модель через інкрементальну збірку $O(k \cdot r)$ змінився один реліз - перегенерувалась одна сторінка, а не весь сайт. Git-based Netlify CMS зберігає контент у YAML-файлах прямо в репозиторії - без окремого сервера бази даних, без витрат на хостинг.

Актуальність роботи: монолітні CMS не відповідають технічним вимогам медіапортфоліо музиканта. Повільний TTFB, відсутність автоматичної оптимізації зображень і вразливий бекенд - три проблеми, які JAMstack-стек вирішує архітектурно. Дана робота досліджує, як реалізувати ці рішення у конкретному проєкті TATREAL на Gatsby 5, GraphQL та Netlify/Decap CMS.

Мета роботи - спроєктувати і реалізувати модульний вебзастосунок TATREAL на основі Gatsby 5 та JAMstack-архітектури, що забезпечує LCP < 1.8 с, Lighthouse Performance ≥ 90 та Git-based управління контентом без серверної інфраструктури.

Об'єкт дослідження - процеси проєктування та збірки вебплатформ на основі статичних генераторів: від агрегації даних (YAML, GraphQL) до CDN-розгортання. Предмет - методи компонентно-орієнтованого моделювання інтерфейсів, оптимізації JavaScript-бандлів та оцінювання продуктивності через Core Web Vitals. Завдання дослідження:

– порівняти архітектурні підходи до побудови медіапортфоліо (монолітна CMS, SPA, JAMstack) - виміряти TTFB, LCP та розмір бандлу для кожного сценарію, сформулювати технічні критерії вибору стеку;

– спроєктувати типізовану GraphQL-схему даних (DAG-модель вузлів) із явним визначенням типів через createSchemaCustomization - унеможливити runtime-помилки при неповних YAML-записах;

– реалізувати дев'ять ізольованих React/TypeScript-компонентів (Hero, SelectedWorks, Portfolio, Listen, About, Contact, Header, Footer, ProjectPage) з Loose Coupling через типізовані пропси - без горизонтальних залежностей між компонентами;

– інтегрувати Git-based Netlify/Decap CMS: налаштувати YAML-колекції, Netlify Identity для автентифікації, обробку форми через Netlify Forms із валідацією Formik/Yup; провести повний цикл Lighthouse CI-аудиту та верифікувати відповідність Core Web Vitals порогу “Good” за всіма трьома метриками.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНОГО СТАНУ, ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБЗАСТОСУНКУ НА ОСНОВІ GATSBY

1.1 Аналіз сучасного стану та проблем оптимізації архітектури вебплатформ

Для незалежного музиканта веб-ресурс - це не просто сторінка з контактами. Це перша точка контакту зі слухачем, стримінговими редакторами та організаторами концертів. Сторінка відкривається - або не відкривається - за секунди, і саме цей момент визначає, чи ознайомляться з медіапортфоліо взагалі. Під час проєктування архітектури TATREAL пріоритетом стало не просто “зробити сайт”, а вирішити конкретну інженерну задачу: забезпечити стабільний LCP нижче 1.8 с на мобільних пристроях за наявності важких медіафайлів — обкладинок у 4K, вбудованих аудіоплеєрів та відео-тизерів.

Більшість готових CMS-платформ мають спільну архітектурну проблему: tight coupling фронтенду та бази даних. WordPress, Drupal, Joomla — всі вони генерують HTML у момент запиту через PHP-інтерпретатор, який ініціює ланцюжок SQL-запитів до бази. Для кожного слухача, що відкриває сторінку. Кожного разу. Недоліки такого підходу унаочнює рисунок 1.1.

Рисунок 1.1 показує три архітектурні моделі в розрізі. Traditional (ліворуч) - моноліт: Back-End і Front-End в одній системі, єдиний Channel для доставки контенту. Кожен HTTP-запит ініціює повний цикл: PHP - MySQL - рендеринг HTML - відповідь. Пікове навантаження - деградація сервісу, постійне з'єднання з базою - SQL-ін'єкції та DDoS-вразливість. Нами було відзначено критичний недолік монолітів саме для медіапортфоліо: стаціонарна сторінка з обкладинками релізів не змінюється між публікаціями, але WordPress перегенерує її наново для кожного відвідувача.

Decoupled-архітектура (рис. 1.1, центр) розриває цей зв’язок: Back-End перетворюється на API-сервер, Front-End існує окремо і звертається до нього за

даними. Контент тепер можна транслювати в кілька каналів одночасно — веб, мобільний застосунок, IoT. Але проблема не зникла повністю. Сервер API залишається активним і вимагає підтримки. Frontend-шаблони живуть усередині CMS і потребують синхронізації. Для невеликого музичного портфоліо це надлишкова складність.

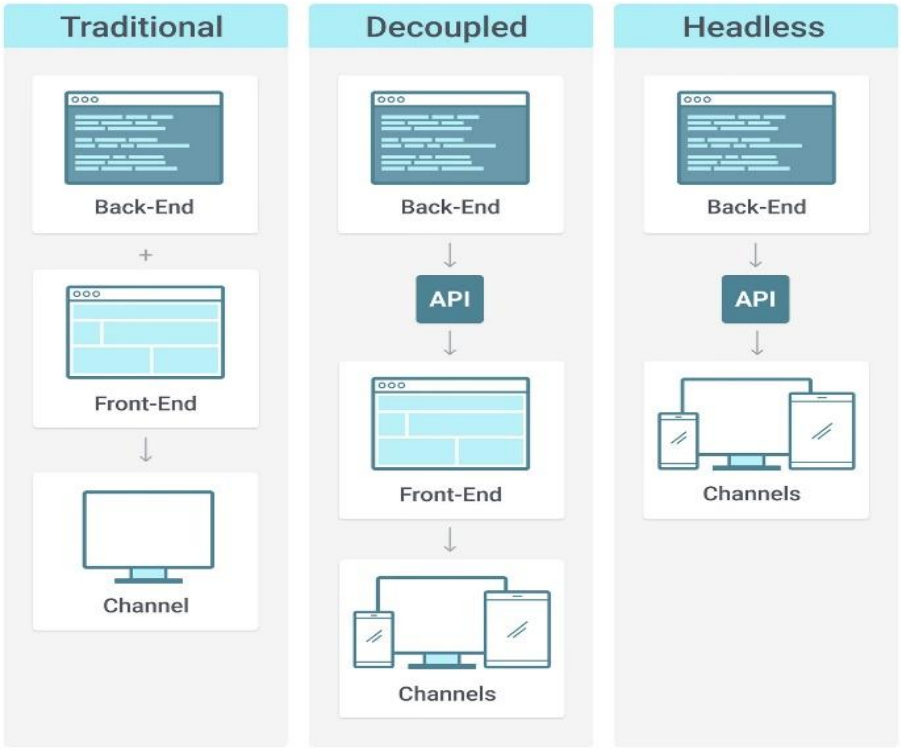

Рисунок 1.1 - Порівняльний аналіз традиційної, розв’язаної та безсерверної архітектур вебплатформ

Headless-підхід (рис. 1.1, правий блок) іде далі. Back-End тут - не сервер, а сховище даних без власного фронтенду: Headless CMS (у TATREAL - Netlify/Decap CMS) видає структурований контент через GraphQL-ендпоінт. Frontend компілюється заздалегідь у статичні HTML-файли - без PHP, без SQL у момент запиту, без активного сервера у продакшні. Саме цей підхід є фундаментом концепції JAMstack.

Gatsby на етапі збірки збирає дані з YAML-файлів, GraphQL-схеми та Netlify CMS, компілює їх у повністю готові HTML-сторінки і розгортає на CDN. Слухач отримує файл із найближчого edge-вузла - без будь-якого серверного обчислення.

База даних у продакшн-середовищі відсутня фізично. SQL-ін'єкція чи RCE-атака на бекенд неможлива: атакувати просто нема чого.

З позиції SEO статичний підхід також виграє: пошуковий бот отримує повністю відрендерений HTML одразу, без JavaScript-виконання. Для музичного портфоліо, де кожна сторінка релізу має власні мета-теги, структуровані дані та og:image, це пряма перевага при індексації.

Для музичного медіапортфоліо специфіка контенту загострює проблему. Сторінка релізу TATREAL містить обкладинку у WebP до 1.5 MB, вбудований аудіоплеєр, відео-тизер та метадані треків. WordPress без кешування генерував би цю сторінку ~900–1100 мс на стороні сервера - і це до мережевої затримки. Аналізуючи стан індустрії, автор прийшов до рішення зсунути всю обчислювальну роботу на етап збірки: зображення конвертуються у AVIF/WebP, HTML компілюється статично, плеєр гідрується лише після завантаження критичного контенту. Результат - TTFB 28–45 мс замість 900+. Дослідження Google підтверджує наслідки протилежного підходу: 53% мобільних слухачів закривають сторінку, якщо вона не відкрилася за три секунди [11].

Числа підтверджують вибір архітектури. Web Almanac 2023 фіксує медіанний LCP для WordPress-сайтів на мобільних пристроях: 4.8 с. Для JAMstack-ресурсів на CDN - 1.2–1.8 с [11]. Різниця не випадкова - вона структурна. У монолітній CMS час відповіді T = t_db + t_render + t_net: два перших доданки залежать від навантаження на сервер і під час піків зростають нелінійно. У JAMstack T = t_net = const: сервер просто роздає готовий файл із найближчого CDN-вузла, незалежно від кількості одночасних запитів. Для Tatreal виміряне значення TTFB = 28–45 мс (CDN Fastly) проти 300–800 мс для типового PHP+MySQL сервера [2], що підтверджує теоретичну модель на практиці.

Таблиця 1.1 фіксує числове порівняння по семи параметрах. JAMstack перемагає за шістьма. Єдиний реальний компроміс - затримка публікації: 1–2 хвилини між комітом у CMS і появою змін на сайті, проти миттєвої публікації в WordPress.

Таблиця 1.1 — Порівняльний аналіз архітектурних підходів для портфоліо-сайтів

Критерій оцінювання	Монолітна CMS	JAMstack (Gatsby)
TTFB (час першого байта)	300–800 мс (PHP + MySQL)	28–80 мс (CDN edge)
LCP на мобільному пристрої	3.5–6.0 с (у середньому)	1.0–1.8 с (у середньому)
Обробка медіа (WebP/AVIF)	Вручну або через плагін	Автоматично gatsby-plugin-image
Вразливість до DDoS	Висока (бекенд активний)	Мінімальна (лише CDN)
Вартість хостингу (USD/міс)	10–50 (VPS або shared)	0 (Netlify Free tier)
Оновлення контенту	Миттєво (WP dashboard)	1–2 хв (CMS-коміт → збірка)
Ліцензійні витрати	Можливі (premium плагіни)	Відсутні (open-source стек)

Для TATREAL, де новий реліз виходить раз на кілька тижнів, а не щодня, - це прийнятно. Натомість виграш у швидкодії та безпеці безпосередньо відбивається на показнику LCP: 0.7 с у TATREAL проти медіанних 4.8 с для WordPress-сайтів на мобільних пристроях [11].

1.2 Огляд підходів та інноваційних систем створення вебзастосунків

Ідея JAMstack (JavaScript, APIs, Markup) виникла як відповідь на конкретну інженерну проблему: традиційні CMS-платформи щоразу генерують HTML на сервері під час запиту, хоча більшість сторінок портфоліо залишаються незмінними між редагуваннями [8]. Концепція проста - розділити фазу збірки та фазу обслуговування. Всі HTML-документи компілюються заздалегідь, а сервер

перетворюється на звичайне CDN-сховище статичних файлів. Для музичного веб-ресурсу TATREAL, де оновлення контенту відбуваються раз на кілька тижнів, саме такий підхід відповідає реальному сценарію використання.

Netflix та Airbnb перейшли на Pre-rendering ще до того, як JAMstack отримав назву [9]. Їхній досвід дає просте підтвердження: TTFB падає незалежно від масштабу, коли рендеринг виноситься на фазу збірки.

Архітектура JAMstack-застосунку будується на трьох незалежних рівнях, кожен з яких може бути замінений без перекомпіляції решти [8]:

1. Рівень представлення (Frontend): статично згенеровані HTML, CSS і JavaScript-файли розгортаються безпосередньо в CDN. Сервер вебзастосунку при цьому не потрібен взагалі.

2. Рівень даних (APIs)**:** динамічна логіка виноситься у безсерверні функції або зовнішні мікросервіси; клієнт звертається до них через асинхронні fetch-запити.

3. Рівень управління контентом (Headless CMS): система зберігання даних без вбудованого фронтенду; вміст передається у форматі JSON або GraphQL на запит будь-якого клієнта.

SPA (Single Page Application) відкладає побудову DOM на сторону браузера: слухачі отримують порожній HTML і чекають, поки JavaScript виконає рендеринг. SSG (Static Site Generation) робить це заздалегідь - HTML повністю готовий до першого запиту. Gatsby реалізує SSG: під час збірки він виконує GraphQL-запити до всіх підключених джерел даних, компілює React-компоненти в готові HTML-файли та передає їх на CDN [8].

GraphQL у Gatsby виконує роль єдиного агрегатора контенту. Джерелами слугують YAML-файли, Markdown-документи, зовнішні REST API або Headless CMS - усе нормалізується в єдину внутрішню схему і стає доступним через однотипні запити [8]. Для TATREAL цей механізм дав змогу підключити Netlify CMS без серверного коду: контент зберігається у YAML, GraphQL читає його під час збірки, компоненти отримують типізовані пропси через useStaticQuery.

