МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Центральноукраїнський державний університет
імені Володимира Винниченка
Факультет математики, природничих наук та технологій
Кафедра інформатики, програмування, штучного інтелекту
та технологічної освіти

Кваліфікаційна робота
на правах рукопису

Агеєнко Роман Олександрович

КВАЛІФІКАЦІЙНА РОБОТА

перший (бакалаврський) рівень вищої освіти

на тему: « **Розробка сайту з продажу авто засобами React.js**»

Виконав: студент IV курсу, групи КН20Б
факультету математики, природничих наук
та технологій
спеціальності 122 Комп'ютерні науки
освітня програма Комп'ютерні науки
(Програмування та адміністрування)
форма навчання денна
керівник: Жебка Вікторія Вікторівна, доктор технічних наук, професор, доцент кафедри інформатики, програмування, штучного інтелекту та технологічної освіти
рецензент: Вишнівський Віктор Вікторович, доктор технічних наук, професор, завідувач кафедри комп'ютерних наук, ДУІКТ

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____
(підпис, ініціали та прізвище здобувача вищої освіти)

Кропивницький – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Центральноукраїнський державний університет
імені Володимира Винниченка

Кафедра інформатики, програмування, штучного інтелекту та технологічної освіти

До захисту допустити
Зав. кафедри_____/Чистякова Л.О./
«_____»_____2024р.

Агеєнко Роман Олександрович

КВАЛІФІКАЦІЙНА РОБОТА

перший (бакалаврський) рівень вищої освіти
на тему: «**Розробка сайту з продажу авто засобами React.js»**

Виконав: студент IV курсу, групи КН20Б
факультету математики, природничих наук та технологій
спеціальності 122 Комп'ютерні науки
освітня програма Комп'ютерні науки (Програмування та адміністрування)
форма навчання денна
науковий керівник:
Жебка Вікторія Вікторівна, доктор технічних наук, професор, доцент кафедри інформатики, програмування, штучного інтелекту та технологічної освіти

Кваліфікаційна робота захищена
з оцінкою «_____» балів,
за шкалою ЄКТС_____,
за національною шкалою_____.

Секретар ЕК_____/_____/
«_____»_____20__р.

Кропивницький – 2024

АНОТАЦІЯ

Агеєнко Р.О. Розробка сайту з продажу авто засобами REACT.JS. Кваліфікаційна робота на правах рукопису.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавра за спеціальністю 122 Комп'ютерні науки. Центральноукраїнський державний університет імені Володимира Винниченка, Кропивницький, 2024.

Кваліфікаційна робота присвячена актуальній тематиці створення ресурсу, який забезпечить користувачам зручний доступ до інформації про автомобілі, дозволить легко переглядати та порівнювати різні моделі. Особлива увага у процесі розробки була приділена створенню зручного користувацького інтерфейсу та забезпеченню високої продуктивності роботи вебсайту. Це дозволило забезпечити максимально позитивний досвід для користувачів, що в свою чергу, сприятиме підвищенню рівня продажів та задоволеності клієнтів.

Ключові слова: продаж авто, користувацький інтерфейс, вебсайт, React.js.

ABSTRACT

Ageyenko R.O. Development of a car sales site using REACT.JS. - Qualification work on manuscript rights.

Qualification work for obtaining a bachelor's degree in the specialty 122 Computer Science. Central Ukrainian State University named after Volodymyr Vinnichenko, Kropyvnytskyi, 2024.

The qualification work is devoted to the current topic of creating a resource that will provide users with convenient access to information about cars, and will allow easy viewing and comparison of different models. During the development process, special attention was paid to creating a convenient user interface and ensuring high performance of the website. This made it possible to provide the most positive experience for users, which in turn will contribute to increasing the level of sales and customer satisfaction.

Keywords: car sales, user interface, website, React.js.

ЗМІСТ

ВСТУП

Розвиток інформаційних технологій значно впливає на всі аспекти сучасного життя, включаючи комерційну діяльність. Однією з ключових тенденцій у бізнесі є перехід до електронної комерції. В умовах швидкого зростання ринку та високої конкуренції створення ефективних і зручних вебресурсів для продажу товарів та послуг стає важливою потребою.

Автомобільна індустрія також активно інтегрує новітні технології, щоб відповідати вимогам сучасного споживача. Сучасний покупець прагне отримати якомога більше інформації про цікавий йому товар з мінімальними затратами часу та зусиль. У зв'язку з цим створення зручного, функціонального та естетично привабливого вебсайту для продажу автомобілів є актуальним завданням.

Метою даної дипломної роботи є розробка вебсайту для продажу автомобілів з використанням сучасних технологій. Це передбачає створення ресурсу, який забезпечить користувачам зручний доступ до інформації про автомобілі, дозволить легко переглядати та порівнювати різні моделі.

Особлива увага у процесі розробки буде приділена створенню зручного користувацького інтерфейсу та забезпеченню високої продуктивності роботи вебсайту. Це дозволить забезпечити максимально позитивний досвід для користувачів, що, своєю чергою, сприятиме підвищенню рівня продажів та задоволеності клієнтів.

РОЗДІЛ 1. ЗАГАЛЬНИЙ ОПИС ПРОЄКТУ

1.1. Опис задач, які вирішує вебсайт

Вебсайт компанії CarS, з використанням React.js, покликаний вирішити низку важливих задач, що забезпечують ефективну взаємодію між компанією та її клієнтами, полегшують процес ознайомлення з наявними автомобілями та сприяють залученню нових клієнтів. Основною метою даного вебсайту є створення зручної та інтуїтивно зрозумілої платформи, яка дозволяє потенційним клієнтам отримати всю необхідну інформацію про автомобілі, представлені компанією CarS, а також порівняти різні марки та моделі, ознайомитися з відгуками інших клієнтів і, в кінцевому результаті, прийняти обґрунтоване рішення щодо купівлі.

Однією з ключових задач, яку вирішує вебсайт, є надання детальної інформації про наявні автомобілі. Кожен автомобіль на вебсайті має власну сторінку з докладним описом характеристик, фотографіями високої якості, інформацією про комплектацію, доступні кольори та інші важливі деталі. Це дозволяє клієнтам отримати повне уявлення про автомобіль, який їх цікавить, не виходячи з дому. Крім того, вебсайт надає можливість переглядати автомобілі у форматі 360 градусів, що сприяє кращому візуальному сприйняттю.

Ще однією важливою задачею є можливість порівняння різних марок та моделей автомобілів. Сайт пропонує зручний інструмент для порівняння, який дозволяє користувачам обрати кілька автомобілів та переглянути їхні характеристики поруч. Це допомагає потенційним клієнтам швидко та легко порівнювати важливі параметри, такі як ціна, потужність двигуна, витрата пального, оснащення та інші характеристики, що є важливими при виборі автомобіля.

