

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ПЕРЕВІРКИ

АВТОМОБІЛІВ ДИЛЕРАМИ

DEVELOPMENT OF A WEB APPLICATION FOR DEALERS TO

INSPECT VEHICLES

Виконав(ла): студент(ка) 4 курсу, групи КНТ-229

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

СТАТЄВ М. В.

(ПРИЗВИЩЕ та ініціали)

Керівник КАПЛІСНКО Т. І.

(ПРИЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О. В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
 Кафедра Програмних засобів
 Ступінь вищої освіти Бакалавр
 Спеціальність 122 Комп'ютерні науки
(код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри **ПЗ, д.т.н., проф.**

Сергій СУББОТІН

« 04 » квітня 20 23 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

СТАТЄВА Микити Володимировича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка вебзастосунку для перевірки автомобілів дилерами

Development of a Web Application for Dealers to Inspect Vegicles

керівник проєкту (роботи) к.т.н., доцент, КАПЛІЄНКО Тетяна Ігорівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від « 04 » квітня 20 23 року № 87

2. Строк подання студентом проєкту (роботи) 07 червня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження.

2. Матеріали і методи 3. Опис програми 4. Експлуатація, тестування та експериментальне дослідження програми

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	КАПЛІЄНКО Т. І., доцент		
Нормоконтроль	КАМІНСЬКА Ж. К., асистент		

7. Дата видачі завдання «__04__» ____квітня____20__23__року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки.		
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження	5 тиждень	Розділ 4, 5
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Микита СТАТЄВ
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна КАПЛІЄНКО
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 125 с., 41 рис., 6 табл., 2 дод., 47 джерел.

ВЕБЗАСТОСУНОК, ПЕРЕВІРКА АВТОМОБІЛІВ, ВЖИВАНІ АВТОМОБІЛІ, JAVASCRIPT, NODEJS, REACT, NEXTJS, EXPRESSJS

Об'єкт дослідження – програмні засоби для адміністрування вживаних автомобілів, оцінювання їх стану, пошкоджень і іншого, та подальша можливість використовувати ці дані для ознайомлення клієнтів.

Предмет дослідження – методи та технології розробки вебзастосунку для адміністрування вживаних автомобілів, які допоможуть дилерам зменшити ризик покупки неякісного автомобіля та підвищити якість їхньої продукції.

Мета роботи – розробка вебзастосунку, який дасть можливість дилерам створювати "точки інтересу" для перевірки вживаних автомобілів, додавати фотографії та інші деталі про автомобіль, які важливі для перевірки якості, а також відстежувати стан перевірки та отримувати повідомлення про зміни.

Матеріали, методи та технічні засоби: мова програмування JavaScript, база даних MongoDB, бібліотека інтерфейсу Ant Design, фреймворм NextJS, фреймворк express.

Результати. Розроблено фронтенд і бекенд частини вебзастосунку для перевірки вживаних автомобілів, що полегшить роботу дилерів і покупців.

Висновки. Створене програмне забезпечення для галузі ринку вживаних автомобілів, що покращує взаємозв'язок між бізнесом і клієнтом.

Галузь використання – ринок вживаних автомобілів, що стає все більш привабливим для багатьох покупців, що надають можливість економити кошти, одночасно задовольняючи потреби в особистій транспортній зручності.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 125 pages, 41 figures, 6 tables, 2 appendixes, 47 sources.

WEB APPLICATION, VEHICLE CHECK, USED VEHICLES, JAVASCRIPT, NODEJS, REACT, NEXTJS, EXPRESSJS

Object of study is a web application for managing used cars, assessing their condition, damage, etc., and the subsequent possibility of using this data to familiarize customers.

Subject of study is the methods and technologies for developing a web application for used car management that will help dealers reduce the risk of buying a low-quality car and improve the quality of their products.

The purpose of the work is to develop a web application that will allow dealers to create "points of interest" for used car inspection, add photos and other details about the car that are important for quality inspection, as well as track the status of the inspection and receive notifications of changes.

Materials, methods and tools: JavaScript programming language, MongoDB database, Ant Design interface library, NextJS framework, express framework.

Results. The front-end and back-end parts of a web application for checking used cars have been developed, which will facilitate the work of dealers and buyers.

Conclusion. Have been created software for the used car market industry that improves communication between business and customers.

The field of use is the used car market, which is becoming increasingly attractive to many buyers, providing an opportunity to save money while meeting the needs for personal transportation convenience.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК	9
ВСТУП.....	10
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	12
1.1 Поняття і ризики вживаних автомобілів	12
1.1 Аналіз потреб користувача	13
1.2 Вимоги щодо розроблюваного програмного забезпечення	13
1.3 Аналіз аналогів і їх порівняння	13
1.4 Висновки з розділу	16
2. ВИБІР І ОБҐРУНТУВАННЯ ІНСТРУМЕНТІВ І АРХІТЕКТУРИ	18
2.1 Вибір мови програмування	18
2.2 Вибір фронтенд фреймворку	20
2.3 Вибір бекенд фреймворку	23
2.3.1 ExpressJS.....	24
2.3.2 Koa	24
2.3.3 NestJS	24
2.3.4 Napi	25
2.3.5 Підсумок.....	25
2.4 API архітектура	26
2.4.1 Rest	27
2.4.2 SOAP	27
2.4.3 GraphQL.....	28
2.5 Вибір бази даних	29
3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	30
3.1 Основні відомості про розробку	30
3.2 Реалізація інтерфейсу	33
3.2.1 Інтерфейс таблиці даних.....	35
3.2.2 Інтерфейс навігаційного меню.....	36
3.2.3 Інтерфейс форми створення/редагування.....	37
3.3 Система маршрутизації	39
3.4 Структура бази даних	41
3.5 Розроблені контролери.....	44
3.6 Висновки за розділом	45

4 КЕРІВНИЦТВО ПРОГРАМІСТА	46
4.1 Розгортання на локальному комп'ютері	46
4.2 Інсталяція застосунку	46
4.3 Розгортання бази даних.....	46
4.4 Запуск сервера.....	47
4.5 Запуск клієнта	49
4.4 Висновки за розділом	50
5 КЕРІВНИЦТВО КОРИСТУВАЧА	51
5.1 Загальні відомості	51
5. 2 Висновки за розділом	57
ВИСНОВКИ.....	58
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	59
ДОДАТОК А Слайди презентації.....	63
ДОДАТОК Б Код програми	71
Б.1 Файл DataTable.tsx.....	72
Б.2 Файл Drawer.tsx	73
Б.3 Файл EmptyPlaceholder.tsx.....	76
Б.4 Файл RouteGuard.....	77
Б.5 Файл Sidebar.tsx	78
Б.6 Файл DrawerContext.tsx.....	80
Б.7 Файл useActivity.ts	81
Б.8 Файл useDebounce.ts.....	83
Б.9 Файл useDrawer.ts	84
Б.10 Файл useWidget.ts	85
Б.11 Файл AuthLayout.tsx	86
Б.12 Файл CommonLayout.tsx	87
Б.13 Файл ImagePlaceholder.tsx	88
Б.14 Файл pinView.tsx.....	88
Б.15 Файл pages/Inspection.tsx	90
Б.16 Файл pages/Login.tsx.....	103
Б.17 Файл _app.tsx.....	105
Б.18 Файл api.ts.....	106
Б.19 Файл cloudinary.ts	108
Б.20 Файл app.reducer.ts.....	110

Б.21 Файл store.ts.....	110
Б.22 Файл inspection.redux.ts.....	111
Б.23 Файл app.js.....	115
Б.24 Файл connection.js.....	116
Б.25 Файл routes.js.....	116
Б.26 Файл inspection/controller.create.js.....	117
Б.27 Файл inspection/controller.deleteById.js.....	118
Б.28 Файл inspection/controller.getAll.js	118
Б.29 Файл inspection/controller.getById.js	120
Б.29 Файл inspection/controller.updateById.js	121
Б.30 Файл inspection.model.js	122
Б.31 Файл errorMiddleware.js	124
Б.32 Файл schemaTransformPlugin.js	124
Б.33 Файл index.js.....	125

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;
BSON – Binary encoded JavaScript Object Notation;
CRUD – Create, Read, Update, Delete;
DOM – Document Object Model;
HTML – HyperText Markup Language;
JSON – JavaScript Object Notation;
MVC – Model View Controller;
OEM – Original Equipment Manufacturer;
REST – Representational State Transfer;
SOAP – Simple Object Access Protocol;
XML – eXtensible Markup Language;
ООП – Об’єктно Орієнтоване Програмування;

ВСТУП

Ринок вживаних автомобілів є значним та зростає кожен рік. Згідно зі звітом компанії Cox Automotive, попит на вживані автомобілі у США зросла на 5,1% у 2020 році [1], що свідчить про постійне зростання цього ринку. Однак, з підвищенням попиту на вживані автомобілі також зростає ризик придбання неякісного автомобіля. Це особливо стосується дилерів автомобілів, які регулярно купують вживані автомобілі на аукціонах та інших ресурсах.

У зв'язку з цим, розробка вебзастосунку, який допоміг би дилерам перевіряти якість вживаних автомобілів, є важливим завданням. Застосунок, що розробляється, дасть можливість дилерам адмініструвати інформацію про вживані автомобілі, дозволить додавати фотографії та інші деталі про автомобіль, які важливі для перевірки якості. Це дозволить дилерам зменшити ризик покупки неякісного автомобіля та забезпечить підвищення якості їхньої продукції.

Об'єктом дослідження є програмні засоби для адміністрування вживаних автомобілів, оцінювання їх стану, пошкоджень і іншого, та подальша можливість використовувати ці дані для ознайомлення клієнтів.

Предметом дослідження є методи та технології розробки вебзастосунку для адміністрування вживаних автомобілів, які допоможуть дилерам зменшити ризик покупки неякісного автомобіля та підвищити якість їхньої продукції.

Метою дослідження є розробка вебзастосунку, який дасть можливість дилерам створювати "точки інтересу" для перевірки вживаних автомобілів, додавати фотографії та інші деталі про автомобіль, які важливі для перевірки якості, а також відстежувати стан перевірки та отримувати повідомлення про зміни. Метою дослідження є також аналіз ефективності користування вебзастосунком дилерами, що допоможе в подальшому вдосконалити продукт та розширити його можливості.

Для розробки вебзастосунку для адміністрування вживаних автомобілів, було виконано наступні кроки:

- аналіз вимог і потреб користувачів: для розуміння того, які функції та можливості повинен мати застосунок, було проведено аналіз вимог та потреб користувачів;
- аналіз конкурентів: було проведено аналіз існуючих на ринку вебзастосунків для адміністрування вживаних автомобілів. Це дозволило визначити переваги та недоліки конкурентів, а також знайти можливості для покращення вебзастосунку;
- розробка архітектури вебзастосунку: була розроблена архітектура вебзастосунку, включаючи визначення структури бази даних та взаємодії між фронтендом та серверною частинами;
- розробка функціоналу вебзастосунку: було розроблено функціонал вебзастосунку, включаючи створення точок інтересу для перевірки якості вживаних автомобілів, додавання фотографій та інших деталей про автомобілі, відстеження стану перевірки та повідомлення про зміни;
- тестування та відлагодження: після розробки було проведено тестування вебзастосунку та виявлено та виправлено помилки та недоліки;
- впровадження та підтримка: після успішного тестування було проведено впровадження вебзастосунку та забезпечено його підтримку та розвиток.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття і ризики вживаних автомобілів

Вживані авто - це автомобілі, які були використані попереднім власником і продаються на ринку вживаних автомобілів. Існує кілька варіантів, які можна розглянути при покупці вживаного автомобіля, включаючи покупку від приватного власника, покупку від дилера або взяття автомобіля в оренду. Перед покупкою вживаного автомобіля, важливо здійснити тест-драйв та перевірити його історію. Крім того, варто розглянути можливість придбання автомобіля з гарантією або з програмою обміну автомобіля. Необхідно враховувати ризики, пов'язані з покупкою вживаного автомобіля, такі як його історія використання, аварії та ремонт.

Вибір вживаного автомобіля - це складний процес, який потребує уважності та обачності. Нижче наведено декілька пунктів на які покупець має звертати свою увагу у першу чергу:

- історія автомобіля. Існує багато сервісів які надають інформацію про історію автомобіля, його попередніх власників, дату випуску, пробіг, його ремонти і аварії;
- технічний стан. Перед тим, як купувати вживаний автомобіль, важливо перевірити його технічний стан. Перевірте двигун, трансмісію, гальма, підвіску та інші вузли автомобіля;
- перед тим, як купувати вживаний автомобіль, важливо зробити тест-драйв та переконатися, що автомобіль справний і підходить покупцю. Тест-драйв допоможе оцінити технічний стан автомобіля, комфортність та безпеку руху.

Загалом, вибір вживаного автомобіля - це важлива та відповідальна річ, яка потребує уважності та обачності. Варто ретельно перевіряти історію та технічний стан автомобіля, враховувати свої потреби та бюджет, не забувати про додаткові витрати та процедури розмитнення та реєстрації. Важливо також дізнатися про можливість купівлі з гарантією та програмою обміну автомобіля.

1.1 Аналіз потреб користувача

По-перше, програмне забезпечення повинно бути простим для інтуїтивного розуміння, задля того, щоб користувач міг швидко і просто виконати всі потрібні йому дії.

По друге, застосунок повинен мати якомога найпотрібнішу інформацію при користуванні. Наприклад, однією з основних сутностей є двигун, абстракція якого повинна бути в застосунку, а, наприклад, диски коліс це не настільки важлива інформація, яку можна записати в описі до іншої сутності.

Також для більш зручного користування, повинні бути різні види сортувань і фільтрів.

1.2 Вимоги щодо розроблюваного програмного забезпечення

Необхідно розробити функціонал додавання, редагування, видалення та перегляду автомобілів і їх перевірок.

Наявність системи реєстрації та авторизації користувачів.

1.3 Аналіз аналогів і їх порівняння

Існують багато вебзастосунків для купівлі та продажу автомобілів, наприклад, Autotrader, Cars.com, eBay Motors та багато інших. Проте, ці платформи зазвичай не забезпечують можливості адміністрування вживаних автомобілів. Зазвичай вони орієнтовані на продаж нових автомобілів або вживаних автомобілів від дилерів.

Водночас існують також спеціалізовані вебзастосунки для продажу вживаних автомобілів, такі як CarGurus, AutoTrader.com та CarsDirect. Ці платформи забезпечують можливості пошуку та перегляду вживаних автомобілів, а також можуть надавати інформацію про історію обслуговування та відгуки користувачів про конкретний автомобіль.

Проте, ці платформи зазвичай не забезпечують можливості адміністрування вживаних автомобілів, які відповідають потребам дилерів та приватних власників автомобілів.

CarGurus - це сервіс, який надає можливість купувати нові чи вживані автомобілі, продавати свої автомобілі та проводити тест-драйв автомобіля [2], він зображений на рис. 1.1.

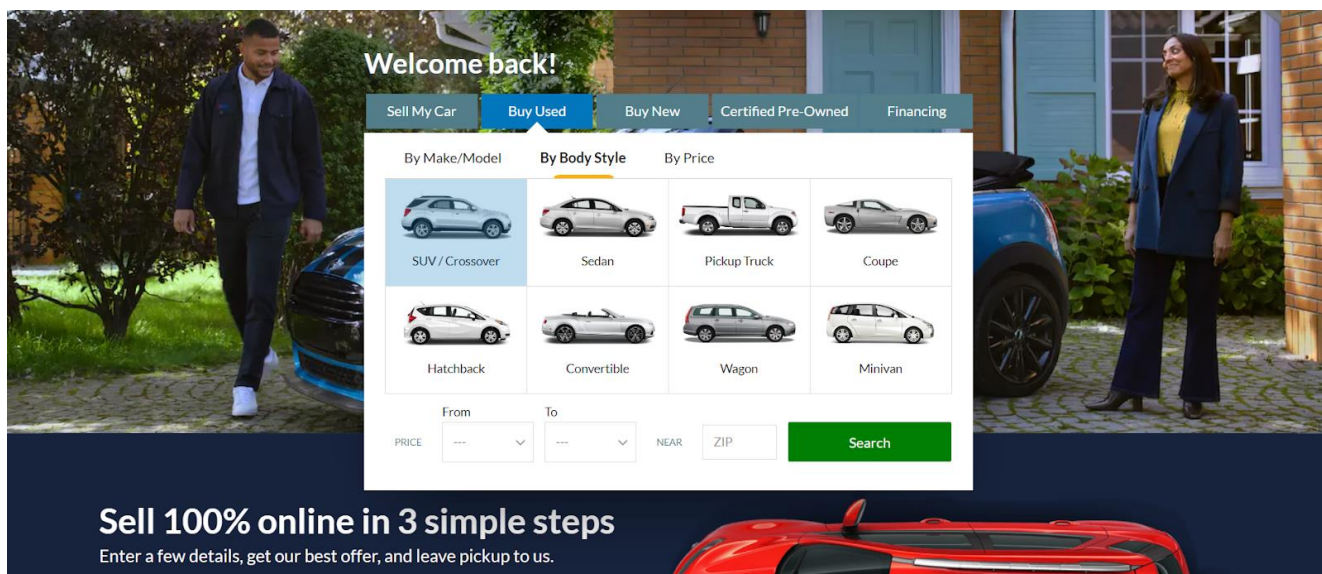


Рисунок 1.1 – Вебзастосунок CarGurus [2]

Мінусом цього сервісу є те, що користувач не може власноруч вказати потрібні йому місця перевірки, через що можна сподіватися лише на довіру.

Сервіс AutoTrader, зображений на рис. 1.2 та рис. 1.3 має переваги порівняно з попереднім, бо тут користувач може подивитися всі основні специфікації автомобіля, які завірені після перевірки, через що можна дуже докладно вивчити, наприклад, декілька варіантів однієї моделі авто [3].

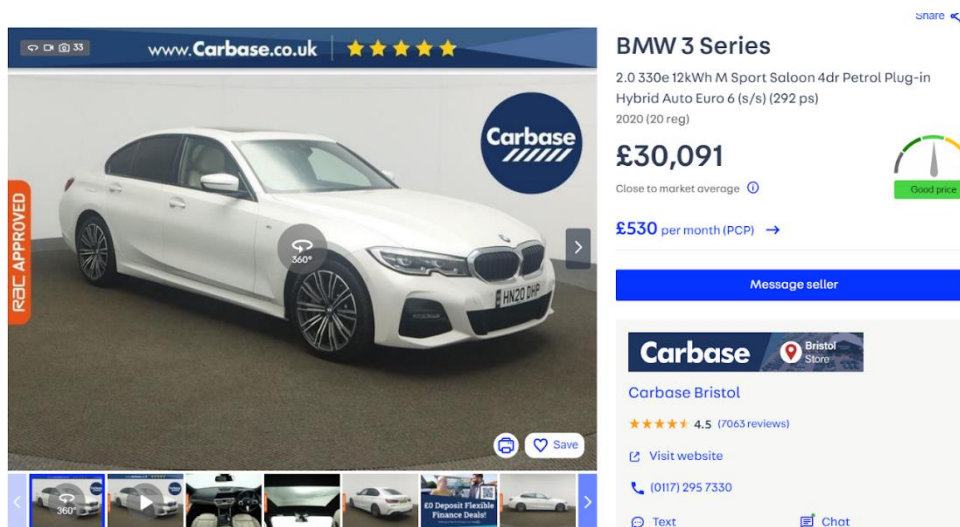


Рисунок 1.2 – Сторінка автомобіля на сервісі AutoTrader [3]

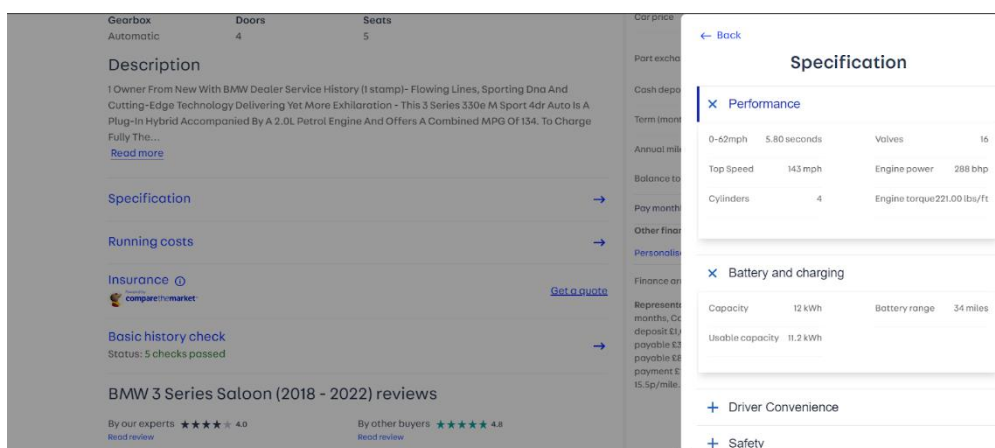


Рисунок 1.3 – Детальна інформація про автомобіль на сервісі AutoTrader [3]

Сервіс CarsDirect, що зображений на рис 1.4 та рис. 1.5 має такий же недолік як і CarGurus - він надає недостатньо інформації про стан автомобіля і його компонентів, користувач не може сам перевірити якісь цікаві йому місця. Але великим плюсом цього сервісу є його інтеграція з Carfax, за допомогою якого можна дізнатися історію автомобіля, всі його аварії або обслуговування [4].

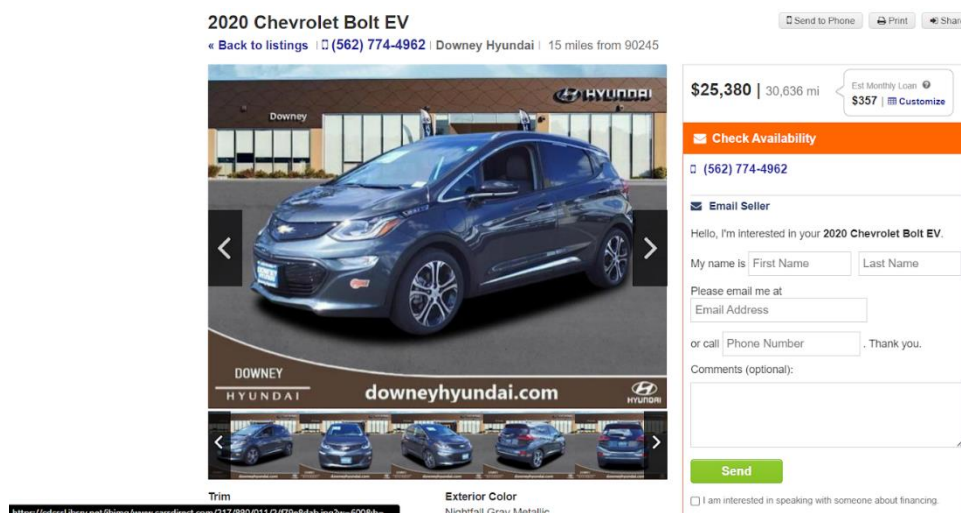


Рисунок 1.4 – Сторінка автомобіля на сервісі CarsDirect [4]

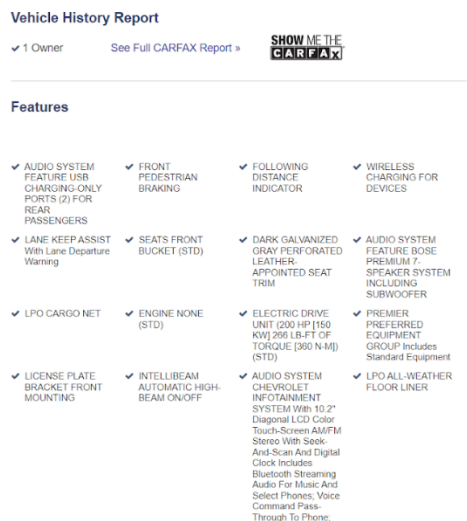


Рисунок 1.5 – Детальна інформація про автомобіль [4]

1.4 Висновки з розділу

Висновки з цього розділу дозволяють зрозуміти, що вебзастосунки для адміністрування вживаних автомобілів можуть бути надзвичайно корисними для дилерів та приватних власників автомобілів, які шукають платформу для ефективного управління та продажу вживаних автомобілів. Ці застосунки можуть допомогти спростити процес продажу та купівлі автомобілів, а також забезпечити більшу безпеку при цьому.