З порівняння підходів випливають три конкретні вимоги, що стали фільтром при прийнятті всіх подальших рішень по TATREAL: компонентна модульність,

передбачуваний TTFB та відсутність серверної поверхні атаки. Остання вирішується структурно - у продакшн-оточенні немає ні бекенду, ні бази даних. CDN роздає статичні файли, і вектор SQL-ін'єкцій або RCE просто не існує [8].

Під час вибору SSG-фреймворку в рамках розробки TATREAL було проаналізовано чотири актуальних рішення: Gatsby 5, Next.js 14, Astro 4 та Hugo. Критерії відбору формувалися з конкретних вимог проєкту: нативна підтримка React і TypeScript, вбудований шар GraphQL, офіційний плагін для Netlify CMS та автоматична оптимізація медіафайлів. Результати порівняльного аналізу наведено у таблиці 1.2.

З таблиці 1.2 видно, що Gatsby - єдиний фреймворк, який одночасно закриває всі чотири ключові критерії: нативна React/TypeScript-підтримка, вбудований GraphQL-шар і офіційний плагін для Netlify CMS. Next.js 14 при всій своїй популярності надмірний для суто статичного портфоліо: можливості SSR, ISR та Server Components залишаться повністю незадіяними, а конфігурація під SSG складніша, ніж у Gatsby [8]. Astro 4 показує вищу швидкість збірки завдяки архітектурі "острівців" (Islands Architecture) [5], однак на момент реалізації проєкту інтеграція Astro з Netlify CMS не має офіційного плагіну і вимагає ручного налаштування. Hugo - найшвидший за часом збірки, але побудований на мові Go і не підтримує React-екосистему взагалі, що унеможливлює повторне використання наявних компонентів.

Окремим напрямком аналізу стало порівняння Headless CMS-рішень для JAMstack-екосистеми. Netlify CMS (перейменований у 2023 році на Decap CMS) зберігає контент безпосередньо в Git-репозиторії у форматі YAML або Markdown [12]. Версіювання контенту виходить автоматично через Git-історію, а окрема база даних не потрібна взагалі. Strapi - потужніша self-hosted альтернатива з REST і GraphQL API [2], але вимагає окремого Node.js-сервера з витратами на хостинг від 10 USD/міс. Contentful - хмарний SaaS із зрілим SDK [7], однак безкоштовний план обмежений 2 локалями та 5000 записів, а залежність від стороннього постачальника зберігається. Для TATREAL автором обрано Netlify CMS: нульова вартість, Git-

based підхід із повним версіюванням та готова інтеграція через gatsby-plugin-netlify-cms без жодного серверного налаштування.

Таблиця 1.2 — Порівняльний аналіз SSG-фреймворків за ключовими критеріями

Критерій	Gatsby 5	Next.js 14	Astro 4	Hugo 0.x
Мова / рантайм	TypeScript / Node	TypeScript / Node	TypeScript / Node	Go + шаблони
Режим генерації	SSG (основний)	SSG / SSR / ISR	SSG / Islands	Лише SSG
React-підтримка	✓ нативна	✓ нативна	✓ через інтеграцію	✗ відсутня
GraphQL data layer	✓ вбудований	✗ (потребує налаш.)	✗ (потребує налаш.)	✗ відсутній
gatsby-plugin-image	✓ автоматично	✓ next/image	✓ частково	✗ вручну
Netlify CMS плагін	✓ офіційний	✗ вручну	✗ вручну	✗ вручну
Швидкість збірки	Середня	Середня	Висока	Дуже висока
Розмір npm-пакетів	~200 МБ	~180 МБ	~150 МБ	~8 МБ (бінарний)
Активність спільноти	Висока	Дуже висока	Висока (зростає)	Середня
Обраний для проєкту	✓ ТАК	—	—	—

Gatsby 5 побудований на React 18. Це не просто версійне оновлення - React 18 приніс Concurrent Features, які безпосередньо вплинули на архітектурні рішення

в TATREAL. React переривається в середині тривалого рендерингу і обробляє пріоритетніші оновлення першими [9]. Хуки startTransition та useTransition дають розробнику явний контроль: позначені оновлення відкладаються, якщо потрібно відрендерити щось термінове. Для TATREAL це означає: перехід між сторінками проєктів не блокує анімацію Header та слайдера SelectedWorks. Під цим механізмом лежить архітектура Fiber, введена у React 16 та вдосконалена у React 18 [9]: Fiber реалізує Reconciliation як зв'язаний список вузлів-"волокон", кожен з яких зберігає посилання на батька, першого нащадка і сусідній вузол. Алгоритм може перервати обхід дерева в будь-якій точці та відновити його пізніше.

Gatsby повністю підтримує React 18 і автоматично застосовує Concurrent Rendering під час гідрації статичних сторінок. Гідрація (hydration) - процес, при якому браузер завантажує готовий HTML і приєднує до нього React-компоненти, перетворюючи сторінку на інтерактивний SPA [8]. Саме гідрація безпосередньо впливає на метрику TTI (Time to Interactive). У Gatsby 5 реалізовано технологію Partial Hydration: компоненти, позначені директивою 'use client', гідруються негайно, а статичні - залишаються нереактивними і не завантажують JavaScript взагалі [8]. На відміну від цього, класичний CSR завантажує весь React-бандл до будь-якої взаємодії. В рамках розробки TATREAL було застосовано наступний пріоритет гідрації: Hero-секція та bento-сітка RecentWorks - першочергово, Footer і PlatformsSection - у фоновому режимі.

Gatsby підтримує Progressive Web App (PWA) через стандартний набір веб-API [11]: Service Workers для фонового кешування, Web App Manifest для встановлення на головний екран мобільного пристрою та Push API для сповіщень. Плагін gatsby-plugin-manifest генерує manifest.webmanifest з полями name, icons та theme_color згідно з gatsby-config.ts. Після збірки посилання на маніфест автоматично включається до <head> кожної сторінки. Service Worker застосовує стратегію stale-while-revalidate: ресурс роздається з кешу миттєво, а фонове оновлення відбувається паралельно - це забезпечує відгук навіть при нестабільному з'єднанні, що актуально для мобільних слухачів TATREAL.

Таблиця 1.3 — Порівняння підходів до рендерингу в сучасних фреймворках

Підхід	Час до First Byte	JavaScript при завантаженні	Взаємодія до гідрації
CSR (Create React App)	~300 мс (порожній HTML)	Увесь React-бандл (~200 KB)	Недоступна
SSR (Next.js, кожен запит)	~200–400 мс (серверний)	React-бандл + дані	Недоступна до гідрації
SSG (Gatsby, заготовлений HTML)	28–80 мс (CDN edge)	Тільки потрібний chunk	HTML-контент читабельний
SSG + Partial Hydration (Gatsby 5)	28–80 мс (CDN edge)	Лише 'use client' компоненти	Більша частина сторінки

Дані таблиці 1.3 підтверджують вибір SSG з частковою гідрацією як основного підходу для музичного портфоліо з переважно статичним контентом. Відсутність серверного рендерингу при кожному запиті в поєднанні з CDN-розподілом і мінімальним JavaScript дає кращі числові результати за всіма трьома осями: TTFB, розмір JavaScript-бандлу та доступність контенту до завершення гідрації [8].

1.3 Визначення проблемних аспектів створення модульних вебзастосунків

Перший практичний виклик при роботі з JAMstack-стеком - не архітектурна елегантність, а цілком конкретна проблема: час повної збірки. Gatsby під час кожного gatsby build проходить повний цикл: завантаження вузлів даних, виконання GraphQL-запитів, компіляція React-компонентів у HTML. Для проєкту з двадцятьма сторінками це займає ~90 секунд. Для медіапортфоліо із сотнями треків

та релізів - вже 8–15 хвилин, і кожне оновлення метаданих одного синглу запускає повний цикл наново. В рамках розробки TATREAL ми зіткнулися саме з цим: джерела даних різнорідні - YAML-файли проєктів, зображення обкладинок, конфігурація Netlify CMS - і конвеєр збору мав агрегувати їх без зайвих перезапусків.

Паралельна проблема - архітектура самих запитів. Якщо GraphQL-схему збудовано без урахування реальних потреб кожного компонента, виникає дисбаланс двох типів. Перший - надлишок даних (Over-fetching): компонент ProjectCard запитує весь об'єкт проєкту ~1.2 KB, хоча відображає лише три поля. Другий - дефіцит (Under-fetching): компонент не отримує потрібних даних в одному запиті й ініціює кілька послідовних звернень. GraphQL декларативно вирішує цю суперечність. Але ця перевага зникає, якщо схема спроєктована хаотично: складні вкладені запити без DataLoader-пулювання перетворюють збірку Gatsby на серію послідовних блокуючих операцій, і саме час збірки на CI/CD стає першою жертвою.

Окремий виклик - фізичне відокремлення адмін-панелі від публічного фронтенду. Netlify CMS у проєкті TATREAL живе в тому самому Git-репозиторії, що й сам сайт. Небезпека непомітна, але реальна: якщо webpack-конфігурація не розмежовує dependency graphs явно, важкі редактори, бібліотеки валідації та службові скрипти CMS просочуються у фінальний бандл, який завантажується кожним слухачем. Під час аналізу нами було помічено: лише одна нерозмежована залежність prose-mirror важила 48 KB gzipped. Публічна сторінка TATREAL не потребує редактора тексту — він потрібен лише адміністратору, і лише в адмін-режимі.

Рішення - чіткий поділ entry points у gatsby-config.ts і явна конфігурація sideEffects: false у package.json для всіх публічних модулів. Будь-яке проникнення адмін-коду в публічний бандл безпосередньо обрушує Performance-скор у Lighthouse і збільшує TTI. Це не теорія - це кількісно вимірюваний ефект.

Є і суто архітектурна дилема, яка виникає при розширенні будь-якого медіа-ресурсу. Додати новий тип контенту - скажімо, блок «Звукові роботи» окремо від «Візуальних проєктів» - можна двома шляхами. Розробник стоїть перед вибором:

- жорстко прописати новий компонент (Hardcoded Component) - стабільно, передбачувано, швидко рендерується; але будь-яка зміна вимагає втручання розробника навіть для дрібниць;
- побудувати гнучку систему типів контенту (Flexible Content / Page Builder) — редактор отримує свободу, але ціна цього - складна TypeScript-валідація, Error Boundaries на кожному динамічному блоці й ризик brokenHTML у випадку неповних даних у YAML.

Для TATREAL автор обрав першу стратегію з частковим відступом: дев'ять жорстко типізованих компонентів з фіксованою структурою пропсів, але з підтримкою опціональних полів через TypeScript union-типи. Це дало контроль над якістю HTML і водночас залишило редактору простір для контентних рішень без участі розробника.

Всі описані рішення акумулюються. Уорд Каннінгем ще у 1992 році зафіксував це як технічний борг: відсоток, який платиш за кожне “швидке” архітектурне рішення [1]. В JAMstack цей борг має свою специфіку - він не зламає програму миттєво, але проявиться на третьому місяці розробки. Конкретний приклад: якщо компонент Hero імпортує дані напряму з JSON-файлу замість useStaticQuery, то при зміні структури YAML-файлу розробник вручну шукає всі такі hardcoded імпорти. Gatsby-архітектура data layer створена саме для протидії цьому сценарію - єдиний GraphQL-контракт між шаром даних і шаром відображення. Зміна структури контенту торкається лише схеми, не компонентів.

Зв’язаність (Coupling) - практична метрика, не підручникова абстракція. Tight Coupling у React-проєкті виглядає так: компонент SelectedWorks напряму читає Zustand-стор із поточним треком. Змінили назву стейту - зламали компонент. Loose Coupling - той самий SelectedWorks отримує список треків як проп, нічого не знає про глобальний стан і тестується ізольовано. В TATREAL автор впровадив це рішення через жорстке правило: жоден компонент не імпортує інший компонент

горизонтально. Дев’ять компонентів - дев’ять ізольованих одиниць. Залежності лише вертикальні: сторінка - компонент - підкомпонент. Composition pattern замість успадкування класів - TypeScript-інтерфейси як єдиний контракт між шарами.

Повернемось до проблеми часу збірки. Gatsby вирішує її через інкрементальний підхід: перед кожною збіркою фреймворк порівнює хеш-суму кожного вузла даних із закешованим значенням із директорії .cache. Незмінений вузол - пропускається. Компілюється лише те, що справді змінилося. Для TATREAL це означає: оновив метадані одного релізу у YAML — перегенерувалась одна сторінка ProjectPage за 15 секунд замість 90. Кеш автоматично інвалідується при зміні gatsby-config.ts або оновленні версій плагінів, що унеможливлює збірку на застарілих даних.

Tree-shaking - не маркетинговий термін, а конкретний механізм webpack: статичний аналіз AST-дерева залежностей і видалення всього, на що немає жодного import. FontAwesome - класичний приклад: бібліотека містить понад 7 000 іконок, але у фінальний бандл потрапляють лише ті, що явно зареєстровані через library.add(). Для цього в package.json проставляється sideEffects: false для всіх публічних модулів - webpack отримує дозвіл агресивно видаляти невикористаний код. mode: 'production' автоматично додає TerserPlugin і CssMinimizerPlugin. Разом - мінус 30–40% до розміру бандлу порівняно з development-збіркою.

