Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА
до дипломного проекту (роботи)

магістр
(освітній ступінь (освітньо-кваліфікаційний рівень))

на тему: **Розробка сайт для сервісного центру з обслуговування комп'ютерної техніки з використанням технології JavaScript**

Виконав: студент (ка) 6 курсу, групи СПм-61
спеціальності (напряму підготовки)
(шифр і назва спеціальності (напряму підготовки))

		Равчак Б.Ю.
	(підпис)	(прізвище та ініціали)
Керівник		Баран І.О.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Бойко І.В.
	(підпис)	(прізвище та ініціали)
Рецензент		Литвиненко Я.В.
	(підпис)	(прізвище та ініціали)

м. Тернопіль – 2019

АНОТАЦІЯ

Магістерська робота на тему «Розробка сайту для сервісного центру з обслуговування комп'ютерної техніки з використанням технології JavaScript» Равчака Богдана Юрійовича. – Тернопільський національний технічний університет імені Івана Пулюя, Факультет комп'ютерно-інформаційних систем і програмної інженерії, Кафедра програмної інженерії, група СПм–61 // Тернопіль, 2019.

Пояснююча записка до звіту з магістерської роботи складається з 75 сторінок, 6 таблиць, 15 рисунків, список використаної літератури з 31 найменувань, 4 додатки, 12 слайдів.

Метою магістерської роботи є створення інформаційного веб-сайту на основі перспективних технологій, щоб підвищити його продуктивність і стійкість до збоїв, викликаних трафіком, а також зменшити вартість розробки та супроводу.

При розробці було використано методологію JAMstack, систему котролю версій Git, робота написана за допомогою технологій JavaScript, GatsbyJS, Contentful, Netlify.

На початку звіту магістерської роботи надається загальна інформація про теоретичну і документаційну складову роботи, а саме дослідження інформації, котра стосується розробки веб-сайту за темою магістерської роботи.

У подальших частинах звіту описується процес вибору середовища, фреймворків JavaScript та API, тестування, а також процес розгортання розробленої системи.

У результаті було здійснено дослідницько-інженерну роботу та завершено розробку програмного забезпечення за темою магістерської роботи.

КЛЮЧОВІ СЛОВА: ВЕБ-САЙТ, JAVASCRIPT, JAMSTACK, СЕРВІСНИЙ ЦЕНТР, ФРОНТ-ЕНД.

ANNOTATION

Master's thesis on "Development of website for service center of computer equipment using JavaScript technology" by Ravchak Bohdan. – Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, SPm-61 group // Ternopil, 2019.

Explanatory note to the report on master's work contains of 75 pages, 6 tables, 15 figures, a list of used literature of 31 titles, 4 applications, 12 slides.

The purpose of the master's thesis is to create an information website based on advanced technologies to increase its productivity and resilience to traffic-induced disruptions, as well as to reduce the cost of development and maintenance.

The development used the JAMstack methodology, Git version control system, the work was written using JavaScript, GatsbyJS, Contentful, Netlify technologies.

At the beginning of the master's report, general information about the theoretical and documentary component of the work is provided, namely the study of information relevant to the development of a website on the topic of the master's thesis.

The following parts of the report describe the process of selecting the environment, JavaScript and API frameworks, testing, and the process of deploying the developed system.

As a result, research and engineering work was completed and software development on the topic of master's work was completed.

KEYWORDS: WEB SITE, JAVASCRIPT, JAMSTACK, SERVICE CENTER, FRONT-END.

ЗМІСТ

ВСТУП

В даний час все більше потенційних клієнтів шукають необхідні послуги в мережі Інтернет. Тому на сьогодні всі організації заявляють про свою діяльність, в першу чергу саме в мережевих ресурсах (сайти оголошень, електронні довідники організацій та ін.). Крім цього багатьом організаціям (включаючи індивідуальних підприємців і нечисленні організації) доцільно мати власний сайт. За допомогою веб-сайту організація зможе охопити більш велику цільову аудиторію. Великим плюсом наявності власного веб-сайту для компанії є безкоштовне самопросування.

Сучасні інформаційні технології дозволяють створити сайт, який буде забезпечувати вирішення комплексу завдань: реклами послуг, інформування потенційних клієнтів, ведення клієнтської бази, автоматизації прийому заявок на послуги. Крім того, наявність сайту організації, згідно з існуючими результатами опитувань, підвищує її престиж і рейтинг. Основна мета, для якої створюються сайти - це залучення додаткових клієнтів, які повинні отримати достатню для прийняття рішення інформацію. Тому в процесі проектування сайту необхідно представити детальний аналіз предметної області діяльності організації. Управління в сфері сервісу підпорядковане завданням ринкового розвитку виробництва послуг (організація праці персоналу, підвищення продуктивності праці працівників, досягнення конкурентоспроможності послуг, їх прибутковість і ін.). Одночасно воно націлюється на реалізацію різноманітних запитів і потреб великих груп людей, конкретних індивідуумів, які вступають з виробником

Користувачі комп'ютерів постійно стикаються з різними типами проблем. Деякі проблеми пов'язані з обладнанням, а інші - з програмним забезпеченням або мережею. Якщо вони не можуть самостійно діагностувати справжню причину проблеми, то зазвичай звертаються до сервісного центру з обслуговування комп'ютерної техніки. Вони можуть доставити комп’ютер у сервісний центр або замовити сервіс на місці. Послуга на місцях обійдеться

трохи дорожче, але тоді не доведеться перевозити апаратне забезпечення комп'ютера аж до сервісного центру. Підприємства, які часто потребують послуг для своїх комп’ютерів, практикують підписання щорічного контракту на обслуговування. Це допомагає зменшити витрати, пов'язані з ремонтом комп’ютерів та сервісами.

Проблеми, які зазвичай потребують ремонту комп’ютера, включають несправні апаратні компоненти, програмні помилки, несумісність драйверів, проблеми з шпигунським та зловмисним програмним забезпеченням, проблеми з підключенням до мережі, оновлення операційної системи та повний ремонт комп'ютера.

Якщо комп'ютер був заражений вірусом, шпигунським або шкідливим програмним забезпеченням, то знадобиться послуга видалення вірусів. Доцільно встановити на комп'ютер антивірусну програму, яка регулярно оновлюється через Інтернет мережу.

На комп'ютері багато електронних деталей, а також кілька рухомих частин. Багато таких деталей вимагають частого налаштування, навіть основна чистка вентиляторів, процесора та інших компонентів може допомогти уникнути багатьох поширених проблем з комп'ютером. Сервісний центр також пропонує налаштування програмного забезпечення різного рівня, яке допомагає покращити продуктивність комп'ютерів – вони перестануть поглинати надлишкову потужність, пам'ять та інші ресурси.

Мережеві послуги, як правило, потрібні клієнтам, які хочуть з'єднати кілька комп'ютерів разом, що допомагає їм отримувати доступ до файлів, які зберігаються на будь-якому мережевому комп'ютері чи сервері. Комп'ютери можуть бути мережевими через провід або бездротовий зв’язок, все залежить від конкретних потреб бізнесу та типу системи мереж, яка є економічною, ефективною та доступною в даній місцевості. Мережа комп'ютерів підвищує ефективність роботи та продуктивність, працівники можуть співпрацювати краще, коли вони мають доступ до всіх комп’ютерів, а також легко поділитися своєю роботою з колегами.

Резервне копіювання даних - це важлива частина будь-якого бізнесу, зараз всі підприємства зобов'язані захищати дані своїх клієнтів та покупців. Такого роду проблем можна уникнути за допомогою ефективного резервного копіювання даних. Це одноразова дія, але вона зазвичай потребує професійної допомоги. Після налаштування фізичної або хмарної системи резервного копіювання даних, вона буде працювати автоматично.

Втрата важливих даних може виявитися згубною. Можливо на комп’ютері були помилково видалені важливі файли, або внаслідок атаки вірусів та зловмисного програмного забезпечення. Щоб відновити втрачені, пошкоджені та пошкоджені дані, звертаються у професійний центр з обслуговування комп'ютерної техніки. Програми відновлення даних та інші методи використовуються для відновлення втраченої та пошкодженої інформації.

Залежно від типу послуг, пропонованих центром обслуговування комп'ютерів, він може надавати такі послуги, як установка операційної системи, її оновлення, установка та адміністрування сервера, віддалена підтримка мережі та очищення системи. Послуги комп'ютерного обладнання та програмного забезпечення доступні для всіх типів комп'ютерів, включаючи ПК, ноутбуки, планшети або смартфони[1].

1. РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

1.1. Аналіз вимог до програмної системи

1.1.1. Аналіз предметної області

Щоб надавати свої послуги в мережі Інтернет, компанія мусить мати веб-сайт. Він буде як веб-сайтом візитівкою, для отримання клієнтами контактних даних і представлення в якій сфері послуг працює дана компанія, так і працювати безпосередньо з діючими клієнтами. Наявність власного веб-сайту позитивно впливає на імідж компанії, клієнти ставляться до неї з більшою довірою. Також веб-сайт допомагає значно збільшити реалізацію послуг, будучи головним інструментом для вирішення різноманітних маркетингових завдань і значно понижує навантаження на офісних працівників, відповідаючи клієнтам на рутинні питання, такі як місцезнаходження, список послуг, контакти і т.д, тому веб-сайт є фактично необхідним атрибутом для конкурентоздатності сервісного центру з обслуговування комп’ютерної техніки.

Інформаційні веб-сайти - це веб-сайти, орієнтовані на вміст та дизайн, які є прекрасним способом обміну інформацією та ресурсами в Інтернеті, а також просування товарів та послуг, важливих дат та подій. Якісний інформаційний веб-сайт ознайомлює його відвідувачів з послугами, які надає компанія, залучає потенційних клієнтів та ініціює контакт між ними та адміністрацією сайту[2].

Створення сайту для сервісного центру з обслуговування комп’ютерної техніки дозволить по-перше, розширити клієнтську базу завдяки представленню організації в мережі Інтернет, а по-друге, автоматизувати прийом заявок на послуги. Сайт повинен ефективно вирішувати поставлені перед ним завдання: представляти в мережі Інтернет загальну інформацію про діяльність сервісного центру, його контактні дані і весь комплекс послуг,

які в ньому надаються, а також дозволяти зворотній зв’язок між користувачами та адміністрацією сайту.

1.1.2. Постановка задачі

Представлення точної і актуальної інформації, яку користувач буде отримувати при відвідуванні сайту є найважливішим завданням веб-сайту. Всі послуги, мають розміщуватися на одній сторінці, тому інша супутня інформація, така як прайслист, контактні дані та форма зворотного зв’язку, задля зручності сприйняття, має бути рознесена. Ця обставина зумовлює необхідність продумування простого та інтуїтивно зрозумілого веб-дизайну для навігації на сайті, що теоретично дозволить суттєво зменшити середній час, який витрачається користувачем під час його переходу між сторінками. Також, крім відображення списку послуг, на сторінці опціонально додати панелі розширення для кожного елементу списку, які будуть містити додаткову інформацію про ту чи іншу послугу.

Для того щоб усунути однотипні питання, пов’язані із замовленням послуг або діагностикою комп’ютерної техніки, які можуть виникнути у потенційних клієнтів під час відвідування ресурсу, мають бути передбачені різноманітні способи зв’язку з адміністрацією сайту. З цією метою, крім контактних номерів телефонів та електронної пошти сервісу, мають бути подані посилання на офіційні мессенджери.

Через відносну нескладність розробки інформаційних веб-сайтів, доцільно використовувати доволі нетрадиційний підхід до створення сайтів – за допомогою генератора статичних сайтів, який дозволяє розробити безсерверну архітектуру і адаптивний дизайн для кросплатформної роботи.

1.1.3. Пошук акторів та варіантів використання

Опис динамічної поведінки (її компонентів) в будь-який момент часу – є найважливішим аспектом для моделювання системи. Для цього годиться стандарт, який в основному застосовується, щоб створювати об'єктно-

орієнтовані, змістовні моделей документації для будь-якої програмної системи, присутньої в реальному світі – UML.

UML налічує п'ять діаграм, які дозволяють моделювати динамічну поведінку, одна з них - це діаграма варіантів використання або діаграма прецедентів. Вона фіксує функціональність та вимоги системи за допомогою акторів та варіантів використання. Прецеденти моделюють служби, завдання, функції, які потрібно виконати системі. Приклади використання відображають функціональні можливості високого рівня та те, як користувач буде обробляти систему, це основні концепції моделювання мови уніфікованого моделювання[3].

Діаграма варіантів використання складається з системи, поєднаних прецедентів та акторів, і зв’язує їх один з одним для візуалізації: що описується? (система), хто використовує систему? (актори) і чого хочуть досягти актори? (варіантів використання). Таким чином, прецеденти допомагають забезпечити розробку правильної системи шляхом відображення вимог з точки зору користувача. Діаграма зазвичай проста, і не відображає деталі елементів, з яких складається, а лише узагальнює деякі взаємозв'язки між варіантами використання, акторами та системами, а також не показує порядок виконання кроків для досягнення цілей кожного прецеденту.

Прецеденти використовуються для представлення функцій високого рівня і того, як користувач поводиться з системою. Варіант використання являє собою чітку функціональність системи, компонента, пакета або класу. Він позначається овальною формою з назвою, записаною всередині овальної форми.

Актор - це суб'єкт, який ініціює варіант використання поза межами прецеденту. Це може бути будь-який елемент, який може викликати взаємодію із варіантом використання. Один актор може бути пов’язаний із кількома випадками використання в системі.

Участь актора у варіанті використання позначається підключенням його до прецеденту суцільною лінією відношення. Актори можуть бути підключені до варіантів використання асоціаціями, що вказує на зв’язок актора та прецеденту при допомозі обміну повідомленнями[4].

Для веб-сайту з обслуговування комп’ютерної техніки, головним актором є відвідувач. Згідно з вимогами, це адаптивний інформаційний сайт, тобто його легко можна відкрити на будь-якій платформі. Така система не передбачає взаємодії між відвідувачами, і тому закономірно припустити, що вона не має важких архітектурних рішень. Цільова аудиторія програмної системи – це широке коло потенційних клієнтів сервісу, що диктує необхідність інтуїтивно зрозумілого та неперевантаженого деталями дизайну інтерфейсу.

Крім відвідувача в системі буде присутній ще один актант – адміністратор, який додає, видаляє та редагує інформацію на сайті, а також приймає обробляє інформацію надану відвідувачами через форму зворотного зв’язку.

1.1.4. Опис варіантів використання

В програмній системі є два актори – Відвідувач і Адміністратор (рис. 1.1).

Рисунок 1.1 – Актори системи

Короткий опис акторів представлений в таблиці. 1.1.

Табл. 1.1 Виявлення акторів

Актор	Короткий опис
Відвідувач	Переглядає веб-сайт, та може взаємодіяти з окремими його елементами, такими як вкладки з інформацією про послуги, форми зворотного зв’язку та секція для коментарів.
Адміністратор	Робота із оформленням замовленої послуги, адміністрування контенту на веб-сайті, зміна його стану та модерування коментарів.

Після аналізу роботи двох акторів в системі, можна виявити наступний набір прецедентів, який наведено в таблиці 1.2.

Табл. 1.2 Виявлення акторів

Основний актор	Найменування	Формулювання
Відвідувач	Переглянути список послуг	Дозволяє клієнтові переглянути список наявних на сайті послуг.
Відвідувач	Переглянути категорії послуг	Клієнт має змогу переглянути список категорій, на які поділені всі послуги, що реалізуються на сайті.
Відвідувач	Переглянути інформацію про окрему послугу	Надає можливість клієнтові переглянути інформацію про окрему послугу та її ціну.

Відвідувач	Відправити запит на консультацію	Дозволяє відвідувачу заповнити скористатися формою зворотного зв’язку або месенджером для контакту з адміністрацією сайту.

Продовження табл. 1.2

Відвідувач	Переглянути коментарі	Надає можливість відвідувачеві переглянути на сайті секцію з відгуками інших відвідувачів про роботу сервісного центру
Відвідувач	Залишити коментар	Дозволяє відвідувачеві, за умови авторизації у соціальній мережі, залишити свій відгук
Адміністратор	Прийняти запит про консультацію	Адміністратор має змогу обробити інформацію, яку надав відвідувач
Адміністратор	Переглянути коментарі	Надає можливість адміністратору сайту модерувати секцію коментарів на предмет невідповідного вмісту
Адміністратор	Залишити коментар	Дозволяє адміністратору відповідати на відгуки, залишені відвідувачами
Адміністратор	Керування контентом веб-сторінок	Адміністратор може створювати, видаляти і змінювати вкладки з описом послуг, додавати рекламні слайдери та редагувати прайслист на послуги
Адміністратор	Зміна стану сайту	Адміністратор може обмежувати доступ до інформації на веб-сайті

Всі варіанти використання приведені на рисунку 1.2.

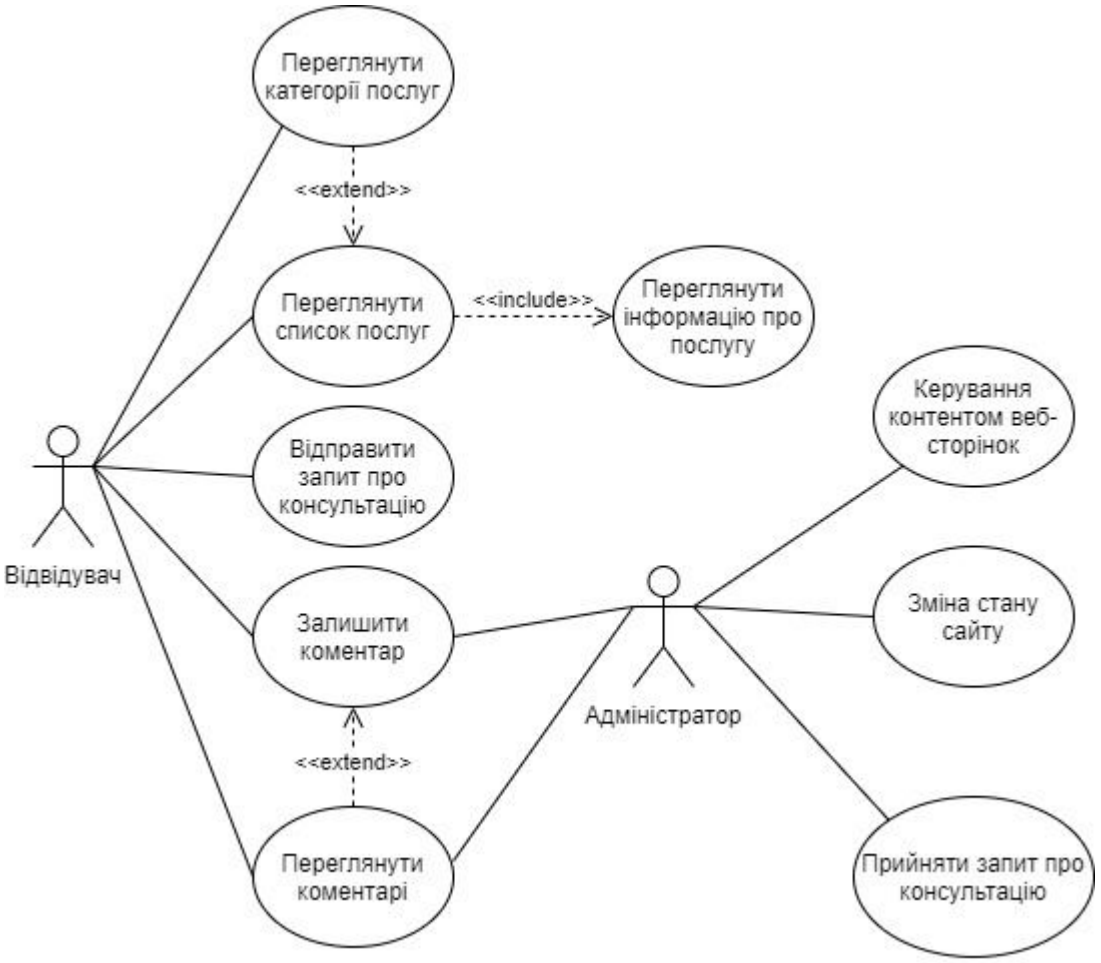

Рисунок 1.2 – Діаграма варіантів використання системи

1.2. Проектування програмної системи

1.2.1. Вибір процесу розробки

Побудова програмного забезпечення - це складне і складне завдання, кожна команда та організація підходить до проблеми по-різному, але дотримуючись стандартизованих методологій.

Проте чотири пункти є частиною кожного процесу розробки програмного забезпечення:

- специфікація або вимоги: саме тут команда та інші зацікавлені сторони проекту визначають основні функції програмного забезпечення, яке вони планують створити, а також обмеження, які вони ставлять перед процесом;
- розробка та реалізація: команда розробляє та впроваджує програмне забезпечення відповідно до заздалегідь визначеної специфікації;
- перевірка та верифікація: саме тут команда переконується, що розроблене програмне забезпечення відповідає специфікаціям та потребам клієнта, які визначені у бізнес-вимогах;
- технічне обслуговування та правки: команда піклується про технічне обслуговування програмного забезпечення, модифікуючи та змінюючи його так, щоб воно відповідало мінливим вимогам споживачів та ринку[5].

Для розробки веб-сайту була вибрана модель швидкої розробки застосунків RAD (Rapid Application Development) - це гнучкий підхід до розробки програмного забезпечення, який зосереджується більше на поточних програмних проектах та відгуках користувачів, і менше на дотриманні суворого плану. Як такий, він заохочує швидке складання прототипів замість дорогого планування. Хоча часто його часто плутають з конкретною моделлю, RAD - це ідея, що можна отримати користь, обробляючи програмні проекти, як глина, а не сталь, як це розглядають традиційні практики розробки.