Залучення нових клієнтів є ще однією важливою задачею. Для цього використовуються різноманітні маркетингові інструменти, такі як SEO-оптимізація, інтеграція з соціальними мережами, спеціальні пропозиції та акції. Вебсайт також містить розділ з відгуками клієнтів, де потенційні покупці можуть ознайомитися з досвідом інших клієнтів, що сприяє підвищенню довіри до компанії та її продукції.

Крім того, вебсайт надає можливість клієнтам залишати свої відгуки та оцінки, що дозволяє компанії отримувати зворотний зв’язок і постійно вдосконалювати свої послуги. Важливим аспектом є також наявність блогу, де публікуються новини, статті та поради щодо вибору та експлуатації автомобілів, що допомагає залучати більше відвідувачів та утримувати їхню увагу.

1.2. Актуальність теми

У сучасному цифровому світі вебсайти відіграють ключову роль у сфері автоторгівлі, забезпечуючи компаніям ефективний спосіб залучення клієнтів та підтримання конкурентоспроможності. Актуальність вебсайтів у цій галузі зумовлена кількома основними факторами, які сприяють зростанню попиту на онлайн-платформи для продажу автомобілів.

Завдяки інтернету, компанії можуть представити свої автомобілі потенційним клієнтам з різних регіонів, що значно розширює їхній ринок. Наприклад, велика кількість автодилерів, таких як CarMax і AutoTrader, використовують свої вебсайти для демонстрації асортименту автомобілів, залучення покупців та проведення маркетингових кампаній. Вебсайти дозволяють клієнтам швидко знайти необхідний автомобіль, порівняти його з іншими моделями та прийняти рішення щодо купівлі.

Завдяки інтеграції таких функцій, як онлайн-консультації, чати з представниками компанії та інтерактивні форми зворотного зв'язку, автодилери можуть оперативно реагувати на запити клієнтів, що підвищує рівень задоволеності та лояльності. Наприклад, вебсайт компанії Carvana надає

можливість клієнтам спілкуватися з консультантами в режимі реального часу, отримуючи відповіді на всі запитання щодо вибору та придбання автомобілів.

Розміщення на сайті детальної інформації про кожен автомобіль, включаючи фотографії, технічні характеристики, історію обслуговування та відгуки клієнтів, допомагає потенційним покупцям приймати обґрунтовані рішення. Це також знижує ризик шахрайства та підвищує довіру до продавця. Наприклад, компанія TrueCar на своєму вебсайті публікує реальні ціни на автомобілі, надані іншими покупцями, що забезпечує прозорість процесу купівлі.

Завдяки інтеграції з CRM-системами, платіжними сервісами та іншими інструментами, компанії можуть автоматизувати багато аспектів своєї діяльності, що знижує витрати та підвищує ефективність. Наприклад, Tesla використовує свій вебсайт для автоматизації процесу замовлення автомобілів, включаючи вибір конфігурації, оформлення замовлення та оплати.

Вебсайти забезпечують аналітику та моніторинг поведінки користувачів. Використання інструментів вебаналітики дозволяє компаніям відстежувати, як користувачі взаємодіють з їхнім сайтом, які сторінки вони відвідують, які автомобілі викликають найбільший інтерес. Це дозволяє коригувати маркетингові стратегії, покращувати контент та підвищувати конверсію. Наприклад, компанія Edmunds використовує аналітичні інструменти для збору даних про поведінку відвідувачів свого сайту та оптимізації маркетингових кампаній.

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ДО СИСТЕМИ

2.1. Функціональні вимоги

Функціональні вимоги є критично важливою складовою процесу розробки програмного забезпечення, оскільки вони визначають основні функції та можливості, які повинна забезпечувати система для задоволення потреб

користувачів і замовників. У випадку вебсайту компанії CarS, що призначений для продажу автомобілів, функціональні вимоги мають охоплювати широкий спектр можливостей, необхідних для ефективної взаємодії з клієнтами та забезпечення високої якості послуг.

Важливим є функціональність пошуку та фільтрації. Вебсайт повинен надавати користувачам можливість швидко і легко знаходити потрібні автомобілі за допомогою розширеного пошуку та фільтрів. Фільтри мають охоплювати різноманітні параметри, такі як марка, модель, рік випуску, ціна, пробіг, тип кузова, колір та інші характеристики. Крім того, система пошуку повинна підтримувати автозаповнення та пропонувати релевантні результати під час введення запиту.

Другою функціональною вимогою є реалізація можливості порівняння автомобілів. Користувачі повинні мати змогу обирати кілька автомобілів та порівнювати їхні характеристики на окремій сторінці. Порівняння має бути зручним і наочним, з відображенням усіх ключових параметрів автомобілів у вигляді таблиці або діаграми, що дозволяє легко оцінити їхні відмінності та переваги.

Вебсайт повинен надавати можливість користувачам залишати відгуки про автомобілі та виставляти їм оцінки. Система відгуків має бути модерованою, з можливістю фільтрації та видалення некоректних або спамних повідомлень. Відгуки повинні відображатися на сторінці кожного автомобіля, а також можуть бути згруповані за різними критеріями (наприклад, за датою, рейтингом).

Четвертою функціональною вимогою є реалізація функціонала облікових записів користувачів. Система повинна забезпечувати реєстрацію та аутентифікацію користувачів, зберігання їхніх персональних даних та історії переглядів. Користувачі повинні мати можливість створювати власні списки обраних автомобілів, залишати відгуки, зв'язуватися з продавцями та отримувати персоналізовані пропозиції на основі їхніх вподобань.

Сьомою вимогою є функціональність зв'язку з клієнтами. Вебсайт повинен надавати можливість потенційним покупцям зв'язуватися з представниками компанії через різні канали, такі як форми зворотного зв'язку, онлайн-чати, електронна пошта та телефон. Система повинна підтримувати збереження історії комунікацій та автоматичне створення заявок для подальшої обробки.

Тобто, функціональні вимоги до вебсайту компанії CarS визначають основні можливості та сервіси, які необхідно реалізувати для забезпечення ефективної роботи системи та задоволення потреб користувачів. Вони охоплюють широкий спектр функцій, що дозволяє створити зручну та багатофункціональну платформу для продажу автомобілів.

2.2. Нефункціональні вимоги

Нефункціональні вимоги є важливими для забезпечення якісної та стабільної роботи вебсайту. Вони визначають такі аспекти системи, як продуктивність, безпека та масштабованість, що суттєво впливають на користувацький досвід та загальну ефективність вебсайту.

Продуктивність є одним з ключових аспектів нефункціональних вимог. Вебсайт компанії CarS повинен забезпечувати високу швидкість завантаження сторінок та швидкий відгук на дії користувачів. Це включає оптимізацію часу завантаження основних компонентів, таких як HTML, CSS, JavaScript та зображення, а також ефективне управління базою даних та серверними запитами. Вебсайт має підтримувати високу продуктивність навіть під час пікових навантажень, коли кількість одночасних користувачів може значно збільшуватися. Для досягнення цього можуть бути використані різні техніки оптимізації, такі як кешування, стиснення даних, асинхронне завантаження ресурсів та балансування навантаження.