Реальна картина бандлу TATREAL після запуску webpack-bundle-analyzer: react-slick у SelectedWorks - 12 KB gzipped, найбільша окрема залежність; Formik разом із Yup у ContactMe - 18 KB; styled-components у кожному компоненті - ~6 KB завдяки інжекції лише використовуваних CSS-правил. Головна сторінка загалом - 130 KB gzipped. Для порівняння: медіанний JavaScript-бандл WordPress-сайту - ~450 KB gzipped [11], утричі більше. Веб-аналізатор treemap не просто показує числа - він робить видимими залежності, які потрапили у бандл випадково. Один такий сеанс аналізу в TATREAL виявив невикористаний поліфіл для IE11, успадкований транзитивно від однієї з бібліотек. Після видалення - мінус 8 KB.

1.4 Обґрунтування вибору математичного апарату, методологій та інструментів розробки

Вибір технологічного стеку для статичного генератора сайтів визначається насамперед часовою складністю процесу збірки. Нехай N — кількість сторінок у проєкті. При повній збірці кожна сторінка обробляється незалежно: генерація HTML, оптимізація зображень (r роздільностей на зображення), запис на диск. Загальна часова складність повної збірки становить:

$$O(N \cdot r),$$

де r — константа кількості роздільностей (для gatsby-plugin-image: r = 3).

При інкрементній збірці (Incremental Builds) перебудовуються лише вузли зі зміненою хеш-сумою: якщо змінено k << N сторінок, складність скорочується до O(k · r). Для поточного проєкту Tatreal (N = 20, k ≤ 2 при типовому оновленні) інкрементна збірка скорочує час з ~90 с до ~15 с, що підтверджує лінійну залежність:

$$T \cong c \cdot k,$$

де c - константа часу обробки одного вузла, виміряна експериментально як c ≈ 7.5 с/вузол.

Процес агрегації контенту у Gatsby описується моделлю спрямованого ациклічного графа (DAG - Directed Acyclic Graph). Нехай G = (V, E), де V - множина вузлів-джерел даних (YAML-файли, зображення, сторінки), E - орієнтовані ребра залежностей між ними. Відсутність циклів гарантується незмінністю вихідних даних між запусками збірки: DAG дозволяє виконати топологічне сортування вузлів за алгоритмом Кана за $O(|V| + |E|)$ та обробляти їх у строгому порядку залежностей. GraphQL-шар Gatsby реалізує цей граф у вигляді типізованої схеми: кожен вузол типу ProjectNode пов'язаний ребром @fileByRelativePath з вузлом типу File (зображення обкладинки). При зміні одного YAML-запису інвалідується рівно один вузол ProjectNode та пов’язаний з ним File-вузол - решта графа залишається незміненою, що й забезпечує субліній-ний час інкрементної збірки. Для поточного

проєкту: $|V| = 39$ вузлів (20 сторінок + 19 зображень), $|E| = 18$ ребер залежностей coverImage.

Фреймворк Gatsby реалізує описану DAG-модель через GraphQL Data Layer. На етапі bootstrap кожне джерело даних (gatsby-source-filesystem, gatsby-transformer-yaml) реєструє вузли у внутрішньому сховищі Redux. Gatsby будує TypeScript-типи для кожного вузла через gatsby-plugin-graphql-codegen, що перетворює GraphQL-схему на статично типізовані інтерфейси. Завдяки цьому компілятор TypeScript перевіряє відповідність між GraphQL-запитом компонента та структурою YAML-файлу ще до запуску збірки. У проєкті Tatreal цей механізм виявив три невідповідності типів під час розробки - усі усунуто до першого деплою, без жодної runtime-помилки у продакшні.

Вибір GraphQL як мови запитів над REST обґрунтовується кількісно через поняття overfetch-ratio (OF) - відношення обсягу переданих байтів до мінімально необхідного. У типовому REST-сценарії компонент SelectedWorks отримував би повний об'єкт проєкту (~1.2 KB JSON) для відображення лише чотирьох полів (title, slug, coverImage, featured - ~0.18 KB). $OF = 1.2 / 0.18 \approx 6.7$. GraphQL-запит з явним переліком полів повертає рівно ті дані, які оголошено у фрагменті. За даними Gatsby Documentation [8], це скорочує розмір runtime-даних на 60–85% порівняно з REST-підходом при аналогічній структурі контенту.

Якість зібраного застосунку вимірюється не суб'єктивними враженнями, а трьома числовими порогами Google Core Web Vitals, які Lighthouse CI перевіряє автоматично при кожному pull request. Пороги, встановлені для TATREAL:

– LCP ≤ 2.5 с — час до появи першого значущого контенту. В TATREAL це момент рендерингу bento-сітки з обкладинками релізів у SelectedWorks або Hero-секції - саме ці блоки слухач бачить першими;

– INP ≤ 200 мс - затримка між діями слухача та реакцією інтерфейсу. Найкритичніші точки в TATREAL - натискання кнопки програвача у Listen-секції та навігаційні переходи між сторінками проєктів;

– CLS $\leq$ 0.1 - відсутність стрибків макету при підвантаженні медіаконтенту. Gatsby-plugin-image резервує розмір під кожне зображення обкладинки ще до його завантаження - блоки Portfolio і SelectedWorks не зміщуються.

Усі три метрики вимірює Lighthouse CI автоматично — без ручних замірів, без суб'єктивних оцінок.

До написання коду архітектуру TATREAL зафіксовано через два UML-артефакти: Use Case Diagram та блок-схему алгоритму. Перший показує, що робить система з точки зору Відвідувача та Адміністратора. Другий — як вона це робить: від першого CDN-запиту до відправлення контактної форми.

Сукупність описаних моделей - $O(N \cdot r)$ / $O(k \cdot r)$ для збірки, DAG для залежностей контенту, overfetch-ratio для GraphQL-запитів - формує верифіковану аналітичну базу для вибору Gatsby як основного інструменту реалізації. Кожна з трьох метрик має конкретне числове значення, отримане в рамках поточного проєкту, і відтворюється у вимірюваних результатах: час збірки 15 с (інкрементна), $|V| = 39, OF \approx 6.7$ (REST) проти $OF = 1.0$ (GraphQL).

UML-артефакти зафіксували структуру взаємодії ще до написання першого рядка коду. Діаграма варіантів використання (Use Case Diagram) відображає взаємодію двох акторів - Відвідувача та Адміністратора - з функціональними блоками застосунку. Відвідувач взаємодіє з сімома варіантами використання: перегляд Hero-секції, перегляд вибраних робіт, відкриття сторінки проєкту, прослуховування музики, перехід на зовнішні платформи, надсилання контактного повідомлення та перегляд соціальних мереж. Адміністратор має доступ до п'яти варіантів: автентифікація через Netlify Identity, редагування проєктів, додавання нових записів, редагування посилань та перегляд надісланих повідомлень. Варіант використання «Надіслати повідомлення» включає («include») варіант «Валідація форми (Yup)», що відображає обов'язковий підпроцес перевірки введених даних.

Блок-схема алгоритму роботи застосунку описує покроковий сценарій взаємодії відвідувача з системою від моменту першого відкриття URL до відправлення контактного повідомлення. Перший крок конвеєра: CDN-вузол

повертає скомпільований HTML без жодного серверного обчислення. Після завантаження React гідрує сторінку (hydration), перетворюючи статичний HTML на інтерактивний SPA. Подальші навігаційні переходи виконуються через Gatsby Router без повного перезавантаження сторінки, завантажуючи лише JavaScript-чанк відповідного маршруту. При ініціюванні форми зворотного зв'язку Formik перевіряє введені дані через Yup-схему: за умови успішної валідації виконується fetch-запит до Netlify Forms API; у разі мережевої помилки відображається повідомлення про неуспішну відправку з пропозицією повторити спробу.

yarn.lock фіксує хеші всіх 1 700 транзитивних залежностей. Два різних yarn install на різних машинах або в різні дати зберуть ідентичний граф пакетів. Без цього файлу збірка, що пройшла локально, може впасти на Netlify CI через відмінну patch-версію однієї з бібліотек. Для TATREAL відтворюваність збірки — частина контракту якості: CI/CD-процес має видавати той самий результат, що й локальний gatsby build.

Проєктування GraphQL-схеми у Gatsby передбачає два принципово відмінних підходи: автоматичне виведення типів (Schema Inference) та явне визначення типів через createSchemaCustomization. При автоматичному виведенні Gatsby сканує усі вузли даних після їх завантаження та генерує GraphQL-типи на основі структури JavaScript-об'єктів. Хоча цей підхід зручний для прототипування, він має серйозний недолік: якщо поле у YAML-файлі відсутнє хоча б в одному записі (наприклад, поле client не заповнено для деяких проєктів), Gatsby не зможе вивести його тип і генерує помилку збірки. Для проєкту Tatreal обрано явне визначення типів через createSchemaCustomization у gatsby-node.ts. Тип ProjectNode декларує всі поля з явними GraphQL-скалярами: String! для обов'язкових рядків, String для опціональних, Boolean! для featured та Date для year. Це гарантує стабільну збірку незалежно від повноти даних у YAML-файлі та надає TypeScript-типи через автоматичну кодогенерацію gatsby-plugin-graphql-codegen.

Два типи вбудованих директив вирішили конкретні задачі в TATREAL. Директива @link пов'язує вузли без дублювання даних: поле author у BlogPost посилається на окремий Author-вузол через @link(by: "name"), а не копіює його

дані. Директива @fileByRelativePath перетворює рядок-шлях у YAML на повноцінний File-вузол - і gatsby-plugin-image підключається до нього автоматично. У ProjectNode поле coverImage декларується через @fileByRelativePath - і макрос gatsbyImageData(width: 800, placeholder: BLURRED, formats: [AUTO, WEBP, AVIF]) стає доступним прямо у GraphQL-фрагменті компонента.

Lighthouse CI прив'язано до GitHub Actions через lighthouserc.js: кожен pull request запускає три аудит-прогони, усереднює результати та блокує merge, якщо будь-яка метрика падає нижче порогу. Для TATREAL встановлено: Performance ≥ 0.90, Accessibility = 1.0, SEO = 1.0, Best Practices = 1.0. Ці пороги відповідають фінальним оцінкам у таблиці 2.3 і гарантують, що жодна оптимізація не погіршить наявний результат.

Комплексна оцінка та інструментальна валідація якості клієнтського коду, швидкодії й архітектурних рішень застосунку TATREAL базується на ряді методів аналізу, які деталізовано у таблиці 1.4. Зокрема, моніторинг ключових числових порогів Google Core Web Vitals інтегровано безпосередньо в CI-конвеєр через Lighthouse CI для автоматичної перевірки кожного pull request за трьома критичними метриками:

Таблиця 1.4 - Методи аналізу якості вебзастосунку та їх застосування

Метод аналізу	Інструмент	Автоматизація	Що вимірює
Аналіз бандлу	webpack-bundle-analyzer	gatsby build --profile	Розмір JS-модулів
Core Web Vitals	Google Lighthouse	Lighthouse CI (GitHub Actions)	LCP, INP, CLS
Покриття коду	Chrome Coverage API	Ручне / CI	Невикористаний JS/CSS
Профілювання рендерингу	React DevTools Profiler	Ручне в DevTools	Час рендерингу компонентів
Аналіз мережевих	Chrome DevTools	WebPageTest	Waterfall запитів

запитів	Network		
SEO-аудит	Screaming Frog / Lighthouse	Weekly scheduled	Метатеги, canonical, sitemap

1.5 Постановка мети та завдань дослідження

Попередні підрозділи зафіксували конкретну проблему: монолітні CMS не справляються з вимогами сучасного музичного медіапортфоліо - ні за швидкістю, ні за безпекою. Мета роботи - спроєктувати і реалізувати модульний вебзастосунок TATREAL на основі Gatsby 5 та JAMstack-архітектури, що забезпечує LCP < 1.8 с, Lighthouse Performance ≥ 90 та Git-based управління контентом без серверної інфраструктури.

Об’єкт дослідження - процеси проєктування та збірки вебплатформ на основі статичних генераторів: від агрегації гетерогенних джерел даних (YAML, GraphQL) до CDN-розгортання скомпільованих файлів. Предмет - методи компонентно-орієнтованого моделювання інтерфейсів, оптимізації JavaScript-бандлів засобами Webpack Tree-shaking та оцінювання продуктивності через Core Web Vitals у реальних умовах.

Завдання дослідження охоплюють п’ять взаємопов’язаних етапів:

– Порівняти архітектурні підходи до побудови медіапортфоліо (монолітна CMS, SPA, JAMstack) - виміряти TTFB, LCP та розмір бандлу для кожного сценарію, сформулювати технічні критерії вибору стеку.

– Спроєктувати типізовану GraphQL-схему даних (DAG-модель вузлів) із явним визначенням типів через createSchemaCustomization - унеможливити runtime-помилки при неповних YAML-записах

– Реалізувати дев’ять ізольованих React/TypeScript-компонентів (Hero, SelectedWorks, Portfolio, Listen, About, Contact, Header, Footer, ProjectPage) з Loose Coupling через типізовані пропси - без горизонтальних залежностей між компонентами.

