Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЄКТ

на тему:

«Проектування web-орієнтованої системи автомобільної компанії»

Студента 2 курсу, 2м групи, спеціальності 121 «Інженерія програмного забезпечення» спеціалізації «Інженерія програмного забезпечення» _____ підпис студента — Новака Владислава Віталійовича

Науковий керівник кандидат технічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Рассамакін Володимир Якович

Гарант освітньої програми доктор економічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Токар Володимир Володимирович

КИЇВ – 2021

Київський національний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмногозабезпечення та кібербезпеки
Криворучко О. В.
«10» листопада 2020 р.

Завдання
на випускний кваліфікаційний проєкт студентові
Новаку Владиславу Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проекту «Проектування web-орієнтованої системи автомобільної компанії»
2. Затверджена наказом ректора від "30" листопада 2020 р. № 3224

2. Строк здачі студентом закінченої проекту 25 листопада 2021р.

3. Цільова установка та вихідні дані до проекту

Мета проекту: створення веб-інтерфейсу для адміністрування бази даних

Об'єкт дослідження: процес розробки веб-додатку

Предмет дослідження: розробка веб-інтерфейсу для адміністрування баз даних

1. Консультанти проєкту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Зміст випускного кваліфікаційного проєкту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПІДХОДІВ ДО СТВОРЕННЯ ВЕБ-ДОДАТКУ

1.1. Характеристика систем та платформ для створення веб-додатків

1.2. Аналіз мов веб-програмування веб-орієнтованих систем, додатків та платформ

1.3. Висновки до розділу 1

РОЗДІЛ 2. ЕТАПИ ТА ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

2.1. Етапи розробки веб-орієнтованого додатку

2.2. Технічні вимоги до веб-додатку

2.3. Загальна концепція веб-додотку

2.4. Верстка інтерфейсу та кодування логіки додатку

2.5. Встановлення MongoDB та налаштування проекту

2.6. Висновки до розділу 2

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ЗАСОБУ АДМІНІСТРУВАННЯ БАЗИ ДАНИХ ЧЕРЕЗ ВЕБ-ІНТЕРФЕЙС

3.1. Елементи користувацького інтерфейсу

3.2. Сторінка логіну

3.3. Інформація про користувача

3.4. Основна сторінка

3.5. Вигляд сторінки додавання записів

3.6. Вигляд сторінки редагування запису

3.7. Вигляд колекції запису у форматі MongoDB

3.8. Вигляд запису в графічному інтерфейсі

3.9. Висновки до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання проєкту

№ пор.	Назва етапів випускного кваліфікаційного проекту	Строк виконання етапів проекту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускного кваліфікаційного проєкту*	*21.09.2020*	*21.09.2020*
2.	*Розробка та затвердження завдання на проєкт магістра (стац/заоч)*	*6.11.2020-2212.202*	*6.11.2020-2212.202*
3.	*Вступ та перелік літературних джерел*	*27.02.2021*	*27.02.2021*
4.	*Розробка технічного завдання*	*20.03.2021*	*20.03.2021*
	Розділ 1. Дослідження підходів до створння веб-додатку	*16.04.2021*	*16.04.2021*
5.	*Розділ 2. Етапи та засоби розробки програмного продукту*	*24.05.2021*	*24.05.2021*
6.	*Розділ 3. Реалізація засобу адміністрування бази даних через веб-інтерфейс*	*20.09.2021*	*20.09.2021*
7.	*Розробка програми та методики тестування*	*18.10.2021*	*18.10.2021*
8.	*Написання наукової статті*	*22.05.2021*	*22.05.2021*
9.	*Керівництво користувача*	*21.10.2021*	*21.10.2021*
10.	*Висновки та пропозиції*	*01.11.2021*	*01.11.2021*
11.	*Здача випускного кваліфікаційного проєкту на кафедру (перша перевірка)*	*03.11.2021*	*03.11.2021*
12.	*Підготовка автореферату та презентації доповіді*	*03.11.2021*	*03.11.2021*
13.	*Попередній захист випускного кваліфікаційного проєкту*	*22.11.2021 – 25.11.2021*	*22.11.2021 – 25.11.2021*
14.	*Здача зброшурованої випускного кваліфікаційного проєкту*	*25.11.2021*	*25.11.2021*
15.	*Зовнішнє рецензування випускного кваліфікаційного проєкту*	*26.11.2021*	*26.11.2021*
16.	*Підготовка до публічного захисту випускного кваліфікаційного проєкту*		

7. Дата видачі завдання «10» листопада 2020 р.