Безпека вебсайту є ще однією важливою нефункціональною вимогою. Оскільки вебсайт обробляє особисті дані користувачів та навіть здійснює фінансові транзакції, він повинен забезпечувати високий рівень захисту від різних типів загроз. Це включає захист від атак, таких як SQL-ін'єкції, міжсайтові скрипти (XSS), атаки типу "людина посередині" (MITM) та атаки розподіленої відмови в обслуговуванні (DDoS). Вебсайт повинен використовувати протоколи безпеки, такі як HTTPS для шифрування даних під час передачі, а також забезпечувати безпечне зберігання даних за допомогою методів хешування та шифрування. Крім того, система повинна підтримувати регулярні оновлення безпеки та аудит безпеки для виявлення та усунення потенційних вразливостей.

Масштабованість вебсайту визначає його здатність підтримувати зростання кількості користувачів та обсягу даних без зниження продуктивності або якості обслуговування. Сайт має бути спроєктований таким чином, щоб легко адаптуватися до зростання бізнесу та змін у вимогах. Це включає використання архітектурних рішень, які дозволяють легко додавати нові функції та модулі, а також забезпечують гнучкість у розподілі ресурсів. Можливе використання хмарних технологій, таких як Amazon Web Services (AWS) або Microsoft Azure, може значно підвищити масштабованість, забезпечуючи автоматичне масштабування ресурсів у відповідь на збільшення навантаження. Також важливим є впровадження мікросервісної архітектури, яка дозволяє розподілити функціональність на незалежні сервіси, що можуть бути розгорнуті та масштабовані окремо.

Таким чином, нефункціональні вимоги є неодмінною частиною розробки, оскільки вони визначають його продуктивність, безпеку та масштабованість. Забезпечення високих стандартів у цих аспектах сприяє створенню надійної, безпечної та ефективної платформи для продажу автомобілів, що відповідає потребам користувачів та забезпечує стійкий розвиток бізнесу.

2.3. Вимоги користувачів

Вимоги користувачів забезпечують орієнтацію системи на кінцевого користувача та максимальне задоволення його потреб і очікувань. У цьому контексті важливо врахувати різноманітні аспекти, пов’язані з функціональністю, зручністю використання, доступністю та ефективністю взаємодії з вебсайтом.

Однією з основних вимог користувачів є інтуїтивно зрозумілий та зручний інтерфейс. Вебсайт повинен мати привабливий дизайн, який відповідає сучасним тенденціям вебдизайну та забезпечує легку навігацію. Користувачі очікують, що всі основні функції, такі як пошук автомобілів, фільтрація результатів, порівняння моделей та перегляд детальної інформації, будуть доступні з кількох кліків. Наприклад, головне меню має бути чітко структурованим, а важливі функції – легко доступними з головної сторінки.

Ще однією важливою вимогою є адаптивність дизайну. Вебсайт повинен коректно відображатися на різних пристроях, включаючи десктопи, планшети та смартфони. Це забезпечує комфортний користувацький досвід незалежно від того, який пристрій використовується. Використання технології responsive design дозволяє автоматично підлаштовувати макет сайту під розмір екрана пристрою, що є важливим для сучасних користувачів, які часто переглядають сайти з мобільних пристроїв.

Користувачі також очікують швидкого доступу до необхідної інформації. Сайт має забезпечувати високу швидкість завантаження сторінок, що особливо важливо для сторінок з великою кількістю зображень та мультимедійного контенту. Кешування, оптимізація зображень та використання CDN (Content Delivery Network) можуть значно покращити продуктивність та зменшити час завантаження.

Персоналізація є ще однією важливою вимогою. Користувачі очікують, що вебсайт буде адаптуватися під їхні індивідуальні потреби та надавати персоналізовані рекомендації. Наприклад, на основі історії переглядів та

пошукових запитів система може пропонувати автомобілі, які найбільше відповідають інтересам користувача. Це теоретично може бути реалізовано за допомогою алгоритмів машинного навчання, які аналізують поведінку користувачів та створюють персоналізовані пропозиції.

Безпека та конфіденційність даних є критичними аспектами для користувачів. Сайт забезпечує захист персональних даних користувачів, використовуючи сучасні методи шифрування та безпеки. Крім того, користувачі повинні мати можливість легко керувати своїми даними, зокрема змінювати особисту інформацію, налаштування облікового запису та видаляти обліковий запис за потреби.

Зручність комунікації з представниками компанії також є важливою вимогою. Користувачі очікують, що зможуть легко зв'язатися з представниками компанії для отримання консультацій, відповіді на запитання або вирішення проблем. Вебсайт має підтримувати різні канали комунікації, такі як онлайн-чати, електронна пошта та телефони, забезпечуючи швидку та ефективну взаємодію.

Таким чином, врахування вимог користувачів є ключовим фактором у процесі розробки вебсайту компанії CarS. Забезпечення зручності використання, адаптивності, швидкого доступу до інформації, персоналізації, безпеки та зручності комунікації сприяє створенню високоякісного продукту, який відповідає очікуванням користувачів та підвищує їхню задоволеність.

РОЗДІЛ 3. ІНСТРУМЕНТИ ДЛЯ РОЗРОБКИ

3.1. Інтегроване середовище розробки Visual Studio Code

Visual Studio Code (VS Code) є одним з найбільш популярних інструментів для розробки сучасних сайтів, включаючи ті, що побудовані зі застосуванням React.js. Цей редактор коду, розроблений компанією Microsoft,

пропонує широкий спектр можливостей, які значно полегшують процес створення та підтримки програмного забезпечення.

Однією з ключових особливостей Visual Studio Code є його інтеграція з системами контролю версій, такими як Git, що дозволяє розробникам ефективно керувати своїм кодом, відстежувати зміни та співпрацювати з іншими членами команди. Інтерфейс користувача VS Code простий у використанні, зручний і налаштовується відповідно до індивідуальних потреб розробника. Завдяки великій кількості доступних плагінів та розширень, Visual Studio Code може бути налаштований для роботи з будь-яким стеком технологій, включаючи JavaScript, TypeScript та React.js.

Серед інших важливих функцій VS Code слід відзначити вбудований термінал, дебагер, та підтримку інтегрованого середовища розробки (IDE). Вбудований термінал дозволяє розробникам виконувати команди прямо з редактора, що значно скорочує час, необхідний для виконання типових задач розробки. Дебагер дозволяє відстежувати виконання коду в режимі реального часу, що допомагає швидко знаходити та виправляти помилки. Підтримка інтегрованого середовища розробки означає, що VS Code може використовуватися для всіх етапів розробки — від написання коду до його тестування та деплойменту.