Хоча точні практики та інструменти відрізняються від конкретних методів RAD, їх основні фази залишаються однаковими:

Крок №1. Визначення вимог

Замість того, щоб витрачати місяці на розробку технічних характеристик з користувачами, RAD починається з визначення вільного набору вимог. Ключовим принципом швидкої розробки застосунків є дозвіл на зміну вимог у будь-якій точці циклу. В основному розробники збирають "суть продукту".

Крок №2. Прототип

У цій швидкій фазі розробки додатків метою розробників є створити щось, що вони можуть продемонструвати клієнту. Це може бути прототип, який задовольняє всі або лише частину вимог (як на ранньому етапі прототипування). Більшість підходів RAD мають завершальний етап, коли розробники закривають технічні «хвости», накопичені на ранніх прототипах.

Крок №3. Накопичення зворотного зв’язку

Коли останній прототип підготований, розробники RAD представляють свою роботу клієнту або кінцевим споживачам. Вони збирають відгуки про все, від інтерфейсу до функціональності - саме тут вимоги до продукту можуть піддаватися ретельному переосмисленню. Клієнти можуть передумати або виявити, що щось, що здавалося правильним на папері, на практиці не має сенсу. Маючи зворотний зв'язок, розробники повертаються до певної форми кроку №2: вони продовжують розробляти прототип. Якщо відгуки суворо позитивні, а клієнт задоволений прототипом, розробники можуть перейти до кроку №4.

Крок №4. Доопрацювання продукту

Під час цього етапу розробники можуть оптимізувати або навіть перепрофілювати реалізацію продукту для покращення стабільності, підтримуваності та ремонтопридатності, і виконувати будь-які інші завдання з технічного обслуговування, перш ніж впевнено передавати виріб.

Модель швидкої розробки застосунків зображено на рисунку 1.3.

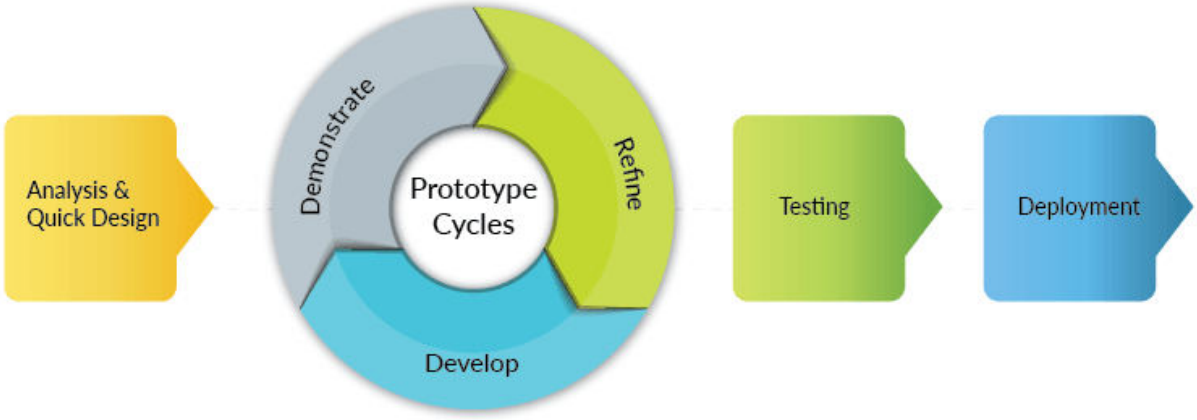

Рисунок 1.3 – Модель швидкої розробки застосунків

До переваг моделі швидкої розробки застосунків можна віднести:

Швидкість. При традиційному водоспадному підході розробники навряд чи поїхали у відпустку після релізу товару. Клієнти неухильно вимагають змін, починаючи від інтерфейсу до функціональності після першого релізу. Завдяки RAD проекти, швидше за все, завершаться вчасно.

Вартість. У RAD розробники будують точно таку систему, яку від них потребує клієнт, і нічого більше. У водоспадному підході IT ризикує створити та розробити складні набори функцій, які клієнт може вибрати усунути з кінцевого продукту. Час, витрачений на побудову таких функцій, ніколи не можна повернути, а це означає, що витрачений на них бюджет втрачається. RAD знижує цей ризик, а отже, зменшує вартість.

Задоволеність розробників. У традиційному водоспадному підході розробники працюють у «вакуумі», позбавлені відгуків та схвалення продукту, який добре зроблений. І коли вони нарешті отримують можливість представити свою роботу клієнту, то він може не прийняти її. Незалежно від того, наскільки пишаються розробники своєю роботою, якщо клієнт незадоволений, то вони не отримують такого фідбеку, який їм так потрібен. У RAD клієнт знаходиться на кожному кроці, і розробник має можливість часто презентувати свою роботу. Це дає їм впевненість, що коли кінцевий продукт буде готовий, їхня робота отримає схвалення.

До недоліків моделі швидкої розробки застосунків можна віднести:

Масштаб. Згуртована команда розробників, дизайнерів та менеджерів може легко використовувати практики RAD, оскільки вони мають прямий доступ один до одного, але коли проект виходить за межі однієї команди або вимагає міжкомандної комунікації, цикл розвитку незмінно сповільнює і заглушує розвиток проекту. Простіше кажучи, важко утримати велику групу людей сконцентрованою, коли ваша вимоги постійно змінюються.

Зобов'язаність. У водному підході клієнт проводив більшу частину свого часу за межами команди розробників після завершення специфікацій. Це дозволило клієнтам зосередитися на своїх основних завданнях, а розробникам – на розробці. У RAD часто цикл прототипів вимагає від розробників та клієнтів брати участь у частих зустрічах, які спочатку можуть здатися непотрібними[6].

1.2.2. Вибір технологій та інструментів для розробки

Система має складатися з декількох ключових компонент, що взаємодіють між собою. Використання мови програмування JavaScript і фреймворків написаних за її допомогою, не лише для реалізації інтерактивності веб-сторінок, але й для генерування статичних сторінок (JAMstack), які замінюють собою повноцінний сервер дозволяє полегшити розробку і подальшу підтримку програмного продукту, зокрема за допомогою платформи з відкритим кодом для виконання високопродуктивних мережевих застосунків Node.js, яка побудована на рушії Chrome V8.

Node.js - це потужний фреймворк JavaScript, розроблений на рушії V8 JavaScript Chrome. Він допомагає при компілюванні JavaScript безпосередньо в натівний машинний код, займає небагато місця і широко використовується на ринку для розробки серверних веб-додатків, а також для широкомасштабної розробки сайтів і потокових сервісів, односторінкових програм та інших веб-додатків. Node.js використовує керовану подіями

модель, яка не блокує введення / виведення, що робить її правильним вибором для інтенсивних додатків у реальному часі.

Як і будь-які інші мови програмування, Node.js використовує різні пакети та модулі. Це не що інше, як бібліотеки, що містять функції, і імпортуються з npm (node package manager, диспетчер пакетів вузлів) в код і використовуються в програмах.

Як правило, такі серверні технології, як PHP, ASP.NET, Ruby & Java-сервери, слідують за багатопотоковою моделлю. У цьому традиційному архітектурному підході кожен запит клієнта створює новий потік або процес. Щоб уникнути цього, Node.js використовує архітектуру моделей циклу подій з одним потоком. Це означає, що всі клієнтські запити на Node.js виконуються в цьому самому потоці. Але ця архітектура не просто однопоточна, вона - керована подіями, це допомагає Node.js одночасно обробляти декілька клієнтів[7].

Хоча термін JAMstack пов'язаний з JavaScript та сучасною архітектурою веб-розробки, він є екосистемою, самостійним набором інструментів. Десятиліття тому, більшість веб-сайтів в Інтернеті були статичними, а отже - швидкими, простими, дешевими і процес їхнього розгортання був легким. JAMstack – це спосіб повернути цю практику, оновивши її сучасними рішеннями веб-дизайну. Є три основи (рис. 1.4), які більше схожі на умовності, а не набір конкретних інструментів, які доведеться використовувати, щоб вписатись у визначення терміну JAMstack:

- JavaScript;
- Прикладний програмний інтерфейс (Application Programming Interface, API);
- Розмітка (Markup).

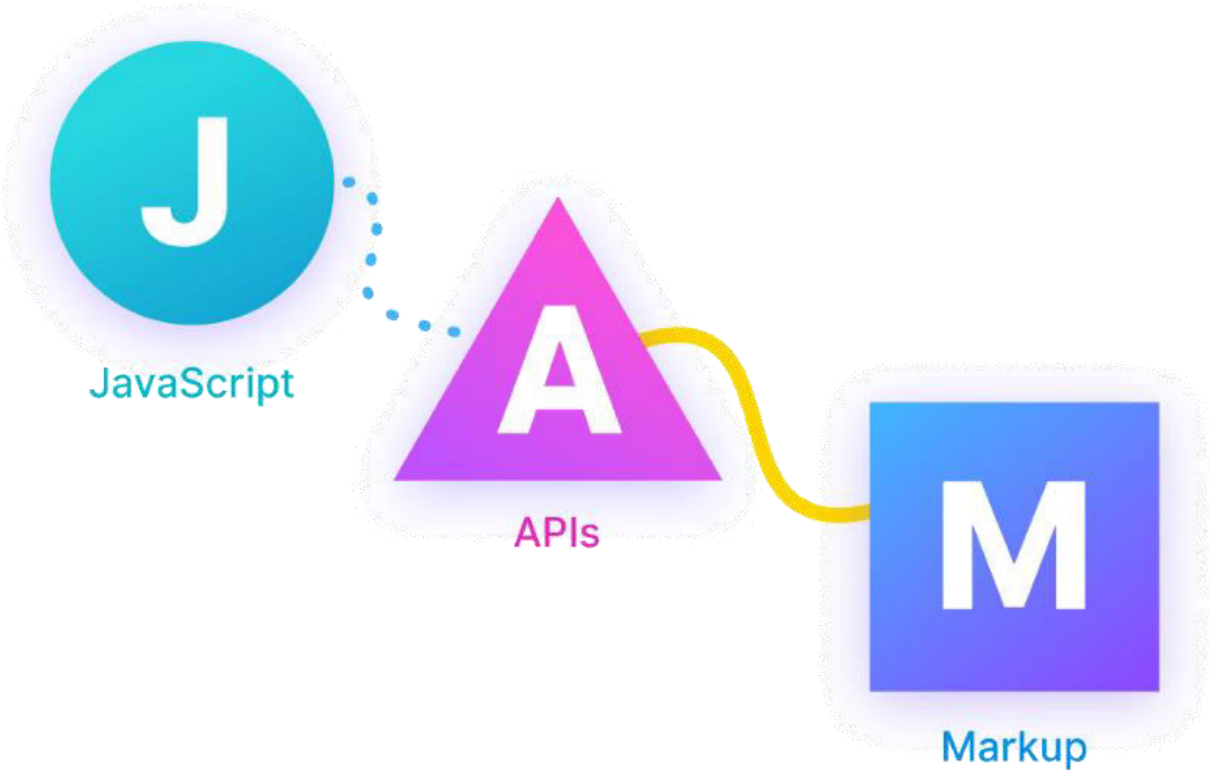

Рисунок 1.4 – Складові JAMstack

JS – це основа всього стеку. Загальновизнаний факт, що JavaScript є однією з найпопулярніших мов програмування в наші дні з надзвичайно яскравим співтовариством. Цикли запитів та відповідей реалізовані на стороні клієнта якраз завдяки цій технології. Можна використовувати чистий JavaScript або будь-який інший фреймворк чи бібліотеку, доступний на ринку, наприклад React або Vue. Це інтерпретована мова, що означає, відсутність потреби у компіляторі для інтерпретації свого коду, як в C або C ++. JavaScript-код працює безпосередньо у веб-браузері з HTML та CSS для створення веб-додатків або веб-сторінок. JavaScript підтримується більшістю сучасних веб-браузерів, таких як Google Chrome, Firefox, Safari, Microsoft Edge, Opera тощо, це стосується також більшості мобільних браузерів для Android та iPhone[8].

JavaScript класифікується як прототипна скриптова мова програмування з динамічною типізацією, але вона також частково підтримує

імперативну та частково функціональну парадигми програмування і деякі архітектурні властивості, наприклад: слабка та динамічна типізація, автоматичне керування пам'яттю, прототипне успадкування, функції як об'єкти першого класу.

Скрипт повинен бути включений у HTML-документ або посилатися на нього, щоб код був інтерпретований браузером. Це означає, що веб-сторінка не повинна бути статичним HTML, але вона може включати програми, які взаємодіють із користувачем, контролюють браузер та динамічно створюють вміст HTML. Фронтенд механізм JavaScript надає багато переваг перед традиційними бекенд CGI скриптами. Наприклад, можна використовувати JavaScript, щоб перевірити, чи користувач ввів дійсну адресу електронної пошти у поле форми. Код виконується, коли користувач подає форму, і лише якщо всі записи дійсні, вони будуть передані на веб-сервер. JavaScript може використовуватися для відслідковування подій, явно або неявно ініційованих користувачем, таких як натискання кнопок та навігація за посиланнями.

Переваги використання JavaScript:

- менше взаємодії з сервером: можна перевіряти введення даних користувача перед тим, як відправити сторінку на сервер. Це економить його трафік, що означає менше навантаження;
- негайний відгук відвідувачів: їм не потрібно чекати перезавантаження сторінки, щоб побачити, чи не забули вони щось ввести;
- підвищена інтерактивність: можна створювати інтерфейси, які реагують, коли користувач наводить на них мишу або активує їх за допомогою клавіатури;

- насичені інтерфейси: можна використовувати JavaScript для включення таких елементів, як компоненти випадання та повзунки, щоб надати відвідувачам свого сайту багатий інтерфейс.

Обмеження JavaScript:

- підтримка браузера: JavaScript іноді по-різному інтерпретується різними браузерами, це ускладнює написання крос-браузерного коду;
- клієнтський JavaScript не дозволяє читати чи записувати файли. Це робиться з міркувань безпеки;
- JavaScript не можна використовувати для мережевих додатків, оскільки такої підтримки немає;
- JavaScript не має функцій багатопотокової чи багатопроцесорної роботи[9].

Наступною основою JAMstack є API. Завдяки цьому можна використовувати функції бекенду, не маючи на своєму сервері бази даних або рушія. Бекенд все ще присутній, але розгортається лише статичний веб-сайт. Можна використовувати будь-який потрібний API - публічний, приватний, або вибрати сторонню програмну систему. До прикладних програмних інтерфейсів можна віднести інструменти аутентифікації (Auth0, Firebase), сервіси оптимізації та зберігання медіа (Cloudinary), для миттєвого розгортання статичних веб-сайтів (Netlify), повністю бекенд сервіси (Backendless), або системи керування вмістом (Contentful, Netlify CMS). Обмеження функціональності сторонніх API для підвищення продуктивності та безпеки в JAMstack є однією з його ключових особливостей[10].

Сьогодні існують сотні статичних хост компаній, але для розробки програмної системи була вибрана Netlify. Вона дозволяє розробникам надзвичайно легко розміщувати веб-сайти, які масштабуються та захищаються. Netlify спрощує розробникам процес розгортання та розміщення веб-сайту. Замість них здійснюється вся робота, на яку вони,

ймовірно, не хочуть витрачати занадто багато часу чи зусиль. Крім цього, Netlify, також дозволяє отримати переваги і для редакторів.

Netlify - це компанія, що займається інфраструктурою та технологіями веб-хостингу, що базується в Сан-Франциско. Цікаво те, що JAMstack спочатку був реалізований співзасновником Netlify. Netlify пропонує веб-хостинг наступного покоління та автоматизацію, які є дуже доступними. Вони також пропонують інфраструктуру веб-хостингу для веб-сайтів JAMstack.

Netlify працює, підключившись до сховища GitHub, щоб витягнути свій вихідний код, тоді він, як правило, запускає процес збирання, щоб попередньо зібрати всі сторінки в статичний HTML. Потім отримані HTML, CSS та JS розгортаються та розподіляються між великою кількостю мереж доставки вмісту. Коли відвідувач намагається отримати доступ до веб-сайту, він автоматично вибирає найближчий центр обробки даних і подає статичні файли.

Переваги Netlify:

- Netlify дешевший і дозволяє отримати швидший сайт. Створення та розміщення статичного веб-сайту допоможе заощадити гроші в довгостроковій перспективі. Крім того, за допомогою Netlify Edge веб-сайт та додатки будуть надзвичайно швидкими, використовуючи глобальний дистрибутив та автоматичне попереднє представлення;
- Netlify Build дозволяє розробникам створювати будь-яку інтеграцію. Кожен раз, коли розробник хоче змінити вміст або додати нову функціональність на свій сайт, Build дозволяє створювати необмежені гілки веб-сайту. Це означає, що якщо веб-сайт розміщено за допомогою Netlify, дасть змогу прокручувати унікальну URL-адресу та дає кілька способів для ваших розробників перевірити будь-які зміни, які вони внесли в різних видах перегляду.

Кожна зміна вмісту дозволяє легко їх переглядати в різних середовищах;

- Простіше запустити сайт за допомогою Netlify, бо він має вбудоване управління DNS та сертифікати SSL, які є безкоштовними і їхній термін дії не закінчується. Це означає меншу складність у запуску веб-сайту, і він розміщений в одному місці[11].

Contentful - це контент-інфраструктура, ця платформа дозволяє створювати, керувати та поширювати вміст на будь-якій іншій платформі. На відміну від CMS, вона надає повну свободу для створення власної моделі вмісту, щоб можна було вирішити, яким контентом керувати. Contenful надає RESTful API, щоб для відправки вмісту на декілька каналів, таких як веб-сайти, мобільні додатки (iOS, Android та Windows Phone) або будь-яку іншу платформу, наприклад Google Glass. Завдяки гармонійному користувальницькому інтерфейсу, Contentful - це ефективний інструмент для створення та керування контентом в Інтернеті, самостійно або в команді. учасникам команди спеціальні ролі та дозволи, Можна давати спеціальні ролі, перевірки залежно від виду вмісту, який потрібно вставляти та додавати медіа, такі як зображення, документи, звуки чи відео.

Проблема систем управління вмістом (CMS) полягає в тому, що вони створюються навколо одного типу контенту, під який оптимізовані для відображення, як правило, це веб-сайт. Більшість CMS стали простими інструментами веб-публікації замість загальних систем управління вмістом. Вони пропонують існуючі моделі вмісту, які звужують можливості та дуже поєднані з одним шаром відображення.

В свою чергу, Contenful має триетапний процес. Перш за все, визначається модель контенту, яка не залежить від будь-якого шару відображення, який установлює, яким типом контенту треба керувати. На другому кроці внутрішні чи зовнішні редактори можуть керувати усім вмістом в простому у використанні та інтерактивному інтерфейсі

редагування. І останнє, але не менш важливе, вміст подається незалежно від відображення.

Щоб створити веб-сайт, потрібно розробляти його самостійно і завантажувати вміст з API. Contentful - це платформа, на якій можна оновлювати вміст свого веб-сайту, мобільний додаток або будь-яку іншу платформу, яка відображає вміст. Вона заощаджує час та клопоти, створюючи власний сервіс для керування вмістом та надає багато інструментів, які полегшують фактичне створення веб-сайту чи програми. Контент можна повторно використовувати на будь-які платформі

Contentful – це надійна мережа доправлення і розповсюдження контенту реалізована так, що програми завжди можуть надсилати API-дзвінок і миттєво повертати вміст назад. Платформа тісно пов'язана з JSON. Це означає, вона надає надійний і знайомий формат для будь-якого типу вмісту, який тільки можна придумати. Крім цього вона ефективно вирішує проблему наявності зображень у контенті за допомогою надійного API зображень. API зображень може не тільки доставляти зображення, він також може маніпулювати, обрізати та змінювати формати зображень на ходу та багато іншого.

Також можна сказати, що Contentful - це контентна інфраструктура, або в цьому випадку інфраструктура контенту для програми - JAMstack. І на відміну від традиційних CMS, таких як Drupal та WordPress, програмна система була побудована з архітектурою RESTful API з самого початку. Варто пам’ятати, що у Contentful є SDK-програми для всіх основних мов програмування, що значно спрощує побудову проектів[12].

Розмітка - презентаційний шар веб-сайту. Зазвичай екосистема JAMstack - це статичний генератор сайту, де шаблонна розмітка заздалегідь будується на час збирання. Можна написати свій власний HTML та CSS-код або використати такі фреймворки, як Hugo, Jekyll або Gatsby, які значно оптимізують час розробки шаблонів.

Gatsby - генератор статичних сайтів на основі React, що працює на основі GraphQL. Що це означає? Що ж, фреймворк поєднує в собі найкращі частини React, webpack, react-router, GraphQL та інші інструменти для фроненд розробки, в один дуже приємний досвід для розробників. Але це не просто "генератор статичних веб-сайтів". Цей термін існує деякий час, але Gatsby набагато більше нагадує сучасний фронтенд фреймворк, ніж старий генератор старих сайтів.

Він використовує потужну попередню конфігурацію для створення веб-сайту, який має лише статичні файли для неймовірно швидких завантажень сторінок, service worker, розбиття коду, візуалізацію на стороні сервера, інтелектуальне завантаження зображень, оптимізацію ресурсів та попереднє завантаження даних. І все це «з коробки».