З використанням сучасних технологій розробки програмного забезпечення, можна створити ефективний та безпечний вебзастосунок, який забезпечує

ефективне управління вживаних автомобілів. Більше того, забезпечення простоти та зручності взаємодії з системою може позитивно позначитись на користуваності платформи та на задоволенні користувачів.

Отже, створення вебзастосунку для адміністрування вживаних автомобілів може зробити процес продажу та купівлі автомобілів більш простим та безпечним для користувачів. Такий застосунок може бути надзвичайно корисним для дилерів та приватних власників автомобілів, які шукають ефективну платформу для управління та продажу вживаних автомобілів, а також може зробити процес купівлі вживаних автомобілів більш доступним та зручним для покупців.

2. ВИБІР І ОБҐРУНТУВАННЯ ІНСТРУМЕНТІВ І АРХІТЕКТУРИ

2.1 Вибір мови програмування

У сучасному цифровому світі веброзробка стала ключовою складовою успіху бізнесу. Вебзастосунки і адміністративні панелі відіграють важливу роль у взаємодії з користувачами та ефективному управлінні даними. Згідно з дослідженням Eastern Peak [5], серед найпопулярніших мов програмування у 2022 році є JavaScript, Python, Java та PHP. Якщо йдеться про веброзробку застосунку для адміністрування, то JavaScript є однією з найбільш популярних мов програмування [6].

Node.js – це середовище виконання JavaScript, яке дозволяє використовувати JavaScript на серверному боці. Використання Node.js дозволяє розробникам використовувати одну мову програмування для розробки як клієнтської, так і серверної частини застосунку [7].

JavaScript є однією з найбільш популярних мов програмування для веброзробки [6]. Вона має широкий спектр функцій та багато корисних бібліотек. JavaScript може бути використаний для створення динамічних інтерфейсів користувача, а також для розробки серверної частини застосунку з використанням Node.js [7].

Таким чином, використання JavaScript + Node.js є гарним вибором для розробки сучасної адмінки.

Ось кілька плюсів JavaScript у порівнянні з іншими мовами програмування для веброзробки:

- широке застосування: JavaScript може бути використаний для розробки як клієнтської, так і серверної частини вебзастосунку [8]. Це дозволяє розробникам використовувати одну мову програмування для розробки як клієнтської, так і серверної частини вебзастосунку;
- широкий спектр функцій: JavaScript має широкий спектр функцій та багато корисних бібліотек [8]. Це дозволяє розробникам створювати складні функції та ефективно працювати з даними;

- простота вивчення: JavaScript є досить простою мовою програмування для вивчення [8]. Вона має простий синтаксис та легко засвоюється новачками;
- широка спільнота розробників: JavaScript має широку спільноту розробників, яка постійно розвивається та поповнюється новими ідеями та ресурсами [8]. Це дозволяє розробникам швидко знаходити відповідь на свої запитання та ефективно працювати з іншими розробниками;
- динамічна природа: JavaScript є динамічною мовою програмування, що дозволяє створювати динамічні інтерфейси користувача [8]. Це дозволяє розробникам створювати багатофункціональні та ефективні інтерфейси користувача.

Порівняння найбільш популярних мов програмування наведено у таблиці 2.1.

Таблиця 2.1 – Порівняння мов програмування у сфері веброзробки

Характеристика	Java	JavaScript	Python	PHP
1	2	3	4	5
Тип мови	Сильно типізована	Слабо типізована	Сильно типізована	Слабо типізована
Використання	Загальне	Фронтенд, бекенд	Загальне	Серверні сценарії, веб
Синтаксис	Об'єктно-орієнтований, Схожий на C	Схожий на C, Синтаксис заснований на об'єктах	Простий, Читабельний	Синтаксис заснований на C
Популярність	Висока	Дуже висока	Висока	Висока
Підтримка спільноти	Велика	Дуже велика	Велика	Велика

Продовження таблиці 2.1

1	2	3	4	5
Використання в веброзробці	Так	Так	Так	Так
Використання в наукових дослідженнях	Так	Так	Так	Так
Розширюваність	Висока	Висока	Висока	Висока
Швидкодія	Висока	Помірна	Помірна	Помірна
Підтримка ООП	Так	Так	Так	Частково
Використання в Big Data	Так	Так	Так	Так

2.2 Вибір фронтед фреймворку

У сучасному світі фронтед розробка стала надзвичайно важливою галуззю програмування. Завдяки розширенню Інтернету та появі великої кількості вебзастосунків, потреба в швидкій та ефективній розробці користувацького інтерфейсу стала особливо актуальною. JavaScript, як мова програмування, здобув велику популярність в цьому контексті, а фреймворки для фронтед розробки на JavaScript забезпечують розробникам потужні інструменти для побудови вебзастосунків [9].

Нижче у таблиці 2.2 наведено порівняльну таблицю найбільш популярних вебфреймворків для фронтед розробки.

Таблиця 2.2 – Порівняння фреймворків для веброзробки

Фреймворк	Вебсайт	Версія	Архітектура	Величина спільноти	Підтримка
React	reactjs.org	17.0.2	Компонентна	Велика	Активна
Angular	angular.io	12.0.0	MVC	Велика	Активна
Vue.js	vuejs.org	3.2.20	Компонентна	Велика	Активна
Ember.js	emberjs.com	3.28.0	MVC	Середня	Активна
Svelte	svelte.dev	3.38.3	Компонентна	Зростаюча	Активна

React є одним з найпопулярніших фреймворків для фронтенд розробки на JavaScript. Він базується на концепції компонентної архітектури, що дозволяє розбити інтерфейс на незалежні компоненти, які легко підтримувати та розширювати. React має велику спільноту розробників, що сприяє швидкій розробці та постійному вдосконаленню фреймворку. Він також має багатий екосистема з великою кількістю сторонніх бібліотек і інструментів. Одним з прикладів є NextJS. Фреймворк Next.js заснований на React та дозволяє розробникам створювати повноцінні вебзастосунки з включеними готовими рішеннями для стилізації та маршрутизації. Основні переваги використання Next.js полягають у підтримці серверного та попереднього рендерингу, що дозволяє поліпшити швидкість завантаження сторінок та забезпечити кращу оптимізацію для пошукових систем [10, 11].

Інші переваги використання Next.js включають:

- простоту використання та гнучкість в налаштуванні, що дозволяє розробникам швидко розпочати проєкт та змінювати його відповідно до потреб;
- можливість попередньої генерації сторінок, що дозволяє покращити швидкість завантаження сторінок та забезпечити кращу оптимізацію для пошукових систем;
- підтримку статичного рендерингу, що дозволяє створювати вебзастосунки зі складними динамічними інтерфейсами, які можуть бути зроблені доступними для пошукових систем;

- підтримку серверного рендерингу, що дозволяє створювати вебзастосунки зі складними динамічними інтерфейсами та забезпечити кращу безпеку та оптимізацію;
- велику та підтримувальну спільноту розробників, що сприяє постійному вдосконаленню фреймворку.

Next.js є одним з найбільш популярних фреймворків для розробки вебзастосунків на основі React, завдяки своїм перевагам та можливостям, які дозволяють розробникам створювати швидкі та оптимізовані вебзастосунки будь-якої складності [10, 11].

Angular, розроблений компанією Google, є ще одним популярним фреймворком для фронтенд розробки [12]. Він використовує архітектурний шаблон MVC (Model-View-Controller) для побудови вебзастосунків. Angular надає потужні можливості для створення складних застосунків, зокрема, вбудовує в себе систему директив, модулів і сервісів. Angular також має широку підтримку з боку спільноти розробників та активно оновлюється.

Vue.js є легким та простим у використанні фреймворком для фронтенд розробки на JavaScript [13]. Він також ґрунтується на компонентній архітектурі, але має меншу вагу порівняно з React та Angular. Vue.js заслуговує уваги завдяки своїй простоті та інтуїтивно зрозумілому синтаксису. Він швидко набирає популярність і має активну спільноту розробників, що сприяє постійному вдосконаленню фреймворку.

Ember.js є повноцінним фреймворком для фронтенд розробки на JavaScript, який використовує архітектурний шаблон MVC [14]. Він надає широкий функціонал для розробки складних вебзастосунків та має велику кількість вбудованих інструментів. Ember.js має середнього розміру спільноту розробників порівняно з іншими фреймворками, але все ще підтримується та активно розвивається.

Svelte є новим фреймворком для фронтенд розробки на JavaScript, який відрізняється від інших фреймворків [15]. Замість того, щоб виконувати розрахунки на стороні клієнта, Svelte перетворює компоненти у оптимізований

JavaScript-код під час компіляції. Це дозволяє створювати швидкі та ефективні вебзастосунки. Спільнота розробників Svelte зростає, і фреймворк активно розвивається.

Вибір фреймворка для фронтенд розробки на JavaScript залежить від конкретних потреб проєкту. Нижче наведено декілька вказівок для вибору:

- якщо розробнику потрібен потужний та розширюваний фреймворк з великою спільнотою, треба розглянути варіант React або Angular;
- якщо важливим фактором є легкість та простота у використанні фреймворк з активною спільнотою, краще підійде Vue.js або Svelte;
- якщо ж для розробки потрібен повноцінний фреймворк з розширеним функціоналом та готовими інструментами, розгляньте Ember.js.

Найкращий фреймворк залежить від потреб розробника, та вимог проєкту.

Фреймворки для фронтенд розробки на JavaScript забезпечують потужні інструменти та можливості для швидкого та ефективного створення вебзастосунків. React, Angular, Vue.js, Ember.js та Svelte - це лише декілька з численних фреймворків, доступних для розробників. Кожен фреймворк має свої особливості та функціонал.

Для вирішення даної проблеми було обрано React, один з найпопулярніших фреймворків для фронтенд розробки на JavaScript з його бібліотекою NextJS. Обираючи React, було враховано його компонентну архітектуру, яка дозволяє легко розбити інтерфейс на незалежні компоненти і керувати ними окремо. Це робить розробку та підтримку коду більш простими та зрозумілими. Крім того, React має велику та активну спільноту розробників, що забезпечує доступ до безлічі сторонніх бібліотек, розширень та ресурсів для підтримки та прискорення процесу розробки.

2.3 Вибір бекенд фреймворку

У світі розробки вебзастосунків бекенд часто відіграє ключову роль у створенні потужних та масштабованих систем. Вибір правильного фреймворку для

розробки бекенду є важливим завданням, оскільки він визначає ефективність, безпеку та швидкість застосунку. У цьому розділі буде порівняно кілька найпопулярніших бекенд фреймворків для розробки на JavaScript: Express, Кoa, NestJS та Напі, розглянемо їх особливості, функціонал та переваги.

2.3.1 ExpressJS

Express є одним з найпоширеніших та найпростіших у використанні бекенд фреймворків на JavaScript [16]. Він має мінімальну кількість абстракцій та дозволяє розробникам використовувати різноманітні бібліотеки для вирішення конкретних завдань. Express пропонує просту маршрутизацію, обробку HTTP-запитів та інтеграцію з різними зовнішніми модулями. Він має широку підтримку спільноти та велику кількість сторонніх пакетів, що робить його гнучким та розширюваним варіантом для розробки бекенду на JavaScript [17].

2.3.2 Кoa

Кoa є модернізованою альтернативою до Express, створеною тим же розробником [18]. Він покликаний розв'язати певні проблеми та недоліки Express, пропонуючи більш простий та елегантний спосіб розробки бекенду. Кoa базується на концепції генераторів ES6, що дозволяє зручно управляти асинхронним кодом та потоком керування. Він пропонує розширену підтримку middleware та добре підходить для створення сучасних API-серверів. Кoa зарекомендував себе як легкий, елегантний та простий у використанні фреймворк для бекенд розробки на JavaScript [19].

2.3.3 NestJS

NestJS є прогресивним та масштабованим фреймворком для розробки бекенду на JavaScript та TypeScript [20]. Він використовує концепції ООП, функціонального програмування та реактивного програмування для побудови

міцних та модульних застосунків. NestJS надає широкий функціонал, включаючи вбудовану маршрутизацію, валідацію даних, автентифікацію та авторизацію. Він підтримує різноманітні бази даних та забезпечує легку інтеграцію з іншими бібліотеками та фреймворками. NestJS є потужним вибором для розробки бекенду на JavaScript, особливо для середніх та великих проєктів.

2.3.4 Нарі

Нарі є фреймворком, що зосереджений на побудові солідних та безпечних вебзастосунків [21]. Він має декларативний підхід до роутінгу, що робить його зручним для розробки та підтримки. Нарі надає багато вбудованих функціональних можливостей, таких як валідація даних, кешування, обробка помилок та багато іншого. Він також пропонує гнучку архітектуру, яка дозволяє розробникам розширювати та налаштовувати фреймворк під свої потреби. Нарі є надійним та зрілим вибором для розробки бекенду на JavaScript.

2.3.5 Підсумок

Вибір правильного бекенд фреймворку для розробки на JavaScript є важливим етапом у створенні сучасних та потужних вебзастосунків. Кожен з розглянутих фреймворків - Express, Koa, NestJS та Нарі - має свої особливості та переваги. Express є простим та гнучким варіантом, Koa пропонує більш елегантний підхід до розробки, NestJS надає масштабовану та модульну архітектуру, а Нарі зосереджений на безпеці та надійності. При виборі фреймворку, варто враховувати особливості вашого проєкту, потреби команди розробників та майбутні перспективи розширення. За допомогою цього порівняльного аналізу, ви зможете зробити обґрунтований вибір та успішно розробити потужний бекенд для свого проєкту на JavaScript.

Нижче наведено таблицю 2.3 з порівнянням популярних бекенд фреймворків.

Таблиця 2.3 – Порівняння бекенд фреймворків

Фреймворк	Основні особливості	Переваги	Недоліки
Express	Мінімальна кількість абстракцій	Простота використання	Менша функціональність порівняно з іншими фреймворками
Koa	Генератори ES6 для керування асинхронним кодом	Елегантний підхід до розробки	Є вимога до розуміння концепції генераторів ES6
NestJS	Прогресивний та модульний фреймворк	Підтримка ООП та реактивного програмування	Вимагає вивчення TypeScript
Napi	Декларативний підхід до роутінгу	Вбудовані функціональні можливості	Менша популярність порівняно з іншими фреймворками

Із цих порівнянь було обрано Express, оскільки він відповідає нашим потребам та вимогам для розробки бекенду на JavaScript. Express забезпечує мінімальну кількість абстракцій та простоту використання, що дозволяє нам швидко створювати розширювальні та ефективні вебзастосунків. Його широка підтримка спільноти та наявність багатьох сторонніх пакетів роблять його гнучким та розширюваним інструментом для розробки. Таким чином, обрання Express дозволить ефективно розробляти та підтримувати бекенд проєкту на JavaScript.

2.4 API архітектура

Архітектура API (Application Programming Interface) - це набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення [22]. API може бути створений для взаємодії між програмними компонентами, або між програмними компонентами та користувачем. Наприклад,

соціальні мережі часто надають API, щоб розробники могли отримувати дані користувачів, публікувати повідомлення або взаємодіяти з іншими функціями платформи. Архітектура API була створена для спрощення взаємодії між програмними компонентами та забезпечення їхньої стандартизації. API дозволяє розробникам легко інтегрувати свої програми з іншими системами, що зменшує час розробки та забезпечує високу якість програмного забезпечення [22, 23]. Окрім REST існують і інші архітектури, їх буде приведено нижче:

2.4.1 Rest

REST API - це спосіб взаємодії двох комп'ютерних систем за допомогою технологій HTTP, що використовуються у веббраузерах і серверах [24]. Це архітектурний стиль для прикладного програмного інтерфейсу (API), який використовує HTTP-запити для доступу та використання даних [25, 26]. REST розшифровується як Representational State Transfer (передача представницького стану) і був створений комп'ютерним вченим Роем Філдінгом (Roy Fielding) [26, 27]. REST API дозволяє взаємодіяти з RESTful вебсервісами. Найчастіше REST API використовують HTTP-запити для виконання роботи з базою даних, наприклад, отримати якийсь запис, або створити новий. Ці функції називаються CRUD (create, read, update, delete). REST API складається з набору сервісів, які відповідають за свою зону, і цей набір відомий як ресурсна модель REST API [27]. Коли клієнтський запит робиться через RESTful API, він передає представлення стану ресурсу запитувачу або кінцевій точці, і ця інформація доставляється в одному з декількох форматів через HTTP, таких як JSON, HTML, XML або звичайний текст [26].

2.4.2 SOAP

SOAP (Simple Object Access Protocol - простий протокол доступу до об'єктів) - це протокол повідомлень, який дозволяє розподіленим елементам програми взаємодіяти між собою [29]. Це протокол на основі XML для доступу до вебсервісів через HTTP [30, 31]. SOAP може передаватися за допомогою різних стандартних

протоколів, включаючи пов'язаний з Інтернетом протокол передачі гіпертексту (HTTP) [29]. SOAP був розроблений як допоміжний інструмент, щоб застосунки, побудовані на різних мовах програмування, могли легко обмінюватися даними і були прості у розробці. SOAP полегшує зв'язок у гетерогенних системах і використовується для обміну структурованою інформацією у вебсервісах [32]. По суті, SOAP упаковує виклики методів у XML-рядки і доставляє їх екземплярам компонентів через HTTP [31]. XML-документи SOAP структуровані навколо кореневих елементів, дочірніх елементів зі значеннями та інших специфікацій [31].

2.4.3 GraphQL

GraphQL - це мова запитів для API і середовище виконання для виконання цих запитів з наявними даними [33]. Це мова запитів до даних для API з відкритим вихідним кодом, яку створила компанія Facebook і яка зараз широко використовується в індустрії [34]. На відміну від REST або SOAP, GraphQL - це не стиль або протокол, а мова запитів для API [35]. GraphQL дозволяє розробникам декларативно визначати дані, які потрібно отримати з сервера [36]. Запити GraphQL отримують доступ не лише до властивостей одного ресурсу, але й плавно слідують за посиланнями між ними. У той час як типові REST API вимагають завантаження з декількох URL-адрес, GraphQL API отримують всі дані, необхідні застосунку, в одному запиті [33]. Застосунки, що використовують GraphQL, працюють швидко і стабільно, тому що вони контролюють дані, які отримують, а не сервер. GraphQL надає повний і зрозумілий опис даних у вашому API, дає клієнтам можливість запитувати саме те, що їм потрібно, і нічого зайвого, а також полегшує розробку API з часом [33]. GraphQL створює уніфікований API для всього вашого застосунку, не обмежуючись конкретним механізмом зберігання даних [33]. Загалом, GraphQL ставить на перший план досвід розробки та клієнтську сторону [35].

2.5 Вибір бази даних

Системи управління базами даних (СУБД) можуть бути різних типів, таких як реляційні, документо-орієнтовані, ключ-значення тощо [37-39]. Реляційна база даних - це інформація, де дані впорядковані та пов'язані між собою певними відносинами [37]. Документо-орієнтована база даних - це база даних, де дані зберігаються у вигляді документів, які містять ключові значення та набір полів [38, 39]. Ключ-значення база даних - це сховище даних, яке зберігає пари ключ/значення [40]. Три найпопулярніші бази даних для веброзробки: MySQL, PostgreSQL, MongoDB [41, 42].

MongoDB є документоорієнтованою нереляційною базою даних, що зберігає дані у вигляді JSON-подібних документів, які зберігаються в бінарному форматі BSON [42-45]. MongoDB є відкритим програмним забезпеченням і дозволяє оперувати документами, які зберігаються в колекціях [46]. База даних MongoDB не вимагає опису схеми таблиць, що дозволяє легко змінювати структуру даних [44]. MongoDB дозволяє використовувати горизонтальне масштабування для зберігання і пошуку файлів [42]. MongoDB є дуже популярною базою даних для веброзробки і використовується в багатьох великих проєктах [43].

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Основні відомості про розробку

У сучасній веброзробці дуже важливу роль грає архітектура закладена у проєкт. Це дозволяє набагато легше його розроблювати і підтримувати у майбутньому, що дуже сильно зменшує затрати бізнесу на розробку продукту.

Архітектура фронтенд і бекенд застосунків відрізняються одна від одної і мають різні регламентовані рекомендації. Також важливою частиною є форматування коду, що теж впливає на швидкість і якість розробки. Для цього найчастіше використовують ESLint - інструмент лінтерів, що підключається та налаштовується, який виявляє проблемні патерни в JavaScript-кодi.

Загальна файлова система фронтенд частини застосунку наступна:

- файли конфігурацій;
- файли перевикористовуваних компонентів;
- файли допоміжних функцій;
- файли обгортки для сторінок;
- файли сторінок;
- файли сервісів;
- файли сховища;
- файли стилів;
- файли типів.

У таблиці 3.1 наведено детальну файлову структуру з вказанням імен.

Таблиця 3.1 – Детальна файлова структура проєкту

Тип файлу	Назва файлу
1	2
Файл конфігурацій	next.config.js
	.eslintrc.json
	tsconfig.json

Продовження таблиці 3.1

1	2
Файл перевикористовуємих компонентів	DataTable.tsx
	Drawer.tsx
	EmptyPlaceholder.tsx
	RouteGuard.tsx
	Sidebar.tsx
	ImagePlaceholder.tsx
Файли допоміжних функцій	activity.ts
	notifications.ts
	routes.ts
	stringHelper.ts
	useActivity.ts
	useColumns.ts
	useDrawer.ts
	useWidget.ts
Файли обгортки для сторінок	AuthLayout.tsx
	CommonLayout.tsx
Файли сторінок	Login.tsx
	Make.tsx
	Model.tsx
	Oem.tsx
	Vehicle.tsx
	Inspections.tsx
Файл сервісів	api.ts
	cloudinary.ts
	app.redux.ts
	inspection.redux.ts
	make.redux.ts
	oem.redux.ts
	vehicle.redux.ts
	_constants.scss
	globals.scss
	styles.module.scss
	DataTypes.d.ts

Файли компонентів, сторінок і обгорток – це файли у форматі `jsx`, які містять у собі JavaScript код який відпрацьовує на стороні браузера і повертає розмітку сторінки, яку потім браузер відображає на екрані користувача. Файли стилів задають позиції, кольори і інші стилі для відображення. Файли сховища використовуються для передачі даних між різними компонентами і сторінками. Сервіси це класи які містять методи для роботи з зовнішніми інструментами.