8. Науковий керівник випускного кваліфікаційного проекту Рассамакін В.Я.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Токар В.В.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Новак В.В.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проєкту

Науковий керівник випускного кваліфікаційного проекту

_____

(підпис, дата)

Відмітка про попередній захист Рассамакін В.Я.

(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проект

Випускний кваліфікаційний проект студента Новак В.В.

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми Токар В.В.

(прізвище, ініціали, підпис)

Завідувач кафедри Криворучко О. В.

(підпис, прізвище, ініціали)

«____»_____20 ___ р.

АНОТАЦІЯ

Відповідно до мети дослідження робота присвячена розробці універсального засобу адміністрування баз даних через web-інтерфейс.

Розглянуто сучасні тенденції ринку веб-розробок і проведено огляд діючих веб-орієнтованих систем, веб-інтерфейсів, веб-додатків, проаналізовані їх особливості, відзначені переваги та недоліки. Здійснено діагностику та систематизацію мов веб-програмування та програмної платформи MEVN. Візуальний нарис веб-додатку був розроблений за допомогою веб-ресурсу WireFrame. Написання веб-додатку було здійснено за допомогою середовища розробки VScode. Була розгорнута та налаштована серверна платформа node.js.

Готове програмне забезпечення Automobile-app було успішно протестовано відповідно до функціональних вимог.

Ключові слова: *веб-сервер, браузер, адміністрування, веб-орієнтована система, веб-додаток, база даних.*

ABSTRACT

According to the purpose the research is devoted to developing of a universal tool for database administration through a web-interface. The modern tendencies of the market of web developments are considered and the review of the operating web interfaces, web applications is carried out, their features are analyzed, shortcomings are noted.

The analysis of web programming development languages and MEVN software platform is performed. The visual outline of the web application was developed using the web resource WireFrame. The web application was written using VScode. The node.js.platform was deployed and configured.

The finished Automobile-app software has been successfully tested according to the functional requirements.

Keywords: *web-server, browser, administration, web application, database*

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – це набір інструкцій, даних або програм, які використовуються для роботи комп’ютера та виконання конкретних завдань.

IDE (англ. Integrated Development Environment) – інтегроване середовище розробки

API – (англ. Application Programming Interface) – це набір функцій, які дозволяють програмам отримувати доступ до даних і взаємодіяти із зовнішніми програмними компонентами, операційними системами або мікросервісами.

ЗМІСТ

					КНТЕУ 121 02м-15.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		*22.12.20*	*Проектування web-орієнтованої системи автомобільної компанії*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Рассамакін В.Я.		*22.12.20*		*3*	*3*	*49*
Гарант		Токар В.В.		*22.12.20*		*Факультет інформаційних технологій 2м курс, 2м група*		
Розробив		Новак В.В.		*22.12.20*	*Зміст*			

ВСТУП

Актуальність. в умовах формування Індустрії 4.0., глобальної цифровізації діяльності суб'єктів бізнесу веб-технології стають невід'ємною складовою розроблення ефективних управлінських рішень та взаємодії із споживачами, постачальниками та стратегічними партнерами. Вагомий внесок глобальної мережі internet на теперішній світ та глобальну економіку не має еволюційних аналогів. Тренд сьогодення – це зростаюча епоха електронного проникнення в усі сфери людського життя та суспільних процесів. Веб-технології переформатували та модернізували уявлення про роботу з інформацією, з комп'ютерною технікою, з засобами комунікацій та зв'язку.

У даний час автомобільним компаніям доводитися оперувати великою кількістю інформації, яку досить складно акумулювати, обробляти наявними засобами офісного забезпечення, до того ж це зазвичай буває неефективно та не практично. Від цього робимо висновок, що гостро постає питання автоматизувати діяльності такого харакетру, що надалі сприятиме швидкому прийняттю управлінських рішень через використання інформаційних систем, вивільнення певної кількості персоналу, удосконаленню процесів інформаційної взаємодії та безпеки.

Створення веб-додатків з кожним роком набуває все більшої актуальнотсі, так як спрямоване на оптимізацію роботи користувача та надає постійний доступ до інформаційних ресурсів , та економить час на реалізацію управлінських завдань. Тому використання веб-додатку для роботи як для автомобільної компанії так і в інших сферах бізнесу на даний момент являється актуальної темою.