Програміст пише та розробляє свій сайт, який Gatsby перетворює в директорію з єдиним HTML-файлом та статичними активами. Ця папка може без проблем бути завантажена у будь-який хостинг-провайдер. Є три речі (рис. 1.5), які роблять Gatsby дуже особливим порівняно з цими його прямими конкурентами в особі генератора статичних сайтів Jekyll, написаного з допомогою мови програмування Ruby та create-react-app.

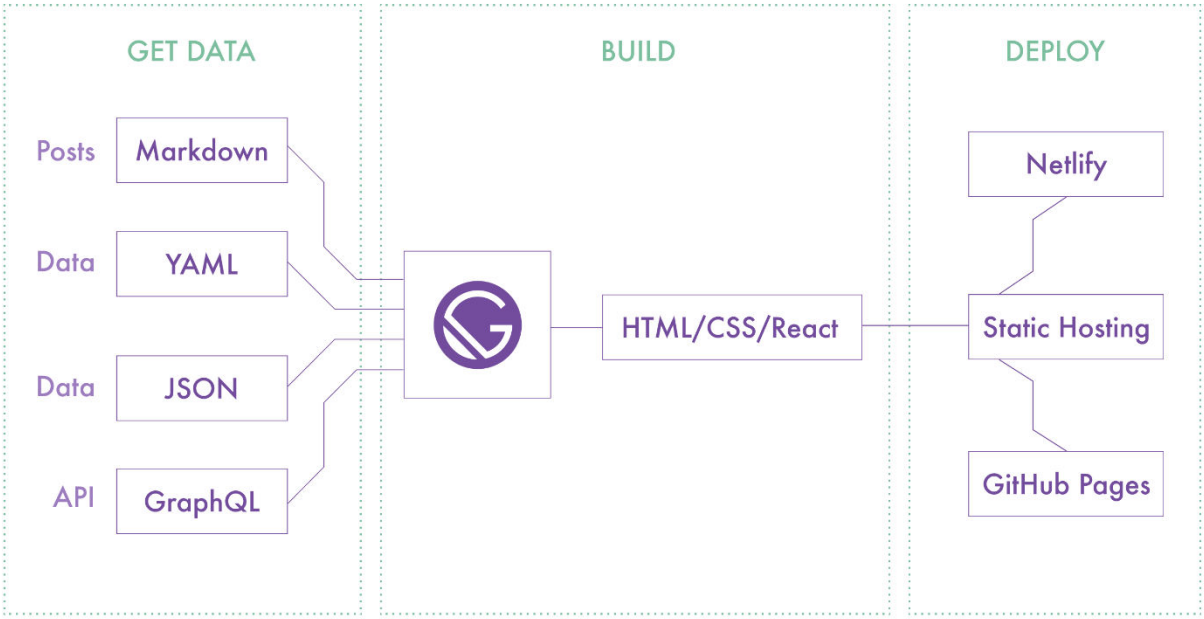

Рисунок 1.5 – Пояснювальна схема GatsbyJS

По-перше, це те, як Gatsby використовує GraphQL для побудови свого рівня даних. Gatsby створений для збору даних з будь-якого місця: Markdown, JSON, будь-яка CMS, сторонні API-програми, словом, будь-що. І під час збирання він створює внутрішній сервер GraphQL з усіх цих даних. Тож у React компонентах всі дані запитуються під час збирання з того самого місця, таким же чином через GraphQL.

Друга причина, чому фреймворк такий особливий - багатство екосистеми Gatsby. Це доволі молодий фреймворк, але він уже може похвалитися чудовою документацією та низкою стартерів, які допоможуть швидко створити сайт. Цей збір даних GraphQL, може здатися залякаючим, але завдяки добре задокументованим плагінам джерел даних Gatsby, він може бути таким же простим, як і декілька рядків коду у конфігураційному файлі.

І нарешті, це її відданість продуктивності. Кожен аспект продуктивності та доступності є важливим моментом, і можна відчути це в кінцевому продукті.

Однією з найбільших перешкод для новачків, які намагаються освоїти Gatsby є відсутність часу та енергії для вивчення додаткових інструментів розробки потрібних для даного фреймворку, таких як React та GraphQL[13].

React JS - це бібліотека JavaScript, яка використовується в веб-розробці для створення інтерактивних елементів на веб-сайтах. Бувають випадки, коли для виконання повторюваних функцій, потрібен JavaScript - ефекти анімації або функції автозаповнення рядка пошуку. Повторне кодування цих функцій кожного разу, коли вони виникають, стає ситуацією "винахід колеса". Вирішити цю проблему дозволяють бібліотеки JavaScript.

Бібліотеки JavaScript - це колекція заздалегідь написаного коду JavaScript, який можна використовувати для поширених завдань JS, дозволяючи обійти час від часу інтенсивний (і непотрібний) процес кодування. Окрім надання коду бібліотеки React для багаторазового використання (економія часу на розробку та зменшення шансів на помилки

під час кодування), React має дві основні функції, які є привабливими для розробників JavaScript:

- JSX;
- Віртуальний DOM.

В основі будь-якого базового веб-сайту лежать HTML-документи. Веб-браузери читають ці документи та відображають їх на комп'ютері, планшеті чи телефоні у вигляді веб-сторінок. Під час цього процесу браузери створюють сутність, що називається Document Object Model (DOM), тобто, дерево уявлення про те, як влаштована веб-сторінка. Потім розробники можуть додавати динамічний контент у свої проекти, змінюючи DOM такими мовами, як JavaScript.

JSX (скорочення JavaScript eXtension) - розширення React, яке дозволяє веб-розробникам змінювати свій DOM за допомогою простого коду в стилі HTML. І оскільки підтримка React поширюється на всі сучасні веб-браузери, то JSX сумісний з ними всіма. Це не лише питання зручності, хоча використання JSX для оновлення DOM призводить до значних покращень продуктивності сайту та ефективності розробки. Вся справа в наступній функції React - віртуальному DOM.

Якщо не застосовується React (та JSX), то веб-сайт використовує HTML, щоб оновити DOM. Це добре працює для простих, статичних веб-сайтів, але для динамічних веб-сайтів, які передбачають важку взаємодію з користувачем, це може стати проблемою, оскільки весь DOM потребує перезавантаження кожного разу, коли користувач натискає функцію, яка вимагає оновлення сторінки.

Однак якщо розробник застосовує JSX для маніпулювання та оновлення DOM, React створює щось, що називається віртуальний DOM - це копія DOM сайту. React використовує цю копію, щоб побачити, які частини фактичного DOM потрібно змінити, коли подія відбувається, наприклад, користувач натискає кнопку. Таке селективне оновлення вимагає менше обчислювальної потужності та меншого часу завантаження[14].

GraphQL – це виразна мова запитів даних, яку ви можна уявити як графік пов'язаних об'єктів та їх полів.

GraphQL дозволяє запитувати дані та отримувати у відповідь саме ту, інформацію, яка потрібна, нічого більше та нічого менше. Тобто, потрібними є лише три поля ресурсу, то API надсилається запит GraphQL із зазначенням цих полів, а результат - документ JSON, який містить лише три поля, які були запитані. Потрібні матеріали через мережу не передаються до потрапляння у фронтенд, а до цього моменту ігноруються.

За допомогою GraphQL можна запитувати кілька пов'язаних ресурсів в одному запиті. Іншими словами, це дає змогу проводити запит за взаємозв’язками і отримувати всі необхідні дані одним махом. GraphQL також надає можливість запитувати пов’язані об’єкти та отримувати відповідь, що має таку ж форму, як і сам запит. Це усуває явище численних запитів, які вимагають перевищення або недоотримання даних. Крім запитів, GraphQL також підтримує мутації для зміни даних та підписок на основі подій для доставки живих каналів даних[15].

З прекрасною документацією та прикладами роботи з цими інструментами, можна впевнено сказати, що якщо розробник знає лише один із них або не знає жодного, то Gatsby насправді є прекрасною точкою входу. Сама структура фреймворку спонукає «мислити в стилі React та GraphQL». Отже, React і GraphQL мають вирішальне значення для Gatsby, але якщо розробник їх не знає, то Gatsby - чудовий інструмент для їх вивчення, а не привід цього не робити.

Gatsby також включає в себе ідею стартерів. Стартери - це щось на кшталт boilerplate та шаблонів для Gatsby. Якщо відомо, що дані надходитимуть з певного джерела, або сайт буде функціонувати певним чином, то, мабуть, є стартер із попередньо налаштованими джерелами даних, а також буде налаштовано стиль та структуру компонентів. Це робить початок роботи та запуск дуже швидким та простим. Якщо придатного стартера не знайшлося, то можна використовувати стартер за замовчуванням

і знайти свій плагін джерела даних та дотримуватися його інструкцій із встановлення.

Чому можуть знадобитися статичні веб-сайти в епоху сучасних та інтерактивних веб-сайтів з багатьма цікавими функціями? Відповідь очевидна: статичні веб-сайти чудові, якщо просто потрібен веб-сайт, який не є складним інтернет-додатком. JAMstack використовує відомі технології, але те, як при цьому створюються веб-сайти, можна назвати нетрадиційним підходом, тому що він дещо відрізняється від того, що було загальноприйнятим роками.

За допомогою JAMstack розробник спершу зосереджується на презентаційному шарі, а потім просто заповнює веб-сайт даними. Не потрібно створювати бекенд, оскільки використовуються сторонні інтерфейси API, і створення інтерфейсу відбувається швидше, оскільки вміст генерується замість того, щоб будуватися з нуля.

Найбільша вигода для розробників, які використовують JAMstack, - це заощаджений час на кожному кроці процесу розробки. JAMstack забезпечує чудовий досвід для розробників, оскільки генерувати код набагато швидше, ніж писати його з нуля. Крім того, процес розгортання простіший, оскільки не потрібно думати про базу даних та інші допоміжні речі, просто розгортається статичний веб-сайт, на якому більша частина вмісту зазвичай вже присутня.

На відміну від традиційного робочого процесу, ми розробка починається з генерування шаблонів та компонування з фреймоворком Gatsby. Наступним кроком є заповнення веб-сайту даними за допомогою зовнішнього API, наданого за допомогою CMS Contentful. Після того, веб-сайт буде готовий, він за лічені секунди безкоштовно розгортаєтья при використанні Netlify[16].

На рисунку 1.6 показано основні відмінності JAMstack від традиційного підходу до написання веб-сайтів.

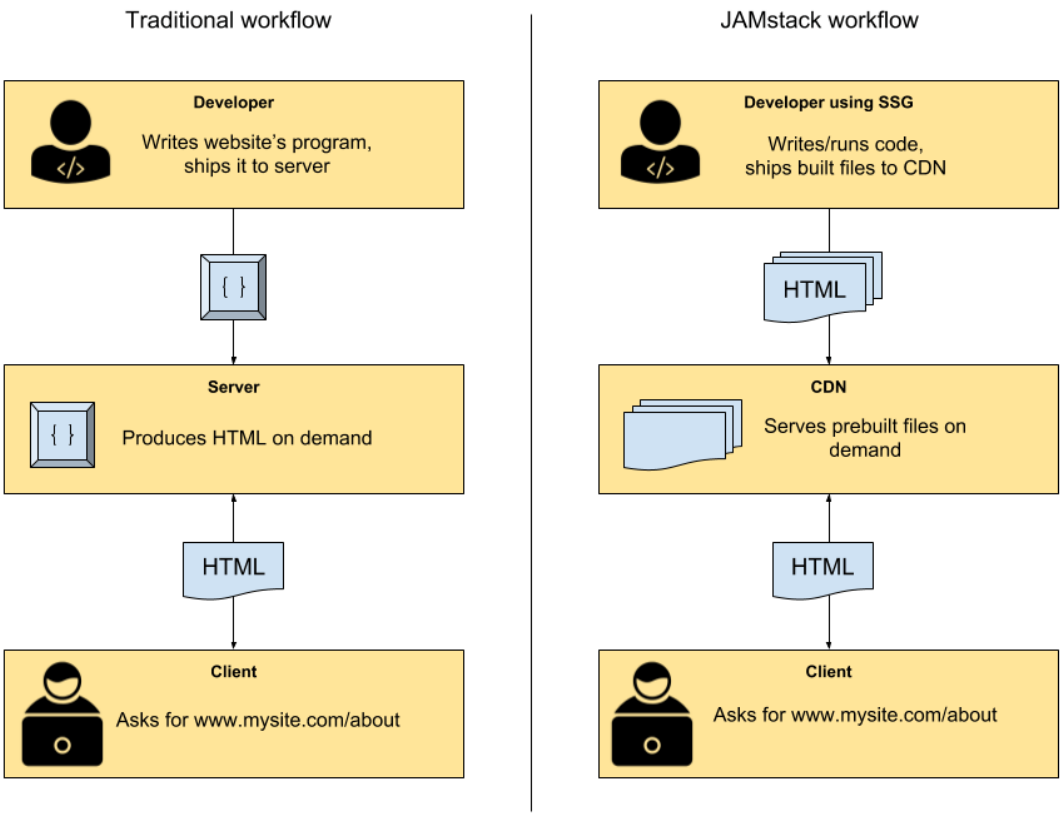

Рисунок 1.6 – Порівняння традиційного та JAMstack підходу до написання веб-сайтів

Як середовище розробки був вибраний Microsoft Visual Studio Code (VS Code). Visual Studio Code - це легкий, але потужний редактор вихідного коду, який оснащений вбудованою підтримкою JavaScript, TypeScript і Node.js, та має багату екосистему розширень для інших мов (таких як C ++, C #, Java, Python, PHP, Go) та режимів виконання (таких як .NET і Unity).

VS Code може виявити, якщо який-небудь фрагмент коду залишився незавершеним. Також загальні синтаксиси змінних та оголошення змінних додаються автоматично. Наприклад, якщо в програмі використовується певна змінна, а користувач забув її оголосити, Intelli-sense зробить це замість нього.

Середовище розробки підтримує кілька мов програмування, що усуває потребу розроників в іншому редакторі коду для різних мов, бо VS Code має вбудовану багатомовну підтримку. Це також означає, редактор легко виявляє, якщо є якась несправність або посилання між мовами. Підтримує всі

мови програмування, але, якщо користувач / розробник хоче використовувати мову програмування, яка не підтримується, він може завантажити розширення та використовувати його, крім того розширення не уповільнює редактор.

Ще одна особливість - це зручність використання Visual Studio Code. Файл розташований ієрархічно і має звичні програмні засоби, такі як панель інструментів, панель стану та бічну панель. Також є плаваюче вікно провідника Windows, яке можна зафіксувати в одному місці відповідно до зручності, яке складається з структури файлів. Ці файли (файли кодів, папки зображень, тощо) можна відкрити або перейменувати у цьому ж вікні, і зміни автоматично відобразяться у сховищі[17].

З постійно зростаючим попитом на написання коду, важливим є безпечне та своєчасне його зберігання.

Git - розподілена система контролю версій, яка дає можливість розробникам відстежувати зміни в файлах і працювати спільно з іншими розробниками. Git відомий своєю швидкістю, простим дизайном, підтримкою нелінійної розробки, повної децентралізацією і можливістю ефективно працювати з великими проектами[18].

Git відокремлений від інших систем контролю версій через підхід до роботи з даними. Більшість інших систем зберігають інформацію у вигляді списку змін в файлах. Замість цього, підхід Git до зберігання даних більше схожий на набір знімків мініатюрної файлової системи. Кожен раз, коли ви зберігаєте стан свого проекту в Git, система запам'ятовує, як виглядає кожен файл в цей момент, і зберігає посилання на цей знімок.

Visual Studio Code пов'язаний з Git або може бути з'єднаний з будь-яким іншим сховищем для витягування або збереження екземплярів.

Переваги Git:

- безкоштовний і open-source. Це означає, що його можна безкоштовно завантажити і вносити будь-які зміни у вихідний код;

- невеликий і швидкий. Він виконує всі операції локально, що збільшує його швидкість. Крім того, Git локально зберігає весь репозиторій в невеликий файл без втрати якості даних;
- резервне копіювання. Git ефективний в зберіганні бекапів, тому відомо мало випадків, коли хтось втрачав дані при використанні Git;
- просте розгалуження. В інших систем контролю версій створення гілок – втомливе і трудомістке завдання, так як весь код копіюється в нову гілку. У Git управління гілками реалізовано набагато простіше і ефективніше.

GitHub - сервіс онлайн-хостингу репозиторіїв, що володіє всіма функціями розподіленого контролю версій і функціональністю управління вихідним кодом - все, що підтримує Git і навіть більше. Зазвичай він використовується разом з Git і дає розробникам можливість зберігати їх код онлайн, а потім взаємодіяти з іншими розробниками в різних проектах.

GitHub Pages існують настільки ж довго, як і сам GitHub і надають сервіс хостингу для статичних веб-сторінок для зареєстрованих користувачів. Сервіс не такий багатий на функції і гнучкий, як Netlify, але швидко налаштовується і передається в GitHub. Якщо проект вже розміщений на GitHub, то процес хостингу стає максимально простим, тому що всі зміни файлів розгортаються миттєво[19].

Також GitHub може похвалитися контролем доступу, багтрекінгом, управлінням завданнями і документами для кожного проекту. Мета GitHub - сприяти взаємодії розробників.

До проекту, завантаженого на GitHub, можна отримати доступ за допомогою інтерфейсу командного рядка Git і Git-команд. Також є й інші функції, такі як документація, запити на прийняття змін (pull requests), історія коммітів, інтеграція з безліччю популярних сервісів, email-повідомлення, емодзі, графіки, вкладені списки завдань, система @ згадок, схожа на ту, що в Twitter, і т.д.

1.3. Конструювання програмної системи

Оскільки програмна реалізація відбувається в рамках фреймворку GatsbyJS, то програмісту слід реалізувати певний функціонал плагінів системи, щоб мати змогу виконати всі заплановані функції.

Існує багато варіантів для додавання функціональних коментарів, кілька з яких спеціально орієнтовані на статичні сайти. Хоча цей список аж ніяк не вичерпний, він слугує гарною відправною точкою для ілюстрації того, що доступно:

- Disqus;
- Commento;
- Facebook Comments;
- Staticman;
- JustComments (офіційний плагін для Gatsby);
- TalkYard;
- Gitalk.

Найзручнішою у користуванні з GatsbyJS, який заснований на ReactJS є секція коментування від Facebook, тобто від авторів оригінального ReactJS. Для того щоб мати змогу користуватися секцією треба завантажити необхідний плагін через npm. Це можна зробити скориставшись командою:

```
npm install react-facebook -g
```

За замовчуванням FacebookProvider негайно завантажує скрипт facebook разом з componentDidMount. Можна використовувати лише один зразок FacebookProvider на своїй сторінці[20]. Реалізація додавання коментарів на сторінку (лістинг 1.1).

Лістинг 1.1 – Додавання коментарів Facebook

```
import React, { Component } from "react";
import {
  FacebookProvider,
  Like,
  Share,
  ShareButton,
  Comments,
```

Продовження лістингу 1.1

```
  CommentsCount,
  LoginButton,
  Profile
} from "react-facebook";

export default class commentSection extends Component {
  handleResponse = data => {
    console.log(data);
  };

  handleError = error => {
    this.setState({ error });
  };
  render() {
    return (
      <FacebookProvider appId="commentSection">
        <Comments href="http://www.facebook.com" />
        <CommentsCount href="http://www.facebook.com" />
        <Profile>
          {({ loading, profile }) => (
            <div>
              {profile.picture}
              {profile.name}
            </div>
          )}
        </Profile>
        <LoginButton
          scope="email"
          onCompleted={this.handleResponse}
          onError={this.handleError}
        >
          <span>Login via Facebook</span>
        </LoginButton>
        <Like
          href="http://www.facebook.com"
          colorScheme="dark"
          showFaces
          share
        />
        <Share href="http://www.facebook.com">
          {({ handleClick, loading }) => (
            <button        type="button"        disabled={loading}
onClick={handleClick}>
              Share
            </button>
          )}
        </Share>
        <ShareButton
href="http://www.facebook.com">Share</ShareButton>
```

Продовження лістингу 1.1

```
      </FacebookProvider>
    );
  }
}
```

GatsbyJS побудований на основі React. Тож все, що можливо за допомогою форм в React, можливо і в Gatsby. Для початку треба побудувати зовнішній вигляд форми зворотного зв’язку[21]. Реалізація зовнішнього вигляду наведена у лістингу 1.2

Лістинг 1.2 – Реалізація зовнішнього вигляду форми зворотного зв’язку

```
import React from "react"
export default class IndexPage extends React.Component {
  state = {
    name: "",
    phoneNumber: "",
  }
  handleInputChange = event => {
    const target = event.target
    const value = target.value
    const name = target.name
    this.setState({
      [name]: value,
    })
  }
  handleSubmit = event => {
    event.preventDefault()
  }
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Ваше ім’я
          <input
            type="text"
            name="name"
            value={this.state.name}
            onChange={this.handleInputChange}
          />
        </label>
        <label>
          Ваш номер телефону
          <input
            type="text"
```

Продовження лістингу 1.2

```
            name="phoneNumber"
            value={this.state.phoneNumber}
            onChange={this.handleInputChange}
          />
        </label>
        <button type="submit">Замовити консультацію</button>
      </form>
    )
  }
}
```

Ця форма на даний момент нічого не робить, окрім того, що показує інформацію про користувача, яку він щойно ввів. Тепер треба перенести цю форму в компонент, надіслати стан форми на бекенд сервер або додати надійну перевірку. Також можна використовувати бібліотеки форм React, такі як Formik або Final Form, щоб прискорити процес розробки.