Загальна структура файлів бекенд частини застосунку:

- файли конфігурацій;
- файли маршрутизації;
- файли моделей;
- файли контролерів;
- файли допоміжних функцій.

У таблиці 3.2 наведено детальну структуру з вказанням імен.

Таблиця 3.2 – Детальна файлова система проєкту

Тип файлу	Назва файлу
1	2
Файл конфігурацій	<code>.eslintrc.js</code>
	<code>tsconfig.json</code>
Файли маршрутизації	<code>inspection.route.js</code>
	<code>make.route.js</code>
	<code>model.route.js</code>
	<code>oem.route.js</code>
	<code>vehicle.route.js</code>
Файли контролерів	<code>controller.create.js</code>
	<code>controller.deleteById.js</code>
	<code>controller.getAll.js</code>
	<code>controller.getById.js</code>
	<code>controller.updateById.js</code>

Продовження таблиці 3.2

1	2
Файли моделей	inspection.model.js
	make.model.js
	model.model.js
	oem.model.js
	vehicle.model.js
Файли допоміжних функцій	errorMiddleware.js
	Joi.js
	getNext.js
	schemaTransformPlugin.js

Система маршрутизації дозволяє керувати вхідними запитами до сервера, створюючи унікальні адреси для кожного запиту. Залежно від маршруту, по якому надійшов запит, викликаються функції контролерів. Основним завданням цих функцій є передача змін до класів моделей та отримання даних з них. Отримані дані форматуються в спеціальний вигляд і відправляються як відповідь з сервера.

Для роботи з базою даних використовуються моделі, які описують поля елементів, їх тип.

3.2 Реалізація інтерфейсу

Інтерфейс застосунку створений з компонентів - це невеликі частини інтерфейсу, які можна багаторазово використовувати в залежності від ситуації. У React компоненти можуть приймати аргументи, з котрими вони будуть працювати.

Більшість компонентів і стилізації береться із бібліотеки Ant Design – це бібліотека компонентів для React, яка надає широкий спектр готових елементів інтерфейсу, таких як кнопки, форми, таблиці, меню, каруселі та інші. Бібліотека має велику кількість налаштувань та можливостей для кастомізації, що дозволяє легко створювати унікальний дизайн для вебзастосунків. Ant Design має велику спільноту користувачів та розробників, яка активно підтримує та розвиває

бібліотеку. Бібліотека є безкоштовною та відкритою для використання та розповсюдження.

У React компоненти мають життєвий цикл, який проходить через кілька фаз, від ініціалізації до знищення. Цей життєвий цикл поділяється на чотири частини: монтування, оновлення, обробка помилок та демонтування. Коли компонент передається на відображення у DOM, він отримує властивості або пропси, до яких можна отримати доступ за допомогою `this.props`. Після цього починається весь процес життєвого циклу. Зверніть увагу, що React-компонент може не пройти через усі фази, і може бути змонтований і демонтований наступної хвилини без будь-яких оновлень або обробки помилок.

Під час фази монтування викликаються декілька методів життєвого циклу, такі як `constructor()`, `render()`, `componentDidMount()` та `getDerivedStateFromProps()`. Фаза оновлення запускається, коли стан або пропси компонента оновлюються, і її методи життєвого циклу включають `shouldComponentUpdate()`, `render()`, `getSnapshotBeforeUpdate()`, `componentDidUpdate()` і `getDerivedStateFromProps()`.

На додаток до стандартних методів життєвого циклу React, React Native Navigation покращує зареєстровані компоненти двома додатковими подіями життєвого циклу: `componentDidAppear` та `componentDidDisappear`. Розуміння життєвого циклу компонента є важливим у розробці React, оскільки це допомагає керувати змінами стану, виконувати дії та оптимізувати продуктивність.

Хуки - це новий підхід до життєвого циклу компонентів у React, який дає змогу використовувати стан, життєвий цикл, контекст і рефи без написання класів. Хуки не замінюють класи, але надають більш прямий доступ до API вже знайомих понять, таких як аргументи, стан, контекст, посилання на інші компоненти і життєвий цикл. Хуки дають змогу функціональним компонентам мати внутрішній стан і виконувати побічні ефекти, як-от завантаження даних і підписка на події. Хуки стану, як-от `useState()`, `useEffect()`, `useContext()`, `useReducer()`, `useCallback()`, і `useMemo()`, є основними хуками, які використовуються в React. Хуки стають головним способом написання React-компонентів, і їх використання

рекомендується замість класів. Життєвий цикл React компонентів показаний на рис. 3.1.

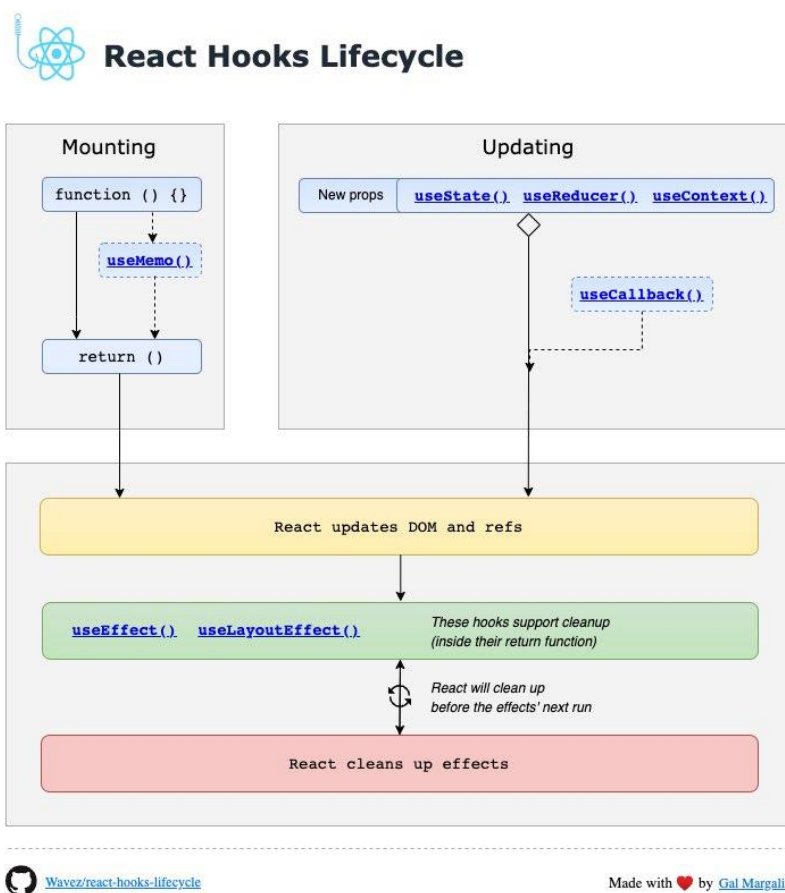


Рисунок 3.1 – Життєвий цикл компонентів React [47]

3.2.1 Інтерфейс таблиці даних

Таблиця даних – компонент інтерфейсу з бібліотеки Ant Design, вона виводить рядок з назвами колонок і порядково показує записи. Для додаткових функцій таблиця має колонку “Actions”, у якій є кнопки редагування і видалення запису. Також таблиця має колонку з картинкою асоційованою з відповідним записом.

Інтерфейс зображено на рис. 3.2.

Add new

Refresh

	Id	Title	Image	Actions
☐	6463...	Toyota		 
☐	6463...	Toyota 2		 
☐	646f...	Volkswagen		 
☐	646f...	Audi		 
☐	646f...	fSDFSDf		 

Рисунок 3.2 – Розроблений інтерфейс таблиці даних

3.2.2 Інтерфейс навігаційного меню

Навігаційне меню - це один з найважливіших компонентів у вебсайтах. Якщо навігація продумана і гарно реалізована, то користувач зможе орієнтуватися по сайту на інтуїтивному рівні. І навпаки, якщо навігація реалізовано складно, має дуже багато випадających підменю – це відштовхне користувача.

В даному вебзастосунку навігаційне меню реалізовано як додаткова секція з краю екрана, яку можна розширити і сховати. Самі навігаційні елементи не мають випадających списків. Такий підхід дуже легко сприймається користувачем і не займає багато простору.

Інтерфейс зображено на рис. 3.3.

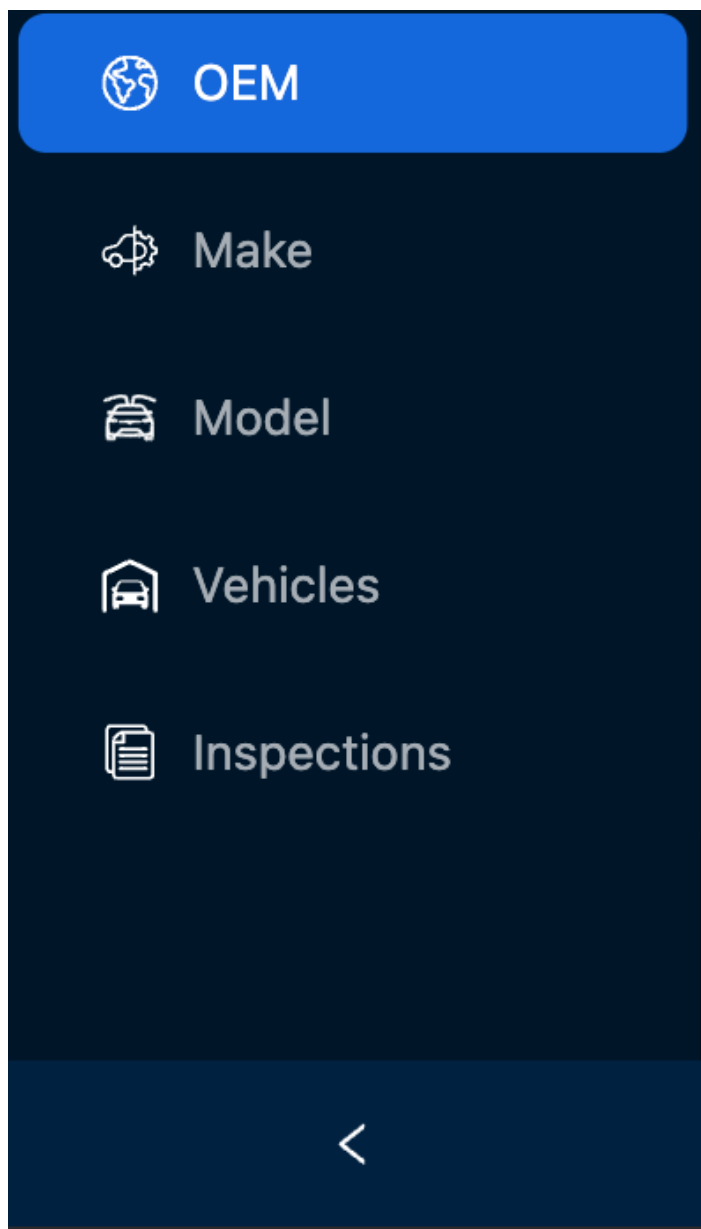


Рисунок 3.3 – Розроблений інтерфейс навігаційного меню

3.2.3 Інтерфейс форми створення/редагування

Форма створення і редагування - це компонент, що з'являється коли користувач натискає кнопку, яка виконує дію створення.

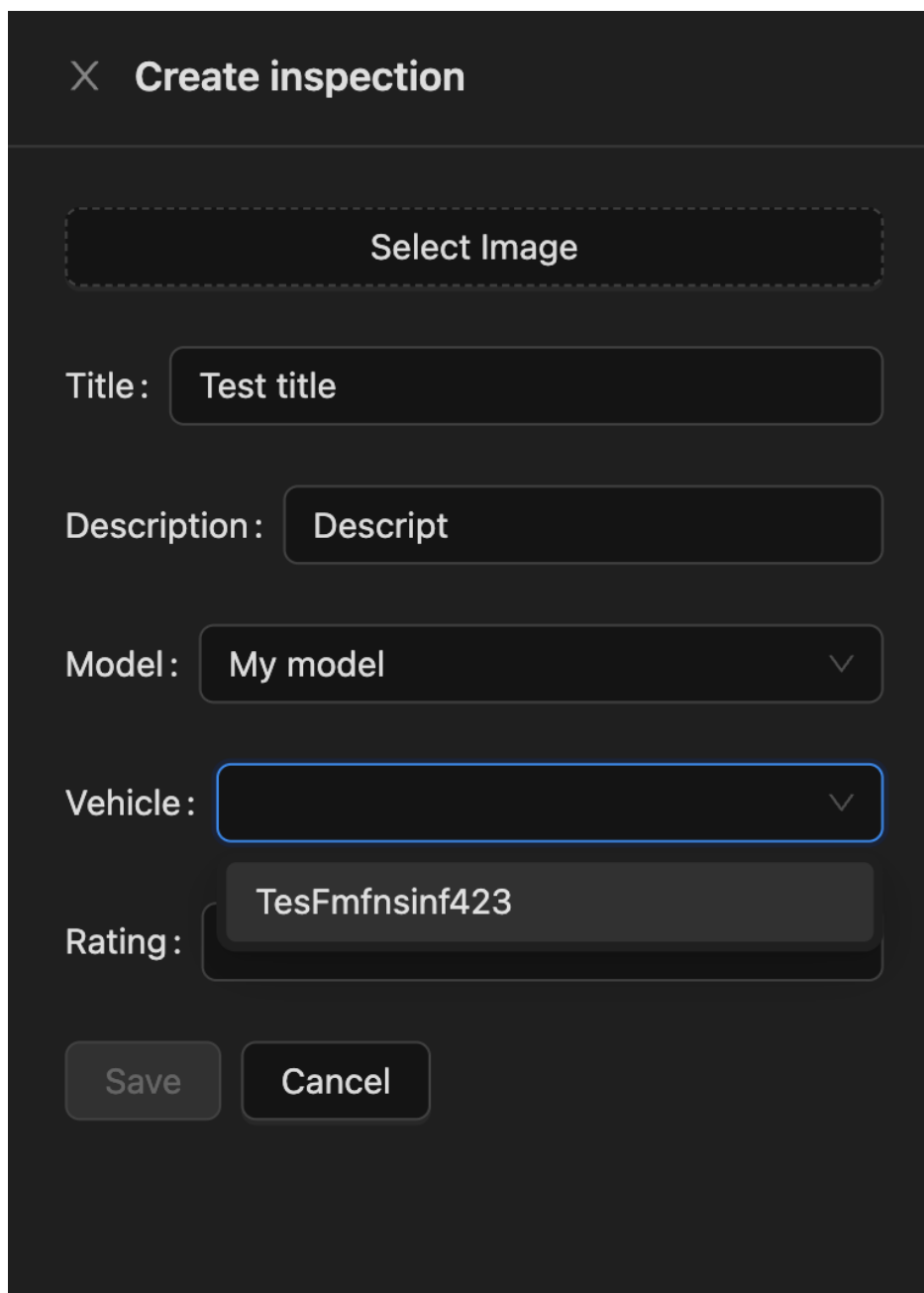
Ця форма реалізована як випливаюче вікно, що може містити декілька типів компонентів для вводу даних:

- текстове поле;
- поле з випадаючим списком для вибору;
- завантаження зображення;

– кнопки.

Після заповнення даних і виклику дії відправлення – дані формують спеціальний об’єкт, який у свою чергу відправляється на сервер для збереження.

Інтерфейс зображено на рис. 3.4.



The image shows a dark-themed modal window titled "Create inspection" with a close button (X) in the top left corner. The form contains the following elements:

- A dashed rectangular box with the text "Select Image" in the center.
- A "Title:" label followed by a text input field containing "Test title".
- A "Description:" label followed by a text input field containing "Descript".
- A "Model:" label followed by a dropdown menu showing "My model" with a downward arrow.
- A "Vehicle:" label followed by a dropdown menu that is currently empty with a downward arrow.
- A "Rating:" label followed by a text input field containing "TesFmfnsinf423".
- At the bottom, there are two buttons: "Save" and "Cancel".

Рисунок 3.4 – Розроблений інтерфейс форми створення запису

3.3 Система маршрутизації

Всі вебзастосунки базують свою роботу на системі маршрутизації. Система маршрутизації складається з набору адрес, які сервер слухає для отримання запитів, виконання відповідних дій і надсилення результату. Це є основою клієнт-серверної архітектури.

Запити в системі можуть мати чотири типи, відповідно до принципу CRUD:

- GET - використовується для отримання (читання) записів;
- POST - використовується для створення нових записів;
- PATCH - використовується для оновлення існуючих записів;
- DELETE - використовується для видалення записів.

В таблиці 3.3 представлений список основних розроблених маршрутів разом з їх описом та типами.

Таблиця 3.3 – Розроблені маршрути

Адреса	Тип запиту	Опис
/api/v1/:entity	POST	Створення запису
	GET	Отримання всіх записів
/api/v1/:entity/:id	PUT	Редагування запису
	DELETE	Видалення запису
	GET	Отримання запису за ідентифікатором

Нижче наведено приклади деяких маршрутів і їх об'єднання

```
import { Router } from 'express'
import { getAll, getById, create, deleteById, updateById } from './controller'

const router = Router()
const root = Router()

router.get('/', getAll)
router.get('/:id', getById)
router.post('/', create)
router.delete('/:id', deleteById)
```

```

router.put('/:id', updateById)

root.use('/inspection', router)

export default root

import Oem from './oem'
import Make from './make'
import Model from './model'
import Vehicle from './vehicle'
import Inspection from './inspection'

const publicRoutes = [Oem, Make, Model, Vehicle, Inspection]
const privateRoutes = []

export default {
  publicRoutes,
  privateRoutes
}

import express, { json } from 'express'
import routes from '@routes/api/v1/index'
import errorMiddleware from '@middlewares/errorMiddleware'
import cors from 'cors'

const app = express()

app.use(json())
app.use(cors())

app.use('/api/v1', routes.publicRoutes)
app.use(errorMiddleware)

export default app

```

Реалізовані маршрути представляють базову модель сервера з усіма мінімальними можливостями.

3.4 Структура бази даних

Структура бази даних в системі MongoDB відіграє ключову роль у збереженні та організації даних. MongoDB є документоорієнтованою базою даних, де дані представляються у вигляді документів у форматі JSON. Дані зберігаються у колекціях, які відповідають таблицям у реляційних базах даних.

MongoDB має спеціальний тип даних `object-id`, який використовується для збереження ідентифікаторів документів. За допомогою цього типу даних можна створювати ієрархічні структури як наприклад, Користувач-Аккаунт.

Для цього застосунку були розроблені такі моделі:

а) Original Equipment Manufacturer (OEM):

- Title – string – назва;
- Image – string – посилання на зображення;
- createdAt – date – дата і час створення документу;
- updatedAt – date – дата і час останнього оновлення документа.

б) Make:

- 1) Title – string – назва;
- 2) Oem – object-id – ідентифікатор на відповідний документ OEM;
- 3) Image – string – посилання на зображення;
- 4) createdAt – date – дата і час створення документу;
- 5) updatedAt – date – дата і час останнього оновлення документа.

в) Model:

- 1) Title – string – назва;
- 2) Make – object-id – ідентифікатор на відповідний документ Make;
- 3) Image – string – посилання на зображення;
- 4) createdAt – date – дата і час створення документу;
- 5) updatedAt – date – дата і час останнього оновлення документа.

г) Vehicle:

- 1) Vin – string – унікальний реєстраційний номер автомобіля;
- 2) Model – object-id – ідентифікатор на відповідний документ Model;

- 3) createdAt – date – дата і час створення документу;
- 4) updatedAt – date – дата і час останнього оновлення документа.

г) Inspection:

- 1) Title – string – назва;
- 2) Description – string – опис інспекції;
- 3) Rating – number – оцінка інспекції;
- 4) Left – number – позиція маркера відносно X координати;
- 5) Top – number – позиція маркера відносно Y координати;
- 6) Vehicle – object-id – ідентифікатор на відповідний документ

OEM;

- 7) Image – string – посилання на зображення;
- 8) createdAt – date – дата і час створення документу;
- 9) updatedAt – date – дата і час останнього оновлення документа.

На рис. 3.5 зображена схема бази даних.

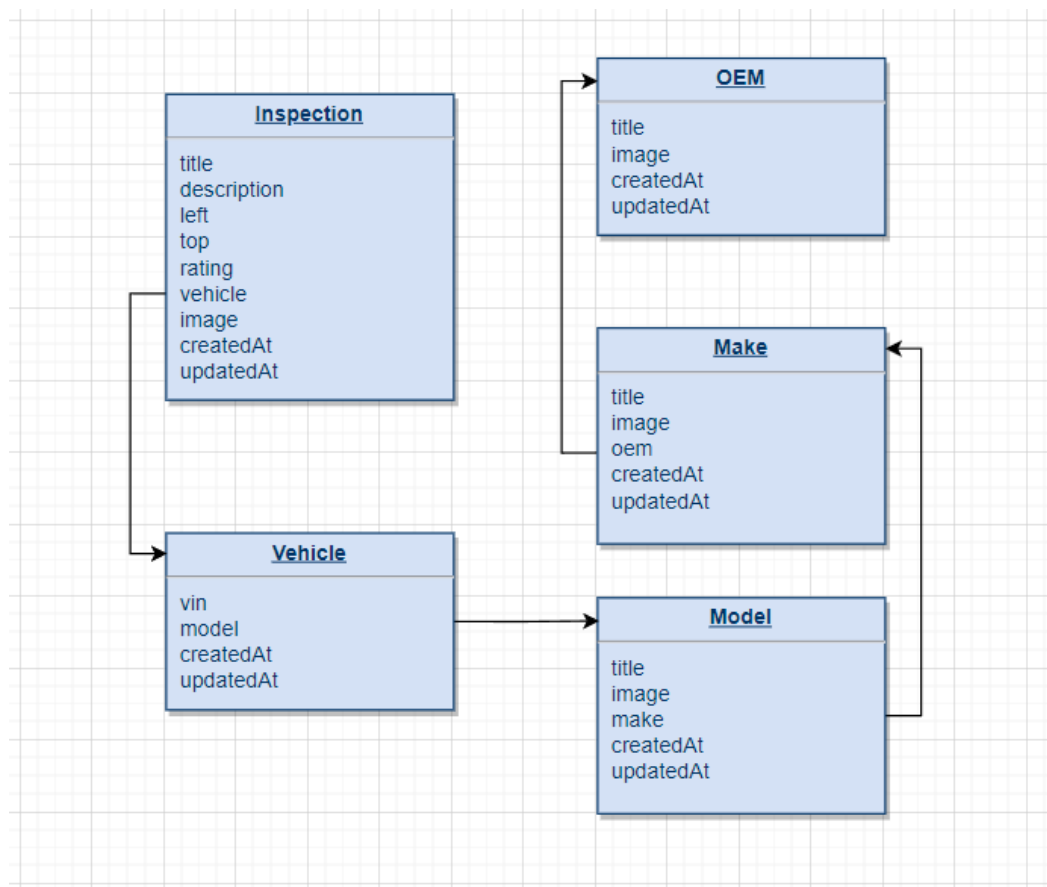


Рисунок 3.5 – Схема бази даних

Нижче приведений приклад створення моделі для бази даних:

```
const schema = new Schema(
  {
    title: {
      type: SchemaTypes.String,
      minlength: 2,
      required: true,
      index: 'text'
    },
    description: {
      type: SchemaTypes.String,
      required: true,
    },
    rating: {
      type: SchemaTypes.Number,
      required: true,
    },
    left: {
      type: SchemaTypes.Number,
      required: true,
    },
    top: {
      type: SchemaTypes.Number,
      required: true,
    },
    image: {
      type: SchemaTypes.String,
      required: false
    },
    vehicle: {
      type: SchemaTypes.ObjectId,
      required: true,
      ref: 'vehicle'
    }
  },
  {
    timestamps: true,
    statics: {
      textSearch(text) {
```

```

        return this.find(
            {
                $text: {$search: text}
            },
            {
                score: {$meta: 'textScore'}
            }
        )
    }
},
methods: {
    rename() {
        console.log(this)
    }
}
}
)

schema.plugin(schemaTransformPlugin)

const m = model('inspection', schema)

export default m

```

3.5 Розроблені контролери

Контролери виконують ключову роль у забезпеченні взаємодії між фронтендом і базою даних. Контролери виступають як посередники, які приймають запити від фронтенд-інтерфейсу, обробляють їх і передають необхідні дані до бази даних. Вони відповідають за обробку різноманітних запитів, валідацію даних, виконання бізнес-логіки і повернення відповідей назад до фронтенду.