Мета: розробка веб-додатків для автомобільних компаній.

*Завдання***:** проведення дослідження, яке покаже можливість використання інструментів та теоретичних знань з розробки веб – додатків.

Об'єкт дослідження: сучасні веб-додатки.

Предмет дослідження є методи та алгоритми створення додатків для автомобільних компаній.

Мета проекту: створення веб-інтерфейсу для адміністрування бази даних.

Об'єкт дослідження: процес розробки веб-додатку.

Предмет дослідження: веб-інтерфейс для адміністрування баз даних.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ПІДХОДІВ ДО СТВОРЕННЯ ВЕБ-ОРІЄНТОВАНИХ СИСТЕМ

1.1. Характеристика систем та платформ для створення веб-додатків

Веб-орієнтована інформаційна система – це комунікаційна платформа, щозабезпечує збирання, пошук, оброблення та пересилання інформації [1], вона забезпечує взаємодію клієнта та суб'єкта бізнесу. Сучасним варіантом такої системи є веб-додаток. Веб- додатки дуже популярні, та є одним з варіантів отримання інформації у мережі Інтернет.

Веб-додаток – додаток, в якому клієнтом є оглядач Інтернета, а сервером — веб-сервер. Оглядач Інтернета може бути реалізацією так званих тонких клієнтів. Він відобрадає веб-сторінки і, як правило, входит до складу операційної системи, а його оновлення та супровід виконує постачальник оперційної системи. Логіка додатка зосереджена на сервері, а оглядача Інтернета найчастіше відповідає лише за відображення інформації, завантаженої з сервера, і за передачу на сервер даних користувача. Отже, веб-додаток – це комп'ютерна програма, яка працює в браузері. Зберігання та обробка інформації при такій організації обчислень відбувається на віддаленому сервері, а веб-переглядач служить програмою-клієнтом і призначеним для користувача інтерфейсом (рис. 1.1).

Рис. 1.1. Типова блок-схема проектування веб-додатку

					КНТЕУ 121 02м-15.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		16.04.21	*Проектування web-орієнтованої системи автомобільної компанії*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Рассамакін В.Я.		*16.04.21*		*Р1*	*5*	*49*
Гарант		Токар В.В.		*16.04.21*		*Факультет інформаційних технологій 2м курс, 2м група*		
Розробив		Новак В.В.		*16.04.21*	*Дослідження підходів до створення веб-орієнтованих систем*			

Однією з ключових особливосей даного методу є той принцип, що клієнти не залежать від виду та особливостей операційної системи юзера, і веб-додатки, таким чином, є міжплатформеними та кросплатформеними сервісами. [1,2].

Веб-додатки мають свої переваги та недоліки (табл.1), що обмежує можливості їх використання.

Таблиця 1.1

Переваги та недоліки Веб-додатків

Переваги Веб-додатків	Недоліки Веб-додатків
веб-додатки розробляються з такими мовами програмування, як HTML та CSS, які методологічно сформовані в IT-просторі	один веб-додаток може використовуватись на всіх пристрях, але повинен бути запрограмований таким чином, щоб він відображався незалежно від операційної системи пристрою
веб-додаток може використовуватись на усіх пристроях, не вимогливий до ресурсів і не пред'являє складних вимог до апаратної платформи	можуть виникнути проблеми під час його відображення на iOS, Android або Windows Phone
веб-додаток запрограмований для роботи в будь-якій операційній системі. адаптується до iOS, Android, Windows Phone та інших систем	підключення до Інтернету буде абсолютно необхідним для запуску веб-додатку
веб-додаток не потребує встановлення на пристрій. Ці програми працюють у власному веб-браузері пристрою через просту URL-адресу	кожне перезавантаження (або оновлення сторінки) викликає помітну затримку, викликану необхідністю встановити HTTP- це спосіб обміну даними між сервером і клієнтом щоб потіи передати у відповідь HTTP-повідомлення і перезавантажити вею-сторінку за допомогою браузера. Це започатковує переривчастий режим роботи та уповільнює виконання веб-додатку
веб-додатки можуть відкривати веб-сайти, не потребують оновлення, як це роблять звичайні програми	
високий рівень розвитку надійності сітьових з'єднань та web-технологій	
розробка веб-додатків - дешевший вид розробки додатків	існують деякі обмеження доступу щодо певних апаратних функцій пристрою, на якому він працює.
веб-додатки дозволяють своїм користувачам бути насправді мобільними	існує багато велика кількість додатків, які не можуть бути реалізовані в Web, наприклад в барузері неможливо створювати складні трьохмірні моделі.