- форма має атрибут data-netlify = "true", який повідомляє Netlify зареєструвати форму під час створення вашого сайту;
- форма має атрибут name, що описує її. Це ім'я Netlify надасть формі при розгортанні її коду;
- атрибут name форми повторюється в прихованому полі імені форми, що є абсолютно необхідним. Якщо пропустити це поле або ввести неправильне ім’я, то записи або повернуть помилку при спробі запиту, або загубляться десь у безодні Інтернету;
- Кожне поле має атрибут name. Поле повинно мати ім'я, щоб ці дані зберігалися в Netlify[22].

Ще один момент - це те, що форми Netlify не працюють при локальній розробці. Коли цей код вперше додається, Netlify не знає про форму, і вона не надсилається до Netlify. Натомість доведеться розгорнути свій код (через Netlify), щоб перевірити форму. Рекомендується робити це за допомогою попереднього перегляду Deploy, який дасть змогу протестувати перед тим, як форма піде в продакшн.

Після комміту і оновлення коду на GitHub, почавши збірку Netlify (якщо все налаштовано правильно), можна буде спостерігати форму після розгортання сайту (або попереднього перегляду). Тоді форма повинна податись належним чином, що буде видно через інтерфейс Netlify.

1.4. Використання програмної системи

1.4.1. Розгортання програмної системи та системні вимоги

Відмінна річ у статичних сайтах - це те, що їх можна розміщувати майже де завгодно. Отримується папку заздалегідь збудованих файлів, які можна завантажувати на будь-який веб-сервер, і все завершено. Навіть можете скинути її у Amazon S3 і заощадити багато грошей за дуже мало роботи.

Тож хостинг простий, але статичні веб-сайти додають ручну роботу для розгортання змін, внесених до коду чи вмісту сайту, на відміну від WordPress або інших традиційних CMS, де зміна вмісту одразу ж починає працювати.

Якщо якась форму автоматизації розгортання не налаштована, то доведеться вручну ініціювати збірку та завантажувати її самостійно. Існує спосіб змусити GatsbyJS запустити збірку у хмарі та розгорнути нову версію сайту на хостингу без переривання робочого процесу. Одним із способів є скористатися Amazon Web Services, після великої роботи з внутрішніми налаштуваннями. Але можна це безкоштовно зробити використавши інший хостинг провайдер з малою внутрішньою конфігурацією або взагалі без неї.

З Netlify це дуже просто. Netlify дуже добре інтегрується з GitHub. У терміналі Node JS запускається команда:

```
$ git init
```

Потім необхідно перейти аккаунту GitHub і створити новий репозиторій gatsby-contentful-deployment, після чого виконати команди для переходу до наявного сховища.

```
$ git add
```

```
$ git commit -m 'initial commit'
$ git remote add origin git@github.com:username/gatsby-contentful-deployment.git
$ git push -u origin master
```

Коли ваш код буде на GitHub, потрібно перейти до Netlify та створити обліковий запис. На панелі інформаційних панелей натиснути на New site from Git, вибрати GitHub як постачальника та пройти процедуру авторизації, вибравши всі необхідні варіанти.

Далі вибрати репозиторій з поданого списку. Для цього треба вибрати Configure the Netlify app on GitHub. Це відкриє спливаюче вікно, яке дозволяє вибрати репозиторій, який необхідно дозволити для використання з Netlify. Після вибору проекту для розгортання, система перенаправить на екран розгортання Netlify, і тепер можна вибрати репозиторій gatsby-contentful-deployment. Далі, все що потрібно, це просто натиснути кнопку Deploy site в кінці сторінки, що показано на рис. 1.7

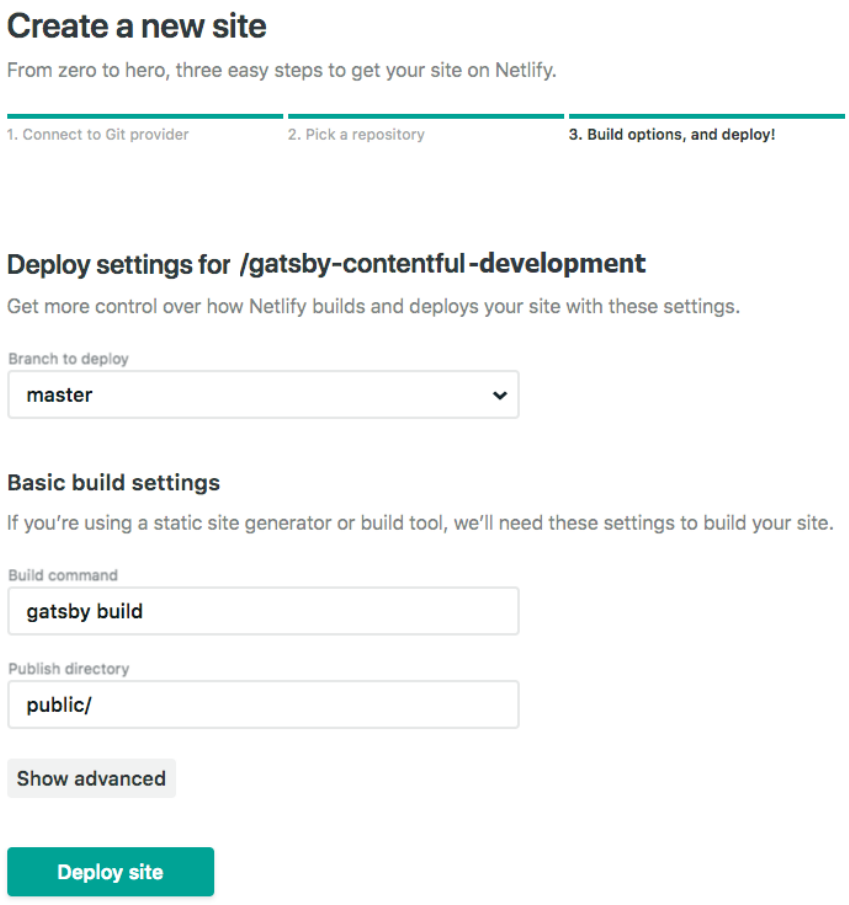

Рисунок 1.7 – Вікно підтвердження розгортання

Netlify дуже легко і з мінімальною конфігурацією запускає сайти написані на GatsbyJS. Новий сайт має розгортатися не більше ніж кілька хвилин (рис 1.8).

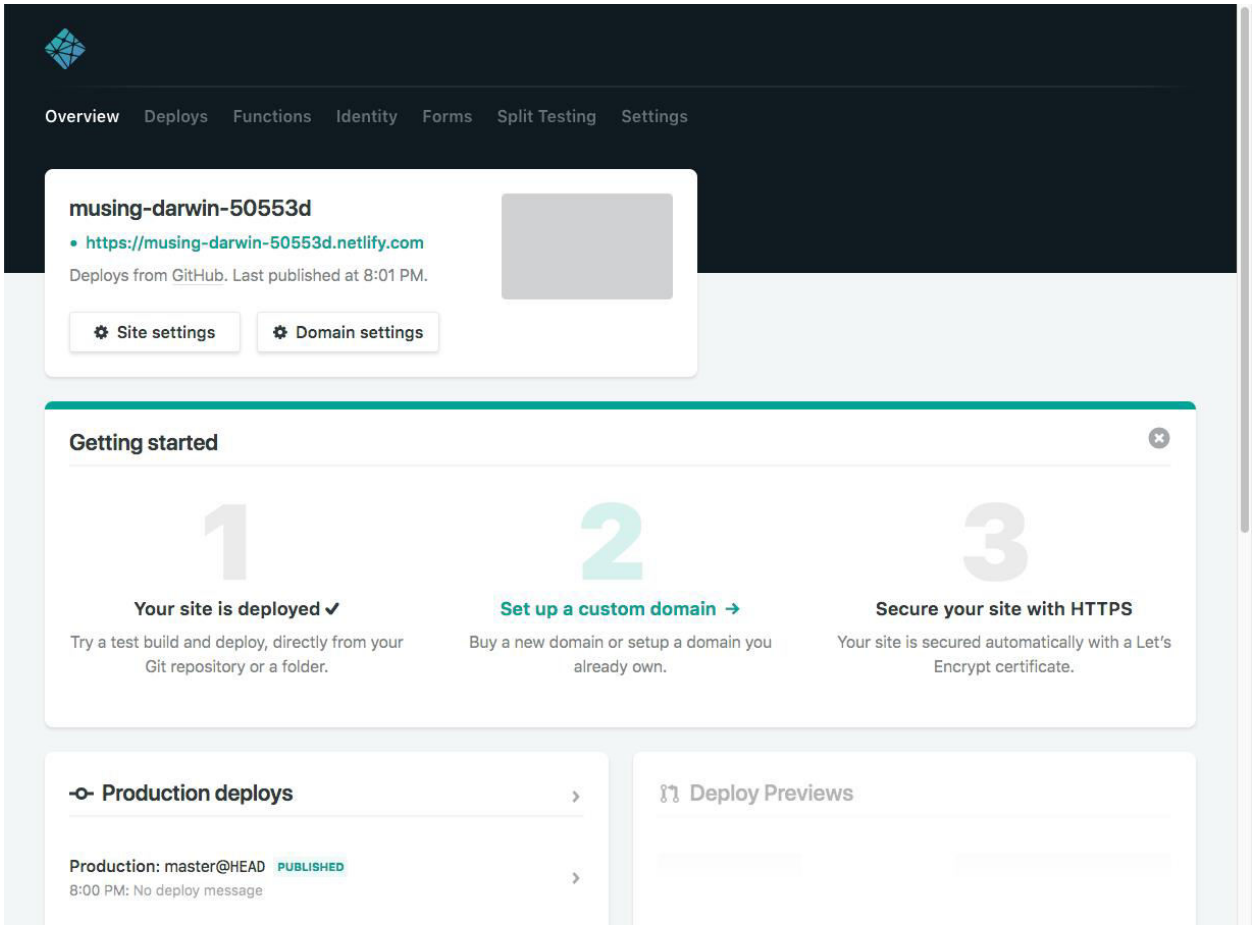

Рисунок 1.8 – Вікно розгортання веб-сайту

Раніше доводилося вимикати сервер і перезапустити його, щоб отримати нові дані. Рішенням того, щоб не запускати повторне розгортання вручну коли хтось додає або змінює вміст у Contenful є використання хуків Contentful для запуску автоматичного перерозподілу сайту Netlify.

Це означає, що нові сторінки будуть додані до веб-сайту автоматично для кожної нової доданої публікації. Крім того, якщо ви використовувати плагін для Gatsby sitemap, нові сторінки будуть включені у мапу сайту, коли вона буде відроджена під час розгортання, що значно полегшить

класифікацію за ключовими словами та допоможе значно мінімізувати суєту з SEO.

Далі потрібно у вікні Netlify натиснути Site settings, а потім у меню ліворуч вибрати Build & Deploy. Після цього знайти кнопку Add build hook і натиснути на неї, даючи ім’я збірному хукові, наприклад contentful, а потім натиснути кнопку Save.

Тепер треба скопіювати URL-адресу збірки та повернутися до змістовної інформаційної панелі, натиснути випадаюче меню та вибрати Webhooks. Екран Webhooks вже має внизу праворуч шаблон для Netlify, на який необхідно натиснути (рис. 1.9).

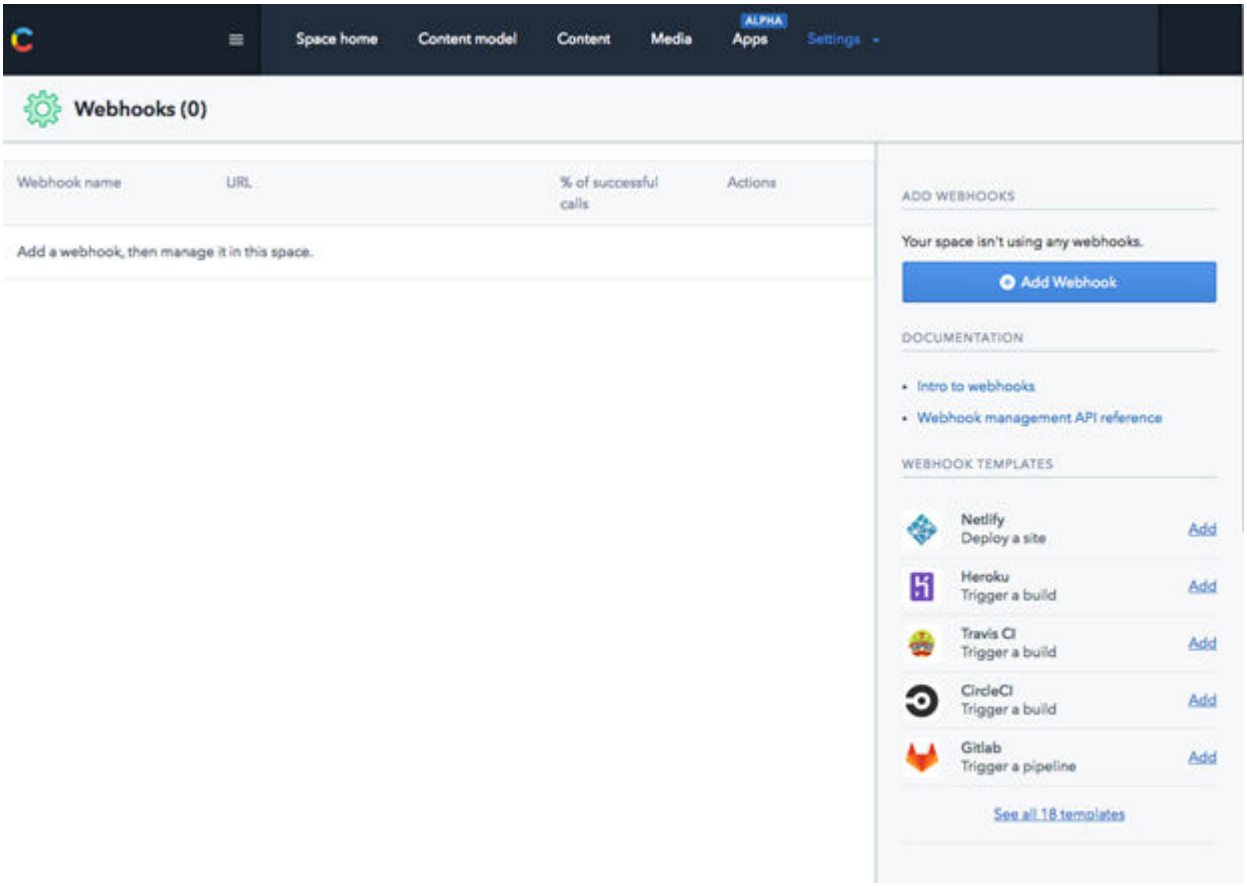

Рисунок 1.9 – Додавання Webhooks

У формі, що з’явиться, додати URL-адресу хука побудови Netlify та натиснути Create Webhook.

Тепер треба повернутися на сторінку вмісту та додати нову публікацію на веб-сайт. Як тільки буде натиснуто на публікацію, Contentful здійснить виклик API на щойно створений хук. Це, в свою чергу, призведе до того, що Netlify буде повторно розгортати сайт. GatsbyJS повторно надасть Contentful дані, які тепер включають нову додану публікацію, і створить нову сторінку на основі нової публікації на веб-сайті[23].

1.4.2. Опис типових схем використання системи

Для клієнта, найтиповішою дією на сайті є відкриття самого сайту, все що необхідно - це перейти за веб-адресою service.center.netlify.com і відкриється головна сторінка сайту (рис. 1.10):

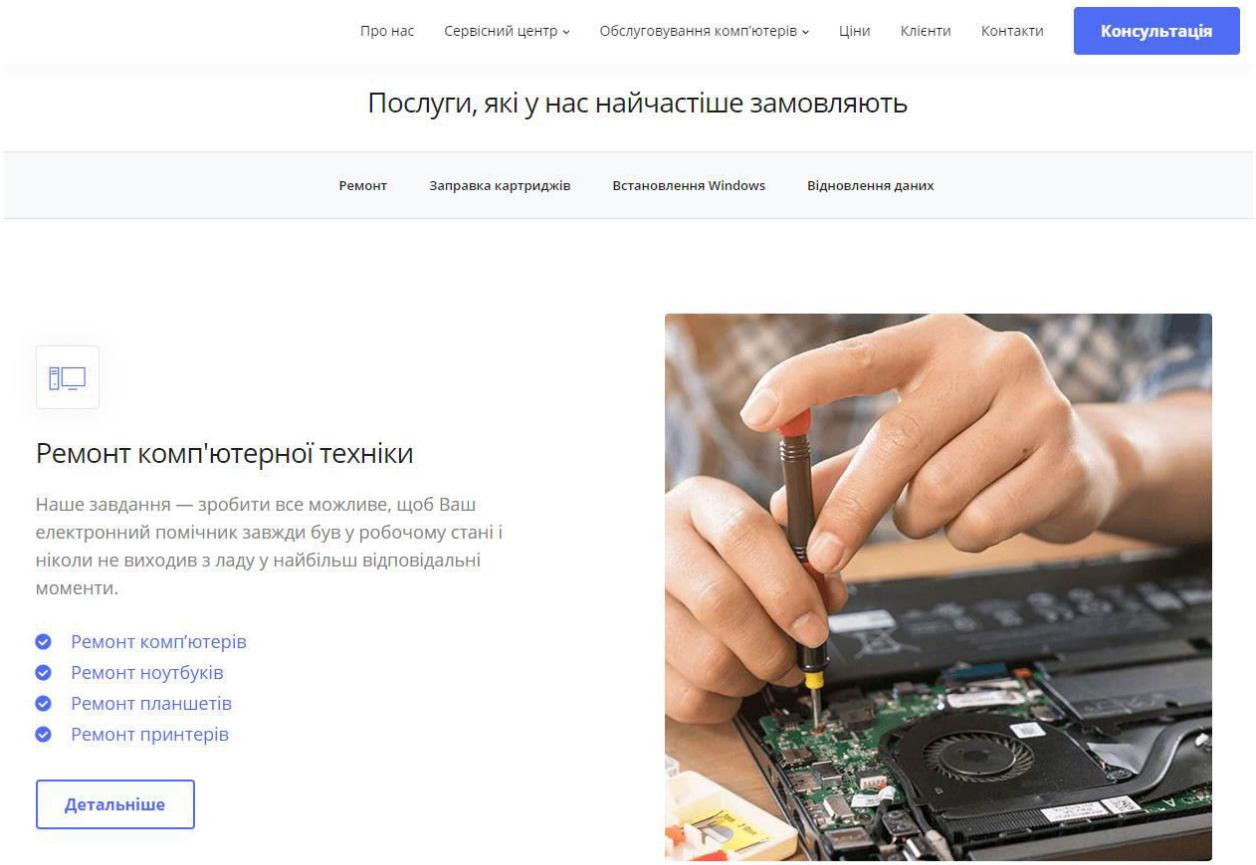

Рисунок 1.10 – Головна сторінка веб-сайту

Наступною дією будь-якого користувача буде перегляд категорій у вкладці, що містить список послуг з обслуговування комп’ютерної техніки,

наприклад, якщо відвідувач шукає Ремонт ноутбуків у категорії Сервісний центр, то йому буде представлена сторінка з видами послуг (рис 1.11).

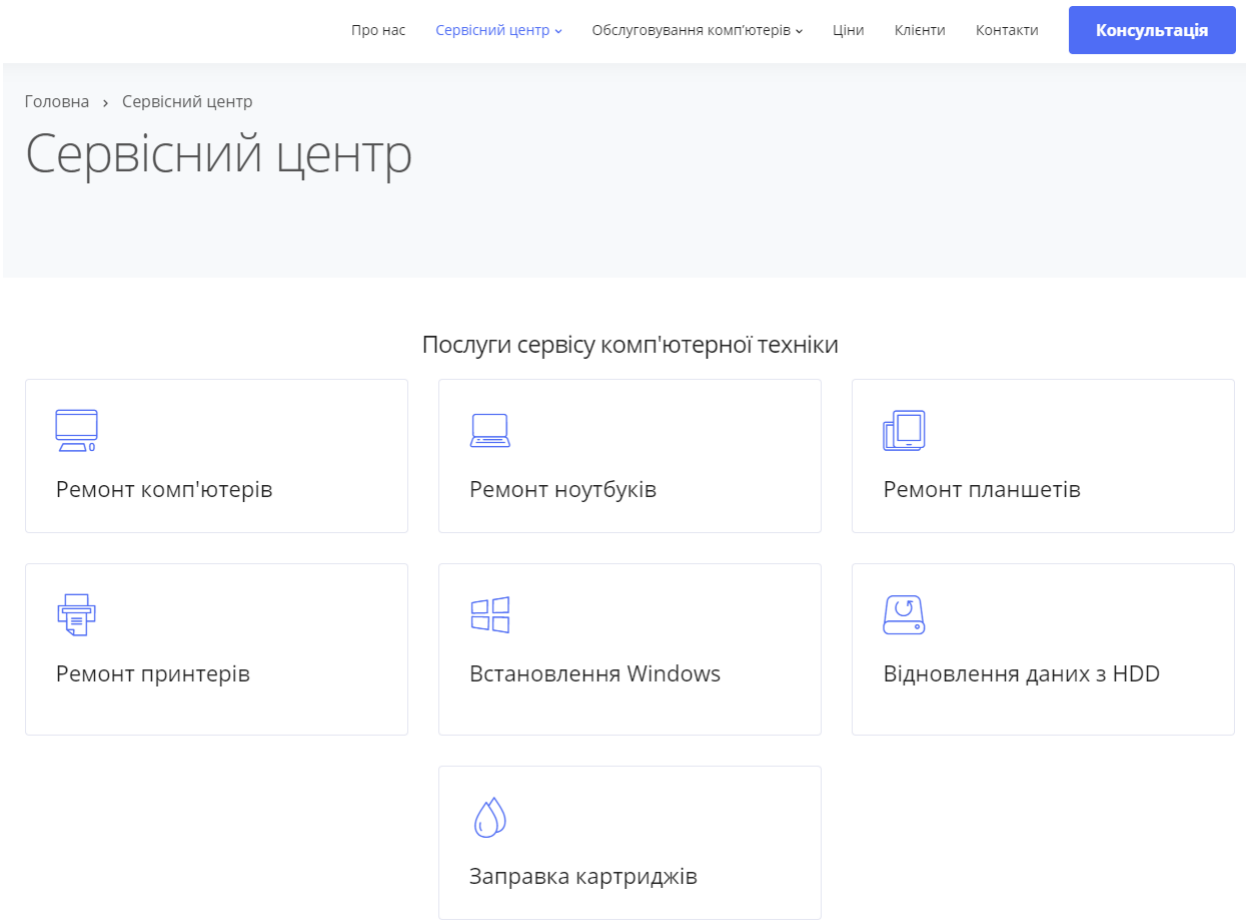

Рисунок 1.11 – Сторінка з видами послуг

Відвідувач вибирає послугу, яка його цікавить, що зумовлює перехід на нову вкладку, де показується вся наявна про цю послугу інформація (рис 1.12) крім ціни на неї. Це зумовлено тим, що ціну визначає складність ремонту та вартість компонентів для заміни, тому для того, щоб ознайомитися з цінами відвідувач має перейти на сторінку з прайслистом, в якому приблизні розцінки ремонту всіх можливих неполадок, які можуть статися з комп’ютерною технікою.