Використання контролерів дозволяє розподілити функціональність системи на логічні блоки, спрощує розробку, тестування і підтримку коду, а також підвищує масштабованість та надійність бекенд застосунків

Ось перелік розроблених контролерів:

- контролер OEM;

- контролер Make;
- контролер Model;
- контролер Vehicle;
- контролер Inspection.

Майже всі контролери які приймають якісь дані з клієнтського запиту мають логіку валідації цих даних, ось приклад такої валідації:

```
const jBody = Joi.object({
  title: Joi.string().min(2).required(),
  vehicle: Joi.objectId().required(),
  description: Joi.string().min(2).required(),
  rating: Joi.number().required(),
  left: Joi.number().required(),
  top: Joi.number().required(),
  image: Joi.string()
})
try {
  await jBody.validateAsync(body)
} catch (e) {
  next(e)
  return
}
```

Цей контролер відповідає за керування документами моделі Inspection

3.6 Висновки за розділом

В даному розділі було наведено основні відомості про розробку фронтенд і бекенд частини вебзастосунку для перевірки вживаних автомобілів. Детально описано структуру маршрутизації, бази даних.

Наведено опис деяких компонентів інтерфейсу з їх функціоналом.

Наведено схему бази даних і її моделей.

Наведено перелік контролерів для керування запитами та їх реалізацію у виді коду.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Розгортання на локальному комп'ютері

Для подальшого вдосконалення створеного продукту потрібно мати базові знання роботи з командним рядком.

Встановити оточення NodeJS і менеджер пакетів npm. Ці інструменти потрібні для розгортання, запуску і роботи застосунку

4.2 Інсталяція застосунку

Для роботи застосунку потрібно мати встановлене оточення NodeJS версії не менше 16.

Для інсталяції застосунку потрібно використовувати менеджер пакетів npm, що забезпечить встановлення усіх потрібних бібліотек для роботи застосунку. Це виконується за допомогою команди “npm ci”

Після даних кроків проєкт готовий для запуску, усі подальші команди також містяться у файлі “README.md”

4.3 Розгортання бази даних

Так як базою даних є MongoDB - її не треба розгортати локально. Для цього існує хмарне рішення, у якому можна розгорнути кластер з базою даних, після чого її можна відразу використовувати.

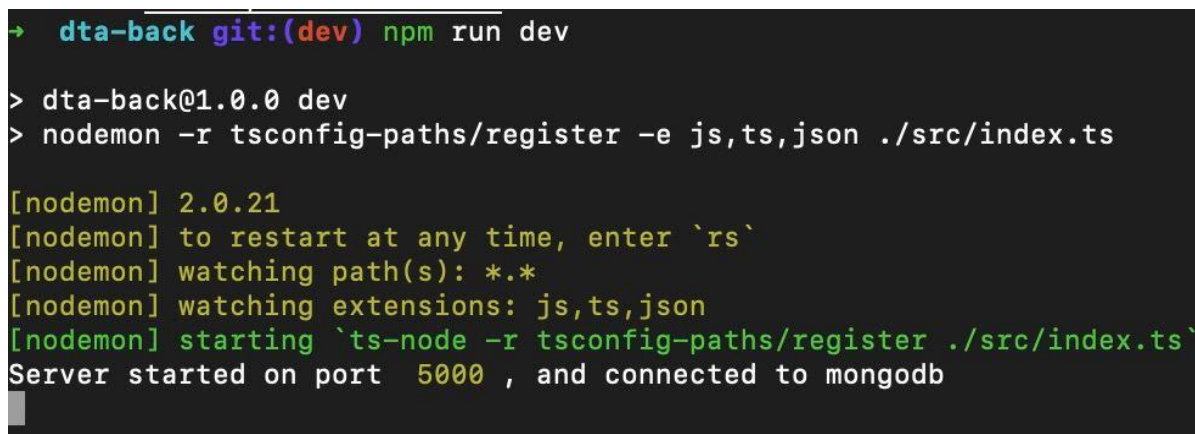
Для цього потрібно зайти на офіційний сайт MongoDB, зареєструватися або авторизуватися, натиснути кнопку “Create cluster”, обрати конфігурацію бази даних, рекомендується залишати стандартні налаштування, з якими можна розгорнути безплатний кластер і використовувати його з деякими обмеженнями.

Після цього можна скопіювати посилання на кластер і скопіювати його у файл .env.

4.4 Запуск сервера

Клієнтська частина застосунку повинна мати зв'язок з сервером, для того щоб отримувати і відсилати дані, якщо сервер не буде запущений, або буде запущений з помилкою – клієнт виведе помилку про підключення.

Для запуску сервера, розробник повинен, у командному рядку, перейти в директорію, яка відповідає директорії розташування файлу package.json бекенд частини і виконати команду “npm run dev”. Після цього розробник має побачити повідомлення про успішний запуск, приклад зображено на рис. 4.1.



```
→ dta-back git:(dev) npm run dev

> dta-back@1.0.0 dev
> nodemon -r tsconfig-paths/register -e js,ts,json ./src/index.ts

[nodemon] 2.0.21
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,ts,json
[nodemon] starting `ts-node -r tsconfig-paths/register ./src/index.ts`
Server started on port 5000 , and connected to mongodb
```

Рисунок 4.1 – Приклад вдалого запуску сервера

У випадку якщо розробник побачить повідомлення про помилку – він має дослідити детальну інформацію помилки, яка буде виведено у консоль, і вирішити її.

Для подальшої перевірки, що сервер працює справно, розробник може виконати запит на отримання стану за допомогою програми Postman або іншим зручним способом. Він розміщений за маршрутом “/api/status”. Приклад запиту і його відповідь при правильній роботі серверу зображено на рис. 4.2.

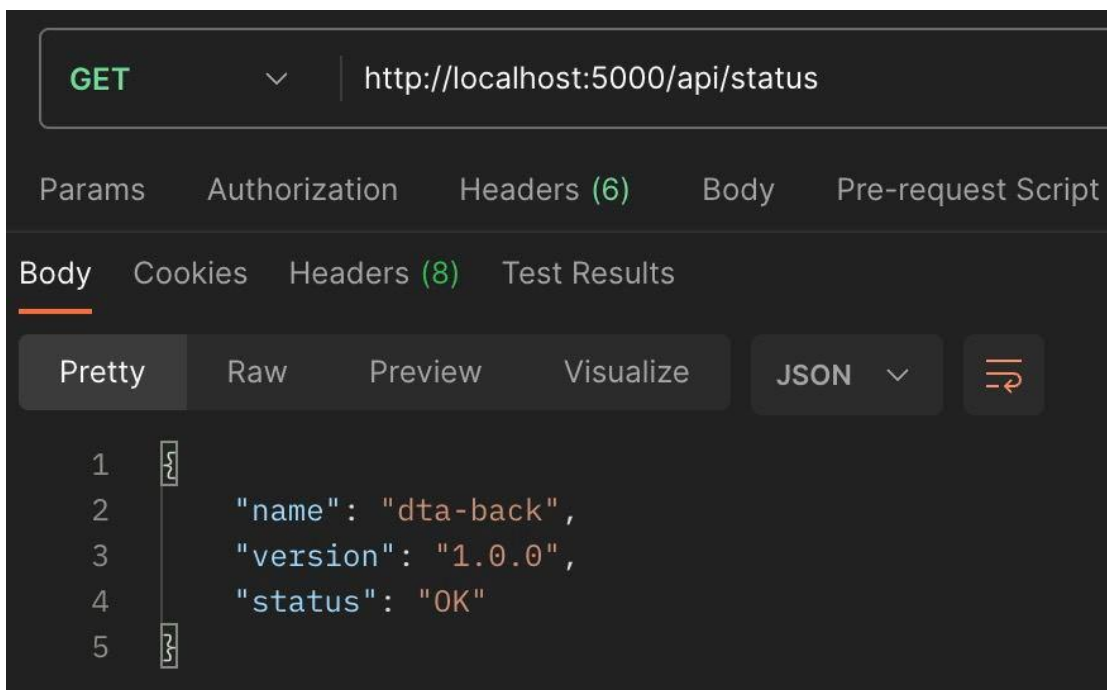


Рисунок 4.2 – Відповідь запиту статусу

Також для перевірки правильного зв'язку сервера з базою даних рекомендується виконати запит на отримання будь-якого типу документа, наприклад, на рис. 4.3 зображено правильну відповідь сервера на запит отримання списку OEM.

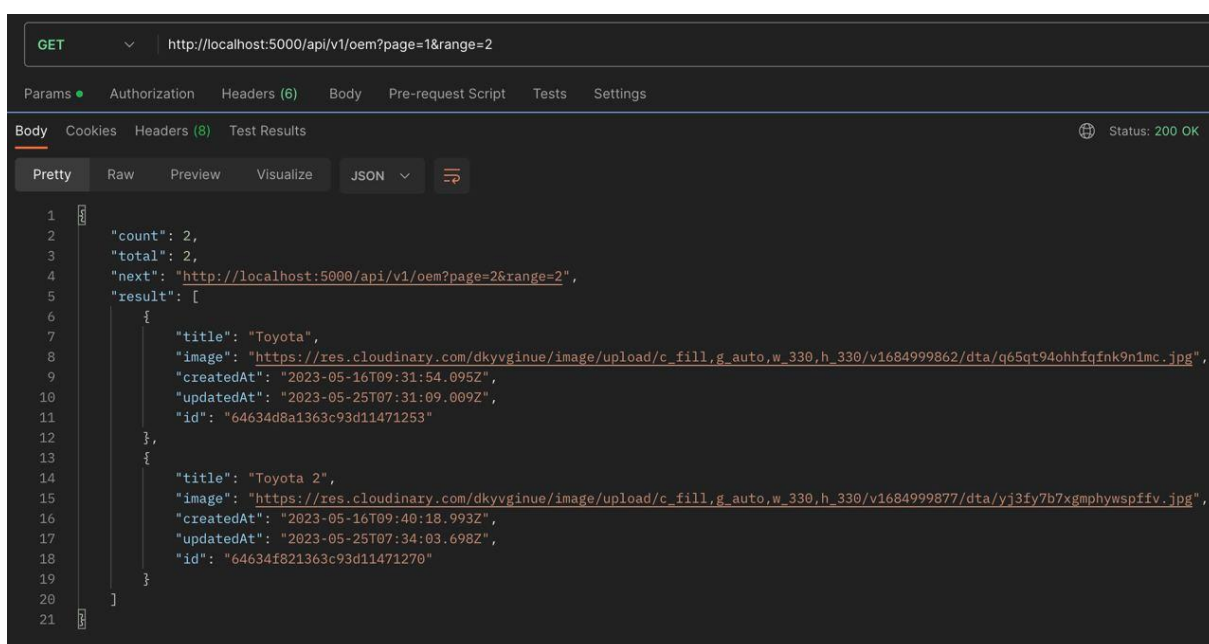


Рисунок 4.3 – Приклад правильної відповіді сервера

4.5 Запуск клієнта

Після вдалого запуску сервера, потрібно запустити фронтенд. Для цього розробник має перейти у директорію, яка відповідає директорії розміщення файлу `package.json` клієнтської частини і виконати команду “`npm run dev`”. Після цього розробник має побачити повідомлення про успішний запуск, приклад зображено на рис. 4.4.

```
→ dta-front git:(connect-redux) npm run dev

> atlas-refactor@0.1.0 dev
> next dev

ready - started server on 0.0.0.0:3000, url: http://localhost:3000
info - Loaded env from /Users/obez/Desktop/dta/dta-front/.env.local
(node:94978) ExperimentalWarning: The Fetch API is an experimental feature. This
feature could change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
event - compiled client and server successfully in 1130 ms (2940 modules)
```

Рисунок 4.4 – Приклад вдалого запуску фронтенда

Якщо запуск був вдалим, розробник має перейти за вебадресою “`localhost:3000`” і побачити вебсайт застосунку за вебадресою “`http://localhost:3000`”. Вигляд сторінки зображено на рис. 4.5.

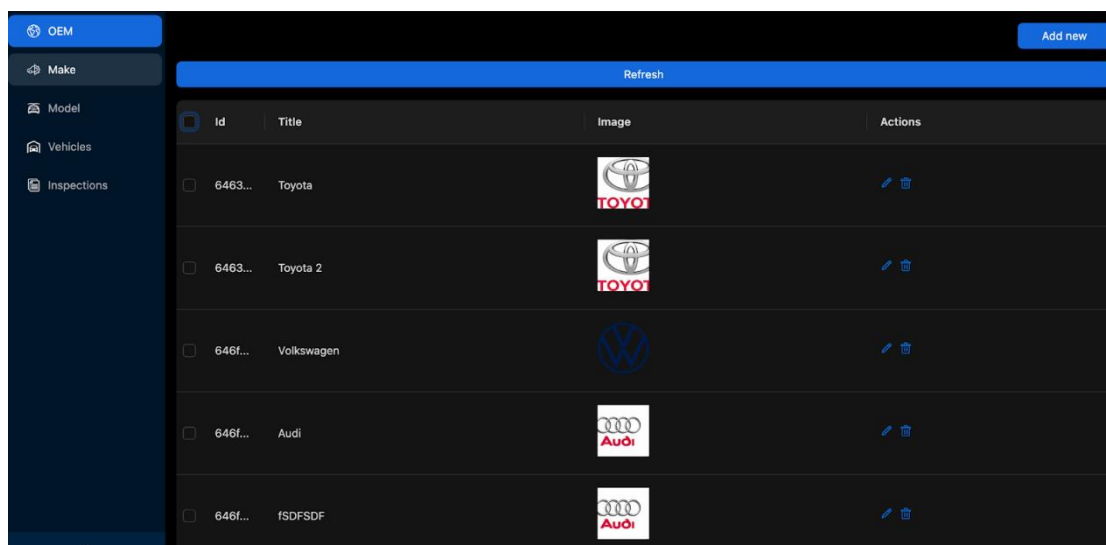


Рисунок 4.5 – Вигляд сторінки після вдалого запуску

У випадку якщо виникнуть якісь помилки, їх можна побачити як у консолі, так і напряму у вікні браузера.

4.4 Висновки за розділом

В даному розділі були розглянуті основні кроки, необхідні для подальшої розробки та вдосконалення застосунку.

Описано принцип розгортання сервера розробки та налаштувань відповідного програмного забезпечення.

Наведено детальну інструкцію з інсталяції проекту і всіх зовнішніх бібліотек

Описано механізми розгортання структури бази даних, її налаштування та підключення до неї генерації даних.

Наведено декілька прикладів запитів до сервера та їх відповіді.

5 КЕРІВНИЦТВО КОРИСТУВАЧА

5.1 Загальні відомості

Для того щоб користувач міг користуватися продуктом, сервера повинні бути розгорнуті і доступ до них має бути за протоколами HTTP/HTTPS

При вході у застосунок користувач побачить екран авторизації, у якому він має буде ввести свої дані для авторизації, сторінка зображена на рис. 5.1.

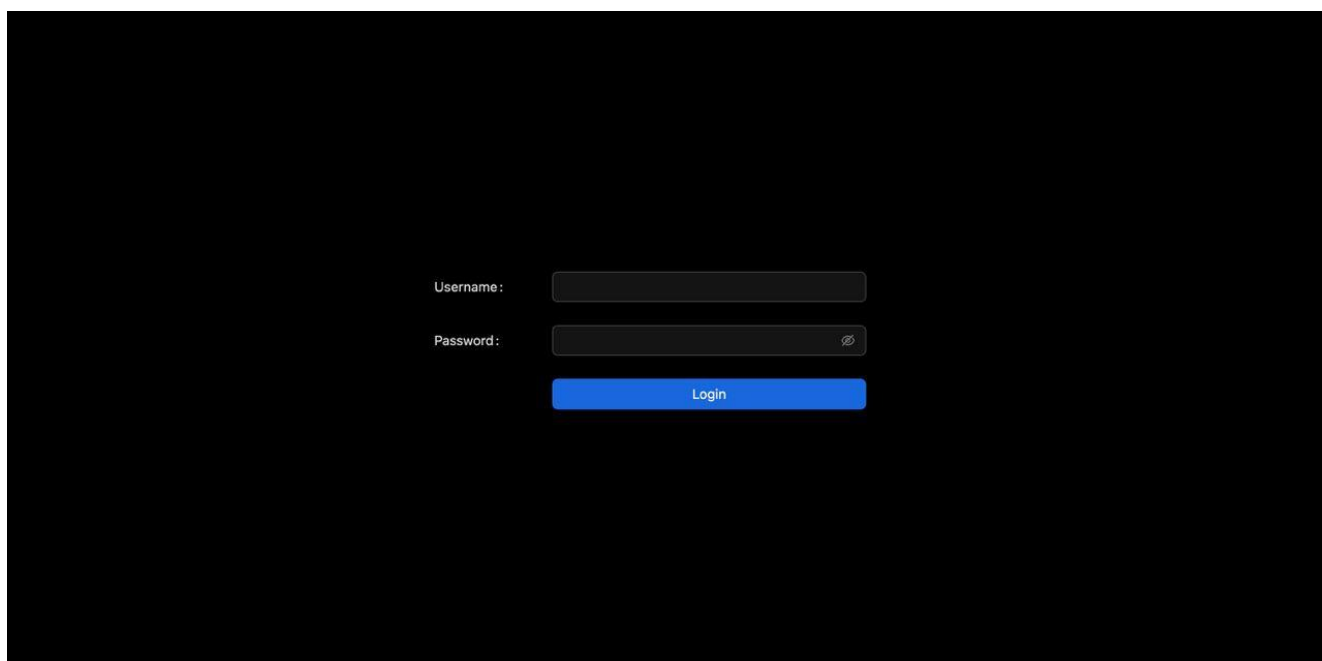


Рисунок 5.1 – Сторінка авторизації

Після успішної авторизації користувач буде перенаправлений на сторінку з таблицею існуючих даних, де він може переглянути їх, відредагувати і створити нові записи. Ця сторінка з описом функціональних елементів зображено на рис. 5.2.

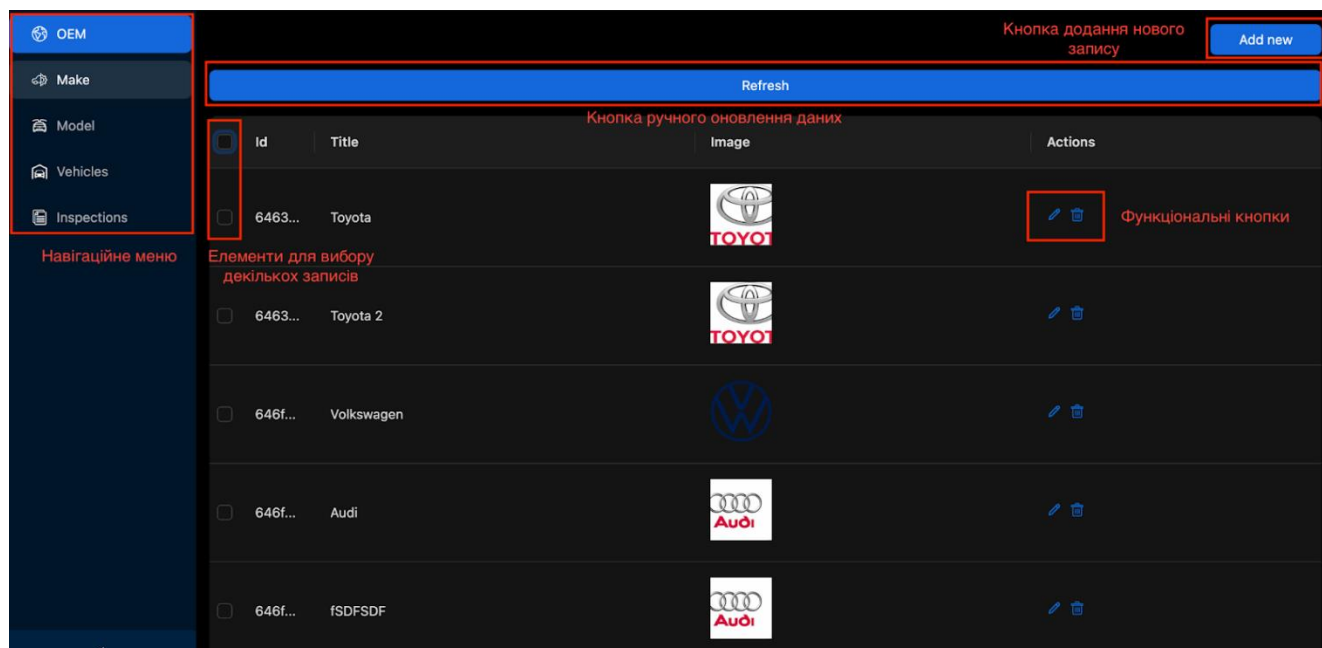


Рисунок 5.2 – Сторінка з описом функціональних елементів

Для створення нового запису Oem, користувач має зайти у розділ OEM і натиснути кнопку “Add new”, після чого відкриється секція зі створенням нового запису, він зображений на рис. 5.3.