Для розробки сторінок на основі React.js, Visual Studio Code пропонує ряд спеціалізованих розширень, таких як ESLint для статичного аналізу коду, Prettier для автоматичного форматування коду, Copilot та інші. Ці розширення допомагають підтримувати високу якість коду, дотримуватися стандартів розробки, що є надзвичайно важливим при роботі в команді або на великих проєктах**.**

3.2. Сервіс для хостингу та керування проєктами програмного забезпечення GitHub

GitHub є провідною платформою для розміщення та спільної роботи над програмним кодом, що дозволяє розробникам з усього світу ефективно співпрацювати та обмінюватися своїми проєктами. Основою GitHub є система контролю версій Git, яка дозволяє відстежувати всі зміни в коді, зберігаючи історію всіх змін та полегшуючи злиття різних гілок розробки. Це особливо корисно при роботі в команді, оскільки забезпечує прозорість та контроль над процесом розробки, дозволяючи кожному учаснику бачити, хто і які зміни вніс.

Один з основних інструментів GitHub — це репозиторії, які слугують місцем для зберігання всіх файлів проєкту та історії їх змін. Кожен репозиторій може містити кілька гілок, що дозволяє розробникам працювати над різними версіями коду паралельно, не заважаючи один одному. GitHub також пропонує зручні інструменти для управління задачами та відстеження помилок, такі як Issues та Projects, які допомагають організувати процес розробки та забезпечити, щоб всі задачі були виконані вчасно.

GitHub Pages є сервісом, що надається GitHub для розміщення статичних вебсайтів безпосередньо з репозиторіїв GitHub. Цей сервіс є ідеальним для хостингу документації проєкту, особистих блогів або портфоліо, а також для розміщення сторінок, створених за допомогою React.js. Використовуючи GitHub Pages, розробники можуть легко деплоїти свої проєкти в інтернет, не витрачаючи час на налаштування серверів або платформи хостингу.

Для створення сайту на GitHub Pages достатньо мати репозиторій з необхідними файлами та налаштувати його відповідним чином. GitHub автоматично обробляє файли та розміщує їх на сервері, роблячи сайт доступним за унікальною URL-адресою. Це дозволяє швидко і безплатно публікувати та оновлювати сайти, що є значною перевагою для розробників, особливо при створенні та тестуванні прототипів.

3.3. Валідація W3C Markup Validation Service та CSS Validation Service

Валідація W3C (World Wide Web Consortium) є процесом перевірки вебдокументів на відповідність міжнародним вебстандартам. Ці стандарти, розроблені W3C, забезпечують сумісність та коректне відображення вебсторінок у різних браузерах та на різних пристроях. Валідація вебсторінок є важливою складовою розробки якісного та доступного вебконтенту.

Основною метою є виявлення помилок у коді HTML, CSS та інших вебтехнологій. Це дозволяє виправити ці помилки до того, як сайт стане доступним для користувачів, що значно покращує користувацький досвід. Крім того, дотримання стандартів W3C забезпечує сумісність вебсайтів з сучасними вебтехнологіями та майбутніми оновленнями браузерів.

Інструменти для валідації, такі як W3C Markup Validation Service та CSS Validation Service, надають розробникам можливість автоматично перевіряти свої вебдокументи. Ці інструменти аналізують код і видають звіти про помилки та попередження, що допомагає швидко ідентифікувати проблемні місця. Валідація також може включати перевірку наявності обов’язкових атрибутів, коректне вкладення елементів та відповідність синтаксичним правилам.

Для розробки за допомогою React.js, валідація W3C є не менш важливою. Хоча React.js генерує HTML-код динамічно, важливо переконатися, що результат відповідає стандартам. Це забезпечує коректне відображення компонентів та уникнення потенційних проблем з доступністю та сумісністю. Інтеграція валідації W3C у процес розробки допомагає підтримувати високу якість коду та сприяє створенню стабільних і надійних сторінок.

Застосування W3C також має позитивний вплив на пошукову оптимізацію (SEO), оскільки пошукові системи надають перевагу сайтам, які відповідають стандартам веброзробки. Це підвищує видимість сайту у пошукових результатах та покращує його рейтинги.

3.4. Платформа Dribbble та переваги використання готових вебшаблонів

Використання готових шаблонів має численні переваги, що дозволяє розробникам значно скоротити час на створення дизайну та структури вебсторінок, зосередившись на функціональності та особливостях проєкту.

Однією з основних переваг використання готових шаблонів є економія часу. Шаблони надають готові до використання дизайни та структури, що дозволяє швидко розпочати роботу над проєктом без потреби витрачати час на створення базового макета з нуля. Це особливо корисно в умовах обмежених термінів або при роботі над кількома проєктами одночасно.

Крім того, готові шаблони часто створюються професійними дизайнерами, що забезпечує високу якість та естетичну привабливість. Використання таких шаблонів дозволяє досягти сучасного та професійного вигляду сайту, що є важливим для залучення користувачів та створення позитивного першого враження. Шаблони також забезпечують адаптивний дизайн, що гарантує коректне відображення вебсторінок на різних пристроях і екранах.

Ще однією значною перевагою є відповідність сучасним вебстандартам. Готові шаблони, як правило, розробляються з урахуванням найкращих практик та стандартів вебдизайну, що сприяє сумісності та стабільності. Це дозволяє уникнути багатьох потенційних проблем, пов'язаних з відображенням і функціональністю сайту на різних платформах і браузерах.

Одним з відомих ресурсів, де можна знайти вебшаблони, є Dribbble.com. Це платформа, що об'єднує дизайнерів з усього світу, на якій вони діляться своїми роботами, включаючи вебшаблони.

РОЗДІЛ 4. ПРОЄКТУВАННЯ СИСТЕМИ

4.1. Архітектура системи, клієнт-серверна модель, мікросервіси

Архітектура системи є фундаментальною складовою процесу розробки вебсайту умовної компанії що займається торгівлею автомобілей, оскільки вона визначає структуру та взаємодію між різними компонентами системи. Вибір архітектурного підходу впливає на продуктивність, масштабованість, надійність та гнучкість системи. У даному контексті розглядається клієнт-серверна модель та мікросервісна архітектура як основні підходи до побудови архітектури системи.

Клієнт-серверна модель є класичною та широко застосовуваною архітектурою в розробці сайтів. У цій моделі система розділена на дві основні частини: клієнтську та серверну. Клієнтська частина, що зазвичай реалізується на основі технологій HTML, CSS та JavaScript (у нашому випадку з використанням React.js), відповідає за інтерфейс користувача та взаємодію з користувачем. Серверна частина, реалізована на базі серверних технологій (наприклад, Node.js та Express.js), обробляє запити від клієнта, виконує бізнес-логіку, взаємодіє з базою даних та повертає відповіді клієнту.