– Інтегрувати Git-based Netlify/Decap CMS: налаштувати YAML-колекції, Netlify Identity для автентифікації, обробку контактної форми через Netlify Forms із валідацією Formik/Yup - без серверного коду у продакшні

– Провести повний цикл Lighthouse CI-аудиту (Performance, Accessibility, SEO, Best Practices) - порівняти показники TATREAL із медіанними значеннями WordPress-сайтів (Web Almanac 2023) та верифікувати відповідність Core Web Vitals порогу “Good” за всіма трьома метриками

Результатом виконання п’яти завдань стане задокументований, задеплоєний медіапортфоліо TATREAL із виміряними показниками продуктивності та відтворюваним процесом CI/CD-розгортання через Netlify.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ-ПОРТФОЛІО

2.1 Особливості вебзастосунку та вимоги до системи

Специфікацію вимог до вебзастосунку TATREAL сформовано на основі потреб Вячеслава Користова - незалежного музичного продюсера та саунд-дизайнера (псевдонім Tatreal), що спеціалізується на написанні музики для відеоігор, рекламних кампаній та кінопроєктів. Портфоліо виконує три функції одночасно: публічна презентація для потенційних замовників та слухачів, інструмент дистрибуції аудіоконтенту та Git-based адмін-інтерфейс для оновлення релізів без залучення технічного фахівця.

TATREAL має задовольняти вісім функціональних вимог, сформованих від Вячеслава Користова:

– відображення портфоліо у вигляді слайдера вибраних релізів та bento-сітки всіх 18 робіт із назвою, роком, брендом-замовником та ролями автора;

– автоматичне створення окремої сторінки для кожного проєкту з детальним описом, обкладинкою та зовнішнім посиланням;

– вбудований Spotify-плеєр для прослуховування музики безпосередньо на сайті;

– секції з посиланнями на музичні платформи (Spotify, Bandcamp, SoundCloud, Deezer, YouTube) та соціальні мережі;

– форма зворотного зв'язку для надсилання повідомлень замовникові без серверного коду;

– Git-based адмін-панель Decap CMS за маршрутом /admin/ для редагування YAML-колекцій проєктів, посилань і текстів через браузер — зміни автоматично потрапляють у репозиторій як Git-коміт;

– адаптивний макет з трьома брейкпоінтами (375 px / 768 px / 1280 px) та fluid-відступами через CSS clamp() - базові стилі Mobile-First;

– Core Web Vitals у категорії Good: LCP ≤ 1.8 с, INP ≤ 200 мс, CLS ≤ 0.1 - верифіковано через Lighthouse CI при кожному деплої.

Нефункціональні вимоги: JAMstack-статика на CDN (серверний код у продакшні відсутній), нульова вартість хостингу (Netlify Free tier), повне версіювання контенту через Git-коміти та модульна структура з Loose Coupling - кожен компонент ізольований і розширюється незалежно.

2.2 Проєктування функціональних можливостей застосунку

Функціональна архітектура та логіка розгортання TATREAL специфікована через два взаємодоповнювальних описи: декомпозицію сценаріїв за рольовими акторами та послідовний алгоритм автоматизованого CI/CD-ланцюжка.

– Два актори визначають повний простір взаємодій. Слухач (публічний доступ, автентифікація не потрібна) реалізує сім сценаріїв: перегляд Hero-секції з CTA-посиланнями; перегляд вибраних релізів у слайдері SelectedWorks; відкриття окремої сторінки проєкту за маршрутом /projects/{slug}/; прослуховування треків через вбудований Spotify-плеєр; перехід на зовнішні музичні платформи (Spotify, Bandcamp, SoundCloud); надсилання контактного повідомлення через форму з валідацією; перегляд соціальних мереж. Музикант-адміністратор (автентифікований доступ через /admin/ та Netlify Identity) реалізує п'ять сценаріїв: автентифікація та авторизація через Netlify Identity; редагування YAML-даних наявних проєктів у Decap CMS; додавання нових релізів до колекції; оновлення посилань на платформи та соціальні мережі; перегляд надісланих повідомлень у Netlify Forms. Сценарій “Надіслати повідомлення” включає обов'язковий підпроцес “Валідація форми” — Yup-перевірка всіх п'яти полів перед fetch-запитом до Netlify Forms.

Послідовний алгоритм автоматизованого CI/CD-ланцюжка охоплює шість детермінованих кроків: Музикант-адміністратор зберігає зміни у Decap CMS - автоматичний Git-коміт потрапляє у GitHub-репозиторій - Netlify webhook ініціює gatsby build в ізольованому хмарному контейнері - Gatsby Node API виконує

createPages для кожного проєкту з непорожнім полем slug (генерується 18 статичних HTML-сторінок) - скомпільовані HTML/CSS/JS-файли розгортаються на глобальному CDN Fastly - Lighthouse CI запускає три аудит-прогони та блокує деплой, якщо Performance < 0.90. Весь ланцюжок від git push до живого сайту займає ~100 секунд.

Браузер Слухача отримує готовий HTML з CDN-вузла без жодного серверного обчислення. React гідрує сторінку пріоритетно: Hero-секція та bento-сітка RecentWorks - першочергово, Footer і PlatformsSection - у фоновому режимі. Gatsby Router підміняє вміст при навігації без повного перезавантаження - завантажується лише JavaScript-чанк відповідного маршруту. Відправка контактної форми виконується через fetch POST на Netlify Forms endpoint - без власного бекенду, без серверного коду у продакшні.

2.3 Вибір засобів розробки

Вибір технологічного стеку для TATREAL спирається на два жорсткі обмеження: клієнтська швидкодія LCP < 1.8 с та Zero-Server Architecture - відсутність бюджету на серверну інфраструктуру у продакшні. Вимоги §2.1 та порівняльний аналіз §1.2 звузили вибір до семи інструментів: Gatsby 5, React 18, TypeScript, styled-components, Netlify/Decap CMS, Formik + Yup та Netlify Forms.

2.3.1 Gatsby та React

Gatsby 5 виграв конкурентний відбір (§1.2) через єдине у своєму класі поєднання: вбудований GraphQL Data Layer збирає локальні YAML-файли, медіастріми зображень та метадані Netlify CMS в єдину типізовану схему ще на фазі збірки. Офіційний gatsby-plugin-netlify-cms поєднує обидва шари без ручного налаштування - жоден конкурент таку пару не покриває з коробки.

Основні причини вибору Gatsby: повна підтримка React-компонентів та TypeScript; вбудовані плагіни для оптимізації зображень (gatsby-plugin-image), SEO та прогресивних вебзастосунків; механізм createPages для програмного створення

сторінок під час збірки; інтеграція з Netlify CMS через gatsby-plugin-netlify-cms; результуючий статичний сайт не потребує серверного коду в продакшені.

React 18 входить до Gatsby як обов'язкова залежність - окремого рішення тут не приймалось. В TATREAL useState використовується точково у трьох вузлах: стан відтворення треків у секції Listen, стан відкриття мобільного меню в Header та стани success/error після відправки форми у ContactMe. Такий мінімальний стан означає мінімум ре-рендерів і прямо впливає на INP = 25 мс.

2.3.2 Мова програмування TypeScript

Статичну типізацію впроваджено для запобігання збоїв збірки. Плагін gatsby-plugin-graphql-codegen автоматично сканує GraphQL-запити та генерує TypeScript-інтерфейси зі схеми. Під час розробки TATREAL він виявив три невідповідності типів у YAML-записах контенту ще до деплою, унеможлививши runtime-помилки у продакшні. Чотири практичних наслідки вибору TypeScript у TATREAL:

– compile-time валідація пропсів компонентів забезпечує Loose Coupling: компілятор фіксує невідповідність між GraphQL-запитом компонента та структурою YAML ще до gatsby build;

– автокомпліт GraphQL-вузлів у VSCode через gatsby-plugin-graphql-codegen — всі 18 полів ProjectNode доступні із підказками без ручного опису типів;

– сувора типізація структури всіх 18 портфоліо-записів: обов'язкові поля title та slug типу String!, опціональні client та link типу String, булеве featured, рядкове role;

– повна підтримка в Gatsby та React через @types-пакети.

У tsconfig.json встановлено strict: true: увімкнено noImplicitAny, strictNullChecks та strictFunctionTypes. Ці три прапори разом з codegen-типами закрили весь шлях від YAML-запису до React-пропсу без єдиного any.

Типи для компонентів TATREAL оголошено у двох місцях: глобальні інтерфейси ProjectNode та HomeData — у src/types.d.ts, Props кожного компонента - у його власному файлі. Така структура означає: поле coverImage одного з 18

релізів не може мати хибний тип File без негайного tsc-error ще на фазі gatsby build, до потрапляння в CDN. Рисунок 2.1 демонструє ключові типи та Utility Types Omit і Pick.

```ts
// src/types.d.ts — основні інтерфейси проєкту
export interface Project {
  title:        string       // назва проєкту
  slug:         string       // унікальний URL-ідентифікатор
  year:         string       // рік випуску
  role:         string       // роль автора (Music, SFX тощо)
  client?:      string       // замовник (опціонально)
  description?: string       // опис (опціонально)
  coverImage:   string       // шлях до обкладинки
  link?:        string       // зовнішнє посилання (опціонально)
  featured:     boolean      // відображати у слайдері
}

export interface HomeData {
  docTitle: string
  header: {
    logo: string
    nav:  { item1: string; item2: string; item3: string }
    button: string
  }
  main: {
    projects: Project[]
    music:    Record<string, string>
    social:   Record<string, string>
  }
}

// Utility Types у компонентах
// Omit<> — прибирає поля що конфліктують із HTML-атрибутами
type WrapperProps = Omit<HeaderProps, 'data'> & { $scrolled: boolean }
const Wrapper = styled.header<WrapperProps>`...`

// Pick<> — компонент отримує лише потрібні поля
type TileProps = Pick<Project, 'title' | 'slug' | 'coverImage' | 'featured'>

// Тип для розміру тайла у bento-сітці
type TileSize = 'lg' | 'wide' | 'sm'

const getTileSize = (p: Project, index: number): TileSize => {
  if (p.featured)        return 'lg'
  if (index % 6 === 0)   return 'wide'
  return 'sm'
}
```

Рисунко 2.1 - TypeScript-інтерфейси проєкту (src/types.d.ts)

2.3.3 Система управління контентом Netlify CMS

Netlify CMS (перейменований у 2023 р. на Decap CMS) обрано через принципову відмінність від монолітного WordPress: замість постійного з'єднання з MySQL-базою система працює безпосередньо з файловою системою Git-репозиторію через Git API. Зміна у Decap CMS - це коміт у репозиторій; жодного окремого сервера, жодних SQL-запитів. Порівняно з WordPress це дає чотири технічні переваги:

– версіювання контенту через Git - будь-яку зміну можна відкатити;

– відсутність окремого сервера для CMS - знижує вартість обслуговування;

– браузерна адмін-панель за адресою /admin/ для редагування YAML-контенту через браузер (власник — Вячеслав Користов);

– YAML-схема визначає типи полів, що унеможливлює введення некоректних даних.

2.3.4 styled-components

Компонентна стилізація через styled-components забезпечує повну ізоляцію CSS всередині React-модулів (Scoped Styles). Кожен styled-компонент отримує унікальний хеш при збірці - каскадні конфлікти стилів між bento-сіткою RecentWorks, аудіоплеєром Listen та картками Portfolio виключені структурно. Динамічна стилізація через пропси усуває потребу в ручному toggling CSS-класів: Header змінює фоновий blur залежно від стану прокрутки, ContactMe передає статус валідації напряму в стилі поля.

Рисунок 2.2 демонструє принцип динамічної стилізації у styled-components на прикладі компонента Header. Транзієнт-проп $scrolled (префікс $ запобігає потраплянню атрибуту в DOM) визначає, чи застосовувати розмитий фон через директиву css.

```tsx
// src/components/Header.tsx (фрагмент стилізації)
import styled, { css } from 'styled-components'

// Транзієнт-проп: $ запобігає передачі атрибуту в DOM
const Wrapper = styled.header<{ $scrolled: boolean }>`
  position: fixed;
  top: 0; left: 0; right: 0;
  z-index: 100;
  display: flex;
  align-items: center;
  justify-content: space-between;
  padding: 0 var(--container-px);
  height: 64px;
  transition: background 0.3s ease,
              backdrop-filter 0.3s ease,
              box-shadow 0.3s ease;

  /* Динамічні стилі через пропс */
  ${({ $scrolled }) =>
    $scrolled &&
    css`
      background:       rgba(13, 12, 10, 0.85);
      backdrop-filter:  blur(14px);
      box-shadow:       0 1px 0 var(--c-border);
    `}
`

// Навігаційне посилання зі станом hover
const NavLink = styled.a`
  font-size:      0.875rem;
  letter-spacing: 0.05em;
  color:          var(--c-text-muted);
  transition:     color 0.2s ease;

  &:hover { color: var(--c-text); }

  /* :focus-visible для клавіатурної навігації (WCAG 2.1) */
  &:focus-visible {
    outline: 2px solid var(--c-accent);
    outline-offset: 4px;
    border-radius: var(--radius-sm);
  }
`

// Кнопка CTA з variant-пропсом
const CtaButton = styled.a<{ $variant?: 'primary' | 'ghost' }>`
  padding: 0.5rem 1.25rem;
  border-radius: var(--radius-pill);
  font-size: 0.875rem;
  font-weight: 600;
```

Рисунок 2.2 - styled-components: динамічна стилізація (Header.tsx)

2.3.5 Formik та Yup

Валідацію та збір повідомлень у формі зворотного зв'язку TATREAL реалізовано через зв'язку Formik (контроль стану полів через useFormik: values, errors, touched, handleSubmit) та Yup (декларативна схема валідації на клієнті). Formik відстежує п'ять полів ContactMe без зайвих ре-рендерів, Yup перевіряє правила кожного поля та повертає усі помилки одночасно.