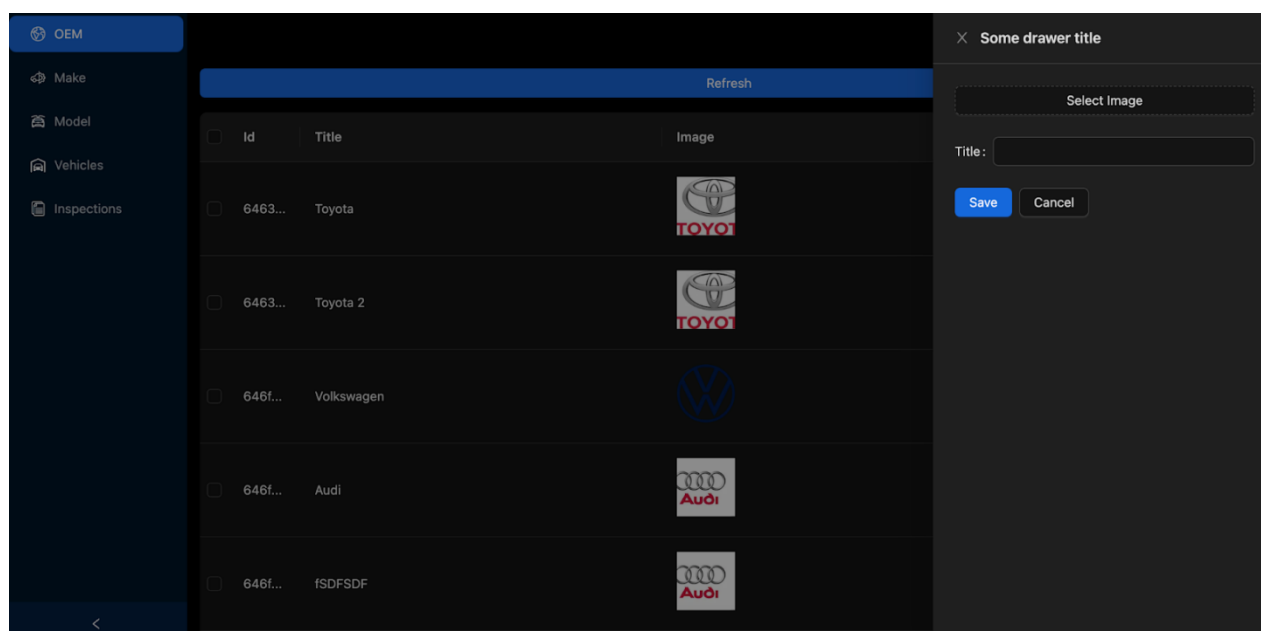


Рисунок 5.3 – Секція створення запису Oem

У цій секції користувач може завантажити зображення запису, натиснувши кнопку “Select Image”, що відкриває вікно завантаження зображення на рис. 5.4.

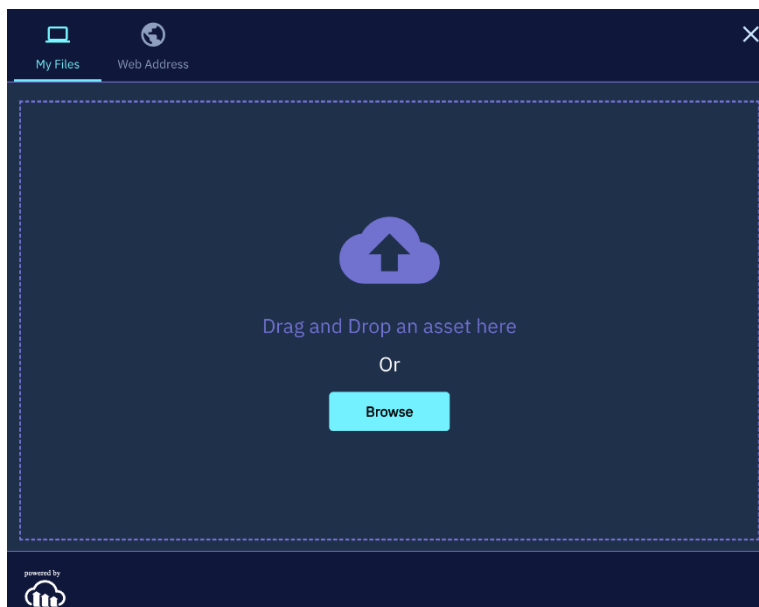


Рисунок 5.4 – Вікно завантаження зображення

Після завантаження зображення, користувач має заповнити всі інші поля.

На прикладі OEM, на рис. 5.5 зображено заповнену форму, після чого користувач може натиснути “Save”.

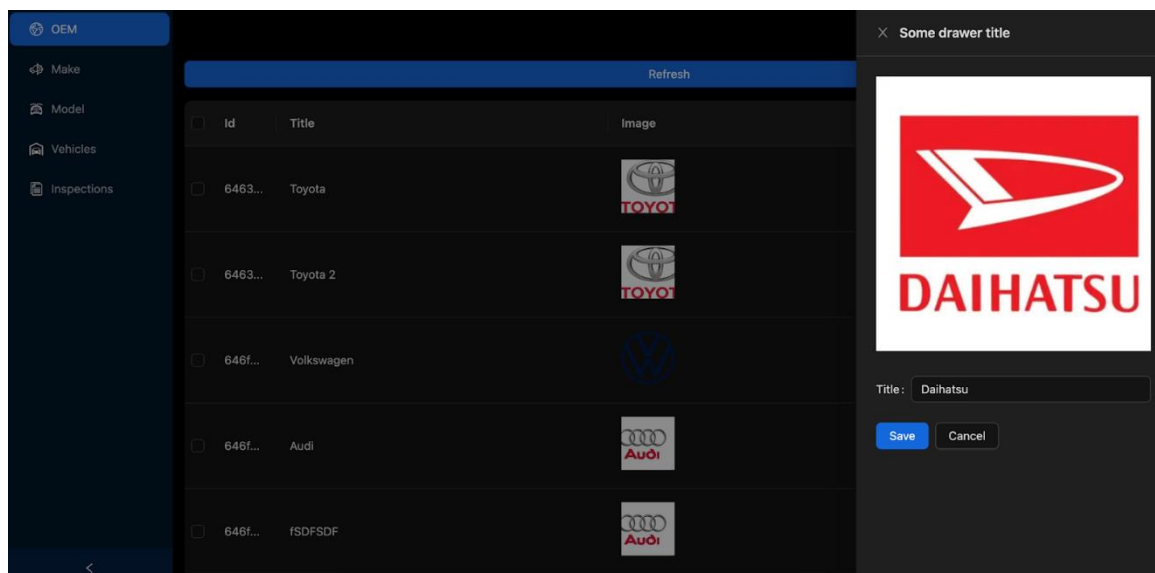


Рисунок 5.5 – Заповнена форма створення OEM

У деяких секціях створення користувач може вибрати інший запис з бази даних за допомогою спеціального елементу інтерфейса, цей варіант зображено на рис. 5.6.

Рисунок 5.6 – Спеціальний елемент інтерфейсу для вибору існуючого запису

Після успішного створення або редагування запису користувач може побачити повідомлення у верхній частині екрану, воно зображено на рис. 5.7.

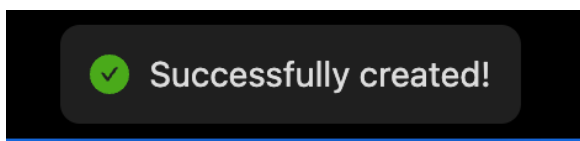
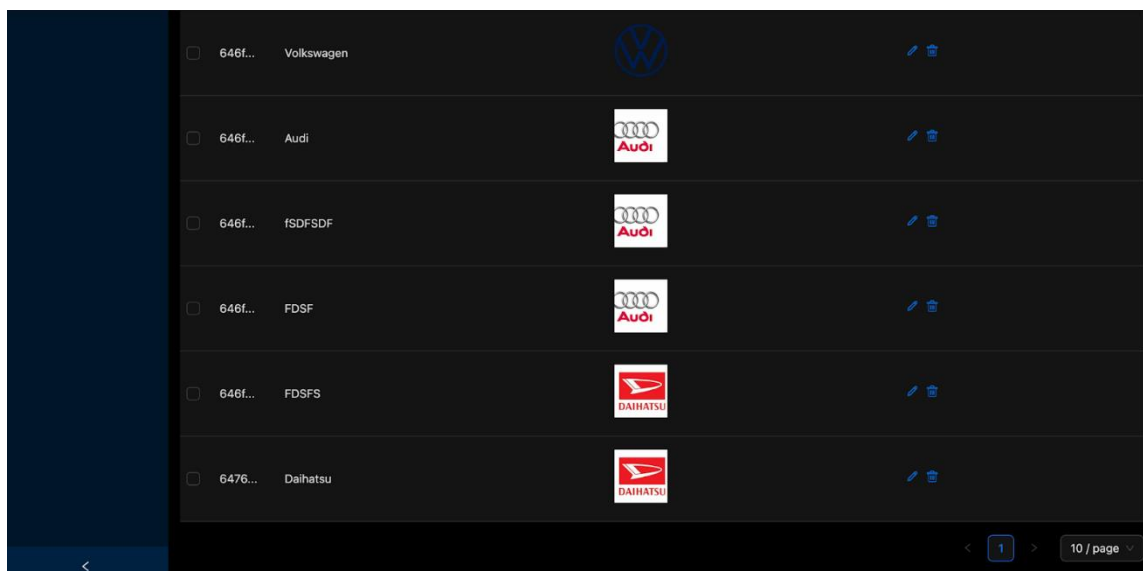


Рисунок 5.7 – Повідомлення про успішне створення

Після цього – запис з’явиться у таблиці зі всіма іншими даними. Приклад на рис. 5.8.



☐	646f...	Volkswagen		 
☐	646f...	Audi		 
☐	646f...	fSDFSDF		 
☐	646f...	FDSF		 
☐	646f...	FDSFS		 
☐	6476...	Daihatsu		 

Рисунок 5.8 – Приклад створеного OEM у таблиці

Якщо ж створення або редагування було не успішним, у випадку коли сервер повернув помилку, користувач може побачити повідомлення про помилку на рис 5.9.

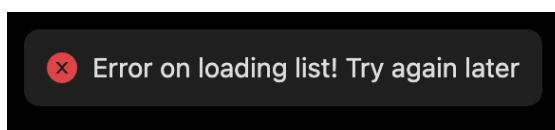


Рисунок 5.9 – Повідомлення про помилку

Після проходження критичного шляху – створення OEM, Make, Model, Vehicle, користувач може створити “Inspection” – точку інтересу для перевірки даного автомобіля. Для цього у розділі “Inspections” потрібно натиснути кнопку “Add new”, і, як і раніше, з’явиться форма для створення нового запису. При завантаженні зображення у цій секції, користувачу буде запропоновано додати точку інтересу на зображенні, це показано на рис. 5.10.

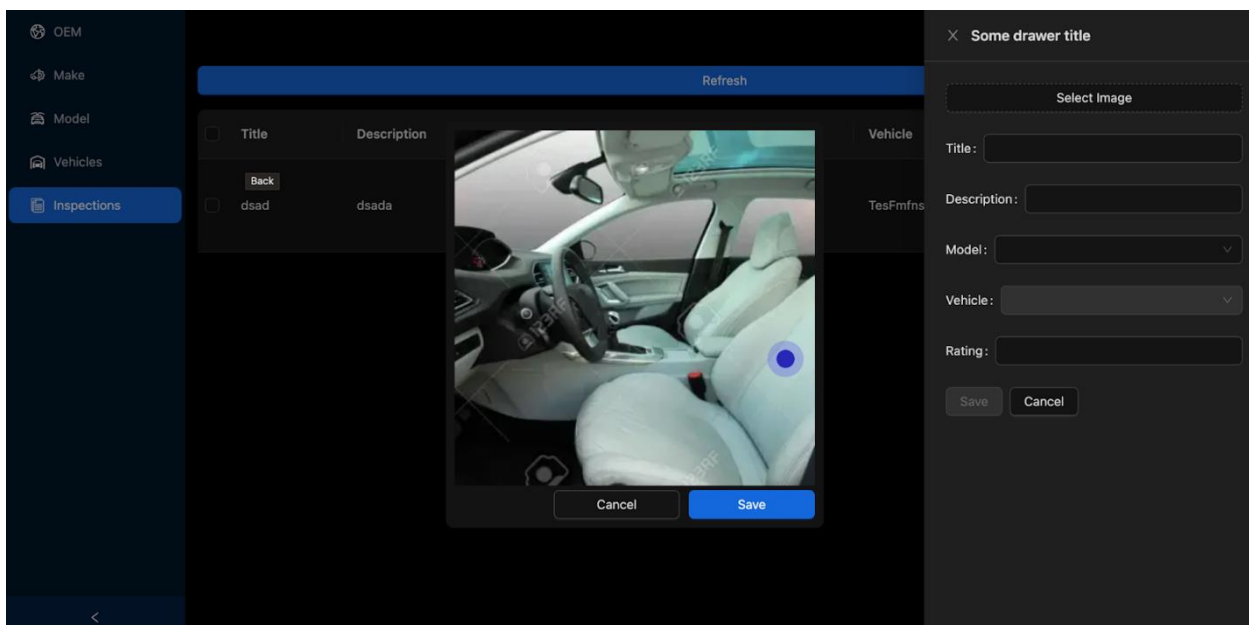


Рисунок 5.10 – Приклад додавання точки інтересу на зображення

Після збереження змін і заповнення всіх полів форми, вона повинна виглядати як на рис. 5.11.

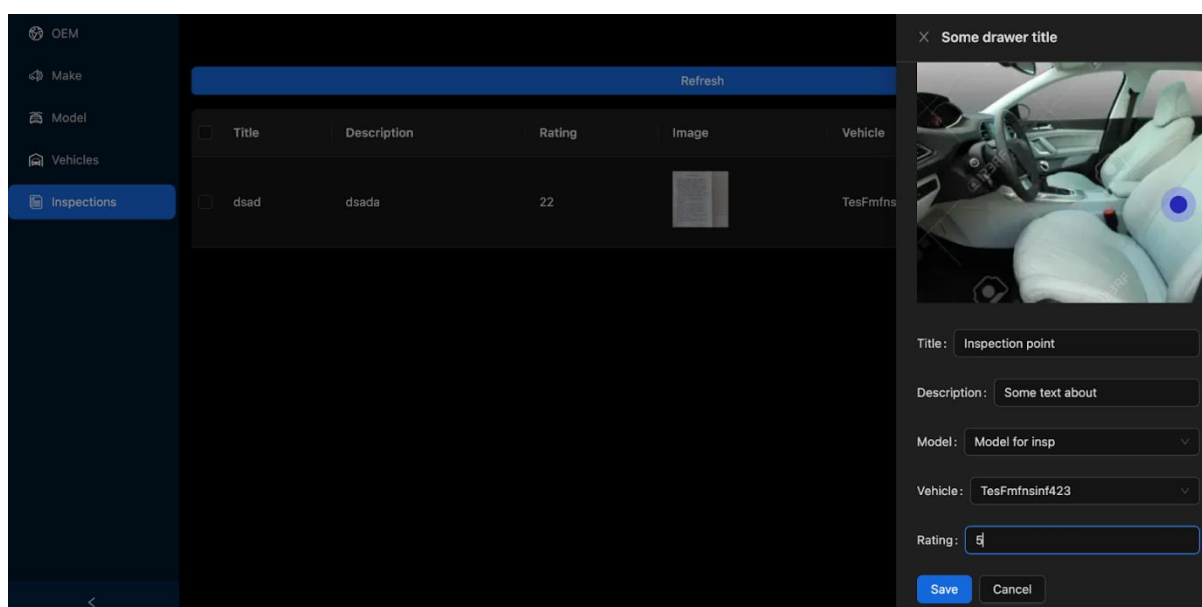


Рисунок 5.11 – Приклад заповнення форми створення інспекції

Після цього, користувач може натиснути кнопку “Save”, і цей запис збережеться у таблицю, де його можна буде видалити або відредагувати. Це зображено на рис. 5.12.

	Title	Description	Rating	Image	Vehicle	Actions
☐	dsad	dsada	22		TesFmfnsinf423	
☐	Inspection point	Some text about	5		TesFmfnsinf423	

Рисунок 5.12 – Відображення створеної інспекції у таблиці

5.2 Висновки за розділом

Таким чином, в розділі було розглянуто основні принципи звернення клієнтського програмного забезпечення до розробленого застосунку. Наведено приклади елементів інтерфейсу і опис функціональних елементів інтерфейсу.

ВИСНОВКИ

У процесі виконання дипломної роботи був розроблений повноцінний вебзастосунок з фронтенд і бекенд.

Розробку вебзастосунку було виконано за допомогою вебфреймворку ExpressJS з визначенням структури даних на основі використання бази даних, вимог до програми, розробленням структури програми і алгоритму роботи на основі цих вимог, реалізацією відповідних одиниць вебзастосунку. Для розробки використано мови програмування JavaScript, мову розмітки HTML, мову стилю сторінок CSS, вебфреймворки Express, React, NextJS, базу даних MongoDB.

Тестування розробленого програмного забезпечення показало, що система функціонує правильно та відповідає всім поставленим вимогам.

Розроблене програмне забезпечення для перевірки вживаних авто є корисним інструментом для покупців, які хочуть уникнути купівлі автомобіля зі скрученим пробігом. Застосунок дозволяє створити запис фірми, яка створила автомобіль, запис моделі автомобіля, зв'язати їх між собою і створити інспекцію, щоб оцінити стан автомобіля. Це допоможе покупцям отримати детальну інформацію про стан транспортного засобу та дізнатися про можливі проблеми, які можуть виникнути в майбутньому.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Estimated monthly used-vehicle SAAR and volume - Cox Automotive Inc [Electronic resource]. – Access mode: <https://www.coxautoinc.com/market-insights/estimated-monthly-used-vehicle-saar-and-volume/>
2. Cargurus [Electronic resource]. – Access mode: <https://www.cargurus.com/>
3. AutoTrader [Electronic resource]. – Access mode: <https://www.autotrader.co.uk/>
4. CarDirect [Electronic resource]. – Access mode: <https://www.cardirect.com.au/>
5. 13 мов програмування, що будуть домінувати у 2022 році [Електронний ресурс]. – Режим доступу: <https://careers.easternpeak.com/blog/popular-programming-languages/>
6. Programming languages used in most popular websites - Wikipedia [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites
7. Web developer - Wikipedia [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Web_developer
8. 15 best programming languages for web development in 2023 [Electronic resource]. – Access mode: <https://www.globalmediainsight.com/blog/programming-languages-web-development/>
9. Comparison of JavaScript-based web frameworks - Wikipedia [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Comparison_of_JavaScript-based_web_frameworks
10. Electronic Kyiv-Mohyla Academy Institutional Repository [Електронний ресурс]. – Режим доступу: <https://ekmair.ukma.edu.ua/server/api/core/bitstreams/9b9964ba-073a-4c38-81bd-cb984d2f9f28/content>
11. Створюємо новий react-додаток – react [Електронний ресурс]. – Режим доступу: <https://uk.legacy.reactjs.org/docs/create-a-new-react-app.html>

12. Angular [Електронний ресурс]. – Режим доступу: <https://angular.io/>
13. VueJS [Електронний ресурс]. – Режим доступу: <https://vuejs.org/>
14. EmberJS [Електронний ресурс]. – Режим доступу: <https://emberjs.com/>
15. Svelte [Електронний ресурс]. – Режим доступу: <https://emberjs.com/>
16. ExpressJS [Електронний ресурс]. – Режим доступу: <https://expressjs.com/uk/>
17. ExpressJS Community [Electronic resource]. – Access mode: <https://expressjs.com/en/resources/community.html>
18. Koa [Електронний ресурс]. – Режим доступу: <https://koajs.com/>
19. Introduction to the Koa.js Framework [Electronic resource]. – Access mode: <https://www.section.io/engineering-education/introduction-to-koajs>
20. NestJS [Electronic resource]. – Access mode: <https://nestjs.com/>
21. Hapi [Electronic resource]. – Access mode: <https://hapi.dev/>
22. API - Wikipedia [Electronic resource]. – Access mode: <https://en.wikipedia.org/wiki/API>
23. Клієнт-серверна архітектура [Електронний ресурс]. – Режим доступу: <https://training.qatestlab.com/blog/technical-articles/client-server-architecture/>
24. Buckler C. What Is a REST API? – SitePoint [Electronic resource] / C. Buckler. – Access mode: <https://www.sitepoint.com/rest-api/>
25. Gillis A. S. What is REST API (RESTful API)? [Electronic resource] / A. S. Gillis. – Access mode: <https://www.techtarget.com/searchapparchitecture/definition/RESTful-API>
26. What is a REST API? [Electronic resource]. – Access mode: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>
27. What is REST - REST API Tutorial [Electronic resource]. – Access mode: <https://restfulapi.net/>
28. What is a REST API? | IBM [Electronic resource]. – Access mode: <https://www.ibm.com/topics/rest-apis>

29. Gillis A. S. What is SOAP? [Electronic resource] / A. S. Gillis. – Access mode: <https://www.techtarget.com/searchapparchitecture/definition/SOAP-Simple-Object-Access-Protocol>
30. SOAP Web Services Tutorial: What is SOAP Protocol? EXAMPLE [Electronic resource]. – Access mode: <https://www.guru99.com/soap-simple-object-access-protocol.html>
31. Simple object access protocol (SOAP) [Electronic resource]. – Access mode: <https://www.microfocus.com/documentation/silk-performer/205/en/silkperformer-205-webhelp-en/GUID-FEFE9379-8382-48C7-984D-55D98D6BFD37.html>
32. Simple Object Access Protocol [Electronic resource]. – Access mode: <https://www.sciencedirect.com/topics/computer-science/simple-object-access-protocol>
33. GraphQL | A query language for your API [Electronic resource]. – Access mode: <https://graphql.org/>
34. What is GraphQL? [Electronic resource]. – Access mode: <https://www.rubrik.com/insights/what-is-graphql>
35. Bakija I. REST, SOAP, or GraphQL [Electronic resource] / I. Bakija. – Access mode: <https://blog.treblle.com/rest-soap-or-graphql/>
36. What is GraphQL? [Electronic resource]. – Access mode: <https://konghq.com/learning-center/api-gateway/graphql>
37. NoSQL – вікіпедія [Електронний ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/NoSQL>
38. Document-oriented database - Wikipedia [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Document-oriented_database
39. Wikiwand - Document-oriented database [Electronic resource]. – Access mode: <https://www.wikiwand.com/en/Document-oriented%20database>
40. Швороб І. Б. Новий підхід до збереження слабоструктурованих медичних даних [Текст] / І. Б. Швороб // Науковий вісник НЛТУ України. – 2016. – Vol. 530, №1. – С. 1-5.
41. Інформаційні системи та реляційні бази даних [Електронний ресурс]. – Режим доступу: <https://www.uzhnu.edu.ua/en/infocentre/get/44979>

42. MongoDB – вікіпедія [Електронний ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/MongoDB>
43. Коротко про MongoDB (noSQL) [Електронний ресурс]. – Режим доступу: <https://ukr-technologies.blogspot.com/2020/03/mongodb-nosql.html>
44. Загальна інформація про mongodb [Електронний ресурс]. – Режим доступу: <https://www.ukraine.com.ua/uk/wiki/mongodb/overview/>
45. MongoDB - нереляційна база даних. Energy Blog [Електронний ресурс]. – Режим доступу: <https://energyfirefox.blogspot.com/2012/07/mongodb.html?m=1>
46. COMPUTER-INTEGRATED TECHNOLOGIES: EDUCATION, SCIENCE, PRODUCTION [Electronic resource]. – Access mode: <https://www.cit-journal.com.ua/index.php/cit/article/view/326/417>
47. GitHub - Wavez/react-hooks-lifecycle: React hooks lifecycle diagram: functional components lifecycle explained [Electronic resource]. – Access mode: <https://github.com/Wavez/react-hooks-lifecycle>