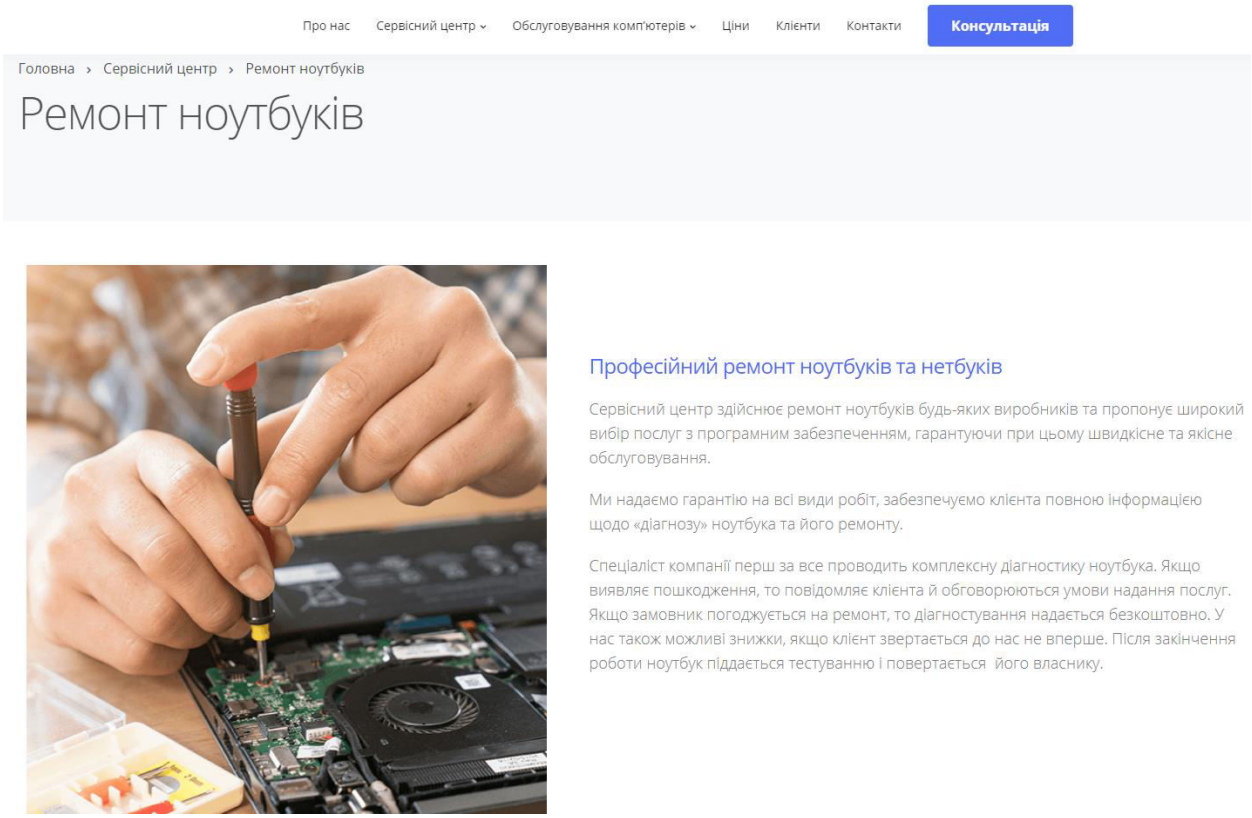

Рисунок 1.12 – Сторінка з видами послуг

Визначившись з видом послуги, відвідувач може замовити консультацію, натиснувши відповідну кнопку на навігаційному меню. Це приведе до відкриття форми зворотного зв’язку, де можна ввести своє ім’я, телефон та описати технічну проблему, яку йому треба вирішити. Після підтвердження надсилання форма буде відправлена в адміністрацію сайту для подальшої обробки (рис 1.13).

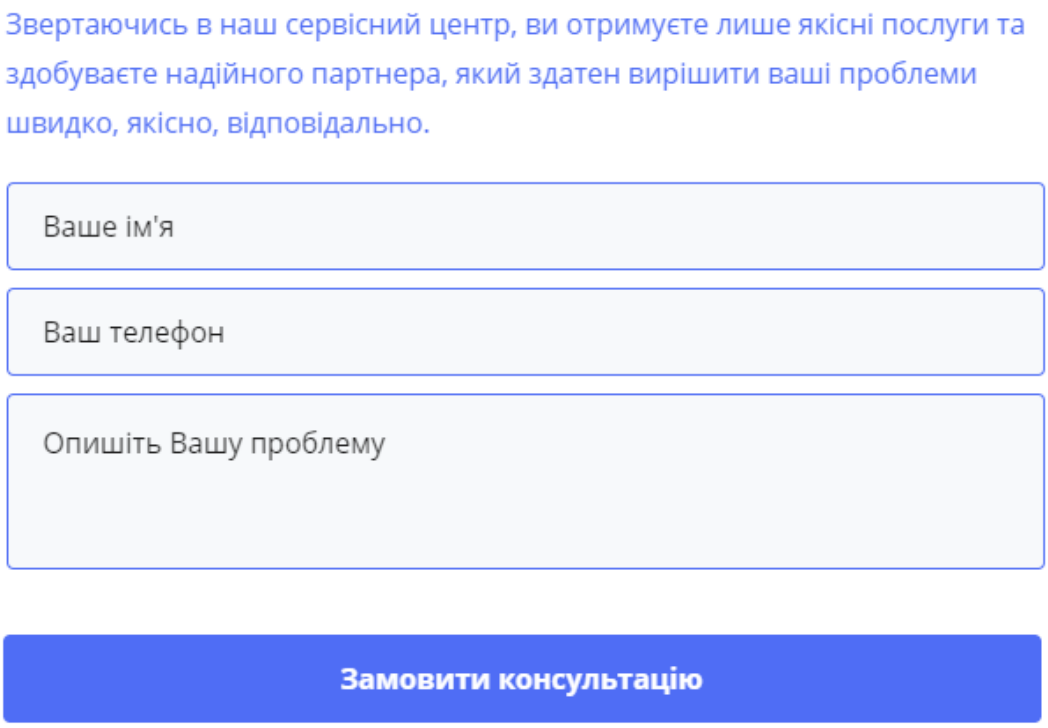

Рисунок 1.13 – Форма зворотного зв’язку

Якщо відвідувач хоче переглянути відгуки про роботу сервісного центру від інших клієнтів, то він може це зробити перейшовши на відповідну сторінку. Крім цього, за умови, що відвідувач авторизований в одній з соціальних мереж, наприклад, Facebook, він має змогу прокоментувати, заповнивши відповідне поле вводу тексту (рис. 1.14).

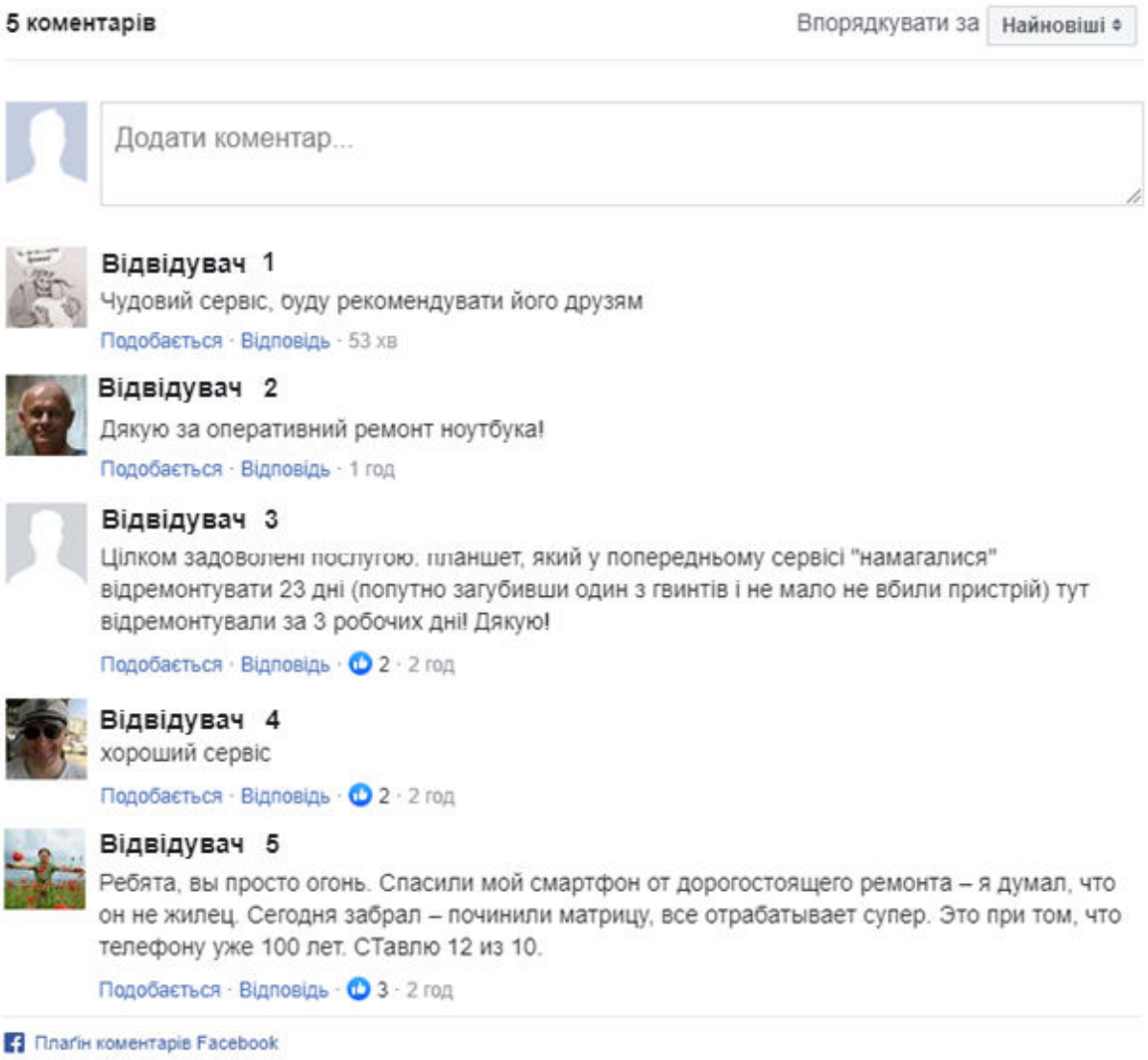

Рисунок 1.14 – Секція коментарів

2. ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1. Розробка Unit-тестів

Unit-тести - це прекрасний спосіб захистися від помилок у коді, перш ніж розгорнути його. У той час як Gatsby не включає підтримку тестування блоку поза «коробкою», для застосування та запуску потрібно лише кілька кроків. Однак є кілька особливостей процесу збирання Gatsby, які означають, що стандартні налаштування Jest не працюють[25].

Найпопулярнішим тестовим фреймворком для React є Jest, яку створив Facebook. Хоча Jest - це тестовий фреймворк JavaScript загального призначення, він має безліч функцій, завдяки яким він особливо добре працює з React.

Спочатку потрібно встановити Jest та ще кілька необхідних пакетів, наприклад, babel-vice і babel-preset-gatsby, щоб гарантувати, що попередньо встановлені пресети babel, що використовуються всередині сайту Gatsby:

`npm install --save-dev jest babel-jest react-test-renderer babel-preset-gatsby identity-obj-proxy.`

Оскільки Gatsby обробляє власну конфігурацію Babel, то потрібно буде вручну повідомити Jest використовувати babel-vice. Найпростіший спосіб зробити це - додати jest.config.js (лістинг 2.1). Можна одночасно налаштувати кілька корисних стандартних параметрів.

Лістинг 2.1 – Конфігураційний файл Jest

```
module.exports = {
  transform: {
    "^.+\\.jsx?$": `<rootDir>/jest-preprocess.js`,
  },
  moduleNameMapper: {
    ".+\\.(css|styl|less|sass|scss)$": `identity-obj-proxy`,
".+\\.(jpg|jpeg|png|gif|eot|otf|webp|svg|ttf|woff|woff2|mp4|webm|wav|mp3|m4a|aac|oga)$": `<rootDir>/__mocks__/file-mock.js`,
  },
  testPathIgnorePatterns: [`node_modules`, `.cache`, `public`],
  transformIgnorePatterns: [`node_modules/(?!(gatsby)/)`],
```

Продовження лістингу 2.1

```
  globals: {
    __PATH_PREFIX__: ``,
  },
  testURL: `http://localhost`,
  setupFiles: [`<rootDir>/loadershim.js`],
}
```

Розділ transform повідомляє Jest про те, що всі файли js або jsx потрібно трансформувати за допомогою файлу jest-preprocess.js (лістинг 2.2) у корені проекту. Це налаштує конфігурацію Babel, можна почати з наступної мінімальної конфігурації:

Лістинг 2.2 – Файл jest-preprocess

```
const babelOptions = {
  presets: ["babel-preset-gatsby"],
}
module.exports = require("babel-
jest").createTransformer(babelOptions)
```

Для написання тестів спочатку треба створити тестовий файл. Можете його помістити або у каталог __tests__, або помістити в інше місце (зазвичай поруч із самим компонентом) з розширенням .spec.js або .test.js. Рішення зводиться до власних уподобань. Для цього тестування буде використовуватися конвенцію про папки __tests__. Щоб перевірити компонент заголовка, треба створити файл header.js у src / components / __ testing __ / (лістинг 2.3):

Лістинг 2.3 – Файл header.js

```
import React from "react"
import renderer from "react-test-renderer"
import Header from "../header"
describe("Header", () => {
  it("renders correctly", () => {
    const tree = renderer
      .create(<Header siteTitle="Default Starter" />)
      .toJSON()
    expect(tree).toMatchSnapshot()
```

Продовження лістингу 2.3

```
  })
})
```

Це дуже простий тест, який використовує react-test-render для візуалізації компонента, а потім генерує його знімок під час першого запуску. Потім він порівнює майбутні знімки з цим, а це означає, що можна швидко перевірити регресію.

Якщо ви поглянути всередину package.json, то, ймовірно, можна виявити, що вже є сценарій для тесту, який просто видає повідомлення про помилку. Щоб використовувати новий виконавчий файл, це потрібно змінити, наприклад: замінити базовий рядок текстом "test": "jest".

Це означає, що тепер можна запускати тести, ввівши npm test. Також можна запустити його з прапорцем, що запускає режим перегляду для перегляду файлів та запуску тестів при їх зміні: npm test - --watch.

Може бути отримане повідомлення про написаний знімок. Він створюється в каталозі __snapshots__ поруч з тестами. Можна побачити, що це JSON-представлення компонента <Header />. Необхідно перевірити файли знімків у системі управління джерелом (наприклад, репортаж GitHub), щоб будь-які зміни відслідковувалися в історії. Це особливо важливо пам’ятати, якщо використовувати систему безперервної інтеграції, наприклад Travis або CircleCI, для запуску тестів, які не дадуть коректних результатів, якщо знімок не буде перевірено в керуванні джерелами. Якщо внести зміни, які означають, що потрібно оновити знімок, це можна зробити, запустивши npm test - -u.

2.2. Оцінка балу Lighthouse

Lighthouse повертає показник ефективності від 0 до 100. 0 - це найнижчий можливий бал. Оцінка 0 зазвичай вказує на помилку в Lighthouse. 100 - найкращий результат. Кожен аудит ефективності, який сприяє оцінці, має свою методологію оцінки. Lighthouse відображає кожний необроблений

бал на число від 0 до 100. Розподіл балів - це нормальний записаний розподіл, отриманий із показників ефективності реальних даних ефективності реального веб-сайту в HTTPArchive.

Наприклад, аудит First Meaningful Paint (FMP) вимірює, коли користувач помічає, що основний вміст сторінки вже видно. Необхідна оцінка для FMP являє собою тривалість часу між ініціацією користувачем завантаження сторінки, і тим, що відображає її основний вміст. На основі реальних даних веб-сайтів, найефективніші веб-сайти відображають FMP приблизно за 1,220 мс, так що значення метрики відображається на балі Lighthouse – 99[26].

Показники, що сприяють оцінці ефективності, не однаково зважуються. Більш важкі аудитори мають більший вплив на загальний показник ефективності. Зважування базується на евристиці. Загальний показник ефективності є середньозваженим рівнем цих аудитів.

Кольорове кодування відображає ці показники ефективності:

- 0 до 49 (повільно): червоний;
- 50 - 89 (середній): помаранчевий;
- 90 до 100 (швидко): зелений.

Під час роботи Lighthouse на реальних сайтах слід очікувати певної мінливості показників продуктивності. Під час кожного відвідування сайт може завантажувати різні оголошення чи сценарії, а умови мережі можуть відрізнятися. Антивірусні сканери, розширення та інші програми, що перешкоджають завантаженню сторінки, можуть спричинити великі зміни. Щоб отримати більш послідовні результати, треба запускати Lighthouse без цих програм.

Lighthouse повертає показник прогресивного веб-застосунку (PWA) від 0 до 100. 0 - це найгірший можливий бал, а 100 - найкращий. Аудити PWA базуються на Базовому контрольному списку PWA, де перераховано 14 вимог. Lighthouse має автоматизований аудит для 11 з 14 вимог. Решта 3

можна перевірити лише вручну. Кожен з 11 автоматизованих аудитів PWA зважується однаково, тому кожен вносить приблизно 9 балів до оцінки PWA.

Оцінка доступності - це середньозважена середня кількість усіх аудитів доступності. Кожен аудит доступності проходить або не проходить. На відміну від аудиту ефективності, сторінка не отримує балів за часткове проходження аудиту доступності. Наприклад, якщо деякі елементи мають зручні для екранізатора імена, а інші - ні, ця сторінка отримує 0 для аудиту зручних для читання екранів імен.

Lighthouse повертає бал кращих практик між 0 і 100. 0 - це найгірший можливий бал, а 100 - найкращий. Аудити найкращих практик однаково зважуються. Щоб обчислити, наскільки кожен аудит сприяє загальному рейтингу найкращих практик, треба підрахувати кількість аудитів найкращих практик, а потім поділити це число на 100. На рис. 2.1 показані показники ефективності для дипломного проекту.

Рисунок 2.1 – Lighthouse показники ефективності проекту

3. ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНА ЧАСТИНА

3.1. Загальний підхід до визначення економічної ефективності розробки

Обов’язковою складовою частиною виконуваного проекту дипломної роботи за темою "Розробка сайту для сервісного центру з обслуговування комп'ютерної техніки з використанням технології JavaScript" є визначення фінансових витрат на різних етапах виконання робіт. Відповідно, важливо вірно здійснити фінансову оцінку передбачуваних витрат, продуктивність, корисність та, в результаті, економічну ефективність проекту.

Наукоємні розробки та дослідження, на відміну від корпоративних, не завжди мають за мету отримання прибутку або ж іншої матеріальної вигоди. В багатьох випадках, проекти наукового спрямування не є економічно вигідними. Однак, вони є рушійною силою прогресу, дослідженням незвіданих галузей та проблем, які, у свою чергу, майже завжди впливають на майбутні різнопланові розробки. Тому дуже часто наукова діяльність стимулюється зовнішніми інвестиціями та підтримкою держаних інститутів або міжнародних грантів. В плані використання результатів досліджень та на основі отриманих моделей можна робити прогнози з впровадження нових методик та принципів організації роботи діагностичних установ. Різноманітні медичні центри зможуть скористатися розробленою системою на практиці для покращення існуючих та отримання нових діагностичних методик.