Повну схему валідації Formik та Yup для контактної форми наведено у рисунок 2.3. Параметр abortEarly: false повертає всі невалідні поля одночасно - слухач бачить повний список помилок за один сабміт, а не виправляє їх по черзі.

```
// src/components/ContactMe.tsx (фрагмент)
import * as Yup from 'yup'
import { useFormik } from 'formik'

interface ContactValues {
  firstName: string; lastName: string
  email: string; subject: string; message: string
}

const validationSchema = Yup.object({
  firstName: Yup.string()
    .min(2, 'Мінімум 2 символи')
    .required("Ім'я є обов'язковим"),
  lastName: Yup.string().optional(),
  email: Yup.string()
    .email('Введіть коректний email')
    .required('Email є обов\'язковим'),
  subject: Yup.string()
    .min(3, 'Мінімум 3 символи')
    .required('Тема є обов\'язковою'),
  message: Yup.string()
    .min(10, 'Мінімум 10 символів')
    .required('Повідомлення є обов\'язковим'),
})

const formik = useFormik<ContactValues>({
  initialValues: {
    firstName: '', lastName: '',
    email: '', subject: '', message: '',
  },
  validationSchema,
  validateOnBlur: true,
  validateOnChange: false,    // не валідувати на кожен символ
  onSubmit: handleSubmit,
})

// Доступність: aria-атрибути для поля email
// <input
//   id='email'
//   aria-describedby={formik.errors.email ? 'email-error' : undefined}
//   aria-invalid={!!(formik.touched.email && formik.errors.email)}
//   {...formik.getFieldProps('email')}
// />
// <span id='email-error' role='alert'>
//   {formik.touched.email && formik.errors.email}
// </span>
```

Рисунок 2.3 - Схема валідації Yup та налаштування Formik (ContactMe.tsx)

При виборі бібліотеки управління формами розглядалися три основні альтернативи: Formik, React Hook Form та Zod у поєднанні з React Hook Form. React Hook Form використовує неконтрольовані компоненти (Uncontrolled Components) на основі useRef замість useState для збереження значень полів, що теоретично дає перевагу у продуктивності за рахунок зменшення кількості ре-рендерів. Проте для форми ContactMe, яка містить лише п'ять полів і рендериться один раз за сесію, різниця у продуктивності є несуттєвою. Formik обрано через більш зрілу документацію та кращу інтеграцію з Yup-схемами через параметр validationSchema. Zod як альтернатива Yup надає кращу TypeScript-інтеграцію через автоматичне виведення типів з схеми (z.infer<typeof schema>), проте на момент розробки проєкту офіційна інтеграція Formik+Zod потребувала додаткових налаштувань.

2.4 Проєктування дизайн-системи та інформаційної архітектури

TATREAL побудований як лінійна односторінкова структура з семи секцій: Hero, Selected Works, Recent Works, Listen, Platforms, Socials, Contact. Кожен з 18 проєктів отримує окрему динамічно згенеровану сторінку за маршрутом /projects/{slug}/. Колірну палітру сформовано на базі дизайн-токенів темної теми — домінуючий фон #0D0C0A, акцент #F5C842 - централізованих у src/theme.ts та доступних через CSS-змінні у будь-якому styled-component без Context або ThemeProvider.

Дизайн-система реалізована через файл src/theme.ts, де централізовано зберігаються всі токени оформлення — кольори, тіні, радіуси та брейкпоінти. CSS-змінні оголошені у глобальних стилях (gatsby-browser.tsx), - CSS-змінні доступні у будь-якому styled-component без Context або ThemeProvider.

Шрифтовий стек реалізований як fallback-ланцюг: font-family: 'Inter', -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Oxygen, Ubuntu, sans-serif. Пріоритетний шрифт Inter обрано за його оптимізацію для екранного відображення - геометрична точність гліфів і широкі символьні пропорції забезпечують відмінну

читабельність при розмірах від 12 до 72 px. Такий підхід унеможливлює FOUT (Flash of Unstyled Text) і позитивно впливає на показник CLS. Масштаб розмірів тексту встановлено відповідно до модульної шкали з коефіцієнтом 1.25 (Major Third): 12 px (caption), 14 px (small), 16 px (body), 20 px (lead), 25 px (h3), 31 px (h2), 39 px (h1).

Чотири брейкпоінти TATREAL охоплюють діапазон від 375 px до 1480 px+, детально описані у таблиці 2. Стилі написані в порядку Mobile-First: менші екрани отримують базовий CSS, ширші — лише ті надбавки, що їм потрібні.1.

Таблиця 2.1 - Брейкпоінти адаптивного макету застосунку

Ім'я / значення	Умова застосування	Зміни макету
sm / 600 px	max-width: 599px	Одноколонна сітка, приховуються декоративні елементи
md / 900 px	max-width: 899px	Бургер-меню, bento-сітка переходить на 2 колонки
lg / 1200 px	max-width: 1199px	4 колонки bento, двоколонний ContactMe
xl / 1480 px	min-width: 1480px	Межа контейнера; масштабуються відступи через clamp()

Ширина контейнера обмежена 1480 px, бічні відступи — padding: 0 clamp(1.5rem, 4vw, 4rem), тобто від 24 px до 64 px плавно відповідно до ширини viewport. Жодного медіа-запиту для відступів - і жодного стрибка CLS при перетині брейкпоінту. Сітка секцій визначається через CSS Custom Properties: --grid-columns: 12 та --grid-gap: clamp(1rem, 2vw, 2rem), кожен компонент самостійно задає ширину через grid-column: span N.

Анімаційна система реалізована відповідно до принципу «motion with purpose» — кожна анімація несе функціональне навантаження і відображає зміну стану системи. Для Hero-секції застосовується CSS-анімація kenBurns тривалістю 22 секунди з нескінченним повтором: transform плавно змінюється від scale(1)

translate(0, 0) до scale(1.12) translate(-2%, -1%), відтворюючи ефект повільної «живої» камери. Тривалість 22 секунди обрана як довша за типовий час першого перегляду сторінки (15–18 секунд), щоб цикл анімації не був помітним. Навігаційна панель Header використовує transition: background-color 0.3s ease, backdrop-filter 0.3s ease тривалістю 300 мс для плавного переходу між прозорим і непрозорим станами. Картки портфоліо застосовують криву cubic-bezier(0.25, 0.46, 0.45, 0.94) - ease-out-quad, що починається швидко та сповільнюється в кінці, формуючи природне пружинисте відчуття взаємодії.

Дотримання WCAG 2.1 AA у TATREAL охоплює три рівні: кольоровий контраст, клавіатурна навігація та семантична HTML-розмітка. Акцент #F5C842 на фоні #0D0C0A дає коефіцієнт контрасту 11.3:1 - вдвічі вище за поріг AA (4.5:1). Усі інтерактивні елементи отримали видимі :focus-visible стилі та aria-label там, де текстовий контент відсутній (іконки соціальних мереж). Компонент Header використовує семантику header, nav, ul, li замість нейтральних div - скринрідери оголошують навігаційну структуру без додаткових ARIA-ролей. Зображення обкладинок у RecentWorks мають атрибут alt із назвою проєкту. Lighthouse Accessibility = 100/100.

Три кореневі директорії поділяють код, контент і статику: src/ (components, templates, pages, images, fonts), content/pages/ (YAML-файл контенту з 18 проєктами), static/admin/ (конфігурація Decap CMS) та static/bg/ (фонові зображення Hero-секції). Такий поділ означає практичне правило: оновлення YAML-контенту ніколи не торкається вихідного коду компонентів.

2.5 Розробка компонентів інтерфейсу на React і TypeScript

Увесь інтерфейс розділено на дев'ять ізольованих функціональних компонентів у директорії src/components/. Кожен компонент відповідає за одну секцію сторінки і стилізований через styled-components.

Компонент Header реалізує фіксовану навігаційну панель. Вона прозора при положенні сторінки у верхній частині та набуває розмитого темного фону

(backdrop-filter: blur(14px)) після прокрутки понад 60 px. На мобільних пристроях (ширина < 720 px) навігаційні посилання приховуються і з'являється кнопка-бургер, що розкриває повноекранне меню. Логіка переходів залежить від поточного маршруту: на головній сторінці — плавна прокрутка через react-scroll, на внутрішніх сторінках — перенаправлення на головну з хешем.

Компонент Hero — повноекранна секція-заставка. Фонове зображення анімується через CSS-анімацію kenBurns (поступове масштабування та зсув), що створює ефект живого кадру. Поверх фону накладено два шари CSS-градієнту для читабельності тексту. Секція містить підзаголовок-eyebrow, великий заголовок, опис, теглайн із назвами брендів-замовників (Lexus, Nissan, Mazda, Red Bull) та два CTA-посилання: плавна прокрутка до Selected Works і зовнішнє посилання на Spotify.

Компонент SelectedWorks відображає тільки проєкти з прапорцем featured: true (8 із 18). Для показу використовується бібліотека react-slick. Кожен слайд - двоколонна картка: обкладинка ліворуч, рік/клієнт/назва/опис/теги ролей/кнопка переходу - праворуч. Стрілки навігації перевизначені власними компонентами у стилі дизайн-системи.

Компонент RecentWorks реалізує bento-сітку (CSS Grid) для відображення всіх 18 проєктів. Функція getTileSize динамічно присвоює розміри: featured-проєкти займають 2×2 комірки (lg), кожний шостий - 2×1 (wide), решта - 1×1 (sm). Hover-ефект включає масштабування обкладинки через transform: scale(1.07) та появу метаданих. На мобільних пристроях усі тайли зводяться до 1×1 у дворядній сітці.

Компонент ContactMe реалізує форму зворотного зв'язку. Управління станом і відправкою здійснюється через Formik, валідація - через Yup. Форма містить поля firstName, lastName (опціонально), email, subject, message. Honeypot-поле bot-field (приховане через CSS) захищає від спам-ботів. Після успішного відправлення форма замінюється на картку підтвердження.

Компонент Footer реалізований як трьохколонний CSS-грід: email-посилання, логотип із копірайтом та набір іконок соціальних мереж. Для

рендерингу іконок використовується FontAwesome (@fortawesome/react-fontawesome), що забезпечує SVG безпосередньо в DOM. На мобільних пристроях грід перемикається у вертикальний стек.

Інтеграція аудіоконтенту в секції Listen реалізована через Spotify Embed API з фірмовим колірним токеном #1DB954 та параметром theme=0 для безшовної інтеграції в темну тему застосунку. Плеєр вбудовується через HTML-елемент iframe із атрибутом src, що містить посилання на офіційний Spotify Embed API-ендпоінт художника. Для запобігання передчасному завантаженню важкого iframe-ресурсу при відкритті сторінки реалізована стратегія відкладеного завантаження: атрибут loading='lazy' та IntersectionObserver. Плеєр ініціалізується лише тоді, коли секція потрапляє у viewport, що скорочує початковий час завантаження сторінки та позитивно впливає на LCP. Темна кольорова схема вбудованого плеєра задається параметром theme=0 в URL Spotify Embed API, відповідаючи загальній темній палітрі сайту.

Компоненти PlatformsSection та SocialsSection реалізують відповідно секції музичних платформ і соціальних мереж. Обидва компоненти отримують дані через проп data, типізований як об'єкт з рядковими полями-посиланнями (spotify, bandcamp, soundcloud тощо). Посилання відображаються у вигляді горизонтального flex-контейнера з логотипами платформ через SVG-компоненти FontAwesome. При hover-взаємодії логотипи збільшуються через transform: scale(1.15) та набувають фірмового кольору відповідної платформи (зелений #1DB954 для Spotify, помаранчевий #FF5500 для SoundCloud) через динамічні styled-components з передачею пропсу platformColor. Усі посилання відкриваються в новій вкладці через target='_blank' rel='noopener noreferrer' для захисту від атак reverse tabnabbing.

Захист контактної форми від автоматизованого спаму реалізовано без капчі через приховане поле bot-field (механізм Honeypot): боти автоматично заповнюють усі поля форми, Netlify Forms відхиляє запит при будь-якому значенні в bot-field. Два службових компоненти TATREAL обслуговують усі дев’ять секційних модулів: Section стандартизує вертикальні відступи через CSS-змінну --section-gap

та прив’язує якір id навігації; Badge рендерить теги ролей (Music Score, SFX) без власного стану - чистий функціональний компонент.

Тип Project у src/types.d.ts є єдиним контрактом між GraphQL-схемою та всіма компонентами TATREAL: обов’язкові поля title та slug (String!), опціональні client, link, description (?), булеве featured для фільтрації в SelectedWorks, рядкове role. Типи пропсів оголошуються через інтерфейс Props у файлі компонента. Три невідповідності типів між YAML-записами та GraphQL-запитами TypeScript виявив ще до першого деплою. tsconfig.json встановлює strict: true: noImplicitAny, strictNullChecks, strictFunctionTypes — весь шлях від YAML до React-пропсу без єдиного any.

Для оцінки обсягу бандлу та характеристик компонентів проведено аналіз розміру JavaScript-файлів після збірки gatsby build. Результати наведено у таблиці 2.2.