Веб-додаток - клієнт-серверний додаток, в якому клієнт взаємодіє з веб-сервером через використання браузера. Логіка веб-додатку розподілена між Frontend-side(клієнтською частиною) і Backend-side(та розробкою на стороні сервера), зберігання даних здійснюється, зазвичай, на сервері, обмін даними відбувається по глобалній мережі з використанням цифрових технологій.

Веб-інтерфейс - веб-сторінка або сукупність веб-сторінок, що надає користувальницький інтерфейс для взаємодії з сервісом або пристроєм за допомогою протоколу HTTP і веб-браузера. Веб-інтерфейси набули широкого поширення в зв'язку з ростом популярності всесвітньої павутини і відповідно - бути широко розповсюдженим веб-браузерів. Одним з основних вимог до Web-interface є їх незмінна зовнішня досконаліть і однакова функціональність при використанні в різних браузерах.

1.2. Аналіз мов веб-програмування веб-орієнтованих систем, додатків та платформ

Для створення сучасних веб-орієнтованих систем, додатків та платформ використовуються 5 типових підходів та методологій:

- *HTML;*
- *за допомогою програмних засобів розробки сайтів та веб-додатків;*
- *з використанням інструментальних систем таких як CMS;*
- *з використанням фреймворків;*
- *на SaaS-платформах у CLOUD.*

1. Використання методології HTML - це не мова програмування та засіб розмітки тексту. На сьогоднішній день HTML залишається найбільш поширеним та універсальним, навіть незамінним засобом розмітки гіпертексту, а, отже, і публікації в Інтернет.

HTML5 - це нова специфікація мови розмітки, що використовується в створенні веб-сторінок. На відміну від попередніх версій це не просто специфікація мови для гіпертекстової розмітки, а набір різнопланових модулів - від HTML-

елементів до відео-, аудіо-, векторної графіки SVG, растрової JavaScript-графіки Canvas, локальних баз даних і навіть різних API браузера. Крім визначення розмітки, в HTML 5 визначені application programming interface, API. Існуючі інтерфейси D0M (Document object Model - «об'єктна модель документа») розширені, також були додані нові API:

- малювання 2D-картинок в реальному часі;

- контроль над відтворенням медіафайлів, який може використовуватися, наприклад, для синхронізації субтитрів з відео;

- зберігання даних у браузері;

- редагування;

- Drag--and--drop;

- робота з мережею.

2. Розробка сайтів та веб-додатків за допомогою програмних засобів (Dreamweaver, FrontPage). Існують рішення для більш швидкої і зручної розробки веб-продуктів, що надають можливість вілтворювати html розмітку, розробляти сайт у режимі візуашьного вітворення і мають багато інших зручних інстументів для використання.

Інструментальними системами для розробки HTML є:

• *програми, що мають у сооєму складі візуальні редактори (design-based editor)* - засоби, які автоматично формують необхідний HTML-код, дозволяючи розробляти Web-сторінки в режимі WYSIWYG;

•*програми-редактори (code-based editors)*, які надають редактор і допоміжні засоби для автоматизації написання коду.

Найбільш популярні design-based *редактори:*

• *Adobe DreamWeaver* - один з кращих візуальних редакторів, що генерують HTML код. Він дозволяє працювати в декількох режимах одночасно, з HTML кодом або у візуальному режимі. Але основним недоліком є те, що програма генерує занадто "важкий" код, додаючи багато зайвого. Але, якщо знайомі з HTML,

тоді текст розмітки можна відредагувати. Ця програмна система випускалася до 2005 року компанією Macromedia, після чого була поглинути компанією Abode.

• *Microsoft FrontPage* - це простий в засвоєнні і зручний Web-редактор для проектування, підготовки і публікації Web-сайтів. Завдяки інтеграції з сімейством продуктів MS Office, звичного інтерфейсу і великої кількості шаблонів програма дозволяє швидко засвоїти роботу навіть початківцям, які знайомі з основами роботи в MS Word. При цьому FrontPage не можна назвати рішенням для «чайників»: програма надає широкі функціональні можливості та різноманітні засоби оптимізації при колективній розробці. Вона дозволяє швидко створювати динамічні комплексні Web-вузли практично будь-якої складності.