У клієнт-серверній моделі клієнт відправляє запити до сервера через протокол HTTP/HTTPS, а сервер обробляє ці запити та повертає відповідні відповіді. Така архітектура забезпечує чіткий розподіл відповідальності між клієнтом і сервером, що спрощує розробку, тестування та підтримку системи. Крім того, клієнт-серверна модель дозволяє легко масштабувати систему, додаючи додаткові серверні ресурси у разі збільшення навантаження.

Мікросервісна архітектура є сучасним підходом до побудови великих та складних систем, що дозволяє забезпечити високу масштабованість, гнучкість та надійність. У мікросервісній архітектурі система розбивається на невеликі, незалежні сервіси, кожен з яких відповідає за конкретну функціональність або бізнес-логіку. Ці сервіси взаємодіють між собою через чітко визначені

інтерфейси (API), зазвичай за допомогою протоколів HTTP/HTTPS або інших методів взаємодії, таких як повідомлення (message queues).

Кожен мікросервіс у такій архітектурі може розгортатися, оновлюватися та масштабуватися незалежно від інших, що забезпечує високу гнучкість і стійкість системи. Наприклад, вебсайт компанії CarS має складатися з окремих мікросервісів для управління користувачами, обробки замовлень, управління контентом, обробки (теоретично) платежів та аналітики.

Переваги мікросервісної архітектури включають підвищену стійкість до збоїв, оскільки збій одного мікросервісу не призводить до зупинки всієї системи, а також можливість використання різних технологій та мов програмування для різних мікросервісів, що дозволяє обирати оптимальні інструменти для кожної конкретної задачі. Крім того, мікросервіси полегшують процес безперервної інтеграції та розгортання (CI/CD), що забезпечує швидке впровадження нових функцій та виправлення помилок.

Таким чином, архітектура системи вебсайту компанії CarS, заснована на клієнт-серверній моделі та мікросервісній архітектурі, забезпечує високу продуктивність, гнучкість, масштабованість та надійність. Ці підходи дозволяють створити сучасну та ефективну платформу для продажу автомобілів, яка відповідає потребам користувачів та бізнесу.

4.2. Діаграми ERD, діаграми активностей, послідовностей, компонентів

Діаграми є важливою частиною процесу проєктування та документування системи. Вони допомагають візуалізувати структуру, поведінку та взаємодію різних компонентів системи, що сприяє кращому розумінню її функціонування та спрощує процес розробки. У даному розділі розглянемо ключові типи діаграм, які використовуються для моделювання вебсайту умовною компанії CarS: діаграми сутність-зв'язок (ERD), діаграми активностей, діаграми послідовностей та діаграми компонентів.

Діаграма сутність-зв'язок (ERD) використовується для моделювання структури бази даних системи. Вона показує сутності (таблиці) та зв'язки між ними, визначаючи основні атрибути кожної сутності та типи зв'язків (один до одного, один до багатьох, багато до багатьох). Для цього вебсайту ERD може включати сутності, такі як "Автомобіль", "Користувач", "Замовлення", "Відгук" та "Платіж".

Розглянемо приклад коду, що визначає ці сутності в базі даних за допомогою ORM (Object-Relational Mapping), такого як Mongoose для MongoDB:

```
// Визначення схеми для entity "Автомобіль"
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const CarSchema = new Schema({
  id: { type: Schema.Types.ObjectId, auto: true },
  brand: { type: String, required: true },
  model: { type: String, required: true },
  year: { type: Number, required: true },
  price: { type: Number, required: true },
  description: { type: String }
});

const Car = mongoose.model('Car', CarSchema);

// Визначення схеми для entity "Користувач"
const UserSchema = new Schema({
  id: { type: Schema.Types.ObjectId, auto: true },
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
```

```
  password: { type: String, required: true },
  role: { type: String, enum: ['customer', 'admin'], default: 'customer' }
});

const User = mongoose.model('User', UserSchema);
```

Цей приклад коду показує, як можна визначити сутності "Автомобіль" та "Користувач" у базі даних за допомогою Mongoose. Схема для автомобіля включає атрибути "brand" (марка), "model" (модель), "year" (рік випуску), "price" (ціна) та "description" (опис). Схема для користувача включає атрибути "name" (ім'я), "email" (електронна пошта), "password" (пароль) та "role" (роль).

Діаграма активностей використовується для моделювання бізнес-процесів та логіки виконання задач у системі. Вона показує послідовність дій та їхні можливі варіанти, визначаючи початок, кінець та проміжні стани процесу. Для вебсайту компанії CarS діаграма активностей може описувати процес покупки автомобіля, включаючи такі дії, як перегляд автомобілів, вибір конкретної моделі, додавання її до кошика, заповнення форми замовлення, обробка платежу та підтвердження замовлення. Кожен з цих етапів може бути деталізований з урахуванням можливих варіантів (наприклад, успішна обробка платежу або відмова в оплаті).

Діаграма послідовностей використовується для моделювання взаємодії між різними компонентами системи під час виконання конкретного сценарію. Вона показує об'єкти, які беруть участь у сценарії, та повідомлення, які вони обмінюються між собою. Для вебсайту компанії CarS діаграма послідовностей може описувати сценарій реєстрації нового користувача. Цей сценарій включає взаємодію між клієнтським інтерфейсом (React.js), сервером (Node.js/Express), базою даних та службою електронної пошти для підтвердження реєстрації. Діаграма покаже, як клієнт надсилає запит на реєстрацію, сервер обробляє цей запит, зберігає дані в базі даних та надсилає підтвердження електронною поштою.

Діаграма компонентів використовується для моделювання фізичної архітектури системи, показуючи, як різні частини програмного забезпечення взаємодіють між собою. Вона описує основні компоненти системи та їхні зв'язки, що допомагає зрозуміти структуру додатка на високому рівні. Для вебсайту компанії CarS діаграма компонентів може включати такі компоненти, як клієнтський інтерфейс (React.js), серверний додаток (Node.js/Express), база даних (MongoDB), служби автентифікації, платіжні шлюзи та зовнішні API для отримання додаткових даних (наприклад, інформації про моделі автомобілів або поточні курси валют).

Таким чином, діаграми ERD, активностей, послідовностей та компонентів є важливими інструментами для візуалізації та документування архітектури та функціонування вебсайту. Вони допомагають розробникам та зацікавленим сторонам краще зрозуміти систему, спрощують процеси аналізу, проєктування та впровадження, а також сприяють ефективній комунікації між учасниками проєкту.