Оцінка вартості дослідницьких розробок базується на витратному підході: використанні первісної вартості об'єктів, виходячи з фактичних витрат на розробку та доведення до комерційного використання з урахуванням амортизації. Результати роботи у вигляді математичних моделей та реалізованого на їх основі ПЗ не будуть використовуватися з комерційною метою та не підлягатиме продажу, а становлять наукову та інтелектуальну цінність, відповідно прибутку від продажу ПЗ та розробки як такого не передбачається. Інакше кажучи, всі вкладені кошти та витрати на

розробку даного рішення є не самоокупними, а лише несуть витрати у кількості залучених ресурсів та матеріальних засобів.

Згідно зі статтею 8 Закону № 3792-12 передбачено, що твори наукового характеру та комп'ютерні програми є об'єктами авторського права[27]. У разі реєстрації виконаної роботи як інтелектуальної власності та авторського права у державній службі інтелектуальної власності України необхідно буде сплатити збори за державну реєстрацію.

Для отримання відмінних результатів експериментів та доцільності розробки такого спеціалізованого ПЗ потрібні відповідні витрати на дослідження та розробку, що становитиме основу витрат, які будуть здійснені протягом підготовки та виконання реалізації даного рішення.

Уявно модель витрат можна поділити на дві основні частини: витрати, пов'язані з дослідженням предметної області, побудовою математичних моделей, отриманням попередніх результатів експериментів, та частину реалізації програмної системи, архітектури і тестування.

Враховуючи залежність якості кінцевого продукту від кваліфікації програмістів, потрібно сконцентрувати увагу на якості та результативності розробки. Для цього потрібно залучити інноваційні технології, провести грунтовний аналіз предметної області, тестування та оцінку результатів розробки. Для забезпечення хорошої результативності при розробці доводиться йти на додаткові заходи заохочення та стимулювання у вигляді преміювання працівників, підтримання наукового дослідження досвідом іноземних науковців, дорогоцінних лабораторних дослідів.

До створення ПЗ можуть бути залучені позаштатні розробники як зареєстровані, так і не зареєстровані підприємцями. В обох випадках співпраця з ними здійснюється на підставі цивільно-правового договору, найчастіше – договорами підряду. Оподаткування виплат за договором підряду залежить від того, чи зареєстрований виконавець підприємцем. Важливим етапом розробки ПЗ є його тестування, яке виконується тестувальником за певну винагороду. Якщо тестувальники не перебувають з

підприємством у трудових відносинах, оплата виконується на основі договору підряду на виконання робіт з тестування ПЗ. Сам же результат розробки не оподатковується, адже не є комерційним проектом і не спрямований на продаж.

3.2. Розрахунок вартості процесу розробки та оцінка економічної ефективності проекту

Для оцінки нематеріальних активів використовують міжнародні стандарти оцінки розрахунку вартості об’єктів інтелектуальної власності, розроблені TIAVIS (The International Assets Vabulation Standarts Commitie).

Виконання розробки програмного забезпечення з огляду економічної моделі можна виконувати двома способами: процедурним та об’єктно-орієнтованим. Обидва підходи потребують залучення ресурсів у вигляді програмістів-розробників, тестувальників, керівника проекту, наукового ресурсу. Різниця виникає в схемі розробки, тривалості періоду розробки та відповідній вартості. Процедурний підхід для розробки ПЗ в основі якого лежать процедури і функції передбачає розробку програми як монолітного композиту, що в подальшому, як правило, вимагає великих витрат на супровід та модернізацію. Об’єктно-орієнтований підхід, що грунтується на основі об’єктів певних класів, що описують певну область і певну поведінку (методи) та володіють властивостями (атрибутами), орієнтовані на варіанти використання та покроковий процес розробки.

Для початку робіт необхідно скласти технічне завдання на розробку, яке є основним документом, що регламентує подальшу роботу, та містить докладний опис необхідних функцій програми, інтерфейс, технології, інше. Вартість складання технічного завдання переважно складає до 10% від планованої вартості розробки. Роботу зі складання технічного завдання веде керівник проекту разом з програмістами та консультуючись із замовником.

Усі програмісти, що працюють у штаті підприємства-розробника мають певний встановлений посадовий оклад. Місячний оклад, денна

заробітна плата, трудомісткість (днів) і основна заробітна плата кожного учасника техпроцесу представлені у таблиці 3.1. Всі суми наведені в національній валюті – в гривні.

Таблиця 3.1 – Розрахункова вартість технологічного процесу розробки

Посада	Місячний оклад, грн.	Денна зар. плата, грн.	Об'єктно-орієнтований підхід		Процедурний підхід	
			Днів	Сума, грн.	Днів	Сума, грн.
Фронт-енд розробник	11550,00	525,00	19	9975,00	24	12600,00
Інженер-програміст	12760,00	580,00	19	11020,00	24	13920,00
Тестувальник	7700,00	390,00	4	1560,00	4	1560,00
Додаткова зар. плата 20%			23	4511,00	28	5616,00
Фонд оплати праці 36,77%				8293,47		10325,01
Всього витрат на зар. плату				35359,47		43566,01
Військовий збір 1,5%				530,39		653,49
Єдиний соціальний внесок 3.6%				1272,94		1568,38
ПДВ, 15%				5303,92		6534,90
Всього				**42466,72**		**52322,78**

Згідно з вимогами та прорахованою кількістю необхідних ресурсів на виконання, розробку, тестування та дослідницьку роботу було отримано основні часові рамки роботи над проектом. Так для об'єктно-орієнтованого підходу загальна тривалість роботи над ПЗ становить 19 робочих днів (під робочим днем розуміється 8-ми годинний робочий день), що включає роботу інженера-програміста, який, в свою чергу, є і керівником розробки, фронт-енд розробника роботу та тестувальника.

Сума витрат на заробітню плату становить 42466,72 гривень включаючи всі види додаткових оплат. Для процедурного підходу до розробки суми дещо більші, адже затрачається більше часу на розробку. Так, при використанні процедурного підходу сумарна тривалість часу розробки становить 24 робочі дні, а витрати у вигляді виплат заробітної плати становлять 52322,78 гривень.

Витрати на науково-дослідницьку роботу та здійснення розробки програмних продуктів і об'єктно-орієнтованим, і процедурним способом включають:

Основна заробітна плата:

$ЗП_{осн\ 1} = 22555$ грн; $ЗП_{осн\ 2} = 28080$ грн.

Додаткова заробітна плата обчислюється як $ЗП_{дод} = 0,2 \cdot ЗП_{осн}$.

$ЗП_{дод\ 1} = 0,2 \cdot 22555 = 4511,00$ грн; $ЗП_{дод\ 2} = 0,2 \cdot 28080 = 5616,00$ грн.

Нарахування на фонд оплати праці (ФОП):

$ФОП_{ЄСВ} = 0,3677 \cdot ФЗП$

$ФОП_{ЄСВ1} = 0,3677 \cdot 22555 = 8293,47$ грн;

$ФОП_{ЄСВ1} = 0,3677 \cdot 28080 = 10325,01$ грн.

Всього витрат:

$В_{ЗП1} = ЗП_1 + ФОП_{ЄСВ1} + ЗП_{дод1} = 35359,47$ грн;

$В_{ЗП2} = ЗП_2 + ФОП_{ЄСВ2} + ЗП_{дод2} = 43566,01$ грн.

З цієї суми утримуються обов'язкові відрахування на заробітну плату: єдиний соціальний внесок, який складає 3,6% від суми нарахованої заробітної плати та податок на доходи фізичних осіб, що складає 15% від суми нарахованої заробітної плати, зменшеної на суму єдиного внеску на загальнообов'язкове соціальне страхування та податкової соціальної пільги, військовий збір у розмірі 1,5%, від суми нарахувань[28, 29]. До окремих витрат також відносяться витрати на куповані вироби (матеріальне забезпечення) та спец обладнання для підтримки експерименту, накладні витрати. Витрати, що будуть супроводжувати проект розробки, порівнюватимемо в двох можливих підходах розробки.

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни (формула 3.1).

$$M_{Bi} = q_i \cdot p_i \quad (3.1)$$

де q_i – кількість витраченого матеріалу i-го виду; p_i – ціна матеріалу i-го виду.

Матеріальні витрати в рамках проекту наведені в таблиці 3.2. Загальна сума матеріальних витрат становить 2449 гривень.

Таблиця 3.2. – Матеріальні витрати

Найменування ресурсу	Кількість, шт.	Ціна одиниці, грн	Загальна сума, грн
Флешка 4Gb	2	80,00	160,00
Папір для друку А4, арк	100	0,158	15,80
Фарба для принтера	1	90,00	90,00
Дошка для записів	1	750,00	750,00
Перманентний маркер	5	17,00	85,00
Всього			1100,80

Розрахунок витрат на електроенергію одиниці обладнання визначаються за формулою:

$$З_в = W * T * S \quad (3.2)$$

де W – необхідна потужність, кВт; T – кількість годин роботи обладнання; S – вартість кіловат-години електроенергії, S = 2,50 грн/кВт·год.

$З_{в1}$ = 0.7 * 152 * 2.50 = 266 грн;

$З_{в2}$ = 0.7 * 192 * 2.50 = 336 грн;

Розрахунок суми амортизаційних відрахувань. Комп’ютери та оргтехніка належать до четвертої групи основних фондів. Для цієї групи

річна норма амортизації дорівнює 60%, вартість яких перевищує 1000 грн. і визначається:

$$A = \frac{C_{\text{Б}} \cdot N_{\text{А}} \cdot T_{\text{ФАК}}}{\text{Т}_{\text{ГОД}}} \quad (3.3)$$

де $C_{\text{Б}}$ – балансова вартість обладнання, грн; N_A – норма амортизаційних відрахувань в рік, %; $\text{Т}_{\text{ФАК}}$ – фактичний час роботи обладнання по написанню програми, год; $\text{Т}_{\text{ГОД}}$ – річний робочий фонд часу, год.

У даній формулі норма відрахувань на амортизацію рівна N_A = 0,6. Балансова вартість обладнання вказана в таблиці 3.3 і рівна $C_{\text{Б}}$ = 49948 гривень. Річний робочий фонд часу приймемо за $\text{Т}_{\text{ГОД}}$ = 2200 годин. З них реальний фактичний робочий час для об'єктно-орієнтованого та процедурного підходів становить $\text{Т}_{\text{ФАК1}}$ = 152 та $\text{Т}_{\text{ФАК2}}$ = 192 відповідно. Згідно з вищезгаданою формулою, витрати на амортизацію для об'єктно-орієнтованого підходу становлять 2070,57 гривень, а для процедурного - 2615,45 гривень.

Таблиця 3.3 – Перелік необхідного обладнання

Найменування	Кількість, шт	Ціна, грн	Сума, грн	
Комп'ютер	3	12600,00	37800,00	
Принтер Samsung SL-M2026	1	1648,00	1648,00	
Операційна система Windows 10	3	3500,00	10500,00	
Середовища розробки	3	безкоштовно	безкоштовно	
Всього більше 1000 грн.			49948,00	
Всього витрат на амортизацію			2070,57	2615,45
Всього			52018,57	52563,45

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління та створення необхідних умов праці та закупівлю ресурсів та обладнання для розробки наведені в таблиці 3.3.

Залежно від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників. Нехай вона буде дорівнювати 35%, що становить 9473,10 грн для об’єктно-орієнтованого і 11793,60 грн для процедурного підходу розробки.

Проведемо розрахунок вартості створюваного програмного продукту. Вартість продукції включає у собі собівартість і планований прибуток. Найважливішим моментом для розробника, з економічної точки зору, є процес встановлення ціни.

Можна отримати загальні значення витрат на розробку та реалізацію проекту, враховуючи всі вище описані витрати та нарахування. Цей вид витрат складається з сум витрат на оплату праці (всього витрати на оплату праці), матеріальні витрати, витрати на електроенергію, накладні витрати, витрати на обладнання, враховуючи амортизації обладнання на час виконання проекту.

Собівартість продукції – це сума грошових витрат підприємства (фірми) на виробництво і збут одиниці продукції, виконання робіт та надання послуг[30]. Повна собівартість програмного продукту дорівнює сумі усіх витрат на його виробництво: 95852,09 грн використовуючи об’єктно-орієнтований підхід, 106323,03 грн при процедурному підході розробки.

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Економічна ефективність (Е) полягає у відношенні результату виробництва до затрачених ресурсів:

$$E = \frac{\text{П}}{\text{Св}} \tag{3.4}$$

де *П* – прибуток, *П= В - Св; Св* – собівартість.

У випадку наявної розробки, маючи некомерційний проект без економічно корисного результату, можна прогнозувати, що економічна ефективність прямує до 0 у обох випадках. Однак це не є причиною для негативного економічного висновку щодо проекту, адже такого плану розробки приносять користь у вигляді інтелектуальних ресурсів, і, переважно, фінансуються або виконуються на замовлення організацій, зацікавлених в отриманні результатів досліджень та розробок. Однак за надходження коштів на реалізацію ззовні можна вважати ефективність проекту рівною 1 (Е = 1), що означає перекриття витрат на розробку у повній мірі, тобто фінансування.

Якщо ринкова вартість програмного продукту рівна прийнятій, то економічна ефективність визначається встановленим рівнем прибутку. Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень ($T_{ок}$):

$$T_{\text{ок}} = \frac{1}{E} \tag{3.5}$$

У випадку, який розглянутий в дипломній роботі, прямого прибутку не існує. Прибуток можна прогнозувати на підприємствах, організаціях чи відомствах, що зацікавлені в дослідженні. Окупність же для розробки ПЗ за ефективності рівній одиниці можна вважати теж рівній 1 згідно з формулою 3.4.

Виходячи із експертних оцінок і складності програми, приймемо величину витрат на супровід і модернізацію програмного забезпечення, створеного за об’єктно-орієнтованим методом 20% (19170,41 грн) від початкових витрат, а за процедурним – 25% (26580,75 грн).

Однак варто зауважити, що розробка спрямована на короткотривалу підтримку та не прогнозує модернізації. Тобто у разі короткотермінової підтримки все ж за доцільніше обирати об’єктно-орієнтовану модель розробки, адже вартість короткотермінової підтримки згідно з цією моделюю не вплине значно на сукупну вартість розробки.

Здана в експлуатацію система не завжди цілком завершена, її треба змінювати протягом терміну експлуатації. Внаслідок змін система стає більш складною і погано керованою. Об’єктно-орієнтоване представлення програми дозволяє навіть програмісту з середніми професійними навичками швидко і ефективно супроводжувати і вдосконалення програми, що значно скорочує подальші витрати на супровід і модернізацію [30].

Сумарні дані економічного розрахунку розробки дипломного проекту наведені в таблиці 3.4.

Таблиця 3.4 – Загальні витрати

Вид витрат	Об’єктно-орієнтований підхід, грн	Процедурний підхід, грн
Зарплата основна	22555,00	28080,00
Зарплата додаткова	4511,00	5616,00
Фонд заробітної плати	27066,00	33696,0
Відрахування на ФОП	8293,47	10325,01
Разом на оплату праці	42466,72	52322,78
Матеріальні витрати	1100,80	1100,80
Електроенергія	266,00	336,00
Амортизація	2070,57	2615,45
Накладні витрати	9473,10	11793,60
Обладнання	52018,57	52563,45
Разом на ін. витрати	54089,14	55178,90

Собівартість	95852,09	106323,03
Прибуток	0,00	0,00
Економічна ефективність	1	1

Продовження таблиці 3.4 - Загальні витрати

Термін окупності	1	1
Вартість розробленого ПЗ	124607,71	138219,93
Порівняльна економія витрат	13612,22	-
Модернізація і супровід	19170,41	26580,75
Загальні витрати (для споживача)	143778,12	164800,68
Порівняльна економія витрат (для споживача)	21022,56	-
Дохідність проекту для споживача за витратною частиною	-140519,15	-160281,95
Економія	19762,8	-

Загальна вартість пропонованих робіт становить 143778,12 гривень для процедурного і 164800,68 гривень для об’єктно-орієнтованого підходів розробки. У випадку реалізації дипломного проекту варто вибрати об’єктно орієнтований підхід для розробки такого ПЗ, адже фінансово це більш вигідно. Також у даній методиці розробки кращі часові рамки та перспективи підтримки і модернізації. У оцінці вартості продукту варто враховувати можливість фінансування та спонсорування проекту, що дозволить гнучко та ефективно підійти до розробки та організації праці.

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

При виконанні робіт із розробки сайту для сервісного центру з обслуговування комп'ютерної техніки з використанням технології JavaScript необхідно працівнику виділити робоче комп'ютеризоване місце середньої важкості IIб відповідно до ДСН 3.3.6.042-99.

Відповідно до НПАОП 0.00-7.15-18 « Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», роботодавець повинен проінформувати працівників під розписку про умови праці та наявність на робочих місцях небезпечних та шкідливих виробничих факторів (фізичних, хімічних, біологічних, психофізіологічних), які виникають під час роботи з екранними пристроями та ще не усунуто, а також про можливі наслідки їх впливу на здоров’я працівників.

До обов'язків роботодавця, відповідно до вимог НПАОП 0.00-7.15-18, також входить забезпечення навчання і перевірки знань працівників з питань охорони праці та безпечного використання екранних пристроїв до початку роботи з ними, а також у випадках модифікації та організації роботи обладнання.

Роботодавець за рахунок тривалості робочої зміни організує внутрішні регламентовані перерви для відпочинку відповідно до Державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПІН 3.3.2.007-98. Робоче місця працівника з екранними пристроями облаштовується таким устаткуванням, яке не створює зайвого шуму та не виділяє надлишкового тепла. Рівні шуму на робочих місцях осіб, які працюють з екранними пристроями, відповідають вимогам Санітарних норм виробничого шуму, ультразвуку та інфразвуку ДСН 3.3.6.037-99. В приміщенні з робочими місцями середня температура повітря становить +20°С, вологість повітря 50%, швидкість повітря 0,2 м/с, тобто вона відповідає п.1.1.1 ДСН 3.3.6.042-99.

Згідно з п.1.2.2 ДСН 3.3.6.042-99, температура внутрішніх поверхонь робочої зони (стіни, підлога, стеля) не виходить більш ніж на 2°С за межі оптимальних величин температури повітря для даної категорії робіт, тобто приміщення відповідає п.1.1.2 ДСН 3.3.6.042-99. В холодний період року в приміщеннях з робочими місцями температура повітря +20°С, відносна вологість 75%, швидкість руху повітря – 0,2 м/с. В теплий період року в приміщеннях з робочими місцями температура повітря +25°С, відносна вологість 70%, швидкість руху повітря – 0,3 м/с. Таким чином робоче місце відповідає п.2.3 ДСН 3.3.6.042-99.

Відповідно до п.2.4 ДСН 3.3.6.042-99, приміщення з робочими місцями з надлишком (явного) тепла використовує природну вентиляцію (аерацію).

Вимірювання параметрів мікроклімату на робочих місцях з розробки сайту для сервісного центру з обслуговування комп'ютерної техніки з використанням технології JavaScript проводяться на висоті 0,5 – 1,0 м від підлоги – при роботі сидячи, 1,5 м від підлоги – при роботі стоячи, таким чином відповідаючи п.3.3 ДСН 3.3.6.042-99.

Відповідно до п.3.6 ДСН 3.3.6.042-99, температура та відносна вологість повітря вимірюються приладами, заснованими на психрометричних принципах. Можливе використання тижневих і добових термографів і гігрографів.

Відповідно до п.3.7 ДСН 3.3.6.042-99, швидкість руху повітря вимірюється анемометрами ротаційної дії. Малі величини швидкості руху повітря (менше 0,3 м/сек.), особливо при наявності різноспрямованих потоків, вимірюються електроанемометрами, циліндричними або кульовими кататермометрами.

Відповідно до п.3.8 ДСН 3.3.6.042-99, температура поверхонь обгороджувальних конструкцій (стін, стелі, підлоги) або обладнань (екранів і т. ін.) вимірюються приладами, що діють за принципом термоелектричного ефекту.

Роботодавець забезпечує за свій рахунок проведення медичних оглядів працівників відповідно до вимог Порядку проведення медичних оглядів працівників певних категорій, затвердженого наказом Міністерства охорони здоров'я України від 21 травня 2007 року № 246.

Згідно з п.2.13 ДСН 3.3.6.042-99, роботодавець забезпечує проходження попередніх (при прийомі на роботу) та періодичних медичних оглядів відповідно з діючим наказом МОЗ України.

Приміщення для роботи з ЕОМ розташоване не у підвальних приміщеннях та не на цокольних поверхах. Тобто, робоче місце відповідає п.2.2 ДСанПіН 3.3.2-007-98.

Відповідно до п.2.3 ДСанПіН 3.3.2-007-98, площа на одне робоче місце в приміщенні, де йде розробка сайту для сервісного центру з обслуговування комп'ютерної техніки з використанням технології JavaScript становить 8 $м^2$, а об’єм – 25 $м^3$.

Природне освітлення здійснюється через світлові прорізи, орієнтовані переважно на північ та північний схід і забезпечують коефіцієнт природної освітленості (КПО) 3%. Тобто, КПО робочого місця відповідає п.2.5 ДСанПіН 3.3.2-007-98.

Згідно з п.2.9 ДСанПіН 3.3.2-007-98, віконні прорізи приміщень для роботи з ВДТ обладнані регульованими пристроями (жалюзі, завіски, зовнішні козирки).

Відповідно до п.2.15 ДСанПіН 3.3.2-007-98, у робочому приміщенні щоденно робиться вологе прибирання.

Згідно з п.2.16 ДСанПіН 3.3.2-007-98, робоче приміщення оснащене аптечками першої медичної допомоги.