Популярні code-based редактори:

•*Adobe HomeSites* - це потужний пакет, до складу якого входить багато корисних функцій і підпрограм. Об'ємний дистрибутив редактора включає в себе, крім самого редактора, редактор TopStyle для редагування таблиць CSS, перевірку орфографії та багато іншого.

•*HotDog* - цілком професійний редактор. Вбудована підтримка дуже широкого набору інструментів, що використовуються в Web-дизайні: HTML, CSS, JavaSript, VBScript, ASP, а також DoM - об'єктну модель документа, що використовується при програмуванні на VBScript і JavaScript. При цьому перевірка синтаксису цих інструментів може налаштовуватися в досить широких рамках. Наприклад, HTML можна перевіряти на відповідність версії 3.2, 4, або на "перегляді" тільки в Internet Explorer та інше.

•*AceHTML* - основні функціональні можливості - подібно HomeSite і FirstPage. З цінних якостей AceHTML треба відзначити вмілу роботу з кодуваннями російської мови. Другий плюс - дуже непогана вбудована утиліта для перегляду графічних файлів у комп'ютері. У просторому вікні відображаються ескізи всіх картинок в директорії, а також їх параметри і розмір у пікселях.

3. Використання інструментальних систем CMS для створення динамічних Web-сайтів та додатків. Для створення динамічного сайту

використовують 2 шляхи: написання власних програм, які відповідають за створення потрібних шаблонів і підтримують необхідні функції, проте можливо вимагатиме великих програмістських зусиль і часу, та через використання існуючих систем, які і називаються системами управління веб-контентом. Другий шлях є основним на цей час для створення складних, сучасних сайтів, порталів, Web-додатків. Це метод застосовується з використанням CMS. Вікіпедія дає наступне визначення CMS - це система керування вмістом (контентом) (англ. Content management system, CMS) , тобто комп'ютерна програма чи система, що використовується для забезпечення і організації сумісного процеса створення, редагування та управління текстовими і мультимедійними документами (вмісту чи контента) [17]. Цей вміст розглядається як не структуровані данні предметної задачі в протилежність структурованим даним, що звичайно знаходяться під керуванням СУБД. Звичайно, що встановлення CMS робиться вже на вибраному хостінзі. При цьому як мінімум вимагається FTP доступ та дозвіл роботи MySQL.

Таким чином, відділення дизайну від контенту є головною відміною особливістю динамічних сайтів. На цій основі можливі подальше втілення структури сайту, такі як визначення різних призначених для користувача функцій і автоматизація бізнес-процесів, а саме головне, контроль контенту, що надходить в офіційній документації систем управління Web-контентом їх параметри згрупували в такі категорії:

- *Розробка контенту* - є одним з ключових компонентів всієї системи. Саме тут починається життєвий цикл будь-якого матеріалу що публікується на сайті. На цьому етапі відбувається створення, редагування та затвердження контенту, а роль системи полягає в автоматизації цих процесів.

- *Управління сайтом* - відбувається розробка самого сайту, попередній перегляд і публікація підготовленого контенту. Тут розробляється зовнішній вигляд, готуються шаблони, розподіляються ролі користувачів і класифікація необхідної бізнес-інформації (наприклад, товари, ціни). Важливими компонентами

цього рівня є служби що підтримують своєчасність надходження необхідного контенту.

- *Доставка контенту.* Коли сайт повністю підготовлений до публікації, необхідні ресурси для динамічного формування Web-сторінок в залежності від виду конкретних користувачів. У зв'язку з цим, одним з важливих компонентів даного етапу є персоналізація або розподіл профілів, щоб кожен користувач отримував тільки ту інформацію, яка відповідає його ролі. CMS перетворилися із просто систем управління контентом в повнофункціональні системи управління сайтом. В більшості випадків CMS будуються із програмного ядра — «движка» та модулів, що підключаються. «Движок» CMS — це програма, яка підпорядкована певному алгоритму введення і виведення інформації. Цей алгоритм у кожної CMS різний Модульна структура дозволяє розробникам охопити велике коло користувачів з різними вимогами до функційності порталу, програмістам - створювати свої модулі під конкретні задачі чи замовлення, а користувачам - отримати зручність і можливість мобільно реалізовувати різні свої ідеї.