4.3. Опис основних модулів та їх взаємодії

Вебсайт компанії CarS складається з кількох основних модулів, кожен з яких виконує специфічні функції та взаємодіє з іншими модулями для забезпечення загальної функціональності системи. Основними модулями вебсайту є модуль користувачів, модуль автомобілів, модуль замовлень, модуль відгуків та модуль платежів. Кожен з цих модулів реалізовано з використанням сучасних технологій, таких як Node.js, Express.js та MongoDB, та взаємодіє через чітко визначені інтерфейси API.

Модуль користувачів відповідає за управління обліковими записами користувачів, їхньою автентифікацією та авторизацією. Він дозволяє користувачам реєструватися, входити в систему, змінювати особисту інформацію та керувати своїм профілем. Нижче наведено приклад коду для реалізації основних функцій цього модуля.

```
// userModel.js
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const UserSchema = new Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  role: { type: String, enum: ['customer', 'admin'], default: 'customer' }
});

module.exports = mongoose.model('User', UserSchema);

// userController.js
const User = require('../models/userModel');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');

exports.register = async (req, res) => {
  const { name, email, password } = req.body;
  const hashedPassword = await bcrypt.hash(password, 10);
  const newUser = new User({ name, email, password: hashedPassword });
  await newUser.save();
  res.status(201).json({ message: 'User registered successfully' });
};

exports.login = async (req, res) => {
  const { email, password } = req.body;
  const user = await User.findOne({ email });
```

```
    if (user && await bcrypt.compare(password, user.password)) {
      const token = jwt.sign({ id: user._id, role: user.role }, 'secret_key', {
expiresIn: '1h' });
      res.status(200).json({ token });
    } else {
      res.status(401).json({ message: 'Invalid email or password' });
    }
  };
```

Модуль автомобілів відповідає за управління інформацією про автомобілі, доступні для продажу. Він дозволяє адміністраторам додавати нові автомобілі, редагувати існуючі записи та видаляти автомобілі з бази даних. Користувачі можуть переглядати інформацію про автомобілі, використовуючи фільтри та пошук.

```
// carModel.js
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const CarSchema = new Schema({
  brand: { type: String, required: true },
  model: { type: String, required: true },
  year: { type: Number, required: true },
  price: { type: Number, required: true },
  description: { type: String }
});

module.exports = mongoose.model('Car', CarSchema);

// carController.js
```

```
const Car = require('../models/carModel');

exports.addCar = async (req, res) => {
  const { brand, model, year, price, description } = req.body;
  const newCar = new Car({ brand, model, year, price, description });
  await newCar.save();
  res.status(201).json({ message: 'Car added successfully' });
};

exports.getCars = async (req, res) => {
  const cars = await Car.find();
  res.status(200).json(cars);
};
```

Модуль замовлень відповідає за обробку замовлень, зроблених користувачами. Він дозволяє користувачам створювати нові замовлення, переглядати історію замовлень та отримувати підтвердження про успішне завершення транзакції.

```
// orderModel.js
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const OrderSchema = new Schema({
  user: { type: Schema.Types.ObjectId, ref: 'User', required: true },
  car: { type: Schema.Types.ObjectId, ref: 'Car', required: true },
  orderDate: { type: Date, default: Date.now },
  status: { type: String, enum: ['pending', 'completed', 'cancelled'], default: 'pending' }
});
```

```
module.exports = mongoose.model('Order', OrderSchema);

// orderController.js
const Order = require('../models/orderModel');

exports.createOrder = async (req, res) => {
  const { userId, carId } = req.body;
  const newOrder = new Order({ user: userId, car: carId });
  await newOrder.save();
  res.status(201).json({ message: 'Order created successfully' });
};

exports.getUserOrders = async (req, res) => {
  const { userId } = req.params;
  const orders = await Order.find({ user: userId }).populate('car');
  res.status(200).json(orders);
};
```

Модуль відгуків дозволяє користувачам залишати відгуки про автомобілі, які вони придбали або тестували. Відгуки можуть включати текстові коментарі та оцінки, що допомагає іншим користувачам приймати обґрунтовані рішення при виборі автомобіля.

```
// reviewModel.js
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const ReviewSchema = new Schema({
  user: { type: Schema.Types.ObjectId, ref: 'User', required: true },
```

```
  car: { type: Schema.Types.ObjectId, ref: 'Car', required: true },
  rating: { type: Number, required: true, min: 1, max: 5 },
  comment: { type: String, required: true },
  reviewDate: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Review', ReviewSchema);

// reviewController.js
const Review = require('../models/reviewModel');

exports.addReview = async (req, res) => {
  const { userId, carId, rating, comment } = req.body;
  const newReview = new Review({ user: userId, car: carId, rating, comment
});
  await newReview.save();
  res.status(201).json({ message: 'Review added successfully' });
};

exports.getCarReviews = async (req, res) => {
  const { carId } = req.params;
  const reviews = await Review.find({ car: carId }).populate('user');
  res.status(200).json(reviews);
};
```

Взаємодія між модулями здійснюється через API-запити. Наприклад, коли користувач створює замовлення, модуль замовлень взаємодіє з модулем користувачів для перевірки автентифікації користувача та з модулем автомобілів для перевірки наявності автомобіля.

Таким чином, основні модулі вебсайту взаємодіють між собою для забезпечення цілісної функціональності системи, забезпечуючи зручний та ефективний процес купівлі автомобілів для користувачів. Кожен модуль виконує специфічні завдання та взаємодіє з іншими модулями через чітко визначені інтерфейси, що сприяє гнучкості та масштабованості системи.

4.4. Інтерфейси користувача, UI/UX дизайн

Інтерфейс користувача (UI) та дизайн користувацького досвіду (UX) відіграють ключову роль у забезпеченні зручності використання вебсайту. Дизайн орієнтований на створення інтуїтивно зрозумілого та привабливого інтерфейсу, що полегшує взаємодію користувачів з системою.

Сайт автодилера має чистий, яскравий та сучасний дизайн, зосереджений на візуальній привабливості та функціональності. Основні елементи інтерфейсу розташовані таким чином, щоб користувачі могли легко знайти необхідну інформацію. Меню навігації чітко структуроване, забезпечуючи швидкий доступ до різних розділів сайту, таких як каталог автомобілів, контактна інформація та відгуки клієнтів.

Важливим аспектом UI/UX дизайну є адаптивність сайту, що дозволяє йому коректно відображатися на різних пристроях, включаючи десктопи, планшети та смартфони. Це забезпечує зручний доступ до сайту незалежно від того, який пристрій використовується.

Візуальні елементи, такі як зображення автомобілів високої якості, допомагають користувачам краще ознайомитися з продукцією компанії. Використання великоформатних фотографій та інтерактивних галерей дозволяє детально розглянути кожну модель автомобіля. Крім того, функція порівняння

автомобілів дозволяє користувачам швидко оцінити різні моделі за основними характеристиками, що полегшує прийняття рішень.