Зазначення освітлення освітленості на поверхні робочого столу в зоні розміщення документів становить 400 лк. Тобто, освітленість робочого місця відповідає п.3.2.3 ДСанПіН 3.3.2-007-98.

Конструкція робочого місця забезпечує підтримання оптимальної робочої пози, та відповідає п.4.2 ДСанПіН 3.3.2-007-98.

Розглянуті стандарти охорони праці на робочих місцях розробки сайту для сервісного центру з обслуговування комп'ютерної техніки з використанням технології JavaScript відповідають усім розглянутим основним нормативам для робочих місць даного типу.

4.2. Безпека в надзвичайних ситуаціях

Під час розробки сайту для сервісного центру з обслуговування комп'ютерної техніки з використанням технології JavaScript розробник зіштовхується з багатьма факторами, котрі впливають на його продуктивність під час виконання завдання.

Електричні установки, до яких відноситься практично все обладнання ПК, представляють для людини велику потенційну небезпеку, тому, що в процесі експлуатації або проведенні профілактичних робіт людина може торкнутися частин, що знаходяться під напругою. Будь-який вплив струму може призвести до електричної травми, тобто до пошкодження організму, викликаного дією електричного струму або електричної дуги.

Загальні заходи щодо усунення небезпеки ураження електричним струмом зводяться до правильного розміщення устаткування та електричних дротів. Інші заходи щодо забезпечення електробезпеки, збігаються з загальними заходами пожежо- та електробезпеки.
В якості профілактичних заходів для забезпечення усунення небезпеки займання слід використовувати приховану електромережу, надійні розетки з пожежобезпечних матеріалів, силові мережі живлення устаткування виконувати кабелями, розрахованими на підключення в 3-5 разів більшого навантаження, вмикати і вимикати живлення обладнання за допомогою штатних рубильників. Треба регулярно робити очистку внутрішніх частин комп'ютерів, іншого устаткування від пилу, розташовувати комп'ютери на окремих пожежостійких столах. Для запобігання іскріння необхідно рідше встромляти і виймати штепселі з розеток.

Небезпека ураження електричним струмом існує завжди, якщо є контакт з пристроєм, що живиться напругою 36В і вище, тим більше від електричної мережі 220В. Це може статися через помилку в разі дотику до відкритих струмоведучих частин, але частіше за все через різні причини, такі як перевантаження, не зовсім якісна ізоляція, механічні пошкодження та ін. В процесі експлуатації може погіршитися ізоляція струмоведучих частин, в тому числі і кабелі живлення, в результаті чого вони можуть виявитися під напругою, а випадковий дотик до них загрожує електротравмою, а у важких випадках - загибеллю людини.

Зоною підвищеної небезпеки є місця підключення електроприладів і установок. Нерідко розетки для підключення розташовують на підлозі, що є неприпустимим. Часто відбувається інша помилка - перевантаження розеток за потужністю, і, як наслідок, відбувається порушення ізоляції, що приводить до короткого замикання.

Для запобігання уражень електричним струмом під час роботи з комп'ютером слід встановити додаткові захисні пристрої, що забезпечують недоступність струмопровідних частин для дотику. З метою зменшення небезпеки можна використовувати розділовий трансформатор для розв'язки з основною мережею, і обов'язковим у всіх випадках є наявність захисного заземлення або занулення (захисного відключення) електрообладнання. Для якісної роботи комп'ютерів створюється окремий заземлюючий контур.

Під час роботи з електроустановками поряд з безумовним дотриманням певних організаційних заходів, встановлених правилами технічної експлуатації електроустановок споживачів, слід суворо виконувати всі технічні заходи, що забезпечують безпеку робіт зі зняттям напруги, а саме:

- відключення обладнання на ділянці, виділеній для виконання робіт, і вжиття заходів проти помилкового чи самовільного увімкнення;
- обгородження, за необхідності, робочих місць і залишених під напругою струмоведучих частин;
- вивішування попереджувальних плакатів і знаків безпеки;

- перевірка відсутності напруги;
- накладення заземлення.

При виконанні електромонтажних та ремонтних робіт необхідно також всі види обслуговування ПК проводити одночасно не менше ніж двом фахівцям, щоб у разі електротравми було кому відключити струм і надати першу долікарську допомогу. При цьому налагоджувальник повинен знаходитися на гумовому килимку і перевіряти електричну схему, не торкаючись корпусу і струмоведучих ланцюгів.

Під час ремонту ПК забороняється:

- застосовувати для з'єднання блоків і приладів дроти з пошкодженою ізоляцією;
- здійснювати пайку і установку деталей в обладнанні, що знаходиться під напругою;
- вимірювати напругу і струм переносними приладами з неізольованими дротами і щупами;
- підключати блоки і прилади до обладнання, що знаходиться під напругою;
- замінювати запобіжники при включеному обладнанні;
- працювати на високовольтних установках без захисних засобів.

Для усунення можливої несиметрії напруги в разі аварійної ситуації на інших електроустановках в силовий мережі, для надійного відключення комп'ютерного обладнання від мережі і з метою забезпечення електробезпеки користувача та збереження техніки необхідно виконувати ряд монтажних вимог.

По-перше, всі з'єднання ПК та зовнішнього обладнання повинні проводитися з відключеним електроживленням.

По-друге, всі вузли одного персонального комп'ютера і підключене до нього периферійне устаткування повинні живитися від однієї фази електромережі.

По-третє, корпус системного блоку і зовнішні пристрої повинні заземлюватися окремо на зовнішній контур.

По-четверте, для вимкнення комп'ютерного обладнання має використовуватися окремий щит з автоматами захисту і одним вимикачем.

Людина, одночасно доторкнувшись до незафарбованих металевих частин корпусу комп'ютера і до будь-яких, що мають з'єднання з землею металоконструкцій (наприклад, до батареї опалення), виявиться в ланцюзі струму, який може бути небезпечною для його життя. Ця ж напруга є джерелом різниці потенціалів між пристроями, від якої страждають інтерфейсні схеми.

Якщо з'єднувальні пристрої надійно заземлені через окремий дріт на загальний контур, то проблема різниці потенціалів не виникає. Якщо обидва з'єднувальні пристрої не заземлені, то в разі їх живлення від однієї фази мережі між ними може з'явитися невелика різниця потенціалів, викликана розкидом ємностей конденсаторів в різних фільтрах, однак небезпека для людини в будь-якому випадку залишається. Якщо незаземлені пристрої підключені до різних фаз, то різниця потенціалів зростає і буде вже близько 190В, що загрожує серйозними наслідками для людини. Найбільш важкий випадок - це з'єднання заземленого пристрою з незаземленим, особливо коли останній має потужний блок живлення[31].

Певні проблеми виникають для пристроїв, блоки живлення яких мають шнури з двополюсним штекером і забезпечені мережевим фільтром. У цих фільтрів конденсатори малої ємності, тому струм короткого замикання відносно невеликий - кілька міліампер. Це вирішується використанням мережевих фільтрів, які включаються в триполюсну розетку із заземленням, при цьому вирішується також проблема різниці потенціалів, якщо здійснювати живлення всіх пристроїв, які з'єднуються інтерфейсами, за допомогою одного такого фільтру або їх ланцюгового з’єднання, що пов'язане триполюсними штекерами і розетками.

Для захисту комп'ютерів від неякісного електроживлення (підвищеного або зниженого рівня напруги, провалів і стрибків напруги, відхилення частоти і форми кривої напруги), що є основною причиною збоїв електроніки під час роботи (зависання, помилки при записі або читанні диска), сьогодні застосовують джерела безперебійного живлення (ДБЖ). Їх основне призначення - забезпечення навантаження електроенергією під час аварії в основній мережі. При використанні ДБЖ необхідно, щоб захисний контур (земля) і нейтральний дріт прокладалися окремо. Крім усього іншого, неякісне заземлення знижує захист від електромагнітних завад, що наводяться джерелом на обладнання (монітор). Також, не рекомендується вмикати в ДБЖ лазерні принтери, бо під час розігріву принтера споживаний струм значно перевищує номінальне значення, що може привести до виходу ДБЖ з ладу.

Працюючи за комп'ютером необхідно стежити, щоб руки були сухі, не допускати потрапляння будь-якої рідини на корпус та допоміжні пристрої ПК. Чистити від пилу і витирати увімкнений комп'ютер або монітор категорично заборонено. Не можна класти на комп'ютерну техніку сторонні предмети і закривати вентиляційні отвори.

Дотримання правил і вимог електробезпеки дозволяє максимально забезпечити захист користувача від ураження електричним струмом. Однак, якщо стався нещасний випадок, в першу чергу необхідно будь-яким способом негайно припинити дію струму, для чого треба вимкнути рубильник, відкинути електропровід від потерпілого сухою палицею або чимось подібним і обов'язково викликати лікаря. Якщо потерпілий у свідомості і відчуває деяке нездужання, до приходу лікаря слід забезпечити йому спокій, свіже повітря і тепло.

При важкому стані потерпілого (втрата свідомості, відсутній пульс, дихання переривчасте) необхідно терміново почати робити штучне дихання за способом "з рота в рот" з частотою 12-15 вдувань на хвилину і непрямий масаж серця з частотою одне натискання на секунду і продовжувати ці дії до

покращення стану хворого (діаметр зіниць відновлюється, тобто зменшується до нормального, пульс повертається, дихання нормалізується). Коли людина приходить до тями, треба продовжувати надавати допомогу ще 5-10 хвилин, потім надати йому тепло і давати рясне пиття у вигляді теплого чаю. У будь-якому випадку має бути забезпечено надання кваліфікованої медичної допомоги.

ВИСНОВКИ

Отже, науково-інженерна робота, що була виконана під керівництвом к.т.н, доцента Барана Ігоря Олеговича в Тернопільському національному технічному університеті імені Івана Пулюя успішно завершена. В цьому звіті було викладено процес і результат діяльності інженера програмного забезпечення, з темою магістерської роботи "Розробка сайту для сервісного центру з обслуговування комп'ютерної техніки з використанням технології JavaScript".

Під час виконання магістерської роботи магістра стояло завдання реалізувати статичний інформаційний веб-сайт для сервісного центру і таким чином продемонструвати високий потенціал JAMstack підходу у вирішенні прикладних задач, які виникають при проектуванні веб-застосунків. Для початку, була детально проаналізована предметна область, та вивчені критерії формування характеристик надаваних послуг. Після цього, етап реалізації веб-сайту був розділений на менші логічні блоки, були описані сценарії дій для кожного з варіантів використання, які були виявлені для веб-сайту сервісного центру на етапі аналізу предметної області.

На момент завершення магістерської роботи магістра були виконані практично 100% від всіх виявлених раніше функцій інформаційного статичного веб-сайту. Цього було цілком достатньо щоб наділити сайт базовими функціями. Також було в повній мірі виконане поставлене завдання, щодо створення статичного набору HTML сторінок на основі такого фреймворку JavaScript як GatsbyJS. В результаті виконання комплексу завдань були отримані практичні навички щодо створення веб-сайту для користувацького проекту. Крім цього, це дозволило засвоїти всі набуті протягом навчання навички на прикладі реального проекту.

Що стосується оцінки отриманих результатів, то можна відмітити, що всі здобуті навички в сучасному світі є дуже корисними, оскільки майже все на сьогоднішній день відбувається у веб-мережі. Саме тому вміння

створювати подібні веб-застосунки є дуже корисним для більшості інженерів програмного забезпечення.

Результат виконання поставленого завдання під час проектування та розробки серверного ПЗ відповідає окремій галузі в сфері інформаційних технологій, а саме створенню статичних інформаційних веб-сайтів, що на сьогоднішній день набуває все більшої популярності. Це вміння на даний момент стає все більш затребуваним у представленій сфері, а все тому, в усіх сферах відбувається процес відмови від традиційних систем керування контентом та перевантажених не потрібними для такого проекту функціями фреймворків. Інакше кажучи, завдяки простоті освоєння такої технології, відбувається масовий перехід усього в мережу інтернет, за допомогою генераторів статичних сайтів. На сьогоднішній день, отримані навички під час виконання дипломної роботи магістра користуються великим попитом на ІТ ринку та направлені на всі галузі.

Отже, відповідно до всього вище написаного можна зробити наступний висновок щодо значимості виконання дипломного проекту, який був виконаний протягом інженерного проектування та реалізації статичного інформаційного веб-сайту для сервісного центру. Веб-сайт для будь-якого сервісного центру має дуже велике значення, оскільки саме він надає уявлення про види послуг, які надаються компанією та можливості їх замовлення. Крім того, веб-сайт покращує досвід потенційного клієнта, що призведе до формування гарного враження від сервісного центру, що в свою чергу переконає клієнта знову скористатися його послугами. Саме тому вміння коректно та відповідно до усіх вимог створювати сайти для будь-якої потреби на сьогоднішній дуже цінується роботодавцями в ІТ-галузі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Different types of services for computers by a service center [Електронний ресурс]. – Режим доступу: http://www.tkworld.org/different-types-of-services-for-computers-by-a-service-center/
2. Informational websites [Електронний ресурс]. – Режим доступу: https://www.sideways-designs.com/informational-websites/
3. UML Use Case Diagram: Tutorial with example [Електронний ресурс]. – Режим доступу: https://www.guru99.com/use-case-diagrams-example.html
4. Use case – Learn By Examples [Електронний ресурс]. – Режим доступу: https://medium.com/@warren2lynch/use-case-learn-by-examples-5a63b67fa64d
5. Software development process [Електронний ресурс]. – Режим доступу: https://concisesoftware.com/software-development-process/
6. What is Rapid Application Development? [Електронний ресурс]. – Режим доступу: https://www.outsystems.com/blog/rapid-application-development.html
7. What is Node.js? One Stop Solution for Beginners [Електронний ресурс]. – Режим доступу: https://www.edureka.co/blog/what-is-node-js/
8. What Is JavaScript and How Does It Work? [Електронний ресурс]. – Режим доступу: https://www.makeuseof.com/tag/what-is-javascript/
9. JavaScript - Overview [Електронний ресурс]. – Режим доступу: https://www.tutorialspoint.com/javascript/javascript_overview.htm
10. Useful APIs to know when building a JAMstack app [Електронний ресурс]. – Режим доступу: https://blog.logrocket.com/jamstack-app-apis/
11. What is Netlify and Why Should You Care as an Editor? [Електронний ресурс]. – Режим доступу: https://agilitycms.com/resources/posts/what-is-netlify-and-why-should-you-care-as-an-editor
12. About Contentful [Електронний ресурс]. – Режим доступу: https://www.contentful.com/faq/about-contentful/

13. What is Gatsby.js? [Електронний ресурс]. – Режим доступу: https://www.mediacurrent.com/what-is-gatsby.js
14. Tech 101: What is React JS? [Електронний ресурс]. – Режим доступу: https://skillcrush.com/2019/05/14/what-is-react-js/
15. What is GraphQL? [Електронний ресурс]. – Режим доступу: https://pragmaticstudio.com/tutorials/what-is-graphql
16. What is JAMstack? [Електронний ресурс]. – Режим доступу: https://buttercms.com/blog/what-is-jamstack
17. What is Visual Studio Code? [Електронний ресурс]. – Режим доступу: https://www.educba.com/what-is-visual-studio-code/
18. Git [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Git
19. Modern JAMstack deploymetn? [Електронний ресурс]. – Режим доступу: https://blog.codecentric.de/en/2019/02/modern-jamstack-deployment/
20. React Facebook https://github.com/seeden/react-facebook
21. Adding forms https://www.gatsbyjs.org/docs/adding-forms/
22. How to Use Netlify Forms in a Gatsby Project [Електронний ресурс]. – Режим доступу: https://cobwwweb.com/how-to-use-netlify-forms-with-gatsby
23. Content Management with Gatsby, Contentful & Netlify [Електронний ресурс]. – Режим доступу: https://itnext.io/content-management-with-gatsby-netlify-and-contentful-70f03de41602
24. Biilman M., Hacksworth P. Modern Web Development on the JAMstack / Mathias Biilman., Phil Hacksworth., 2019. – 106 с. – (1-ше). – (978-1-492-05854-0).
25. Unit testing [Електронний ресурс]. – Режим доступу: https://www.gatsbyjs.org/docs/unit-testing/
26. LightHouse Scoring Guide [Електронний ресурс]. – Режим доступу: https://developers.google.com/web/tools/lighthouse/v3/scoring

27. Закон України «Про авторське право і суміжні права» від 23.12.1993 № 3792-XII. – [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/3792-12
28. Закон України «Податковий кодекс України» від 2 грудня 2010 року № 2755-VI. – [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2755-17
29. Закон України «Про внесення змін до Податкового кодексу України та деяких законодавчих актів України щодо податкової реформи» від 28 грудня 2014 року № 71-VIII. – [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/71-19.
30. Методичні вказівки для виконання розділу дипломної роботи щодо техніко-економічного обґрунтування вибору проектного рішення розробки та оцінки якості програмного забезпечення / Упор. Петрик М.Р., Кінах Я.І., Головатий А.І., Рогатинська Л.Р. – Тернопіль: Вид-во ТНТУ ім. І. Пулюя. – 2013. – 34 с.
31. Электробезопасность при работе з ПЭВМ [Електронний ресурс]. – Режим доступу: https://life-prog.ru/1_53829_elektrobezopasnost-pri-rabote-s-pevm.html

ДОДАТКИ

Додаток А - Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

ІМЕНІ ІВАНА ПУЛЮЯ

Кафедра “Програмної інженерії”

ЗАТВЕРДЖУЮ

_____

“___ “_____ 2019 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання дипломної роботи магістра

“ Розробка сайту для сервісного центру з обслуговування комп'ютерної техніки з використанням технології JavaScript”

ПІБД 18. № СПм-18-180. 001 ТЗ

Розробники

Керівник проекту

_____

“ _____” 2019р.

Виконавець

студ. Равчак Б.Ю.

“ _____” 2019р.

Тернопіль 2019

ЗМІСТ

1. ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до календарного плану виконання дипломної роботи магістра на 2019 р. за спеціальністю 121-Інженерія програмного забезпечення.

Тема роботи: “ Розробка сайту для сервісного центру з обслуговування комп'ютерної техніки з використанням технології JavaScript”.

2. ПРИЗНАЧЕННЯ РОЗРОБКИ

Програмний продукт призначений для надання потрібної інформації про послуги в сфері веб-маркетингу. Дане програмне забезпечення буде розроблене без серверної частини, яку замінять елементи технології JAMstack.

Мета розробки полягає в реалізації інформаційного веб-сайту для сервісного центру з обслуговування комп’ютерної техніки для подальшого розгортання на хостингу статичних сайтів.

Даний продукт дозволить продемонструвати швидкодію та простоту розробки статичного інформаційного веб-сайту за допомогою цієї методології на прикладі реального завдання

3. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Функціональні характеристики

Програмний продукт має забезпечити виконання наступних дій:

- Дозволяє відвідувачу переглянути список наявних послуг на сайті;
- Відвідувач має змогу переглянути список категорій, на які поділені всі послуги, що реалізуються на даному сайті;

- Надає можливість клієнтові переглянути інформацію про окрему послугу та її ціну.
- Дозволяє додати обраний товар до кошика, щоб в подальшому оформити покупку даного товару;
- Дозволяє відвідувачеві, за умови авторизації у соціальній мережі, залишити свій відгук
- Адміністратор має змогу обробити інформацію, яку надав відвідувач
- Надає можливість адміністратору сайту модерувати секцію коментарів на предмет невідповідного вмісту
- Дозволяє адміністратору відповідати на відгуки, залишені відвідувачами
- Адміністратор може створювати, видаляти і змінювати вкладки з описом послуг, додавати рекламні слайдери та редагувати прайслист на послуги
- Адміністратор може обмежувати доступ до інформації на веб-сайті

3.2 Склад та параметри технічних засобів

Функціонування програми забезпечується: електронно-обчислювальною машиною з розрядністю процесора x64 або x86, з 4 Гб оперативної пам‘яті, встановленою системою Windows 7/10, та не менш ніж 400 Мб вільного місця на жорсткому диску.

3.3 Інформаційна та програмна сполучність

Програмний продукт повинен коректно функціонувати під керуванням ОС Microsoft Windows 7/10/. Програмний продукт має назву “Веб-сайт

«Service Center»”. і повинен бути написаний мовою JavaScript з використанням фреймворку GatsbyJS.

4. СТАДІЇ РОЗРОБКИ

- - аналіз предметної галузі;
- - вибір інструментів та конструювання програмної системи;
- - розробка прикладного програмного забезпечення;
- - тестування програмного продукту;
- - оформлення супровідної документації;
- - здача продукту.

5. ПРОГРАМНА ДОКУМЕНТАЦІЯ

Для програмного продукту повинні бути розроблені наступні документи:

- Специфікація;

- Технічне завдання;

- Пояснювальна записка;

6. ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Розроблений програмний продукт має виконувати всі вимоги, що складаються з перерахованих у п.3.1 характеристик.

Приймання проводиться спеціально створеною комісією в термін до 26 грудня 2019р.

Додаток Б - Тези

УДК 004.418

Б.Ю. Равчак

Тернопільський національний технічний університет імені Івана Пулюя

ХАРАКТЕРИСТИКА МЕТОДОЛОГІЇ JAMSTACK

Архітектура веб-додатків описує взаємодію між додатками, базами даних та системами проміжного програмного забезпечення в Інтернеті. Як тільки користувач натискає кнопку переходу після введення URL-адреси, сервер надсилає файли в браузер як відповідь на зроблений запит. Потім браузер виконує ці файли, щоб показати потрібну сторінку.[1].