ДОДАТОК А
Слайди презентації

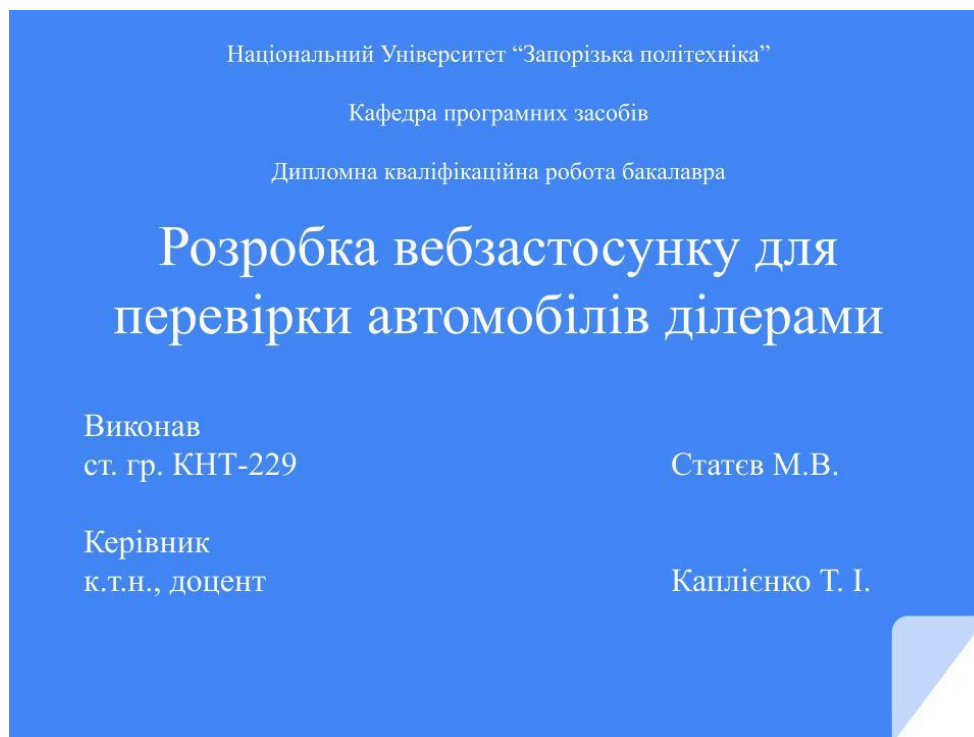


Рисунок А.1 – Слайд вступу

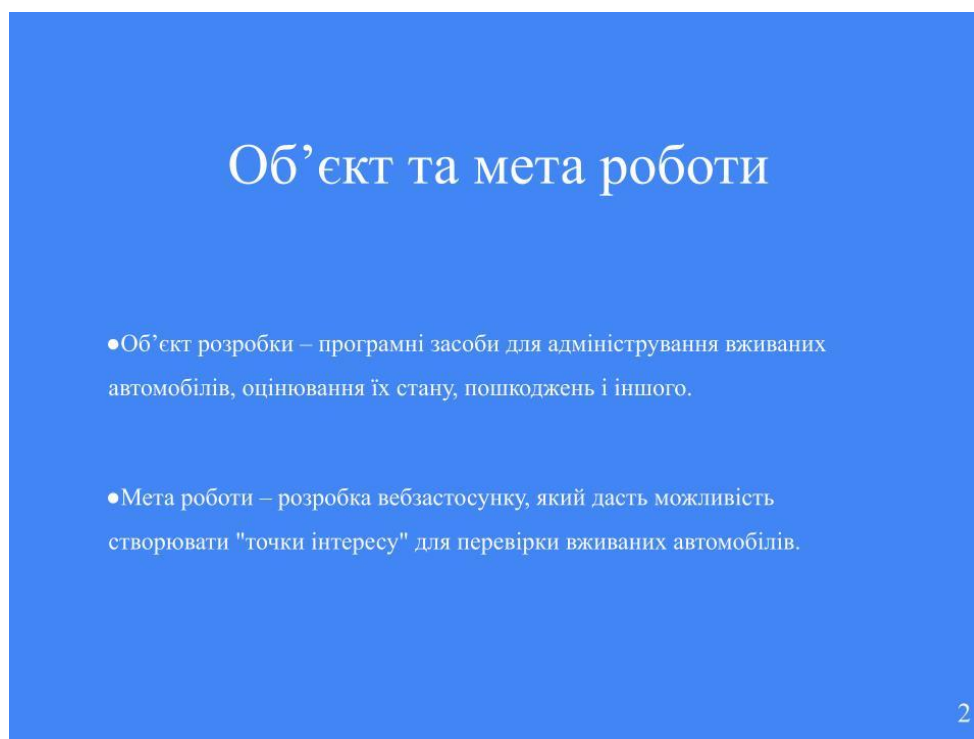


Рисунок А.2 – Слайд мети роботи

Задачі

- Аналіз предметної області
- Розробка структури системи
- Розробка схеми бази даних
- Розробка системи маршрутизації
- Розробка інтерфейсу

3

Рисунок А.3 – Слайд задач

Обґрунтування вибору мови програмування

Характеристика	Java	JavaScript	Python	PHP
Тип мови	Сильно типізована	Слабо типізована	Сильно типізована	Слабо типізована
Використання	Загальне	Фронтенд, бекенд	Загальне	Серверні сценарії, веб
Синтаксис	Об'єктно-орієнтований, Схожий на C	Схожий на C, Синтаксис заснований на об'єктах	Простий, Читабельний	Синтаксис заснований на C
Популярність	Висока	Дуже висока	Висока	Висока

4

Рисунок А.4 – Порівняння мов програмування

Обґрунтування вибору фреймворку

Фронтенд фреймворк

Фреймворк	Вебсайт	Версія	Архітектура	Величина спільноти	Підтримка
React	reactjs.org	17.0.2	Компонентна	Велика	Активна
Angular	angular.io	12.0.0	MVC	Велика	Активна
Vue.js	vuejs.org	3.2.20	Компонентна	Велика	Активна
Ember.js	emberjs.com	3.28.0	MVC	Середня	Активна
Svelte	svelte.dev	3.38.3	Компонентна	Зростаюча	Активна

5

Рисунок А.5 – Слайд порівняння фронтенд фреймворків

Обґрунтування вибору фреймворку

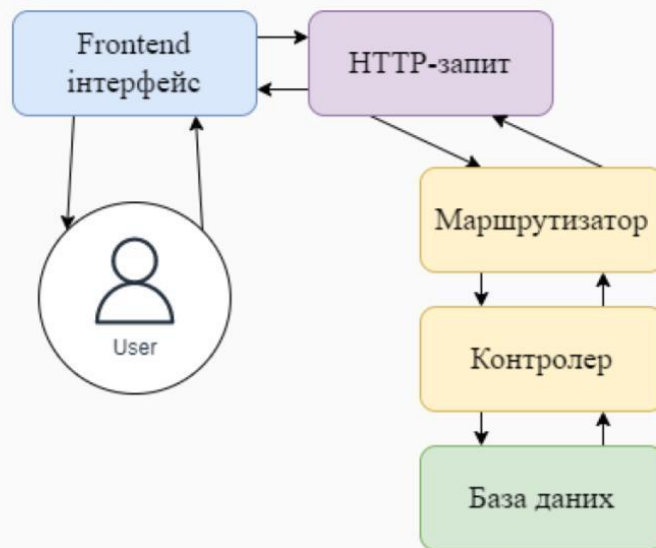
Бекенд фреймворк

Фреймворк	Основні особливості	Переваги	Недоліки
Express	Мінімальна кількість абстракцій	Простота використання	Менша функціональність порівняно з іншими фреймворками
Koa	Генератори ES6 для керування асинхронним кодом	Елегантний підхід до розробки	С вимога до розуміння концепції генераторів ES6
NestJS	Прогресивний та модульний фреймворк	Підтримка ООП та реактивного програмування	Вимагає вивчення TypeScript
Napi	Декларативний підхід до роутінгу	Вбудовані функціональні можливості	Менша популярність порівняно з іншими фреймворками

6

Рисунок А.6 – Слайд порівняння бекенд фреймворків

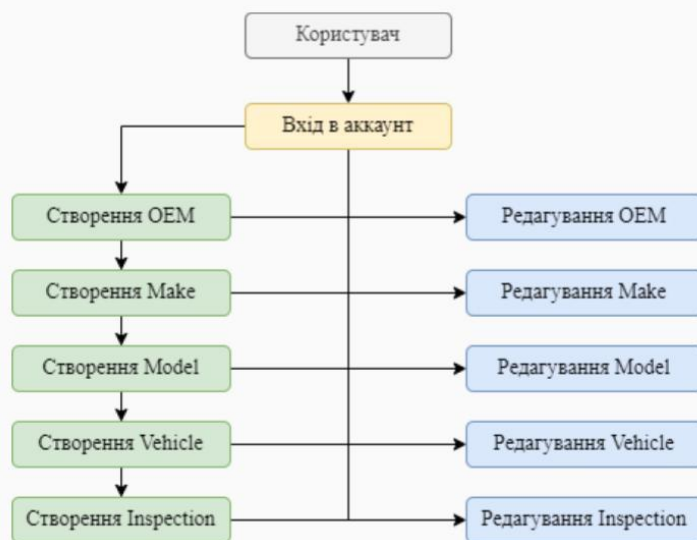
Схема взаємодії компонентів застосунку



7

Рисунок А.7 – Слайд схеми системи

Функціональна схема застосунку



8

Рисунок А.8 – Слайд функціональної схеми

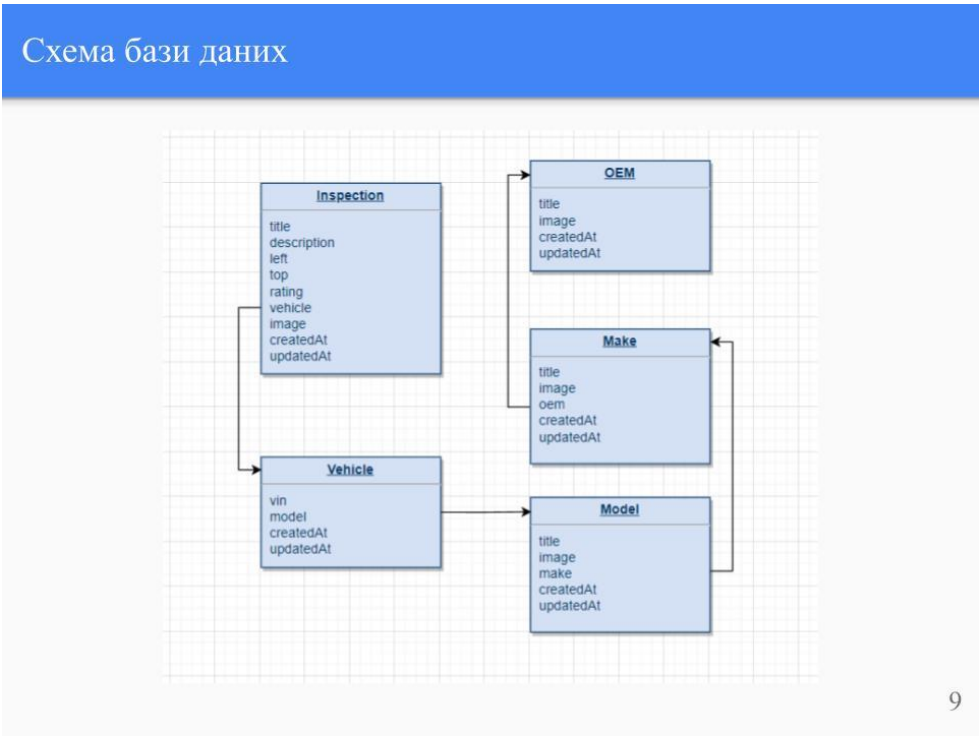


Рисунок А.9 – Слайд схеми бази даних

Розроблені елементи інтерфейсу

Таблиця даних

Refresh					Add new
☐	Id	Title	Image	Actions	
☐	6463...	Toyota		 	
☐	6463...	Toyota 2		 	
☐	646f...	Volkswagen		 	
☐	646f...	Audi		 	
☐	646f...	fsdfSDF		 	

10

Рисунок А.10 – Слайд інтерфейсу таблиці

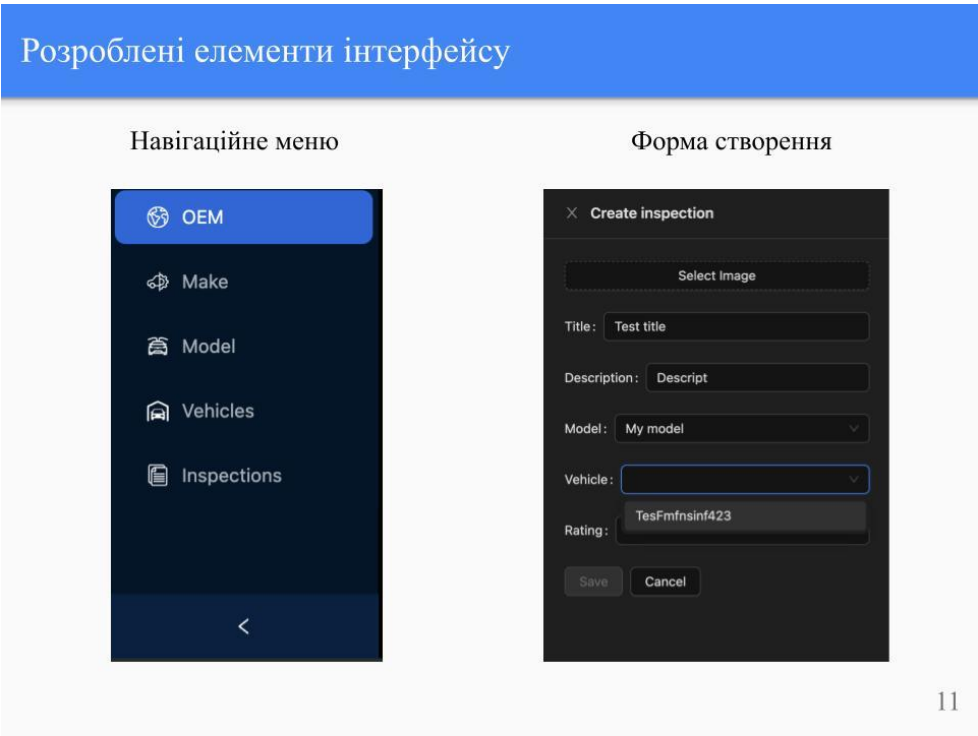


Рисунок А.11 – Слайд елементів інтерфейсу

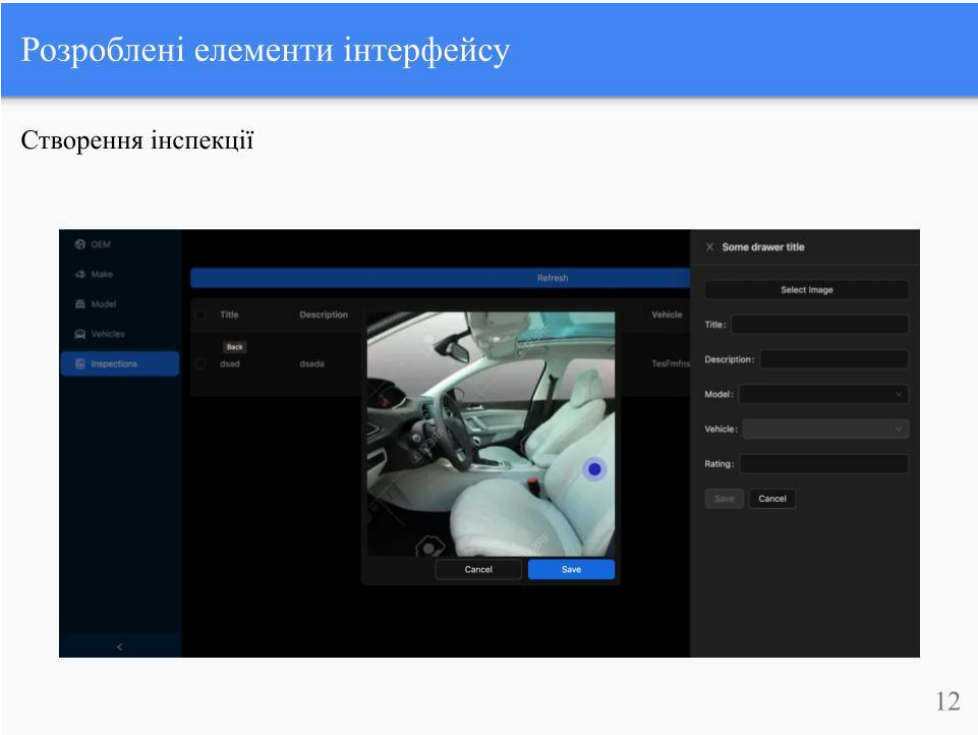
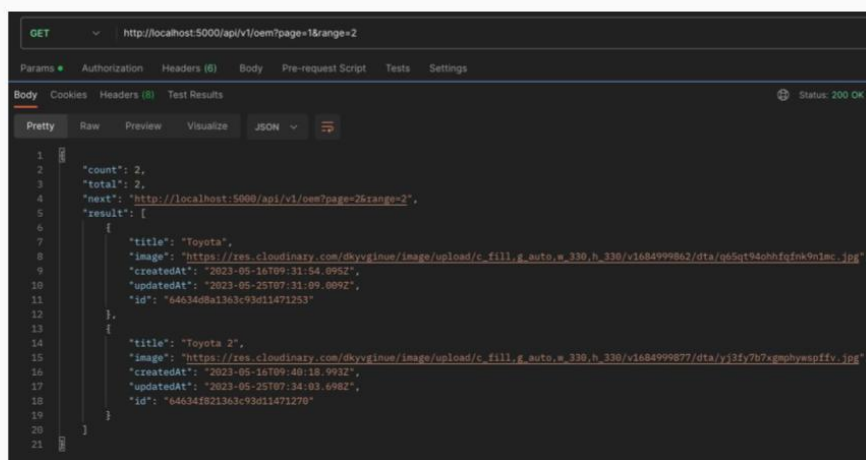


Рисунок А.12 – Слайд інтерфейсу створення інспекції

Приклад взаємодії інтерфейсу з сервером

Запит на отримання списку OEM



```

1 GET http://localhost:5000/api/v1/oem?page=1&range=2
2
3 Params Authorization Headers (6) Body Pre-request Script Tests Settings
4
5 Body Cookies Headers (6) Test Results Status: 200 OK
6
7 Pretty Raw Preview Visualize JSON
8
9 1
10 2 "count": 2,
11 3 "total": 2,
12 4 "next": "http://localhost:5000/api/v1/oem?page=2&range=2",
13 5 "result": [
14 6 {
15 7   "title": "Toyota",
16 8   "image": "https://res.cloudinary.com/dkyvginue/image/upload/c_fill,g_auto,w_330,h_330/v1684999862/dta/qe5qt94chhfgfnk9n1ec.jpg",
17 9   "createdAt": "2023-05-16T09:31:54.095Z",
18 10   "updatedAt": "2023-05-25T07:31:09.009Z",
19 11   "id": "64634d8a1363c93d11471253"
20 12 },
21 13 {
22 14   "title": "Toyota 2",
23 15   "image": "https://res.cloudinary.com/dkyvginue/image/upload/c_fill,g_auto,w_330,h_330/v1684999877/dta/yi3iy7b7xgmphyspfiv.jpg",
24 16   "createdAt": "2023-05-16T09:40:18.993Z",
25 17   "updatedAt": "2023-05-25T07:34:03.698Z",
26 18   "id": "64634f821363c93d11471278"
27 19 }
28 20 ]
29 21

```

13

Рисунок А.13 – Слайд відповіді сервера

Висновки

Проведено дослідження предметної області, окреслено базові поняття, що стосуються об'єкта дослідження. Визначено основну суть об'єкта дослідження, його провідну ідею та цілі.

Обрано та обгрунтовано інструменти розробки. Спроектовано і наведено структуру системи і функціональну схему.

Створено повноцінний вебзастосунок з фронтенд і бекенд сервісами, яке має можливості для масштабування і додавання нових сутностей і функціоналу.

14

Рисунок А.14 – Слайд висновків

ДОДАТОК Б**Код програми**

Б.1 Файл DataTable.tsx

```
import { Layout, Table, TableColumnType } from 'antd'
import styles from './styles.module.scss'
```

```
interface Props {
  data: object[]
  columns: TableColumnType<any>[]
  pageSize?: number
  loading?: boolean
  onSelect?: () => void
  onChangeSize?: (size: number) => void
}
```

```
const DataTable = ({
  data,
  columns,
  pageSize,
  loading,
  onChangeSize
}: Props) => {
  console.log('rerender')
  return (
    <Layout className={styles.wrapper}>
      <Table
        dataSource={data}
        columns={columns}
        rowSelection={{
          type: 'checkbox',
          onChange: (selectedKeys, selectedRow) => {
            console.log(selectedKeys, selectedRow)
          }
        }}
        className={styles.table}
        pagination={{
          pageSize,
          showSizeChanger: true,
          // hideOnSinglePage: true,
          onShowSizeChange: (_, size) => {
            onChangeSize?.(size)
          }
        }}
      />
    </Layout>
  )
}
```

```

        }
      }}
      rowKey="id"
      loading={loading}
    />
  </Layout>
)
}

export default DataTable

```

Б.2 Файл Drawer.tsx

```

import {
  Button,
  Drawer as AntdDrawer,
  Form,
  SelectProps,
  Input,
  Select,
  Space
} from 'antd'
import { memo } from 'react'
import ImagePlaceholder from 'src/modules/ImagePlaceholder'
import pinViewStyles from '../modules/pinView/styles.module.scss'

const MemoImagePlaceholder = memo(ImagePlaceholder)

interface InputField {
  type: 'input'
  value: string
  onChange: (val: string) => void
}

interface SelectField {
  type: 'select' | 'multiselect'
  options: SelectProps['options']
  value: SelectProps['value']
  onChange: (val: { label: string; value: string }) => void
}

```

```

interface ImageField {
  type: 'image'
  src: string
  onChange: (val: string) => void
  pin?: { left: number; top: number }
}

export type Field = (InputField | SelectField | ImageField) & {
  key: string
  label: string
  required?: boolean
  disabled?: boolean
}

export interface DrawerProps {
  title: string
  key: string
  onConfirm: () => void
  onClose: () => void
  onCancel?: () => void
  cleanUp?: () => void
  isOpen: boolean
  fields: Field[]
  loading?: boolean
}

const Drawer = ({
  title,
  onConfirm,
  onClose,
  onCancel,
  isOpen,
  fields,
  loading
}: DrawerProps) => {
  const valid = fields.every((field) =>
    field.required ? field?.value || field?.src : true
  )
}

```

```

return (
  <AntdDrawer title={title} onClose={onClose} open={isOpen}>
    <Form>
      {fields.map((field) => {
        switch (field.type) {
          case 'input':
            return (
              <Form.Item key={field.key} label={field.label}>
                <Input
                  value={field.value}
                  disabled={field.disabled}
                  onChange={(e) => field.onChange(e.target.value)}
                />
              </Form.Item>
            )
          case 'select':
          case 'multiselect':
            return (
              <Form.Item key={field.key} label={field.label}>
                <Select
                  labelInValue
                  value={field.value}
                  options={field.options}
                  onChange={field.onChange}
                  disabled={field.disabled}
                />
              </Form.Item>
            )
          case 'image':
            return (
              <Form.Item key={field.key}>
                <MemoImagePlaceholder
                  value={field.src}
                  onChange={(url) => field.onChange(url)}
                  disabled={field.disabled}
                />
                {field.pin && (
                  <div
                    className={pinViewStyles.pin}
                    style={{
                      left: `${field.pin?.left}%`,

```

```

        top: `${field.pin?.top}%`
      }}
    ></div>
  )}
</Form.Item>
)
default:
  return null
}
}}
<Form.Item>
  <Space>
    <Button
      onClick={onConfirm}
      loading={loading}
      type="primary"
      disabled={!valid}
    >
      Save
    </Button>
    <Button onClick={onCancel || onClose}>Cancel</Button>
  </Space>
</Form.Item>
</Form>
</AntdDrawer>
)
}

export default Drawer