Колірна гамма сайту вибрана таким чином, щоб забезпечити контрастність і легкість читання. Використання акцентних кольорів допомагає виділити важливі елементи інтерфейсу, такі як кнопки заклику до дії (CTA), форми зворотного зв'язку та контактні дані.

Навігація по сайту є інтуїтивно зрозумілою завдяки логічно структурованій інформаційній архітектурі. Користувачі можуть легко переміщатися між розділами сайту та знаходити необхідну інформацію без зайвих зусиль. Це забезпечується за допомогою добре продуманого хедеру, пошукової функції та зручних фільтрів для сортування автомобілів за різними критеріями.

Одним з важливих аспектів UX дизайну є забезпечення швидкої та ефективної комунікації з представниками компанії. На сайті передбачені різні канали зв'язку, такі як форми зворотного зв'язку, онлайн-чати та контактні номери телефонів, що дозволяє користувачам оперативно отримувати відповіді на свої запитання.

РОЗДІЛ 5. РЕАЛІЗАЦІЯ СИСТЕМИ

5.1. Фронт-енд реалізація (React.js, компоненти, маршрутизація, управління станом)

Фронт-енд реалізація вебсайту базується на. React.js, та дозволяє розробляти компонентно-орієнтовані додатки, що сприяє підвищенню гнучкості та повторного використання коду.

Основою React-додатка є компоненти, які являють собою незалежні та повторно використовувані частини інтерфейсу. Кожен компонент може мати свій власний стан (state) та методи для обробки подій. У вебсайті компанії CarS компоненти використовуються для реалізації таких елементів, як заголовок,

навігаційне меню, списки автомобілів, форми зворотного зв'язку та інші інтерактивні елементи. Приклад компонента, що відображає список автомобілів:

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

const CarList = () => {
  const [cars, setCars] = useState([]);

  useEffect(() => {
    fetch('/api/cars')
      .then(response => response.json())
      .then(data => setCars(data));
  }, []);

  return (
    <div className="car-list">
      {cars.map(car => (
        <CarCard key={car.id} car={car} />
      ))}
    </div>
  );
};

export default CarList;
```

Для управління навігацією у додатку використовується бібліотека React Router. Вона дозволяє створювати маршрути, які відповідають різним шляхам у

додатку, і забезпечує плавну навігацію без перезавантаження сторінок. Приклад налаштування маршрутизації:

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import HomePage from './pages/HomePage';
import CarDetails from './pages/CarDetails';
import ContactPage from './pages/ContactPage';

const App = () => (
  <Router>
    <Switch>
      <Route path="/" exact component={HomePage} />
      <Route path="/car/:id" component={CarDetails} />
      <Route path="/contact" component={ContactPage} />
    </Switch>
  </Router>
);

export default App;
```

Для управління станом додатка використовується бібліотека Redux або контекстний API React. Це дозволяє централізовано зберігати стан додатка та легко передавати дані між компонентами. Приклад використання контекстного API для управління станом:

```
import React, { createContext, useReducer, useContext } from 'react';

const initialState = {
  user: null,
```

```
  cars: []
};

const AppContext = createContext();

const reducer = (state, action) => {
  switch (action.type) {
    case 'SET_USER':
      return { ...state, user: action.payload };
    case 'SET_CARS':
      return { ...state, cars: action.payload };
    default:
      return state;
  }
};

export const AppProvider = ({ children }) => {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <AppContext.Provider value={{ state, dispatch }}>
      {children}
    </AppContext.Provider>
  );
};

export const useAppContext = () => useContext(AppContext);
```

Таким чином, фронт-енд реалізація сторінок за допомогою React.js включає використання компонентів для створення інтерфейсу користувача, маршрутизації для управління навігацією та засобів для централізованого управління станом додатка. Це забезпечує гнучкість, масштабованість та зручність розробки, а також сприяє створенню інтерактивного та інтуїтивно зрозумілого користувацького досвіду.

5.2. Бек-енд реалізація (серверна частина, API, інтеграція з базою даних)

Бек-енд реалізація вебсайту автодилера базується на використанні Node.js та Express.js, що забезпечує швидку та ефективну обробку серверних запитів і інтеграцію з базою даних. Основними завданнями бек-енд частини є обробка клієнтських запитів, управління даними та забезпечення безпеки.

Серверна частина, реалізована за допомогою Node.js та Express.js, відповідає за маршрутизацію запитів, автентифікацію користувачів та обробку бізнес-логіки. Express.js забезпечує простий та гнучкий спосіб створення RESTful API, що дозволяє легко обробляти HTTP-запити та взаємодіяти з клієнтською частиною.

```
const express = require('express');
```

```
const app = express();
const mongoose = require('mongoose');
const bodyParser = require('body-parser');

app.use(bodyParser.json());

// Підключення до бази даних
mongoose.connect('mongodb://localhost/carsdb', { useNewUrlParser: true, useUnifiedTopology: true });

// Маршрут для отримання списку автомобілів
app.get('/api/cars', async (req, res) => {
  const cars = await Car.find();
  res.json(cars);
});

// Маршрут для створення нового автомобіля
app.post('/api/cars', async (req, res) => {
  const newCar = new Car(req.body);
  await newCar.save();
  res.status(201).json(newCar);
});

// Запуск сервера
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

RESTful API забезпечує взаємодію між клієнтською та серверною частинами додатка. API надає кінцеві точки (endpoints) для виконання CRUD-операцій (створення, читання, оновлення, видалення) над даними

автомобілів, користувачів, замовлень та відгуків. Кожен маршрут API обробляє певні запити та виконує відповідні операції з базою даних.

Для зберігання даних використовується MongoDB, нереляційна база даних, що забезпечує високу продуктивність та гнучкість. Для взаємодії з MongoDB використовується Mongoose, об'єктно-реляційний мапер (ORM), який спрощує роботу з базою даних.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const CarSchema = new Schema({
  brand: { type: String, required: true },
  model: { type: String, required: true },
  year: { type: Number, required: true },
  price: { type: Number, required: true },
  description: { type: String }
});

const Car = mongoose.model('Car', CarSchema);

module.exports = Car;
```

Цей приклад показує, як визначити схему даних для автомобіля та створити модель, яку можна використовувати для виконання CRUD-операцій з базою даних.

Таким чином, бек-енд реалізація вебсайту забезпечує обробку запитів, управління даними та взаємодію з клієнтською частиною через RESTful API. Використання Node.js, Express.js та MongoDB дозволяє створити надійну та масштабовану серверну частину.

5.3. Зберігання та обробка даних (CRUD операції, взаємодія з БД)

Основні операції з даними включають створення (Create), читання (Read), оновлення (Update) та видалення (Delete) — відомі як CRUD операції. Вони забезпечують повний цикл управління даними в базі даних, що дозволяє ефективно керувати інформацією про автомобілі, користувачів, замовлення та відгуки.