Нинішні веб-сайти, в основному, відносяться до базової моделі мережевого клієнт-серверного програмування з двома загальними елементами:

- сторона сервера. Сюди входить апаратне і програмне забезпечення веб-сервера, а також програмні елементи і вбудовані технології. діапазон цих технологій простягається від простих програм CGI, написаних на Perl, до комплексних багатоланкових додатків на основі PHP. Тут враховуються прикладні технології, наприклад - сервери баз даних, які можуть забезпечувати підтримку веб-сайту.

- сторона клієнта. Сторона клієнта пов'язана з веб-браузером і підтримуваними

ним технологіями, такими як мови HTML, CSS і JavaScript, елементи управління ActiveX і змінні модулі Explorer, які використовуються для створення шаблону сторінки або забезпечення інтерактивних функцій.

На заміну цій моделі прийшла методологія JAMstack. Генератори статичних сайтів стають дуже поширеними, оскільки вони базуються на найсучасніших технологіях та структурах JavaScript, таких як Vue.js або React.

JAMstack - це не технологія. Натомість JAMstack - це новий спосіб створення веб-сайтів та додатків, сучасна архітектура веб-розробки, заснована на стороні клієнта, і не залежить від веб-сервера; статичний HTML-сайт, який автоматично відновлюється щоразу, коли оновлюється вміст, і розгортається безпосередньо на CDN[2].

JavaScript: будь-яке динамічне програмування під час циклу запит / відповідь обробляє JS, повністю працює на клієнті. Це може бути будь-який фронтенд фреймворк чи бібліотека, або навіть натівний JavaScript.

API: усі процеси на сервері або дії з базою даних абстрагуються в API багаторазового використання, доступ до яких здійснюється через HTTPS з JavaScript.

Markup: шаблонна розмітка повинна бути попередньо побудована під час розгортання, як правило, використовуючи генератор веб-сайтів для контентних сайтів або інструмент побудови веб-додатків.

Переваги JAMstack:

- більш швидка та краща продуктивність. Він може генерувати нові сторінки під час розгортання та обслуговувати попередньо розбудовані розмітки та активи над CDN;

- менш дорогі і простіші в масштабі. Менша складність розробки зменшує витрати, а також розміщення статичних файлів є дешевим або навіть безкоштовним;

- вища безпека. З делегуванням операцій на базі сервера та бази даних нам більше не потрібно турбуватися про вразливості;

Література

1. Web-application architecture definition models, types and more – Режим доступу : https://hackr.io/blog/web-application-architecture-definition-models-types-and-more

2. Why the JAMstack is becoming so popular– Режим доступу : https://medium.com/notonlycss/why-the-jamstack-is-becoming-so-popular-a26133b12a30

FaqItem.js

```
import React from 'react';
import Collapsible from 'react-collapsible';

import { Wrapper } from './style';

const FaqItem = ({ title, children }) => (
  <Wrapper>
    <Collapsible
      className="faq"
      openedClassName="faq active"
      triggerClassName="faq-title"
      triggerOpenedClassName="faq-title active"
      triggerTagName="button"
      contentInnerClassName="faq-content"
      trigger={title}
      transitionTime={300}
      easing="ease-out"
    >
      {children}
    </Collapsible>
  </Wrapper>
);

export default FaqItem;
```

Layout.js

```
import React from 'react';
import PropTypes from 'prop-types';
import { ThemeProvider } from 'styled-components';

import SEO from '@common/SEO';

import theme from '@styles/theme';
import GlobalStyles from '@styles/GlobalStyles';

const Layout = ({ children }) => (
  <ThemeProvider theme={theme}>
    <>
      <SEO />
      <GlobalStyles />
      {children}
    </>
  </ThemeProvider>
);

Layout.propTypes = {
  children: PropTypes.node.isRequired,
};
```

```
export default Layout;

Navbar.js
import React, { Component } from 'react';
import AnchorLink from 'react-anchor-link-smooth-scroll';
import Scrollspy from 'react-scrollspy';

import { Container } from '@components/global';
import {
  Nav,
  NavItem,
  Brand,
  StyledContainer,
  NavListWrapper,
  MobileMenu,
  Mobile,
} from './style';

import { ReactComponent as MenuIcon } from '@static/icons/menu.svg';

const NAV_ITEMS = ['Про нас', `Комп'ютерна техніка`, 'Ціни',
'Контакти'];

class Navbar extends Component {
  state = {
    mobileMenuOpen: false,
  };

  toggleMobileMenu = () => {
    this.setState(prevState => ({ mobileMenuOpen:
!prevState.mobileMenuOpen }));
  };

  closeMobileMenu = () => {
    if (this.state.mobileMenuOpen) {
      this.setState({ mobileMenuOpen: false });
    }
  };

  getNavAnchorLink = item => (
    <AnchorLink href={`#${item.toLowerCase()}`}
onClick={this.closeMobileMenu}>
      {item}
    </AnchorLink>
  );

  getNavList = ({ mobile = false }) => (
    <NavListWrapper mobile={mobile}>
      <Scrollspy
        items={NAV_ITEMS.map(item => item.toLowerCase())}
```

```
        currentClassName="active"
        mobile={mobile}
        offset={-64}
      >
        {NAV_ITEMS.map(navItem => (
          <NavItem
key={navItem}>{this.getNavAnchorLink(navItem)}</NavItem>
        ))}
      </Scrollspy>
    </NavListWrapper>
  );

  render() {
    const { mobileMenuOpen } = this.state;

    return (
      <Nav {...this.props}>
        <StyledContainer>
          <Brand>Absurd</Brand>
          <Mobile>
            <button  onClick={this.toggleMobileMenu}  style={{  color:
'black' }}>
              <MenuIcon />
            </button>
          </Mobile>
          <Mobile hide>{this.getNavList({})}</Mobile>
        </StyledContainer>

        <Mobile>
          {mobileMenuOpen && (
            <MobileMenu>
              <Container>{this.getNavList({        mobile:        true
})}</Container>
            </MobileMenu>
          )}
        </Mobile>
      </Nav>
    );
  }
}

export default Navbar;

ExternalLink.js
import React from 'react';

const ExternalLink = ({ href, children, ...other }) => (
  <a href={href} {...other} rel="noreferrer noopener" target="_blank">
    {children}
  </a>
);
```

```
export default ExternalLink;

SEO.js
import React from 'react';
import Helmet from 'react-helmet';

const SEO_DATA = {
  description: 'An absurd gatsby starter',
  title: 'Absurd',
  url: '',
  author: 'ajayns',
  keywords: ['gatsby', 'site', 'absurd'],
  twitter: {
    id: '@ajayns08',
    img: '',
  },
  facebook: {
    id: '',
    img: '',
  },
};

const SEO = () => {
  return (
    <Helmet>
      <meta property="fb:app_id" content={SEO_DATA.facebook.id} />
      <meta property="og:title" content={SEO_DATA.title} />
      <meta property="og:type" content="website" />
      <meta property="og:url" content={SEO_DATA.url} />
      <meta property="og:image" content={SEO_DATA.facebook.img} />
      <meta  property="og:description"  content={SEO_DATA.description}
/>

      <meta name="twitter:card" content="summary_large_image" />
      <meta name="twitter:creator" content="@hackinoutco" />
      <meta name="twitter:site" content="@hackinoutco" />
      <meta name="twitter:title" content={SEO_DATA.title} />
      <meta  name="twitter:description"  content={SEO_DATA.description}
/>
      <meta name="twitter:domain" content={SEO_DATA.url} />
      <meta name="twitter:image:src" content={SEO_DATA.img} />

      <meta name="description" content={SEO_DATA.description} />
      <meta name="keywords" content={SEO_DATA.keywords.join(', ')} />
      <meta name="author" content={SEO_DATA.author} />
      <title>{SEO_DATA.title}</title>
      <html lang="en" />
    </Helmet>
  );
};
```

```
export default SEO;

About.js
import React from 'react';
import styled from 'styled-components';
import { StaticQuery, graphql } from 'gatsby';
import Img from 'gatsby-image';

import { Section, Container } from '@components/global';

const About = () => (
  <StaticQuery
    query={graphql`
      query {
        art_slow: file(
          sourceInstanceName: { eq: "art" }
          name: { eq: "fast" }
        ) {
          childImageSharp {
            fluid(maxWidth: 760) {
              ...GatsbyImageSharpFluid_withWebp_tracedSVG
            }
          }
        }

        art_learn: file(
          sourceInstanceName: { eq: "art" }
          name: { eq: "learn_yourself" }
        ) {
          childImageSharp {
            fluid(maxWidth: 760) {
              ...GatsbyImageSharpFluid_withWebp_tracedSVG
            }
          }
        }

        art_ideas: file(
          sourceInstanceName: { eq: "art" }
          name: { eq: "ideas" }
        ) {
          childImageSharp {
            fluid(maxWidth: 760) {
              ...GatsbyImageSharpFluid_withWebp_tracedSVG
            }
          }
        }
      }
    `}
    render={data => (
      <Section id="про нас">
```

```
        <Container>
          <Grid>
            <div>
              <h2>Speed past the competition</h2>
              <p>
                Gatsby.js builds the fastest possible website. Instead
of
                waiting to generate pages when requested, pre-build
pages and
                lift them into a global cloud of servers — ready to be
delivered
                instantly to your users wherever they are.
              </p>
            </div>
            <Art>
              <Img fluid={data.art_slow.childImageSharp.fluid} />
            </Art>
          </Grid>
          <Grid inverse>
            <Art>
              <Img fluid={data.art_learn.childImageSharp.fluid} />
            </Art>
            <div>
              <h2>Nothing new to learn here</h2>
              <p>
                Enjoy the power of the latest web technologies –
React.js ,
                Webpack , modern JavaScript and CSS and more — all set
up and
                waiting for you to start building.
              </p>
            </div>
          </Grid>
          <Grid>
            <div>
              <h2>Grow and build your ideas</h2>
              <p>
                Waste no more time on tooling and performance. Focus
on the the
                site you want to build and nothing more.
                <br />
                <br />
                Gatsby is fast in every way that matters.
              </p>
            </div>
            <Art>
              <Img fluid={data.art_ideas.childImageSharp.fluid} />
            </Art>
          </Grid>
        </Container>
      </Section>
```

```
    )}
  />
);

const Grid = styled.div`
  display: grid;
  grid-template-columns: 3fr 2fr;
  grid-gap: 40px;
  text-align: right;
  align-items: center;
  justify-items: center;
  margin: 24px 0;

  ${props =>
    props.inverse &&
    `
    text-align: left;
    grid-template-columns: 2fr 3fr;
  `}

  h2 {
    margin-bottom: 16px;
  }

  @media (max-width: ${props => props.theme.screen.md}) {
    grid-template-columns: 1fr;
    text-align: left;
    margin-bottom: 96px;

    &:last-child {
      margin-bottom: 24px;
    }
  }
`;

const Art = styled.figure`
  margin: 0;
  max-width: 380px;
  width: 100%;
`;

export default About;

Contacts.js
import React from 'react';
import styled from 'styled-components';
import { StaticQuery, graphql } from 'gatsby';
import Img from 'gatsby-image';

import { Section, Container } from '@components/global';
```

```
const TEAM = [
const Contacts = () => (
  <StaticQuery
    query={graphql`
      query {
        allFile(filter: { sourceInstanceName: { eq: "team" } }) {
          edges {
            node {
              relativePath
              childImageSharp {
                fluid(maxWidth: 400, maxHeight: 400) {
                  ...GatsbyImageSharpFluid
                }
              }
            }
          }
        }
        art_team: file(
          sourceInstanceName: { eq: "art" }
          name: { eq: "team_work" }
        ) {
          childImageSharp {
            fluid(maxWidth: 1600) {
              ...GatsbyImageSharpFluid_withWebp_tracedSVG
            }
          }
        }
      }
    `}
    render={data => (
      <Section id="контакти">
        <Container style={{ position: 'relative' }}>
          <h1>The Team</h1>
          <TeamGrid>
            {TEAM.map(({ name, image, role }) => {
              const img = data.allFile.edges.find(
                ({ node }) => node.relativePath === image
              ).node;

              return (
                <div key={name}>
                  <Img fluid={img.childImageSharp.fluid} alt={name} />
                  <Title>{name}</Title>
                  <Subtitle>{role}</Subtitle>
                </div>
              );
            })}
          </TeamGrid>
          <Art>
            <Img fluid={data.art_team.childImageSharp.fluid} />
          </Art>
```

```
          <ArtMobile>
            <Img fluid={data.art_team.childImageSharp.fluid} />
          </ArtMobile>
        </Container>
      </Section>
    )}
  />
);

const TeamGrid = styled.div`
  display: grid;
  grid-template-columns: repeat(auto-fill, 200px);
  grid-template-rows: min-content;
  grid-gap: 50px;
  justify-content: space-between;
  width: 60%;
  margin-top: 72px;

  @media (max-width: ${props => props.theme.screen.lg}) {
    justify-content: start;
  }

  @media (max-width: ${props => props.theme.screen.md}) {
    width: 100%;
    grid-template-columns: repeat(auto-fill, minmax(160px, 1fr));
  }

  @media (max-width: ${props => props.theme.screen.xs}) {
    grid-gap: 24px;
  }
`;

const Art = styled.figure`
  width: 800px;
  margin: -80px 0;
  position: absolute;
  top: 0;
  left: 70%;

  @media (max-width: ${props => props.theme.screen.lg}) {
    top: 20%;
  }

  @media (max-width: ${props => props.theme.screen.md}) {
    display: none;
  }
`;

const ArtMobile = styled.figure`
  width: 100%;
  margin: 0;
```

```
  display: none;
  margin-top: 64px;
  margin-bottom: -60%;

  @media (max-width: ${props => props.theme.screen.md}) {
    display: block;
  }
`;

const Title = styled.p`
  margin-top: 16px;
  color: ${props => props.theme.color.black.regular};
`;

const Subtitle = styled.p`
  ${props => props.theme.font_size.small};
  color: ${props => props.theme.color.black.light};
`;

export default Contacts;
```

Header.js

```
import React from 'react';
import styled from 'styled-components';
import { StaticQuery, graphql } from 'gatsby';
import Img from 'gatsby-image';

import { Container } from '@components/global';
import ExternalLink from '@common/ExternalLink';

const Header = () => (
  <StaticQuery
    query={graphql`
      query {
        art_build: file(
          sourceInstanceName: { eq: "art" }
          name: { eq: "build" }
        ) {
          childImageSharp {
            fluid(maxWidth: 1400) {
              ...GatsbyImageSharpFluid_withWebp_tracedSVG
            }
          }
        }
      }
    `}
    render={data => (
      <HeaderWrapper>
        <Container>
          <Grid>
```

```
            <Art>
              <Img fluid={data.art_build.childImageSharp.fluid} />
            </Art>
            <Text>
              <h1>
                Fast in
                <br />
                every way
                <br />
                that matters
              </h1>
              <br />
              <p>
                <StyledExternalLink
href="https://github.com/ajayns/gatsby-absurd">
                  Check out source  &#x2794;
                </StyledExternalLink>
              </p>
            </Text>
          </Grid>
        </Container>
      </HeaderWrapper>
    )}
  />
);

const HeaderWrapper = styled.header`
  background-color: ${props => props.theme.color.primary};
  padding-top: 96px;

  @media (max-width: ${props => props.theme.screen.md}) {
    padding-top: 128px;
  }
`;

const Art = styled.figure`
  width: 100%;
  margin: 0;

  > div {
    width: 120%;
    margin-bottom: -4.5%;

    @media (max-width: ${props => props.theme.screen.md}) {
      width: 100%;
    }
  }
`;

const Grid = styled.div`
  display: grid;
```

```
  grid-template-columns: 1fr 1fr;
  align-items: center;
  grid-gap: 64px;

  @media (max-width: ${props => props.theme.screen.md}) {
    grid-template-columns: 1fr;
    grid-gap: 80px;

    > ${Art} {
      order: 2;
    }
  }
`;

const Text = styled.div`
  justify-self: center;

  @media (max-width: ${props => props.theme.screen.md}) {
    justify-self: start;
  }
`;

const StyledExternalLink = styled(ExternalLink)`
  color: inherit;
  text-decoration: none;

  &:hover {
    color: ${props => props.theme.color.black.regular};
  }
`;

export default Header;

Pricelist.js
import React from 'react';
import styled from 'styled-components';
import { StaticQuery, graphql } from 'gatsby';
import Img from 'gatsby-image';

import { Section, Container } from '@components/global';
import ExternalLink from '@common/ExternalLink';

import { ReactComponent as AirbnbLogo } from
'@images/logos/airbnb.svg';
import { ReactComponent as AppleMusicLogo } from '@images/logos/apple-
music.svg';
import { ReactComponent as CokeLogo } from '@images/logos/coca-
cola.svg';
import { ReactComponent as NodeLogo } from '@images/logos/nodejs.svg';
import { ReactComponent as NikeLogo } from '@images/logos/nike.svg';
```

```
import { ReactComponent as InstagramLogo } from
'@images/logos/instagram.svg';

const LOGOS = [
const UsedBy = () => (
  <StaticQuery
    query={graphql`
      query {
        art_story: file(
          sourceInstanceName: { eq: "art" }
          name: { eq: "tell_story" }
        ) {
          childImageSharp {
            fluid(maxWidth: 1200) {
              ...GatsbyImageSharpFluid_withWebp_tracedSVG
            }
          }
        }
      }
    `}
    render={data => (
      <Section id="ціни" accent>
        <StyledContainer>
          <div>
            <h1>Used by biggest in tech</h1>
            <LogoGrid>
              {LOGOS.map(({ logo, link }) => (
                <ExternalLink key={link} href={link}>
                  {logo()}
                </ExternalLink>
              ))}
            </LogoGrid>
          </div>
          <Art>
            <Img fluid={data.art_story.childImageSharp.fluid} />
          </Art>
        </StyledContainer>
      </Section>
    )}
  />
);

const LogoGrid = styled.div`
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-gap: 64px;
  justify-items: center;
  margin-top: 96px;

  a {
    svg {
```

```
      width: 100%;
    }
  }

  @media (max-width: ${props => props.theme.screen.sm}) {
    grid-template-columns: 1fr;
  }
`;

const StyledContainer = styled(Container)`
  display: flex;
  justify-content: flex-end;
  position: relative;

  @media (max-width: ${props => props.theme.screen.md}) {
    justify-content: center;
  }
`;

const Art = styled.figure`
  width: 600px;
  position: absolute;
  top: -12%;
  right: 50%;

  @media (max-width: ${props => props.theme.screen.lg}) {
    top: 0;
    right: 65%;
    width: 500px;
  }

  @media (max-width: ${props => props.theme.screen.md}) {
    display: none;
  }
`;

export default UsedBy;

Footer.js
import React from 'react';
import styled from 'styled-components';
import { StaticQuery, graphql } from 'gatsby';
import Img from 'gatsby-image';

import { Container } from '@components/global';
import ExternalLink from '@common/ExternalLink';

import GithubIcon from '@static/icons/github.svg';
import InstagramIcon from '@static/icons/instagram.svg';
import TwitterIcon from '@static/icons/twitter.svg';
```

```
const SOCIAL = [
  {
    icon: GithubIcon,
    link: 'https://github.com/ajayns/gatsby-absurd',
  },
  {
    icon: InstagramIcon,
    link: 'https://instagram.com/ajay_ns',
  },
  {
    icon: TwitterIcon,
    link: 'https://twitter.com/ajayns08',
  },
];

const Footer = () => (
  <StaticQuery
    query={graphql`
      query {
        art_pot: file(
          sourceInstanceName: { eq: "art" }
          name: { eq: "customers_pot" }
        ) {
          childImageSharp {
            fluid(maxWidth: 960) {
              ...GatsbyImageSharpFluid_withWebp_tracedSVG
            }
          }
        }
      }
    `}
    render={data => (
      <React.Fragment>
        <Art>
          <Img
            fluid={data.art_pot.childImageSharp.fluid}
            style={{ width: 480, maxWidth: '100%', marginBottom: -16
}}
          />
        </Art>
        <FooterWrapper>
          <StyledContainer>
            <Copyright>
              <h2>Absurd</h2>
              <span>
                Illustrations by
                {` `}
                <ExternalLink
href="https://twitter.com/diana_valeanu">
                  @diana_valeanu
                </ExternalLink>
```

```
              </span>
            </Copyright>
            <SocialIcons>
              {SOCIAL.map(({ icon, link }) => (
                <ExternalLink key={link} href={link}>
                  <img src={icon} alt="link" />
                </ExternalLink>
              ))}
            </SocialIcons>
          </StyledContainer>
        </FooterWrapper>
      </React.Fragment>
    )}
  />
);

const SocialIcons = styled.div`
  display: flex;

  img {
    margin: 0 8px;
    width: 24px;
    height: 24px;
  }

  @media (max-width: ${props => props.theme.screen.sm}) {
    margin-top: 40px;
  }
`;

const FooterWrapper = styled.footer`
  background-color: ${props => props.theme.color.primary};
  padding: 32px 0;
`;

const Copyright = styled.div`
  font-family: ${props => props.theme.font_size.small};
  color: ${props => props.theme.color.black.regular};

  a {
    text-decoration: none;
    color: inherit;
  }
`;
const StyledContainer = styled(Container)`
  display: flex;
  justify-content: space-between;
  align-items: center;

  @media (max-width: ${props => props.theme.screen.sm}) {
    flex-direction: column;
```

```
    text-align: center;
  }
`;

export default Footer
```