```

Б.3 Файл EmptyPlaceholder.tsx

```

import styles from './styles.module.scss'

interface Props {
  title: string;
  create: () => void;
}

const EmptyPlaceholder = ({ title, create }: Props) => {

```

```

return (
  <div className={styles.placeholder_wrapper}>
    <p className={styles.text}>This is just the beginning of the journey!
      <br/>
      Lets    <span    className='action_span'    onClick={create}>add    first
{title}...</span>
    </p>
  </div>
)
}

export default EmptyPlaceholder

```

Б.4 Файл RouteGuard

```

import { useAppSelector } from '@store/index'
import routes from '@utils/routes'
import { useRouter } from 'next/router'
import { PropsWithChildren, useCallback, useEffect } from 'react'

const RouteGuard = ({ children }: PropsWithChildren) => {
  const router = useRouter()
  const isAuthenticated = useAppSelector(state => state.app.isAuthenticated)

  // If user authenticated and on login page - redirect him to home page
  useEffect(() => {
    if (router.pathname === routes.login && isAuthenticated) {
      router.push(routes.oem)
    }
  }, [isAuthenticated, router])

  const resolveRoutes = useCallback(() => {
    const pathname = router.pathname

    const isPublicUrl = [routes.login].includes(pathname)

    return isPublicUrl ? true : !isAuthenticated
  }, [router, isAuthenticated])

```

```

useEffect(() => {
  if (!resolveRoutes()) {
    router.push(routes.login)
  }
}, [resolveRoutes, router])

return resolveRoutes() ? <{children}> : null
}

export default RouteGuard

```

Б.5 Файл Sidebar.tsx

```

import { Layout, Menu, MenuProps } from 'antd'
import Image from 'next/image'
import { useState } from 'react'
import Link from 'next/link'
import routes from '@/utils/routes'
import { useRouter } from 'next/router'

type MenuItem = Required<MenuProps>['items'][number]
const menuItems: MenuItem[] = [
  {
    key: routes.oem,
    label: <Link href={routes.oem}>OEM</Link>,
    icon: <Image src="/assets/icons/oem.svg" alt="oem" width={16} height={16} />
  },
  {
    key: routes.make + '-group',
    label: <Link href={routes.make}>Make</Link>,
    icon: (
      <Image src="/assets/icons/makes.svg" alt="make" width={16} height={16} />
    )
  },
  {
    key: routes.model,
    label: <Link href={routes.model}>Model</Link>,
    icon: (
      <Image src="/assets/icons/model.svg" alt="model" width={16} height={16} />
    )
  }
]

```

```

    },
    {
      key: routes.vehicle,
      label: <Link href={routes.vehicle}>Vehicles</Link>,
      icon: (
        <Image
          src="/assets/icons/vehicles.svg"
          alt="vehicle type"
          width={16}
          height={16}
        />
      )
    },
    {
      key: routes.inspections,
      label: <Link href={routes.inspections}>Inspections</Link>,
      icon: (
        <Image
          src="/assets/icons/insp_points.svg"
          alt="inspection"
          width={16}
          height={16}
        />
      )
    }
  ]

```

```

const Sidebar = () => {
  const [isOpen, setIsOpen] = useState(true)
  const router = useRouter()

```

```

  const onCollapse = (value: boolean) => {
    setIsOpen(!value)
  }

```

```

  return (
    <Layout.Sider
      collapsible
      collapsed={!isOpen}
      onCollapse={onCollapse}

```

```

        theme="dark"
        collapsedWidth={50}
      >
        <Menu
          items={menuItems}
          defaultSelectedKeys={[router.pathname]}
          theme="dark"
          mode="inline"
        />
      </Layout.Sider>
    )
  }

```

```
export default Sidebar
```

Б.6 Файл DrawerContext.tsx

```

import { DrawerProps } from '@components/Drawer'
import { PropsWithChildren, createContext, useState } from 'react'

const initialState = {
  drawers: [] as DrawerProps[],
  pushDrawer: undefined as undefined | ((payload: DrawerProps) => void),
  popDrawer: undefined as undefined | ((payload: {key: string}) => void),
  updateDrawer: undefined as undefined | ((payload: DrawerProps) => void)
}

const DrawerContext = createContext(initialState)

const DrawerContextProvider = ({ children }: PropsWithChildren) => {
  const [drawerList, setDrawerList] = useState(initialState.drawers)

  const pushDrawer = (payload: DrawerProps) => {
    setDrawerList((prev) => prev.concat([payload]))
  }

  const popDrawer = (payload: {key: string}) => {
    setDrawerList(prev => prev.filter(drawer => drawer.key !== payload.key))
  }

```

```

const updateDrawer = (payload: DrawerProps) => {
  setDrawerList(prev => prev.map(drawer => {
    if (drawer.key === payload.key) {
      return { ...drawer, ...payload }
    } else {
      return drawer
    }
  })))
}

return (<DrawerContext.Provider value={{ drawers: drawerList, pushDrawer, popDrawer,
updateDrawer }}>{children}</DrawerContext.Provider>)
}

export default DrawerContext
export { DrawerContextProvider }

```

Б.7 Файл useActivity.ts

```

import { App } from 'antd'
import { AxiosError } from 'axios'
import { useState } from 'react'

interface Options {
  messageOnSuccess?: (t: any) => string;
  messageOnLoading?: string;
  messageOnError?: (e: any) => string;
  key: string;
  throwError?: boolean;
}

const useActivity = <T extends (...args: any[]) => Promise<unknown>>(<
  activity: T,
  { throwError = true, ...options }: Options
) => {
  const [data, setData] = useState<null | unknown>(null)
  const [loading, setLoading] = useState(false)
  const [error, setError] = useState<null | unknown>(null)

```

```

const { message } = App.useApp()

const showSuccess = () => {
  if (options.messageOnSuccess) {
    message.success({
      content: options.messageOnSuccess(data),
      key: `activity_success_${options.key}`,
      duration: 2
    })
  }
}

const showLoading = () => {
  if (options.messageOnLoading) {
    message.loading({
      content: options.messageOnLoading,
      key: `activity_loading_${options.key}`
    })
  }
}

const hideLoading = () => {
  if (options.messageOnLoading) {
    message.destroy(`activity_loading_${options.key}`)
  }
}

const showError = () => {
  if (options.messageOnError) {
    message.error({
      content: options.messageOnError(error),
      key: `activity_error_${options.key}`,
      duration: 2
    })
  }
}

const request = async (...args: any[]) => {
  setLoading(true)

```

```

    setError(null)
    showLoading()

    await activity(...args)
      .then((response) => {
        setLoading(false)
        hideLoading()
        setData(response)
        showSuccess()
      })
      .catch((err: AxiosError) => {
        console.error(err)
        setLoading(false)
        setError(err)
        showError()
        hideLoading()
        if (throwError) throw err
      })
  }

  return { data, loading, error, request: request as (...args: any[]) => Promise<void>
}

}

export default useActivity

```

Б.8 Файл useDebounce.ts

```

import { useEffect, useRef, useState } from 'react'

const useDebounce = <T>(value: T): T => {
  const [deferred, setDeferred] = useState(value)
  const timeout = useRef<ReturnType<typeof setTimeout>>()

  useEffect(() => {
    clearTimeout(timeout.current)

    timeout.current = setTimeout(() => {
      setDeferred(value)
    })
  }, [value])
}

```

```

    }, 300)

    return () => {
      clearTimeout(timeout.current)
    }
  }, [value])

  return deferred
}

export default useDebounce

```

Б.9 Файл useDrawer.ts

```

import { DrawerProps } from '@components/Drawer'
import { DependencyList, useCallback, useContext, useEffect, useMemo } from 'react'
import DrawerContext from 'src/context/DrawerContext'

const useDrawer = (factory: () => DrawerProps, deps: DependencyList) => {
  const memoizedPayload = useMemo(factory, deps)
  const drawerContext = useContext(DrawerContext)

  if (!drawerContext) {
    throw new Error('useDrawer hook must be used within DrawerContextProvider')
  }

  const popDrawer = useCallback(() => {
    drawerContext.popDrawer?.({ key: memoizedPayload.key })
  }, [memoizedPayload])

  const pushDrawer = () => {
    drawerContext.pushDrawer?.({
      ...memoizedPayload,
      onClose,
      onConfirm
    })
  }

  const onClose = useCallback(() => {

```

```

    memoizedPayload.onClose()
    memoizedPayload.cleanUp?.()
    setTimeout(popDrawer, 300)
  }, [memoizedPayload, popDrawer])

const onConfirm = useCallback(async () => {
  try {
    await memoizedPayload.onConfirm()
    onClose()
  } catch (e) {
    console.error('drawer confirmation error: ', e)
  }
}, [memoizedPayload, onClose])

useEffect(() => {
  drawerContext.updateDrawer?>({
    ...memoizedPayload,
    onClose,
    onConfirm
  })
}, [memoizedPayload, popDrawer, onClose, onConfirm])

return pushDrawer
}

export default useDrawer

```

Б.10 Файл useWidget.ts

```

import CloudinarySDK, { Edits } from '@services/cloudinary'
import { useEffect, useState } from 'react'

const useWidget = (transformOptions: Edits, defaultData?: string, onChange?: (val:
string) => void): [() => void, string] => {
  const [showWidget, setShowWidget] = useState<() => void>()
  const [dataUrl, setDataUrl] = useState('')

  useEffect(() => {
    setShowWidget(() =>

```

```

    CloudinarySDK.getWidget((err, result) => {
      if (err) {
        return
      }
      if (result.info.secure_url) {
        setDataUrl(CloudinarySDK.transformUrl(result.info.secure_url,
transformOptions))
        if (onChange) onChange(CloudinarySDK.transformUrl(result.info.secure_url,
transformOptions))
      }
    })
  )
}, [])

return [showWidget as () => void, dataUrl || defaultData as string]
}

export default useWidget

```

Б.11 Файл AuthLayout.tsx

```

import { NextComponentType } from 'next'
import { Layout } from 'antd'
import styles from '../styles.module.scss'

interface Props {
  Component: NextComponentType;
  pageProps: object;
}

const AuthLayout = ({ Component, pageProps }: Props) => {
  return (
    <Layout className={styles.layout}>
      <Component {...pageProps} />
    </Layout>
  )
}

export default AuthLayout

```

Б.12 Файл CommonLayout.tsx

```
import { NextComponentType } from 'next'
import { Layout } from 'antd'
import styles from '../styles.module.scss'
import Sidebar from '@components/Sidebar'
import Drawer from '@components/Drawer'
import { useContext } from 'react'
import DrawerContext from 'src/context/DrawerContext'

interface Props {
  Component: NextComponentType;
  pageProps: object;
}

const CommonLayout = ({ Component, pageProps }: Props) => {
  const drawerContext = useContext(DrawerContext)

  if (!drawerContext) {
    throw new Error('CommonLayout uses DrawerContext, so it must be within
DrawerContextProvider')
  }

  return (
    <Layout>
      <Sidebar />
      <Layout className={styles.layout}>
        <Component {...pageProps} />
      </Layout>
      {drawerContext.drawers.map(drawerProps => <Drawer {...drawerProps}
key={drawerProps.key}/>)}
    </Layout>
  )
}

export default CommonLayout
```

Б.13 Файл ImagePlaceholder.tsx

```
import useWidget from '@/hooks/useWidget'
import { Button } from 'antd'
import Image from 'next/image'

interface Props {
  value: string;
  onChange: (url: string) => void;
  disabled: boolean;
}

const ImagePlaceholder = ({ value, onChange, disabled }: Props) => {
  const [showWidget, data] = useWidget(
    {
      width: 330,
      height: 330
    },
    value,
    onChange
  )

  const onClick = () => {
    !disabled && showWidget()
  }

  return (value ? <Image src={value} alt='image' width={330} height={330}
onClick={onClick}/> : <Button type='dashed' onClick={onClick} style={{ width: '100%'
}}>Select Image</Button>)
}

export default ImagePlaceholder
```

Б.14 Файл pinView.tsx

```
import Image from 'next/image'

import styles from './styles.module.scss'
import { Button } from 'antd'
```

```

import { MouseEvent, SyntheticEvent, useState } from 'react'

interface Props {
  onChange?: (pos: { left: number; top: number }) => void
  imageUrl: string
  onClose: () => void
}

const PinView = ({ onChange, imageUrl, onClose }: Props) => {
  const [pinPos, setPinPos] = useState<{ left: number; top: number }>()
  return (
    <div className={styles.container}>
      <div className={styles.flex}>
        <div
          className={styles.imageWrapper}
          onClick={(e: MouseEvent<HTMLImageElement>) => {
            const bound = (e.target as HTMLImageElement).getBoundingClientRect()
            setPinPos({
              left: ((e.clientX - bound.left - 10) / bound.width) * 100,
              top: ((e.clientY - bound.top - 10) / bound.height) * 100
            })
          }}
        >
          <Image
            fill={true}
            style={{
              objectFit: 'cover',
              objectPosition: 'center'
            }}
            src={imageUrl}
            alt={'image'}
          />
          {pinPos?.left && pinPos?.top && (
            <div
              className={styles.pin}
              style={{
                left: `${pinPos.left}%`,
                top: `${pinPos.top}%`
              }}
            />
          )}
        </div>
      </div>
    </div>
  )
}

```

```

    </div>
    <div className={styles.buttonWrapper}>
      <Button onClick={onClose} type={'default'} className={styles.button}>
        Cancel
      </Button>
      <Button
        onClick={onChange && (() => {onChange(pinPos!); onClose()})}
        disabled={!pinPos?.left || !pinPos.top}
        type={'primary'}
        className={styles.button}
      >
        Save
      </Button>
    </div>
  </div>
</div>
)
}

export default PinView

```

Б.15 Файл pages/Inspection.tsx

```

import DataTable from '@components/DataTable'
import EmptyPlaceholder from '@components/EmptyPlaceholder'
import useActivity from '@hooks/useActivity'
import useDebounce from '@hooks/useDebounce'
import useDrawer from '@hooks/useDrawer.'
import { Api } from '@services/api'
import { useAppDispatch, useAppSelector } from '@store/index'
import {
  createInspectionAction,
  deleteInspectionAction,
  editInspectionAction,
  getAllInspectionAction,
  getOneInspectionAction,
  resetCurrentInspectionAction
} from '@store/inspection.redux'
import { getAllVehicleAction, getOneVehicleAction } from '@store/vehicle.redux'
import { getAllModelAction, getOneModelAction } from '@store/model.redux'

```

```

import getColumns from '@utils/getColumns'
import * as DataTypes from 'DataTypes'
import { Button, Input, Layout, Space } from 'antd'
import { InferGetServerSidePropsType } from 'next'
import Image from 'next/image'
import { ChangeEvent, memo, useEffect, useMemo, useState } from 'react'
import PinView from '../modules/pinView'

const MemoDataTable = memo(DataTable)

export const getServerSideProps = async () => {
  const data = await Api.getAll<DataTypes.Inspection>('inspection', {
    page: 1,
    nPerPage: 10
  })
  .then((res) => res.data.result)
  .catch((e) => {
    console.error(e)
    return [] as Awaited<
      ReturnType<typeof Api.getAll<DataTypes.Inspection>>
    >['data']['result']
  })

  return {
    props: {
      initialData: data
    }
  }
}

const InspectionPage = ({
  initialData
}: InferGetServerSidePropsType<typeof getServerSideProps>) => {
  const [isOpenDrawer, setIsOpenDrawer] = useState(false)
  const [titleInput, setTitleValue] = useState('')
  const [descriptionInput, setDescriptionInput] = useState('')
  const [pinPos, setPinPos] = useState<{ left: number; top: number }>()
  const [ratingInput, setRatingInput] = useState<number>()
  const [modelSelect, setModelSelect] = useState<
    { label: string; value: string } | undefined

```

```

>()
const [vehicleSelect, setVehicleSelect] = useState<{
  label: string
  value: string
}>()
const [imageUrl, setImageUrl] = useState('')
const [isShowPinEnter, setIsShowPinEnter] = useState(false)
const [searchValue, setSearchValue] = useState('')
const deferredSearch = useDebounce(searchValue)

const dispatch = useAppDispatch()

const { list, current } = useAppSelector((state) => state.inspection)
const { list: vehicleList, current: currentVehicle } = useAppSelector(
  (state) => state.vehicle
)
const { list: modelList, current: currentModel } = useAppSelector(
  (state) => state.model
)

const { request, loading: loadingRefresh } = useActivity(
  () => dispatch(getAllInspectionAction({ search: deferredSearch })),
  {
    key: 'getAllInspection',
    messageOnError: () => 'Error on loading list! Try again later',
    throwError: false
  }
)

const { request: postInspection, loading: loadingCreate } = useActivity(
  () =>
    dispatch(
      createInspectionAction({
        title: titleInput,
        image: imageUrl,
        description: descriptionInput,
        rating: ratingInput!,
        left: pinPos!.left,
        top: pinPos!.top,
        vehicle: vehicleSelect!.value
      })
    )

```

```

    ),
    { key: 'postInspection' }
  )
const { request: editInspection, loading: loadingEdit } = useActivity(
  () =>
    dispatch(
      editInspectionAction({
        title: titleInput,
        image: imageUrl,
        description: descriptionInput,
        left: pinPos!.left,
        top: pinPos!.top,
        rating: ratingInput!,
        vehicle: vehicleSelect!.value,
        id: current?.id || ''
      })
    ),
    { key: 'editInspection' }
  )
const { request: getOneInspection } = useActivity(
  (id: string) => dispatch(getOneInspectionAction({ id })),
  { key: 'getInspection' }
)
const { request: deleteInspection } = useActivity(
  (id: string) => dispatch(deleteInspectionAction({ id })),
  { key: 'deleteInspection' }
)
const { request: getModelList } = useActivity(
  () => dispatch(getAllModelAction({})),
  { key: 'getAllModelsForInsp' }
)
const { request: getOneModel } = useActivity(
  (id: string) => dispatch(getOneModelAction({ id })),
  { key: 'getOneModelForInsp' }
)
const { request: getVehicleList } = useActivity(
  () => dispatch(getAllVehicleAction({})),
  { key: 'getAllvehicleForInsp' }
)
const { request: getOneVehicle } = useActivity(
  (id: string) => dispatch(getOneVehicleAction({ id })),

```

```

    { key: 'getOneVehicleForInspection' }
  )

```

```

const cleanUp = () => {
  setTitleValue('')
  setDescriptionInput('')
  setModelSelect(undefined)
  setVehicleSelect(undefined)
  setRatingInput(undefined)
  setPinPos(undefined)
  setImageUrl('')
  setIsOpenDrawer(false)
}

```

```

useEffect(() => {
  if (current) {
    setTitleValue(current.title)
    setDescriptionInput(current.description)
    setRatingInput(current.rating)
    setImageUrl(current.image || '')
    getOneVehicle(current.vehicle)
    setPinPos({
      left: current.left,
      top: current.top
    })
  }
}, [current])

```

```

useEffect(() => {
  if (currentVehicle) {
    setVehicleSelect({
      label: currentVehicle.vin,
      value: currentVehicle.id
    })
    getOneModel(currentVehicle.model)
  }
}, [currentVehicle])

```

```

useEffect(() => {
  if (currentModel) {

```

```

        setModelSelect({
            label: currentModel.title,
            value: currentModel.id
        })
    }
}, [currentModel])

useEffect(() => {
    !list.length && request()
    !modelList.length && getModelList()
    !vehicleList.length && getVehicleList()

    return () => {
        dispatch(resetCurrentInspectionAction())
    }
}, [])

useEffect(() => {
    request()
}, [deferredSearch])

const pushCreateDrawer = useDrawer(
    () => ({
        isOpen: isOpenDrawer,
        key: 'inspection_create_drawer',
        title: 'Some drawer title',
        onClose: () => setIsOpenDrawer(false),
        onConfirm: () => postInspection(),
        cleanup,
        fields: [
            {
                type: 'image',
                onChange: (url) => {
                    setIsShowPinEnter(true)
                    return setImageUrl(url)
                },
                key: 'image',
                label: 'Image',
                src: pinPos?.left && pinPos?.top ? imageUrl : '',
                pin: pinPos
            },

```

```

{
  type: 'input',
  onChange: (val) => setTitleValue(val),
  value: titleInput,
  key: 'title',
  label: 'Title',
  required: true
},
{
  type: 'input',
  onChange: (val) => setDescriptionInput(val),
  value: descriptionInput,
  key: 'description',
  label: 'Description',
  required: true
},
{
  type: 'select',
  onChange: (val) => setModelSelect(val),
  options: modelList.map((model) => ({
    label: model.title,
    value: model.id
  })),
  value: modelSelect,
  key: 'model',
  label: 'Model',
  required: true
},
{
  type: 'select',
  onChange: (val) => setVehicleSelect(val),
  options: vehicleList.map((vehicle) => ({
    label: vehicle.vin,
    value: vehicle.id
  })),
  value: vehicleSelect,
  key: 'vehicle',
  label: 'Vehicle',
  required: true,
  disabled: !modelSelect?.value
},

```

```

    {
      type: 'input',
      key: 'rating',
      label: 'Rating',
      required: true,
      onChange: (val) => {
        setRatingInput(+val.replace(/^[^d]/g, ''))
      },
      value: `${ratingInput || ''}`
    }
  ],
  loading: loadingCreate
}),
[
  isOpenDrawer,
  titleInput,
  descriptionInput,
  loadingCreate,
  imageUrl,
  vehicleSelect,
  modelSelect,
  ratingInput,
  pinPos
]
)
const pushEditDrawer = useDrawer(
  () => ({
    isOpen: isOpenDrawer,
    key: 'inspection_edit_drawer',
    title: 'Edit Inspection',
    onClose: () => setIsOpenDrawer(false),
    onConfirm: () => editInspection(),
    cleanUp,
    loading: loadingEdit,
    fields: [
      {
        type: 'image',
        onChange: (url) => {
          setIsShowPinEnter(true)
          return setImageUrl(url)
        },

```

```

        key: 'image',
        label: 'Image',
        src: pinPos?.left && pinPos?.top ? imageUrl : '',
        pin: pinPos
    },
    {
        type: 'input',
        onChange: (val) => setTitleValue(val),
        value: titleInput,
        key: 'title',
        label: 'Title',
        required: true
    },
    {
        type: 'input',
        onChange: (val) => setDescriptionInput(val),
        value: descriptionInput,
        key: 'description',
        label: 'Description',
        required: true
    },
    {
        type: 'select',
        onChange: (val) => setModelSelect(val),
        options: modelList.map((model) => ({
            label: model.title,
            value: model.id
        })),
        value: modelSelect,
        key: 'model',
        label: 'Model',
        required: true
    },
    {
        type: 'select',
        onChange: (val) => setVehicleSelect(val),
        options: vehicleList.map((vehicle) => ({
            label: vehicle.vin,
            value: vehicle.id
        })),
        value: vehicleSelect,

```

```

        key: 'vehicle',
        label: 'Vehicle',
        required: true,
        disabled: !modelSelect?.value
    },
    {
        type: 'input',
        key: 'rating',
        label: 'Rating',
        required: true,
        onChange: (val) => {
            setRatingInput(+val.replace(/^[^d]/g, ''))
        },
        value: `${ratingInput || ''}`
    }
]
}),
[
    isOpenDrawer,
    titleInput,
    descriptionInput,
    loadingCreate,
    imageUrl,
    vehicleSelect,
    modelSelect,
    ratingInput,
    pinPos
]
)