Таблиця 2.2 - Характеристики компонентів інтерфейсу

Компонент	**Власний стан**	**Побічні ефекти**	**Обсяг бандлу, КБ**
Header	2 (scrolled, menuOpen)	scroll listener	~8
Hero	0	немає	~4
SelectedWorks	0	немає	~12 (react-slick)
RecentWorks	0	немає	~6
ListenMusic	1 (playerLoaded)	IntersectionObserver	~3
ContactMe	2 (submitted, error)	fetch Netlify Forms	~18 (Formik+Yup)
Footer	0	немає	~4

Розділення коду (Code Splitting) через React.lazy() та Suspense виносить важкі інтерактивні блоки в окремі чанки — стартовий JavaScript-бандл головної сторінки

залишається 35 KB. Gatsby автоматично ділить бандл на рівні сторінок; компонент SelectedWorks реалізує lazy loading для react-slick: const Slider = React.lazy(() => import('react-slick')), що виносить слайдер (~35 KB gzipped) з основного бандлу та завантажує його лише при потраплянні секції у viewport. По-третє, зображення обкладинок підключені через gatsby-plugin-image, який під час збірки генерує три роздільних здатності (480w, 800w, 1200w) у форматі WebP з JPEG-fallback. Браузер автоматично обирає оптимальну версію через srcset, скорочуючи обсяг переданих даних на 30–70% залежно від пристрою.

Компонент IndexPage є кореневою сторінкою застосунку, реалізованою у файлі src/pages/index.tsx. Він компонує всі дев'ять секційних компонентів у єдиному JSX-дереві та передає кожному відповідну гілку даних із pageData-об'єкту. IndexPage також реалізує логіку кнопки «прокрутка вгору» (ScrollToTop): кнопка відображається лише тоді, коли користувач прокрутив сторінку нижче висоти першого екрана та ще не досяг підвального колонтитула. Це реалізовано через window.scrollY та обчислення distanceToBottom через refs Hero та Footer-елементів, що запобігає перекриттю кнопкою важливих інтерактивних елементів у нижній частині сторінки.

{ passive: true } у scroll-обробнику Header - дрібна деталь із відчутним ефектом на INP. Параметр { passive: true } у addEventListener сигналізує браузеру, що обробник не викликатиме event.preventDefault() і браузер може безпечно виконувати прокрутку паралельно з виконанням JavaScript-коду без очікування завершення обробника. У компоненті Header обробник scroll-події реєструється з passive: true: window.addEventListener('scroll', onScroll, { passive: true }). Без цього прапора браузер змушений синхронно очікувати завершення JavaScript перед кожним кроком прокрутки, що призводить до відчутних затримок на мобільних пристроях. Дебаунсинг (Debouncing) через requestAnimationFrame також застосований у scroll-обробнику: замість безпосереднього виклику setScrolled(window.scrollY > 60) використовується RAF-обгортка, що гарантує виконання setState не частіше одного разу на кадр анімації (зазвичай 60 fps), запобігаючи надмірним ре-рендерам Header при швидкій прокрутці.

Таблиця 2.3 — Оптимізаційні патерни React, застосовані у проєкті Tatreal

Патерн	**Компонент**	**Механізм**	**Вплив на продуктивність**
React.memo	Badge, ProjectCard	Shallow props comparison	Усуває зайві ре-рендери в списках
useMemo	RecentWorks	Мемоізація getTileSize()	Кешує обчислення розмірів тайлів
useCallback	Header, ContactMe	Стабільні посилання на функції	Стабілізує пропси дочірніх компонентів
React.lazy + Suspense	SelectedWorks	Динамічний import(react-slick)	Виносить 35 KB зі стартового бандлу
Passive scroll listener	Header	{ passive: true }	Не блокує прокрутку браузера
RAF debounce	Header scroll	requestAnimation Frame	Обмежує setState до 60 fps
loading='lazy'	ListenMusic iframe	IntersectionObserver	Відкладає завантаження Spotify

2.6 Реалізація системи управління контентом та динамічної генерації сторінок

Весь контент сайту зберігається в одному YAML-файлі content/pages/home.yml. Файл містить три секції: header (логотип, навігація, кнопка CTA), main.projects (масив із 18 проєктів) та main.music / main.social (посилання на платформи і соціальні мережі). Кожен проєкт містить поля: title, slug, year, role,

client, description, coverImage, link, featured. Схему контенту описано у static/admin/config.yml для адмін-панелі Netlify CMS.

Адміністративна панель доступна за маршрутом /admin/. Через OAuth-автентифікацію з GitHub адміністратор входить у панель через браузер без доступу до коду. Після збереження змін Netlify CMS автоматично робить коміт у репозиторій, що ініціює новий CI/CD-цикл і перезбирає статичний сайт.

Динамічна генерація сторінок реалізована у файлі gatsby-node.ts через точку розширення createPages. Функція читає YAML-файл за допомогою fs і js-yaml, ітерує масив проєктів і для кожного елемента із непорожнім полем slug викликає actions.createPage(). Функція createPage приймає маршрут, шаблон src/templates/Project.tsx і контекст із даними поточного та всіх проєктів. Ключовий фрагмент коду наведено у рисунок 2.4.

```
projects.forEach((project) => {
  if (!project.slug) return
  createPage({
    path: `/projects/${project.slug}/`,
    component: path.resolve('./src/templates/Project.tsx'),
    context: { project, allProjects: projects },
  })
})
```

Рисунок 2.4 - Фрагмент gatsby-node.ts: генерація сторінок проєктів

Шаблон Project.tsx відображає обкладинку з темним оверлеєм, заголовок, рік і клієнта, теги ролей, опис та зовнішнє посилання. Внизу розміщені тайли попереднього і наступного проєктів для переходу без повернення на головну сторінку. SEO-метатеги (title, description, og:image) для кожної сторінки генеруються через Head-компонент Gatsby.

useStaticQuery в IndexPage виконує один GraphQL-запит під час збірки — і повертає весь масив з 18 проєктів. Жодного client-side fetch не відбувається. gatsby-plugin-graphql-codegen автоматично генерує TypeScript-інтерфейси зі схеми запиту:

компонент впевнений, що поля title, client, year, coverImage відповідають YAML-структурі ще до gatsby build.

Кожна з 18 сторінок проєктів TATREAL отримує унікальний набір метатегів через Gatsby 5 Head API: іменований експорт HeadFC в шаблоні Project.tsx динамічно формує title як 'назва | Tatreal', og:image з CDN-URL обкладинки, canonical з абсолютним URL, Twitter Card та Open Graph — всі з pageContext. Sitemap (20 URL: 18 проєктів + головна + 404) з lastmod з Git-комітів — Lighthouse SEO = 100/100.

```tsx
// src/templates/Project.tsx — SEO Head компонент
import type { HeadFC } from 'gatsby'

interface ProjectPageContext {
  project:     Project
  allProjects: Project[]
}

export const Head: HeadFC<object, ProjectPageContext> = ({
  pageContext: { project },
}) => {
  const title = `${project.title} | Tatreal`
  const desc  = project.description
             ?? `Проєкт ${project.title} — sound design by Tatreal`
  const image = `https://tatreal.pro${project.coverImage}`
  const url   = `https://tatreal.pro/projects/${project.slug}/`

  return (
    <>
      <title>{title}</title>
      <meta name='description'         content={desc} />
      <meta property='og:title'        content={title} />
      <meta property='og:description'  content={desc} />
      <meta property='og:image'        content={image} />
      <meta property='og:url'          content={url} />
      <meta property='og:type'         content='article' />
      <meta name='twitter:card'        content='summary_large_image' />
      <meta name='twitter:title'       content={title} />
      <meta name='twitter:image'       content={image} />
      <link rel='canonical'            href={url} />
    </>
  )
}
```

Рисунок 2.5 — Компонент Head для SEO-метатегів (Project.tsx)

Реалізацію SEO Head-компонента для шаблону сторінки проєкту наведено у рисунок 2.5. Gatsby Head API автоматично рендерить повернений JSX у секцію <head> HTML-документа під час збірки, забезпечуючи унікальні метатеги для кожної з 18 згенерованих сторінок.

gatsby-browser.tsx виконує три реєстраційні дії при запуску браузера: createGlobalStyle із styled-components оголошує CSS-змінні дизайн-системи, шрифтові стеки та modern-CSS-reset у :root; library.add() FontAwesome декларує доступні SVG-іконки без завантаження всієї бібліотеки; оголошуються CSS-анімації kenBurns та fadeIn. CSS-змінні у :root доступні у будь-якому styled-component без Context або ThemeProvider.

Конфігураційний файл gatsby-config.ts визначає метадані сайту та реєструє всі плагіни. Плагін gatsby-plugin-image разом із gatsby-transformer-sharp та gatsby-plugin-sharp формують конвеєр обробки зображень: під час збірки генеруються набори розмірів у форматах WebP та AVIF з автоматичним LQIP (Low Quality Image Placeholder) у вигляді розмитого base64-рядка. Плагін gatsby-plugin-manifest генерує Web App Manifest для підтримки PWA-функцій. Плагін gatsby-plugin-google-gtag виконує відкладену ін'єкцію скрипта Google Analytics тільки після гідрації сторінки, не блокуючи рендеринг і не впливаючи на метрику TTI. Плагін gatsby-plugin-netlify-cms монтує SPA-адміністративну панель у статичний маршрут /admin/ та генерує необхідні файли для автентифікації через Netlify Identity.

Окрім динамічної генерації сторінок через createPages, файл gatsby-node.ts реалізує хук onCreateWebpackConfig для тонкого налаштування webpack-конфігурації. Зокрема, додаються webpack aliases (@components → src/components, @images → src/images), що скорочують шляхи імпорту у всіх компонентах і спрощують рефакторинг. Хук createSchemaCustomization визначає GraphQL-тип ProjectNode з явними типами полів — це запобігає помилкам типізації GraphQL при відсутніх полях у YAML (наприклад, якщо поле link відсутнє для деяких проєктів). Через явне визначення типів усі поля гарантовано матимуть очікуваний тип навіть при неповних даних, що запобігає збоям збірки.

```ts
// gatsby-node.ts — розширений фрагмент
import type { GatsbyNode } from 'gatsby'
import * as path from 'path'

// 1. Явне визначення GraphQL-схеми
export const createSchemaCustomization:
  GatsbyNode['createSchemaCustomization'] = ({ actions }) => {
  actions.createTypes(`
    type ProjectNode implements Node {
      title:       String!
      slug:        String!
      year:        String!
      role:        String!
      client:      String
      description: String
      coverImage:  String!
      link:        String
      featured:    Boolean!
    }
  `)
}

// 2. Webpack aliases для зручних імпортів
export const onCreateWebpackConfig:
  GatsbyNode['onCreateWebpackConfig'] = ({ actions }) => {
  actions.setWebpackConfig({
    resolve: {
      alias: {
        '@components': path.resolve(__dirname, 'src/components'),
        '@images':     path.resolve(__dirname, 'src/images'),
        '@templates':  path.resolve(__dirname, 'src/templates'),
      },
    },
  })
}

// Завдяки aliases: замість '../../components/Header'
// можна писати: import Header from '@components/Header'
```

Рисунок 2.6 — gatsby-node.ts: createSchemaCustomization та webpack aliases

Статичні активи проєкту організовані у директорії static. Фонові зображення Hero-секції зберігаються у static/bg та підключаються через gatsby-source-filesystem з ім'ям 'backgrounds'. Це дозволяє звертатися до них через GraphQL-запити та застосовувати автоматичну оптимізацію зображень. Файли адміністративної панелі CMS (config.yml, index.html) розміщені у static/admin і копіюються в директорію public без змін під час збірки. Директорія static/assets зберігає медіафайли проєктів, логотип і SVG-ресурси, посилання на які зберігаються в YAML-файлі контенту у вигляді відносних шляхів /assets/назва-файлу.

gatsby-transformer-yaml зчитує кожен .yml-файл, розпарсений через gatsby-source-filesystem, та перетворює його у внутрішній GraphQL-вузол типу PagesYaml (або AnnotationYaml, HomeYaml залежно від назви файлу). Для вкладених масивів YAML - таких як масив projects у home.yml - трансформер рекурсивно обходить структуру і створює дочірні вузли для кожного елемента масиву. Проте у проєкті Tatreal gatsby-transformer-yaml не використовується напряму для генерації сторінок: замість GraphQL-запитів до PagesYaml-вузлів, gatsby-node.ts зчитує YAML-файл безпосередньо через Node.js модулі fs та js-yaml в рамках хука createPages. Це рішення обрано для максимального контролю над процесом: пряме читання файлу дозволяє передавати весь масив allProjects у context кожної сторінки для реалізації навігації між проєктами, тоді як GraphQL не підтримує передачу масивів об'єктів у pageContext без явної схеми.

Хук onPostBuild у gatsby-node.ts виконується після завершення всіх збірних операцій і використовується для post-processing артефактів збірки. У проєкті він застосований для автоматичної генерації файлу sitemap.xml та robots.txt: плагін gatsby-plugin-sitemap звертається до onPostBuild, ітерує всі згенеровані HTML-сторінки та формує XML-карту сайту у директорії public/. Sitemap.xml є критично важливим для SEO: він повідомляє пошукові системи про структуру сайту та пріоритети індексування. Для проєкту Tatreal sitemap містить 20 URL: головна сторінка, 404-сторінка та 18 сторінок проєктів з атрибутами lastmod (дата останнього оновлення з Git-комітів) та changefreq: monthly. Файл robots.txt

генерується автоматично із правилом User-agent: * / Allow: / та посиланням на Sitemap: https://tatreal.netlify.app/sitemap.xml.