CRUD операції реалізуються за допомогою Mongoose, який забезпечує зручний інтерфейс для взаємодії з MongoDB. Нижче наведено приклади основних CRUD операцій для сутності "Автомобіль".

Для створення нового запису в базі даних використовується метод save(). Це дозволяє додавати нові автомобілі до колекції.

```
const Car = require('../models/carModel');

const addCar = async (req, res) => {
  const newCar = new Car(req.body);
  try {
    await newCar.save();
    res.status(201).json(newCar);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

Для отримання даних з бази використовується метод find(). Це дозволяє отримувати списки автомобілів або окремі записи за певними критеріями.

```
const getCars = async (req, res) => {
  try {
    const cars = await Car.find();
    res.status(200).json(cars);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

Оновлення записів що існують виконується за допомогою методів findByIdAndUpdate(). Це дозволяє змінювати дані про автомобілі, зберігаючи актуальність інформації.

```
const updateCar = async (req, res) => {
  try {
     const updatedCar = await Car.findByIdAndUpdate(req.params.id, req.body, { new: true });
    res.status(200).json(updatedCar);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

Для видалення записів з бази даних використовується метод findByIdAndDelete(). Це дозволяє видаляти автомобілі, які більше не доступні для продажу.

```
const deleteCar = async (req, res) => {
  try {
```

```
    await Car.findByIdAndDelete(req.params.id);
    res.status(200).json({ message: 'Car deleted successfully' });
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

Взаємодія з базою даних відбувається через Mongoose, який забезпечує зручний API для підключення, запитів та управління даними. Підключення до бази даних здійснюється на початку роботи сервера, що гарантує доступність даних для всіх частин додатка.

```
const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost/carsdb', { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => console.log('MongoDB connected...'))
  .catch(err => console.log(err));
```

Таким чином, реалізація CRUD операцій та взаємодія з базою даних є важливою частиною бек-енд функціональності. Це забезпечує ефективне управління даними, підтримку їх актуальності та цілісності, а також надійну роботу системи загалом.

РОЗДІЛ 6. ТЕСТУВАННЯ

6.1. Методології тестування (юніт-тестування, інтеграційне тестування, системне тестування)

Тестування забезпечує його надійність, якість і відповідність програмного забезпечення вимогам користувачів. Для цього проєкту я бажаю застосовувати різні методології тестування, зокрема юніт-тестування, інтеграційне тестування та системне тестування, кожна з яких виконує специфічні завдання та має свої особливості.

Юніт-тестування полягає у перевірці окремих модулів або компонентів системи. Це дозволяє виявляти помилки на ранніх стадіях розробки та забезпечувати правильність роботи окремих функцій. Для вебсайту компанії CarS юніт-тести можуть перевіряти функції обробки даних, компоненти React, бізнес-логіку та інші окремі частини коду. Зазвичай для цього використовуються бібліотеки та фреймворки, такі як Jest для JavaScript.

Інтеграційне тестування перевіряє взаємодію між різними модулями системи. Це дозволяє виявити проблеми, які виникають при інтеграції окремих компонентів. У випадку вебсайту компанії CarS інтеграційні тести можуть перевіряти взаємодію між клієнтською та серверною частинами, правильність виконання API-запитів, взаємодію з базою даних та інтеграцію з зовнішніми сервісами. Для цього можуть використовуватися інструменти, такі як Mocha та Chai.

Системне тестування полягає у перевірці всієї системи як єдиного цілого. Воно охоплює всі аспекти роботи вебсайту, включаючи функціональність, продуктивність, безпеку та зручність використання. Системне тестування допомагає переконатися, що всі компоненти системи працюють разом належним чином і відповідають вимогам користувачів. Для вебсайту компанії CarS системні тести можуть включати перевірку всього процесу купівлі автомобіля, від пошуку та фільтрації до обробки замовлення та оплати.

Застосування різних методологій тестування дозволяє забезпечити всебічну перевірку, виявити та виправити помилки на різних етапах розробки та гарантувати високу якість кінцевого продукту.

ВИСНОВКИ

Розробка вебсайту компанії CarS із застосуванням сучасних технологій, таких як React.js, Node.js та MongoDB, дозволила створити високоякісну та функціональний сайт для продажу автомобілів. Інтерфейс користувача, розроблений з урахуванням принципів UI/UX дизайну, забезпечує інтуїтивно зрозумілу навігацію та привабливий вигляд, що сприяє залученню клієнтів. Завдяки компонентно-орієнтованій архітектурі та розділенню на фронт-енд і бек-енд частини система демонструє високу гнучкість та масштабованість, що полегшує її підтримку та розвиток.

Використання клієнт-серверної моделі та мікросервісної архітектури забезпечує надійність та ефективність взаємодії між компонентами системи. Реалізація CRUD операцій та інтеграція з базою даних MongoDB дозволяють ефективно керувати даними про автомобілі, користувачів, замовлення та відгуки, підтримуючи актуальність та цілісність інформації. Впровадження RESTful API сприяє легкій інтеграції з іншими системами та забезпечує надійний обмін даними між клієнтською та серверною частинами додатка.

Важливою складовою процесу розробки є тестування, яке включає юніт-тестування, інтеграційне тестування та системне тестування. Застосування цих методологій дозволяє виявити та виправити помилки на різних етапах розробки, що забезпечує високу якість кінцевого продукту. Юніт-тестування перевіряє коректність окремих компонентів, інтеграційне тестування забезпечує правильну взаємодію між модулями, а системне тестування перевіряє роботу всієї системи в цілому.

СПИСОК ЛІТЕРАТУРИ

1. React.js Documentation. Офіційна документація. Доступно: https://reactjs.org/docs/getting-started.html.

2. Node.js Documentation. Офіційна документація

Доступно: https://nodejs.org/en/docs/.

3. Express.js Documentation. Офіційна документація. Доступно: https://expressjs.com/en/starter/installing.html.

4. MongoDB Documentation. Офіційна документація. Доступно: https://docs.mongodb.com/.

5. Mongoose Documentation. Офіційна документація. Доступно:

https://mongoosejs.com/docs/guide.html.

6. Jest Documentation. Офіційна документація Доступно: https://jestjs.io/docs/en/getting-started.

7. Mocha Documentation. Офіційна документація. Доступно: https://mochajs.org/.

8. Chai Documentation. Офіційна документація. Доступно: https://www.chaijs.com/.

9. Stripe Documentation. Офіційна документація. Доступно: https://stripe.com/docs.

10. React Router Documentation. Офіційна документація. Доступно:

https://reactrouter.com/web/guides/quick-start.

11. Choi, David. Full-Stack Web Development with React and Node.js: Build Scalable and Maintainable Full-Stack Web Applications Using React, Node.js, and MongoDB. O'Reilly Media, 2020. - 648 с.

ДОДАТКИ

Додатки є власною розробкою та інтелектуальною властістю студента, тому не публікуються.