```

```

const cols = useMemo(
    () => [
        {
            title: 'Title',
            key: 'title',
            dataIndex: 'title'
        },
        {
            title: 'Description',
            key: 'description',
            dataIndex: 'description'
        }
    ]
)

```

```

    },
    {
      title: 'Rating',
      key: 'rating',
      dataIndex: 'rating'
    },
    {
      title: 'Image',
      key: 'image',
      render: (_, record) => {
        return (
          <Image
            src={record?.image}
            width={64}
            height={64}
            alt={'oem image'}
          />
        )
      }
    },
    {
      title: 'Vehicle',
      key: 'vehicle',
      dataIndex: 'vehicle',
      render: (_, item) => item?.vehicle?.vin
    },
    {
      title: 'Actions',
      key: 'actions',
      render: (_, record) => {
        return (
          <Space>
            <button
              className="action_button"
              type="button"
              onClick={() => {
                getOneInspection(record.id)
                pushEditDrawer()
                setIsOpenDrawer(true)
              }}
            />
          >

```

```

        <Image
          src="/assets/icons/edit_button.svg"
          alt="add"
          width={16}
          height={16}
        />
      </button>
      <button
        className="action_button"
        type="button"
        onClick={() => {
          deleteInspection(record.id)
        }}
      >
        <Image
          src="/assets/icons/trash_can.svg"
          alt="trash"
          width={16}
          height={16}
        />
      </button>
    </Space>
  )
}
}
],
[]
)

const handleClickCreate = () => {
  setIsOpenDrawer(true)
  pushCreateDrawer()
}

const handleSearch = (value: string) => {
  console.log('value:', value)
}

const handleChange = (e: ChangeEvent<HTMLInputElement>) => {
  setSearchValue(e.target.value)
}

```

```

}

return (
  <Layout
    style={{
      gap: 16,
      height: '100%'
    }}
  >
    {!initialData.length ? (
      <EmptyPlaceholder title="inspection point" create={handleClickCreate} />
    ) : (
      <>
        <Button
          onClick={handleClickCreate}
          type="primary"
          style={{
            width: '10%',
            alignSelf: 'flex-end'
          }}
        >
          Add new
        </Button>
        <Button type="primary" onClick={request} loading={loadingRefresh}>
          Refresh
        </Button>
        { /*<Input.Search*/ }
        { /* value={searchValue}*/ }
        { /* onSearch={handleSearch}*/ }
        { /* onChange={handleChange}*/ }
        { /* placeholder="input search text"*/ }
        { /* allowClear*/ }
        { /*/>*/ }
        <MemoDataTable
          data={list.length ? list : initialData}
          columns={cols}
        />
      </>
    )}
    {isShowPinEnter && (
      <PinView

```

```

        onChange={{(pos) => {
            setPinPos(pos)
        }}}
        imageUrl={imageUrl}
        onClose={() => {
            setIsShowPinEnter(false)
        }}
    ></PinView>
  )}
</Layout>
)
}

```

```
export default InspectionPage
```

Б.16 Файл pages/Login.tsx

```

import { placeholder } from '@services/api'
import { Button, Form, Input, Layout } from 'antd'

import { passwordRegex, usernameRegex } from '@utils/stringHelper'
import useActivity from '@hooks/useActivity'
import styles from './styles.module.scss'
import { useEffect } from 'react'
import { useAppDispatch } from '@store/index'
import { setAuthAction } from '@store/app.reducer'

const Login = () => {
  const dispatch = useAppDispatch()
  const { request, loading, data } = useActivity(placeholder, {
    key: 'placeholder',
    messageOnLoading: 'Authorization',
    messageOnSuccess: (_) => 'Successfully authorized',
    messageOnError: (err) => err.message
  })
  // TODO: make mobx action for auth

  const onSubmit = async () => {
    request()
  }

```

```

}

useEffect(() => {
  if (data) {
    dispatch(setAuthAction({ isAuth: true }))
  }
}, [data])

// Ant design layout styles
const layout = {
  labelCol: { span: 6 },
  wrapperCol: { span: 16 }
}
const tailLayout = {
  wrapperCol: { offset: 6, span: 16 }
}

return (
  <Layout className={styles.wrapper}>
    <Form onFinish={onSubmit} {...layout} requiredMark={false}>
      <Form.Item
        label='Username'
        name='username'
        rules={[
          { required: true, message: '' },
          {
            type: 'string',
            min: 4,
            max: 16,
            pattern: usernameRegex,
            message: 'Username must be at least 4 symbol up to 16'
          }
        ]}
        labelAlign='left'
      >
        <Input />
      </Form.Item>
      <Form.Item
        label='Password'
        name='password'

```

```

        rules={[
          { required: true, message: '' },
          {
            type: 'string',
            min: 8,
            max: 16,
            pattern: passwordRegex,
            message:
              'Password must contain at least 8 symbols up to 16, at least 1 special
symbol, 1 capital letter, 1 number'
          }
        ]}
        labelAlign='left'
      >
        <Input.Password />
      </Form.Item>
      <Form.Item {...tailLayout} className={styles.button}>
        <Button type='primary' htmlType='submit' block loading={loading}>
          Login
        </Button>
      </Form.Item>
    </Form>
  </Layout>
)
}

export default Login

```

Б.17 Файл _app.tsx

```

import CommonLayout from '@layouts/index'
import { reduxStore } from '@store/index'
import type { AppProps } from 'next/app'

// import RouteGuard from '@components/RouteGuard'
import '@styles/globals.scss'
import { App as AntApp, ConfigProvider, theme } from 'antd'
import Head from 'next/head'
import { DrawerContextProvider } from 'src/context/DrawerContext'

```

```

import { Provider } from 'react-redux'
import RouteGuard from '@components/RouteGuard'

const App = ({ Component, pageProps }: AppProps) => {
  return (
    <>
      <Head>
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <link rel="icon" href="/favicon.ico" />
      </Head>
      <Provider store={reduxStore}>
        <DrawerContextProvider>
          <ConfigProvider
            theme={{
              algorithm: theme.darkAlgorithm
            }}
          >
            <AntApp>
              { /* <RouteGuard> */ }
              <CommonLayout Component={Component} pageProps={pageProps} />
              { /* </RouteGuard> */ }
            </AntApp>
          </ConfigProvider>
        </DrawerContextProvider>
      </Provider>
    </>
  )
}

export default App

```

Б.18 Файл api.ts

```

import { ResponseList, ResponseType, SearchParams } from '@types'
import axios from 'axios'
import * as DataTypes from 'DataTypes'

const client = axios.create({
  baseURL: process.env.NEXT_PUBLIC_BASE_URI
})

```

```
export const placeholder = async (): ResponseType<{ id: string }> => {
  return await axios.get('https://jsonplaceholder.typicode.com/users/1')
}
```

```
export const Api = {
  async getAll<T> (
    type: DataTypes.Entity,
    payload: SearchParams
  ): ResponseList<(T & DataTypes.WithId)[]> {
    const params: any = {
      page: payload.page,
      range: payload.nPerPage
    }

```

```
    if (payload.search) {
      params.search = payload.search
    }

```

```
    return await client.get(type, {
      params
    })
  },

```

```
  async getById<T> (
    type: DataTypes.Entity,
    payload: {
      id: DataTypes.Id;
    }
  ): ResponseType<T & DataTypes.WithId> {
    return await client.get(`${type}/${payload.id}`)
  },

```

```
  async post<T> (
    type: DataTypes.Entity,
    payload: T
  ): ResponseType<T & DataTypes.WithId> {
    return await client.post(type, payload)
  },

```

```

async put<T> (
  type: DataTypes.Entity,
  payload: Partial<T> & DataTypes.WithId
): ResponseType<T & DataTypes.WithId> {
  const { id, ...rest } = payload
  return await client.put(`${type}/${id}`, rest)
},

async delete (
  type: DataTypes.Entity,
  payload: { id: DataTypes.Id }
): ResponseType<void> {
  return await client.delete(`${type}/${payload.id}`)
}
}

```

Б.19 Файл cloudinary.ts

```

export interface Edits {
  resize?: 'scale' | 'fill' | 'fit' | 'crop' | 'thumbnail';
  focus?: 'face' | 'faces' | 'auto';
  height?: number;
  width?: number;
}

const CloudinarySDK = {
  getWidget: (
    cb: (error: any, result: { [key: string]: any }) => void
  ): (() => void) | undefined => {
    if ((window as any) !== undefined) {
      const widget = (window as any).cloudinary.createUploadWidget(
        {
          cloudName: process.env.NEXT_PUBLIC_CLOUDINARY_CLOUD,
          uploadPreset: process.env.NEXT_PUBLIC_UPLOAD_PRESET,
          sources: ['local', 'url'],
          maxImageWidth: 1024,
          multiple: false,
          styles: {
            palette: {
              window: '#10173a',

```

```

        sourceBg: '#20304b',
        windowBorder: '#7171D0',
        tabIcon: '#79F7FF',
        inactiveTabIcon: '#8E9FBF',
        menuIcons: '#CCE8FF',
        link: '#72F1FF',
        action: '#5333FF',
        inProgress: '#00ffcc',
        complete: '#33ff00',
        error: '#cc3333',
        textDark: '#000000',
        textLight: '#ffffff'
    },
    fonts: {
        default: null,
        '\IBM Plex Sans', sans-serif: {
            url: 'https://fonts.googleapis.com/css?family=IBM+Plex+Sans',
            active: true
        }
    }
}
},
cb
)

return widget.open
}
},
transformUrl: (origin: string, options: Edits = {}) => {
    const {
        focus = 'auto',
        resize = 'fill',
        height = 100,
        width = 100
    } = options

    const splitted: string[] = origin.split('/')
    const first = splitted.slice(0, 6)
    const edit = `/c_${resize},g_${focus},w_${width},h_${height}/`
    const second = splitted.slice(6)

```

```

    return first.join('/') + edit + second.join('/')
  }
}

```

```
export default CloudinarySDK
```

Б.20 Файл app.redux.ts

```

import { createSlice, PayloadAction } from '@reduxjs/toolkit'

const initialState = {
  isAuth: false,
  loadCount: 0
}

const appSlice = createSlice({
  name: 'app',
  initialState,
  reducers: {
    setAuthAction: (state, action: PayloadAction<{isAuth: boolean}>) => {
      state.isAuth = action.payload.isAuth
    }
  }
})

export default appSlice.reducer
export const { setAuthAction } = appSlice.actions

```

Б.21 Файл store.ts

```

import { configureStore, combineReducers } from '@reduxjs/toolkit'
import appReducer from './app.redux'
import oemReducer from './oem.redux'
import makeReducer from './make.redux'
import modelRedux from './model.redux'
import inspectionRedux from '@store/inspection.redux'
import vehicleRedux from '@store/vehicle.redux'
import { useDispatch, useSelector, TypedUseSelectorHook } from 'react-redux'

```

```

const reduxStore = configureStore({
  reducer: combineReducers({
    app: appReducer,
    oem: oemReducer,
    make: makeReducer,
    model: modelRedux,
    inspection: inspectionRedux,
    vehicle: vehicleRedux
  })
})

export type RootState = ReturnType<typeof reduxStore.getState>
export type AppDispatch = typeof reduxStore.dispatch
export const useAppDispatch = useDispatch<AppDispatch>
export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector

export { reduxStore }

```

Б.22 Файл inspection.redux.ts

```

import { Api } from '@services/api'
import { createAsyncThunk, createSlice } from '@reduxjs/toolkit'
import DataTypes from 'DataTypes'
import { RootState } from '.'

const initialState = {
  list: [] as (DataTypes.Inspection & DataTypes.WithId)[],
  page: 1,
  nPerPage: 10,
  current: undefined as (DataTypes.Inspection & DataTypes.WithId) | undefined,
  nextPage: undefined as string | undefined
}

const getAllInspectionAction = createAsyncThunk(
  'inspection/getAllInspection',
  async (
    payload: {
      search?: string
    },
    thunkApi

```

```

) => {
  const state = thunkApi.getState() as RootState
  try {
    const response = await Api.getAll<DataTypes.Inspection>('inspection', {
      search: payload?.search,
      page: state.inspection.page,
      nPerPage: state.inspection.nPerPage
    })

    return response
  } catch (err) {
    return thunkApi.rejectWithValue('Request Error')
  }
}
)

```

```

const getOneInspectionAction = createAsyncThunk(
  'inspection/getOneInspection',
  async (payload: { id: DataTypes.Id }, thunkApi) => {
    try {
      const response = await Api.getById<DataTypes.Inspection>(
        'inspection',
        payload
      )

      return response
    } catch (err) {
      return thunkApi.rejectWithValue('Request Error')
    }
  }
)

```

```

const createInspectionAction = createAsyncThunk(
  'inspection/createOneInspection',
  async (payload: DataTypes.Inspection, thunkApi) => {
    try {
      const response = await Api.post<DataTypes.Inspection>(
        'inspection',
        payload
      )
    }
  }
)

```

```

        return response
    } catch (err) {
        return thunkApi.rejectWithValue('Request Error')
    }
}

)

const editInspectionAction = createAsyncThunk(
  'inspection/editInspectionAction',
  async (payload: DataTypes.Inspection & DataTypes.WithId, thunkApi) => {
    try {
      const response = await Api.put<DataTypes.Inspection>(
        'inspection',
        payload
      )

      return response
    } catch (err) {
      return thunkApi.rejectWithValue('Request Error')
    }
  }
)

const deleteInspectionAction = createAsyncThunk(
  'inspection/deleteInspection',
  async (payload: { id: DataTypes.Id }, thunkApi) => {
    try {
      await Api.delete('inspection', payload)

      return payload
    } catch (err) {
      return thunkApi.rejectWithValue('Request Error')
    }
  }
)

const inspectionSlice = createSlice({
  name: 'inspection',
  initialState,

```

```

reducers: {
  resetInspectionStateAction: () => {
    return initialState
  },
  resetCurrentInspectionAction: (state) => {
    state.current = initialState.current
  }
},
extraReducers: (builder) => {
  builder.addCase(getAllInspectionAction.fulfilled, (state, action) => {
    state.list = action.payload.data.result

    if (action.payload.data.next) {
      state.nextPage = action.payload.data.next
    }
  })
  builder.addCase(getAllInspectionAction.rejected, () => {
    throw new Error()
  })
  builder.addCase(getOneInspectionAction.fulfilled, (state, action) => {
    state.current = action.payload.data
  })
  builder.addCase(getOneInspectionAction.rejected, () => {
    throw new Error()
  })
  builder.addCase(createInspectionAction.fulfilled, (state, action) => {
    state.list.push(action.payload.data)
  })
  builder.addCase(createInspectionAction.rejected, () => {
    throw new Error()
  })
  builder.addCase(editInspectionAction.fulfilled, (state, action) => {
    state.list = state.list.map((i) =>
      i.id === action.payload.data.id ? action.payload.data : i
    )
  })
  builder.addCase(editInspectionAction.rejected, () => {
    throw new Error()
  })
  builder.addCase(deleteInspectionAction.fulfilled, (state, action) => {
    state.list = state.list.filter((i) => i.id !== action.payload.id)
  })
}

```

```

    })
    builder.addCase(deleteInspectionAction.rejected, () => {
      throw new Error()
    })
  }
})

export default inspectionSlice.reducer
export const { resetInspectionStateAction, resetCurrentInspectionAction } =
inspectionSlice.actions
export {
  getAllInspectionAction,
  getOneInspectionAction,
  createInspectionAction,
  editInspectionAction,
  deleteInspectionAction
}

```

Б.23 Файл app.js

```

import express, { json } from 'express'
import routes from '@routes/api/v1/index'
import errorMiddleware from '@middlewares/errorMiddleware'
import cors from 'cors'
import packageJson from '../package.json'

const app = express()

app.use(json())
app.use(cors())

app.use('/api/v1', routes.publicRoutes)
app.use('/api/status', (req, res) => {
  return res.status(200).json({
    name: packageJson.name,
    version: packageJson.version,
    status: 'OK'
  })
})

app.use(errorMiddleware)

```

```
export default app
```

Б.24 Файл connection.js

```
import mongoose from 'mongoose'

const uri = process.env.MONGO_URI
const options = {
  user: process.env.MONGO_USER,
  pass: process.env.MONGO_PASS
}

if (!uri) {
  throw new Error('Edit your .env file and add MONGO_URI property')
}

if (!options.user || !options.pass) {
  throw new Error('Edit your .env file and add MONGO_USER and MONGO_PASS')
}

const connect = async () => mongoose.connect(uri, options)

export default connect
```

Б.25 Файл routes.js

```
import Oem from './oem'
import Make from './make'
import Model from './model'
import Vehicle from './vehicle'
import Inspection from './inspection'

const publicRoutes = [Oem, Make, Model, Vehicle, Inspection]
const privateRoutes = []

export default {
  publicRoutes,
```

```
privateRoutes
}
```

Б.26 Файл inspection/controller.create.js

```
import {Inspection, Oem, Vehicle} from '@models'
import Joi from '@lib/Joi'
```

```
const jBody = Joi.object({
  title: Joi.string().min(2).required(),
  vehicle: Joi.objectId().required(),
  description: Joi.string().min(2).required(),
  rating: Joi.number().required(),
  left: Joi.number().required(),
  top: Joi.number().required(),
  image: Joi.string()
})
```

```
const create = async (
  req,
  res,
  next
) => {
  const { body } = req
```

```
  try {
    await jBody.validateAsync(body)
  } catch (e) {
    next(e)
    return
  }
```

```
  try {
    const doc = await Inspection.create(body)
    res.status(200).json(doc)
  } catch (err) {
    next(err)
  }
}
```

```
export default create
```

Б.27 Файл inspection/controller.deleteById.js

```
import { Inspection, Oem, Vehicle } from '@models'
import Joi from '@lib/Joi'

const deleteById = async (req, res, next) => {
  const { id } = req.params

  try {
    await Joi.objectId().required().validateAsync(id)
  } catch (e) {
    next(e)
    return
  }

  try {
    const doc = await Inspection.findOneAndDelete({ _id: id }).exec()

    if (!doc) {
      res.status(404).end()
      return
    }

    res.status(204).end()
  } catch (err) {
    next(err)
  }
}

export default deleteById
```

Б.28 Файл inspection/controller.getAll.js

```
import { Inspection, Oem, Vehicle } from '@models'
import getNext from '@utils/getNext'
```

```

import Joi from '@lib/Joi'

const jquery = Joi.object({
  page: Joi.number().min(1).required(),
  range: Joi.number().min(1).required(),
  search: Joi.string()
})

const getAll = async (req, res, next) => {
  const {
    page,
    range,
    search
  } = req.query

  try {
    await jquery.validateAsync(req.query)
  } catch (e) {
    next(e)
    return
  }

  try {
    const data = await (async () => {
      if (search) {
        return await Inspection.textSearch(search)
          .populate({
            path: 'vehicle',
            populate: {
              path: 'model'
            }
          })
          .limit(range)
          .skip(Number(range) * (Number(page) - 1))
          .exec()
      } else {
        return await Inspection.find()
          .populate({
            path: 'vehicle',
            populate: {

```

```

        path: 'model'
      }
    })
    .limit(range)
    .skip(Number(range) * (Number(page) - 1))
    .exec()
  }
}

res.status(200).json({
  count: data.length,
  total: data.length,
  next: getNext(req, {
    page: `${Number(page) + 1}`,
    range
  }),
  result: data
})
} catch (err) {
  next(err)
}
}

export default getAll

```

Б.29 Файл inspection/controller.getById.js

```

import {Inspection, Oem, Vehicle} from '@models'
import Joi from '@lib/Joi'

const getOne = async (req, res, next) => {
  const { id } = req.params

  try {
    await Joi.objectId().required().validateAsync(id)
  } catch (e) {
    next(e)
    return
  }
}

```

```

const doc = await Inspection.findOne({ _id: id }).exec()

if (!doc) {
  res.status(404).end()
} else {
  res.status(200).json(doc)
}
}

export default getOne

```

Б.29 Файл inspection/controller.updateById.js

```

import {Inspection, Oem, Vehicle} from '@models'
import Joi from '@lib/Joi'

const jBody = Joi.object({
  title: Joi.string().min(2),
  vehicle: Joi.objectId(),
  description: Joi.string().min(2),
  rating: Joi.number(),
  left: Joi.number(),
  top: Joi.number(),
  image: Joi.string()
})

const update = async (
  req,
  res,
  next
) => {
  const { id } = req.params
  const body = req.body

  try {
    await Joi.objectId().required().validateAsync(id)
  } catch (e) {
    next(e)
  }

```

```

    return
  }

  try {
    await jBody.validateAsync(body)
  } catch (e) {
    next(e)
    return
  }

  try {
    const doc = await Inspection.findOneAndUpdate({ _id: id }, body, { new: true })
    if (!doc) {
      res.status(404).end()
      return
    }
    res.status(200).json(doc)
  } catch (err) {
    next(err)
  }
}

export default update

```

Б.30 Файл inspection.model.js

```

import schemaTransformPlugin from '@utils/schemaTransformPlugin'
import {model, Schema, SchemaTypes} from 'mongoose'

const schema = new Schema(
  {
    title: {
      type: SchemaTypes.String,
      minlength: 2,
      required: true,
      index: 'text'
    },
    description: {
      type: SchemaTypes.String,
      required: true,

```

```

    },
    rating: {
      type: SchemaTypes.Number,
      required: true,
    },
    left: {
      type: SchemaTypes.Number,
      required: true,
    },
    top: {
      type: SchemaTypes.Number,
      required: true,
    },
    image: {
      type: SchemaTypes.String,
      required: false
    },
    vehicle: {
      type: SchemaTypes.ObjectId,
      required: true,
      ref: 'vehicle'
    }
  },
  {
    timestamps: true,
    statics: {
      textSearch(text) {
        return this.find(
          {
            $text: {$search: text}
          },
          {
            score: {$meta: 'textScore'}
          }
        )
      }
    },
    methods: {
      rename() {
        console.log(this)
      }
    }
  }

```

```

    }
  }
)

schema.plugin(schemaTransformPlugin)

const m = model('inspection', schema)

export default m

```

Б.31 Файл `errorMiddleware.js`

```

import Joi from '@lib/Joi'

const errorMiddleware = (err, req, res, next) => {
  if (err instanceof Joi.ValidationError) {
    res.status(400).json(
      err.details
    )
    return
  }

  res.status(500).json([
    {
      message: err.message
    }
  ])
}

export default errorMiddleware

```

Б.32 Файл `schemaTransformPlugin.js`

```

const schemaTransform = {
  transform: (doc, ret) => {
    ret.id = ret._id
    delete ret._id
    delete ret.__v
  }
}

```

```
const schemaTransformPlugin = (schema) => {  
  schema.set('toJSON', schemaTransform)  
}
```

```
export default schemaTransformPlugin
```

Б.33 Файл index.js

```
import app from './app';  
import { config } from 'dotenv';  
config();  
import connect from './connection';  
  
const port = Number(process.env.PORT) || 5000;  
  
app.listen(port, '0.0.0.0', () => {  
  connect().then(() => {  
    console.log('Server started on port ', port, ', and connected to mongodb');  
  });  
});
```