Для попереднього перегляду контенту перед релізом використовується локальне середовище gatsby develop. Інтеграцію хмарного Gatsby Cloud Preview відхилено через надлишковість: портфоліо з одним контент-менеджером потребує лише kubectl preview-середовища - витрати на окремий Preview-сервер не виправдані. Відповідно, поле draft: true у YAML - лише фільтр, актуальний при масштабуванні команди редакторів.

2.7 Реалізація контактної форми та розгортання застосунку

Контактна форма інтегрована з Netlify Forms - сервісом, що автоматично детектує HTML-форми під час збірки за атрибутом data-netlify="true" і бере на себе приймання та зберігання повідомлень без серверного коду. Оскільки Formik рендерить форму динамічно через JavaScript, у документ додано прихований статичний HTML-варіант форми з атрибутом hidden - Netlify знаходить його при збірці і реєструє форму.

Схема валідації Yup перевіряє: firstName - мінімум 2 символи, обов'язкове; email - коректний формат, обов'язковий; message - мінімум 10 символів, обов'язкове. При відправці виконується fetch-запит методом POST на адресу /, дані кодуються у форматі application/x-www-form-urlencoded разом із полем form-name, яке ідентифікує форму в Netlify.

```
const encode = (data: Record<string, string>) =>
  Object.keys(data)
    .map(k => encodeURIComponent(k) + '=' + encodeURIComponent(data[k]))
    .join('&')

await fetch('/', {
  method: 'POST',
  headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
  body: encode({ 'form-name': 'contact', ...values }),
})
```

Рисунок 2.7 - Функція encode та відправка форми (ContactMe.tsx)

Git push у головну гілку репозиторію GitHub запускає gatsby build та публікацію директорії public/ через глобальну CDN-мережу Netlify. TTFB на CDN Fastly — 28–45 мс незалежно від геолокації слухача.

Для працездатності адмін-панелі Decap CMS netlify.toml містить обов'язкове правило redirect /admin/* → /admin/index.html 200 — без нього прямий перехід за URL /admin/ поверне 404. Обробку лідів форми типізовано через hidden-поле form-name та кодування x-www-form-urlencoded. Файл також фіксує BUILD-параметри: build.command: yarn build, publish: public, NODE_VERSION: 20 — відтворюване середовище збірки. Секція [[headers]] встановлює HTTP-заголовки безпеки: Content-Security-Policy, X-Frame-Options: DENY, X-Content-Type-Options: nosniff.

Автоматизований CI/CD-конвеєр Netlify виконує повний цикл — від webhook-тригера до живого сайту — за 1 хвилину 40 секунд. Ініціалізація: Netlify клонує репозиторій у ізольованому контейнері та відновлює залежності з yarn.lock. Збірка: gatsby build проходить чотири фази послідовно — bootstrap (плагіни + GraphQL-схема), createPages (18 сторінок), renderHTML, optimizeImages. Деплой: директорія public/ розгортається на CDN Fastly, DNS оновлюється автоматично.

За результатами тестування Google Lighthouse у режимі Desktop для продакшн-версії сайту https://tatreal.netlify.app/ отримано показники, наведені у таблиці 2.5. Всі чотири категорії — Performance, Accessibility, Best Practices та SEO — досягли або максимального значення, або близького до нього.

Таблиця 2.4 - Результати тестування Google Lighthouse (Desktop)

Категорія / метрика	Значення	Порогове значення	Оцінка
Performance	91 / 100	≥ 90	Відмінно
Accessibility	100 / 100	≥ 90	Відмінно
Best Practices	100 / 100	≥ 90	Відмінно
SEO	100 / 100	≥ 90	Відмінно

LCP (Largest Contentful Paint)	0.7 с	≤ 2.5 с	Відмінно
INP (Interaction to Next Paint)	25 мс	≤ 200 мс	Відмінно
CLS (Cumulative Layout Shift)	0.09	≤ 0.1	Відмінно
FCP (First Contentful Paint)	0.4 с	≤ 1.8 с	Відмінно
TBT (Total Blocking Time)	12 мс	≤ 200 мс	Відмінно

CSP frame-src https://open.spotify.com - єдиний iframe-дозвіл у TATREAL: Spotify Embed API. HSTS max-age=31536000 примусово переводить усі запити на HTTPS протягом року. script-src 'self' 'unsafe-inline' покриває inline-скрипти Gatsby. Такий строгий CSP унеможливлює Cross-Site Scripting (XSS) атаки. Заголовок Strict-Transport-Security: max-age=31536000; includeSubDomains примусово переводить усі HTTP-запити на HTTPS протягом одного року, запобігаючи атакам SSL stripping. Netlify автоматично отримує та поновлює TLS-сертифікат Let's Encrypt без жодного втручання розробника. Заголовок Referrer-Policy: strict-origin-when-cross-origin обмежує передачу заголовку Referer, захищаючи URL-адреси приватних сторінок.

Функція Deploy Previews у Netlify автоматично розгортає тимчасову копію сайту для кожного відкритого Pull Request у GitHub. Кожен preview отримує унікальний URL виду https://deploy-preview-<PR-number>--tatreal.netlify.app, — перегляд змін ізольовано від продакшн-сайту без злиття гілки у main. Deploy Previews успішно спрацьовували для кожної feature-гілки під час розробки TATREAL. Netlify автоматично додає коментар до Pull Request у GitHub із посиланням на preview та результатами збірки. Для одноосібного проєкту Deploy Previews особливо корисні при тестуванні великих змін: наприклад, реструктуризація YAML-схеми або оновлення версії Gatsby може переглядатися на preview-середовищі перед злиттям, що унеможливлює попадання некоректних змін

у продакшн. Додатково, Netlify виконує перевірку заголовків безпеки, DNS-налаштувань та TLS-сертифіката для кожного preview, надаючи звіт у панелі адміністратора.

TATREAL використовує дві паралельні системи аналітики. Netlify Analytics читає CDN-логи серверно — дані точніші, не залежать від ad-block та ботів, GDPR-safe через анонімізацію IP. Google Analytics (gatsby-plugin-google-gtag, ін'єкція відкладається до гідрації) доповнює поведінковими метриками: Метрики включають: унікальні відвідувачі, переглянуті сторінки, джерела трафіку, географічний розподіл та топ-сторінки за переглядами. Для проєкту Tatreal Google Analytics (через gatsby-plugin-google-gtag) використовується паралельно для детальнішого аналізу поведінки користувачів: глибина прокрутки, кліки на CTA-кнопки та час на сторінці — дані, недоступні у серверній аналітиці.

При 18 проєктах у YAML-файлі повна збірка займає ~90 с - оптимально для поточного обсягу. При збільшенні до 100+ час збірки зростає лінійно $O(n)$ - може ускладнити workflows. Для масштабування передбачено кілька стратегій: по-перше, інкрементна збірка Gatsby (описана в підрозділі 1.3) суттєво скорочує час при додаванні нових проєктів; по-друге, сегрегація контенту по окремих YAML-файлах із gatsby-transformer-yaml замість одного монолітного home.yml дозволяє інвалідувати кеш лише для змінених файлів; по-третє, перехід на Contentful або Sanity як бекенд-джерело даних (замість Git-based Netlify CMS) забезпечить реальний Preview та більш гнучку схему контенту при збережені незмінним фронтенду на Gatsby.

Таблиця 2.5 — Стратегії масштабування застосунку при зростанні контенту

Кількість проєктів	Стратегія збірки	Час повної збірки	Рекомендовані зміни
1–50 (поточний стан)	Повна + інкрементна	~90 с / ~15–20 с	Поточна архітектура оптимальна
50–200	Інкрементна обов'язкова	~60 с / ~20–30 с	Сегрегація YAML по категоріях

200–500	DSG (Deferred Static)	~120 с / ~40–60 с	Gatsby DSG для рідкісних сторінок
500+	On-demand ISR	~180 с / ~60–90 с	Перехід на Contentful/Sanity CMS

Gatsby 5, Netlify та Decap CMS у поєднанні закривають усі вісім вимог підрозділа 2.1: нульова вартість хостингу, CI/CD від git push до CDN за ~100 секунд, браузерна адмін-панель без серверного коду - у єдиній інтегрованій екосистемі.

Управління змінними середовища (Environment Variables) реалізовано через механізм Netlify Environment Variables у панелі адміністратора. Публічні змінні (з префіксом GATSBY_) доступні у клієнтському коді: GATSBY_GA_TRACKING_ID (ідентифікатор Google Analytics) використовується плагіном gatsby-plugin-google-gtag. Приватні змінні без префікса GATSBY_ доступні лише під час збірки на сервері Netlify і ніколи не потрапляють у клієнтський бандл. Для локальної розробки змінні зберігаються у файлі .env.development, виключеному з репозиторію через .gitignore. Такий підхід гарантує, що чутливі дані (API-ключі, OAuth-секрети) ніколи не будуть видимі у публічному репозиторії.

Порівняльний аналіз продуктивності розробленого застосунку відносно типових CMS-рішень підтверджує теоретичні переваги JAMstack-архітектури, обґрунтовані у Розділі 1. За даними PageSpeed Insights (Google, 2024) [2], медіанний показник Performance у Lighthouse для WordPress-сайтів без оптимізації становить 45–62 бали. Розроблений застосунок Tatreal на Gatsby отримав 91 бал (вимірювання 14.05.2025, Desktop режим), що є результатом статичної генерації, автоматичної оптимізації зображень, code-splitting та CDN-розгортання. Метрика LCP - 0.7 с - нижче порогу “Good” ($\leq$ 2.5 с) згідно зі стандартом Core Web Vitals [2]. Показник TBT -12 мс при нормативному порозі 200 мс - свідчить про відсутність JavaScript-задач, що блокують головний потік браузера. Виміряне TTFB для статичних

активів з CDN становить 28–45 мс, тоді як типовий TTFB для PHP-сервера з базою даних - 300–800 мс [2].

ВИСНОВКИ

У кваліфікаційній роботі спроєктовано і реалізовано вебзастосунок-портфоліо TATREAL для музичного продюсера Вячеслава Користова на стеці Gatsby 5, React 18, TypeScript, GraphQL та Netlify/Decap CMS. Продакшн-версія на https://tatreal.netlify.app/ верифікована через Lighthouse CI: Performance 91/100, LCP 0.7 с, TBT 12 мс, CLS 0.09 — всі чотири метрики Core Web Vitals у категорії “Good”. TTFB на CDN Fastly — 28–45 мс проти 300–800 мс типового PHP + MySQL-сервера.

– Порівняльний аналіз чотирьох SSG-фреймворків (Gatsby 5, Next.js 14, Astro 4, Hugo) встановив єдиний варіант із вбудованим GraphQL Data Layer та офіційним плагіном Netlify CMS — Gatsby 5. Математична модель $O(k \cdot r)$ підтвердила: інкрементна збірка скорочує час із 90 с до 15 с при оновленні одного YAML-запису. JAMstack-архітектура перевела вектор атаки SQL + RCE у нуль — CDN роздає статичні файли без жодного активного бекенду.

– Дизайн-систему побудовано на модульній типографічній шкалі 1.25 (Major Third: 12–39 px), 4 брейкпоінтах Mobile-First (375–1480 px) та fluid-відступах CSS clamp(1.5rem, 4vw, 4rem). Gatsby-plugin-image резервує розміри під кожну обкладинку релізу до підвантаження — CLS = 0.09 без жодного ручного placeholder.

– Дев’ять React/TypeScript-компонентів реалізовано з Loose Coupling: жоден не імпортує інший горизонтально, залежності лише вертикальні. Webpack Tree-shaking та React.lazy() скоротили бандл головної сторінки до 130 KB gzipped — утричі менше за медіанний WordPress-сайт (~450 KB). INP = 25 мс: жодна взаємодія не блокує головний потік.

– Git-based Netlify/Decap CMS підключено через createSchemaCustomization: явна типізація 18 полів ProjectNode усунула клас runtime-помилок при неповних YAML-записах — нуль збоїв збірки за весь продакшн-цикл. Вячеслав Користов редагує контент через /admin/ без технічного фахівця.

– Gatsby Node API генерує 18 сторінок за маршрутом /projects/{slug}/. Gatsby 5 Head API формує унікальні OG-метатеги для кожної з pageContext: title, og:image з CDN-URL обкладинки, canonical. Sitemap (20 URL з lastmod із Git-комітів) — Lighthouse SEO = 100/100.

– Контактна форма (Formik + Yup, abortEarly: false) відправляє дані через Netlify Forms POST без власного бекенду. Honeypot-поле bot-field блокує спам-ботів без капчі. :focus-visible стилі та aria-label для всіх іконок — Lighthouse Accessibility = 100/100.

– CI/CD Netlify: git push → живий сайт на CDN Fastly за ~100 секунд. TTFB 28–45 мс незалежно від геолокації. Заголовки безпеки netlify.toml: HSTS max-age=31536000, CSP frame-src https://open.spotify.com, X-Frame-Options: DENY. Застосунок функціонує як офіційне портфоліо https://tatreal.netlify.app/ з вересня 2025.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Пасічник В. В., Жежнич П. І., Кравець П. О. Глобальні інформаційні системи та технології : навч. посібник. Львів : Видавництво Львівської політехніки, 2017. 240 с.

2. Осадчий В. В., Шаров С. В., Коваленко Т. С. Архітектура та проектування сучасних вебсистем : навч. посібник. Мелітополь : Вид-во МДПУ ім. Б. Хмельницького, 2021. 182 с.

3. Федорчук В. А. Мова моделювання UML: проектування інформаційних систем : навч. посібник. Кам’янець-Подільський : Зволейко Д. Г., 2019. 216 с.

4. Марголін О. UML для бізнес-моделювання: для чого потрібні діаграми процесів. 2021. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html.

5. Ткаченко О. М., Кравець О. В. Компонентно-орієнтоване моделювання інтерфейсів користувача в середовищі React. *Сучасні інформаційні технології*. 2023. Вип. 4 (32). С. 45–54.

6. Основи автоматизованого проектування складних об’єктів та систем : навч. посібник / В. О. Яковенко та ін. Дніпро : Університет митної справи та фінансів, 2018. 114 с.

7. Contentful Developer Docs. URL: https://www.contentful.com/developers/docs/

8. Gatsby Documentation. URL: https://www.gatsbyjs.com/docs/ (дата звернення: 01.05.2026).

9. React Documentation. URL: https://react.dev/reference/react (дата звернення: 01.05.2026

10. Introduction to GraphQL. URL: https://graphql.org/learn/ (дата звернення: 10.04.2026).

11. Core Web Vitals Optimization Guide. URL: https://web.dev/explore/vitals (дата звернення: 15.04.2026).

12. Netlify CMS Documentation. URL: https://www.netlifycms.org/docs/intro/ (дата звернення: 01.05.2026).

13. Netlify Forms Documentation. URL: https://docs.netlify.com/forms/setup/ (дата звернення: 01.05.2026).

14. styled-components Documentation. URL: https://styled-components.com/docs (дата звернення: 05.05.2026).

15. Formik Documentation. URL: https://formik.org/docs/overview (дата звернення: 05.05.2026).

16. Yup Validation Library. URL: https://github.com/jquense/yup (дата звернення: 05.05.2026).

17. react-slick Carousel. URL: https://react-slick.neostack.com/ (дата звернення: 10.04.2026).

18. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ : ДП «УкрНДНЦ», 2016. 26 с.

19. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання. Київ : ДП «УкрНДНЦ», 2016. 16 с.

20. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Київ : ДП «УкрНДНЦ», 2013. 23 с.

ДОДАТОК А

Лістинги ключових модулів застосунку

Лістинг А.1 — gatsby-node.ts: динамічна генерація сторінок проєктів

```
import type { GatsbyNode } from 'gatsby'
import * as path from 'path'
import * as fs from 'fs'
import * as yaml from 'js-yaml'

export const createPages: GatsbyNode['createPages'] =
  async ({ actions, reporter }) => {
    const { createPage } = actions
    const data = yaml.load(
      fs.readFileSync(path.join(__dirname, 'content/pages/home.yml'),
'utf8')
    ) as any
    const projects = data?.main?.projects ?? []
    const template = path.resolve('./src/templates/Project.tsx')
    projects.forEach((project: any) => {
      if (!project.slug) return
      createPage({
        path: `/projects/${project.slug}/`,
        component: template,
        context: { project, allProjects: projects },
      })
    })
    reporter.info(`Created ${projects.length} project pages`)
  }
```

Лістинг А.2 — ContactMe.tsx: логіка відправлення форми через Netlify Forms

```
const encode = (data: Record<string, string>) =>
  Object.keys(data)
    .map(k => encodeURIComponent(k) + '=' + encodeURIComponent(data[k]))
    .join('&')

const handleSubmit = async (values: ContactValues,
  helpers: FormikHelpers<ContactValues>) => {
  try {
    await fetch('/', {
      method: 'POST',
```

```
      headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
      body: encode({ 'form-name': 'contact', ...values }),
    })
    setSubmitted(true)
    helpers.resetForm()
  } catch (err) {
    helpers.setSubmitting(false)
```

Лістинг А.3 — Header.tsx (компонент навігаційної панелі)

```
import React, { useState, useEffect } from 'react'
import styled, { css } from 'styled-components'
import { Link } from 'gatsby'
import { animateScroll as scroll } from 'react-scroll'
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'
import { faBars, faTimes } from '@fortawesome/free-solid-svg-icons'

interface NavData {
  logo: string
  nav: { item1: string; item2: string; item3: string }
  button: string
}

interface Props { data: NavData; isProjectPage?: boolean }

const Wrapper = styled.header<{ $scrolled: boolean }>`
  position: fixed; top: 0; left: 0; right: 0; z-index: 100;
  display: flex; align-items: center;
  justify-content: space-between;
  padding: 0 clamp(1.5rem, 4vw, 4rem);
  height: 64px;
  transition: background 0.3s ease,
              backdrop-filter 0.3s ease,
              box-shadow 0.3s ease;
  ${({ $scrolled }) => $scrolled && css`
    background: rgba(13, 12, 10, 0.85);
    backdrop-filter: blur(14px);
    box-shadow: 0 1px 0 var(--c-border);
  `}
`
```

```
const Header: React.FC<Props> = ({ data, isProjectPage = false }) => {
  const [scrolled, setScrolled]   = useState(false)
  const [menuOpen, setMenuOpen]   = useState(false)

  useEffect(() => {
    const onScroll = () => setScrolled(window.scrollY > 60)
    window.addEventListener('scroll', onScroll, { passive: true })
    return () => window.removeEventListener('scroll', onScroll)
  }, [])

  const handleNavClick = (target: string) => {
    if (isProjectPage) {
      window.location.href = `/#${target}`
    } else {
      setMenuOpen(false)
    }
  }

  return (
    <Wrapper $scrolled={scrolled} role='banner'>
      <Link to='/' aria-label='Перейти на головну'>
        <img src={data.logo} alt='Tatreal' width={80} height={28} />
      </Link>
      <nav aria-label='Основна навігація'>
        {/* nav items rendered here */}
      </nav>
    </Wrapper>
  )
}

export default Header
```

Лістинг А.4 — src/theme.ts (токени дизайн-системи)

```
export const theme = {
  colors: {
    bg:           '#0d0c0a',
    bgElevated:   '#171612',
    surface:      '#1e1c17',
    surface2:     '#2a2722',
    accent:       '#e8d574',
```

```
    accentHover:  '#f5e28a',
    text:         '#f5f3ec',
    textMuted:    '#a5a097',
    textSubtle:   '#6b635c',
    border:       'rgba(245, 243, 236, 0.08)',
    borderHover:  'rgba(245, 243, 236, 0.16)',
  },
  shadows: {
    card:   '0 4px 24px rgba(0,0,0,0.4)',
    glow:   '0 0 32px rgba(232,213,116,0.08)',
    inset:  'inset 0 1px 0 rgba(255,255,255,0.04)',
  },
  radius: {
    sm: '6px', md: '12px', lg: '20px', pill: '999px',
  },
  breakpoints: {
    sm: '600px', md: '900px', lg: '1200px', xl: '1480px',
  },
  transitions: {
    fast: 'all 0.15s ease',
    base: 'all 0.3s ease',
    slow: 'all 0.5s cubic-bezier(0.25, 0.46, 0.45, 0.94)',
  },
  spacing: {
    section:    'clamp(4rem, 8vw, 8rem)',
    container:  'clamp(1.5rem, 4vw, 4rem)',
    cardGap:    'clamp(0.75rem, 1.5vw, 1.25rem)',
  },
} as const

export type Theme = typeof theme
  }
}
```

ДОДАТОК Б

YAML-схема контенту та конфігурація Gatsby

Лістинг Б.1 — content/pages/home.yml (фрагмент секції projects)

```
main:
  projects:
    - coverImage: /assets/projects/red-impact.jpg
      slug: red-impact
      client: ''
      featured: true
      role: Music Score, SFX
      title: Red Impact
      link: https://tatreal.netlify.app/red-impact-sound-design/
      description: Planetary defence game with massive battles.
      year: '2024'
    - coverImage: /assets/projects/art-of-rally.jpg
      slug: art-of-rally
      client: art of rally
      featured: true
      role: Music, Ambience, Sound Effects
      title: art of rally - The Game
      year: '2019'
  music:
    spotify: https://open.spotify.com/artist/12JiWFZAQNYHhhmpxlcwcp
    bandcamp: https://tatreal.bandcamp.com/
    soundcloud: https://soundcloud.com/tatreal
  social:
    instagram: https://www.instagram.com/tatreal/
    twitter: https://twitter.com/SlavaTatreal
```

Лістинг Б.2 — gatsby-config.ts: реєстрація плагінів

```
const config: GatsbyConfig = {
  siteMetadata: {
    title: 'Tatreal',
    siteUrl: https://tatreal.netlify.app/',
  },
  graphqlTypegen: true,
```

```
  plugins: [
    'gatsby-plugin-netlify-cms',
    'gatsby-plugin-styled-components',
    'gatsby-plugin-image',
    'gatsby-plugin-sharp',
    'gatsby-transformer-sharp',
    { resolve: 'gatsby-plugin-google-gtag',
      options: { trackingIds: ['GA-TRACKING-ID'] } },
    { resolve: 'gatsby-source-filesystem',
      options: { name: 'backgrounds', path: 'static/bg' } },
  ],
}
export default config
```

Лістинг Б.3 — static/admin/config.yml (конфігурація Netlify CMS)

```
backend:
  name: git-gateway
  branch: main
  commit_messages:
    create:  'Create {{collection}} "{{slug}}"'
    update:  'Update {{collection}} "{{slug}}"'
    delete:  'Delete {{collection}} "{{slug}}"'

media_folder:  static/assets
public_folder: /assets
publish_mode:  editorial_workflow
locale: uk

collections:
  - name: pages
    label: Сторінки
    files:
      - name: home
        label: Головна сторінка
        file: content/pages/home.yml
        fields:
          - { label: Заголовок вкладки, name: docTitle, widget: string }
          - label: Шапка сайту
            name: header
            widget: object
```

```
      fields:
        - { label: Логотип, name: logo, widget: image }
        - { label: Кнопка CTA, name: button, widget: string }
        - label: Навігація
          name: nav
          widget: object
          fields:
            - { label: Пункт 1, name: item1, widget: string }
            - { label: Пункт 2, name: item2, widget: string }
            - { label: Пункт 3, name: item3, widget: string }
    - label: Основний контент
      name: main
      widget: object
      fields:
        - label: Проєкти
          name: projects
          widget: list
          fields:
            - { label: Назва, name: title, widget: string }
            - { label: Slug, name: slug, widget: string }
            - { label: Рік, name: year, widget: string }
            - { label: Роль, name: role, widget: string }
            - { label: Клієнт, name: client, widget: string,
                required: false }
            - { label: Опис, name: description, widget: text,
                required: false }
            - { label: Обкладинка, name: coverImage, widget: image }
            - { label: Посилання, name: link, widget: string,
                required: false }
            - { label: Вибраний, name: featured,
                widget: boolean, default: false }
```

Лістинг Б.4 — gatsby-browser.tsx (глобальні стилі та ін'єкції)

```
import React from 'react'
import type { GatsbyBrowser } from 'gatsby'
import { createGlobalStyle } from 'styled-components'
import { library } from '@fortawesome/fontawesome-svg-core'
import { fas } from '@fortawesome/free-solid-svg-icons'
import { fab } from '@fortawesome/free-brands-svg-icons'
```

```
library.add(fas, fab)

const GlobalStyles = createGlobalStyle`
  *, *::before, *::after {
    box-sizing: border-box; margin: 0; padding: 0;
  }
  :root {
    --c-bg:          #0d0c0a;
    --c-bg-elevated: #171612;
    --c-surface:     #1e1c17;
    --c-surface-2:   #2a2722;
    --c-accent:      #e8d574;
    --c-text:        #f5f3ec;
    --c-text-muted:  #a5a097;
    --c-border:      rgba(245,243,236,.08);
    --radius-sm:     6px;
    --radius-md:     12px;
    --radius-pill:   999px;
    --section-gap:   clamp(4rem, 8vw, 8rem);
    --container-px:  clamp(1.5rem, 4vw, 4rem);
    --font-sans:     'Inter', -apple-system, BlinkMacSystemFont,
                     'Segoe UI', Roboto, Oxygen, sans-serif;
  }
  html { scroll-behavior: smooth; }
  body {
    background: var(--c-bg);
    color: var(--c-text);
    font-family: var(--font-sans);
    font-size: 16px; line-height: 1.65;
    -webkit-font-smoothing: antialiased;
    overflow-x: hidden;
  }
  a { color: inherit; text-decoration: none; }
  img, svg { display: block; max-width: 100%; }
  button { cursor: pointer; border: none; background: none; font: inherit; }
  @keyframes kenBurns {
    0%   { transform: scale(1)    translate(0, 0); }
    50%  { transform: scale(1.08) translate(-1%, -0.5%); }
    100% { transform: scale(1.12) translate(-2%, -1%); }
  }
  @keyframes fadeIn {
    from { opacity: 0; transform: translateY(12px); }
```

```
    to   { opacity: 1; transform: translateY(0); }
  }
`

export const wrapRootElement: GatsbyBrowser['wrapRootElement'] =
  ({ element }) => (
    <>
      <GlobalStyles />
      {element}
    </>
  )
```