- *Оформлення Веб-сайту.* Художнє та дизайнерське оформлення сайтів беруть на себе так звані "шаблони". Це розроблене дизайнерське та художнє рішення сайту, що пропонується кожним CMS при його встановленні. Розробка нового свого шаблону досить трудомісткий процес. Використання стандартних шаблонів значно полегшує створення сайтів, але зменшує їх індивідуальність. Значно полегшують роботу користувачів з CMS при введенні та редагуванні інформації вбудовані в них редактори (їх часто називають WYSIWYG).

Системи управління контентом, або движки, пропонують компроміс між цими двома варіантами. За рахунок обмежень, накладених на логічну структуру контенту, зовнішній вигляд (дизайн) і функціональні можливості створюваних динамічних сайтів, такі системи дозволяють радикально знизити трудомісткість розробки та підтримки.

Поділяються на такі види: *просту, шаблонну, професійну та універсальну системи управління контентом.*

Проста система управління конвентом. Інтерфейс користувача контент-системи збирається з програмних модулів, набір яких визначається в індивідуальному порядку для кожного окремого проекту. Модулі один раз налаштовуються розробником, чим жорстко закріплюється структура проекту. Від користувача системи потрібне знання розмітки HTML. Система функціонує на основі динамічного формування сторінок (щоразу при запиті користувача) і обмеженій пропускній спроможності (в залежності від апаратного забезпечення) - до 3 000-5 000 відвідувачів на добу.Проста система сумісна з певними платформами і типами СУБД.

Шаблонна система управління конвентом. Інтерфейс користувача системи представлений єдиним модулем або набором модулів з жорстко закріпленою структурою сайту. Шаблонна система функціонує на основі динамічного формування сторінок або використання кешування даних.

Професійна система управління конвентом представлена інтуїтивно зрозумілим для користувача інтерфейсом і розширеними можливостями редагування. На основі таких систем управління контентом створюються самі різноманітні сайти. Надається можливість підключення додаткових модулів, як від розробника, так і прикладного ПЗ. Формовані динамічні сторінки кешуються, що веде до безмежної пропускної здатності, яка залежить тільки від апаратного забезпечення. Система сумісна з різними програмно-апаратними платформами. Рівень пропонованої в системі масштабованості дозволяє підключати додаткові модулі без порушення структури та ідеології керування веб-ресурсом.

Універсальна система управління конвентом. Користувацький сервіс представлений передовими засобами управління контентом, система надає засоби для розробки нових сервісів і можливостей. За технічними характеристиками дана система повністю пристосована до внутрішньокорпоративного використання в зв'язці з ERP-пакетами, що забезпечено наявністю сертифікованої системи безпеки - розмежування прав доступу до контент-системи на внутрішньокорпоративному рівні. У процесі функціонування системи проводиться кешування формованих

динамічних сторінок і при цьому пропускна здатність - не обмежена. Система сумісна з різними програмно-апаратними платформами. Можливості масштабування дозволяють підключати додаткові модулі, що розширюють функціональність ресурсу, у тому числі готові модулі інтеграції з внутрішньокорпоративними системами. Можливе розширення за рахунок кластеризації.

- *Розповсюджені CMS.*

WordPress дозволяє створювати сайти будь-якого типу, так як має гарну розширюваність. Він простий в установці, дозволяє легко змінювати шаблони і теми оформлення, має високі SEO-характеристики

Drupal - безкоштовна CMS, з хорошою SEO-адекватністю, безпекою та розширюваністю. Движок широко використовується при побудові сайтів різного призначення. Його використано при розробці сайтів сенату США та потужного сайту рейтингування Університетів світу Webometrics.

Joomla - хорошу розширюваність, захищеність і відкритість, має великий набір шаблонів. Недолік є високе навантаження на сервер і невисоку швидкість роботи. Проблеми з SEO-адекватністю.

Bitrix - професійна платна CMS, використовують при створенні корпоративних сайтів. Досить складна в налаштуванні і редагуванні, сильно навантажує сервер, є проблеми з дублюванням сторінок і безпекою.

DataLifeEngine - платна CMS, призначена для розробки інформаційних і новинних сайтів.

4. Використання фреймворків. Фреймворк це програмний продукт, який є основою для створення сайтів, але він не має готових рішень для побудови сайтів, не має рішень для виконання певних функцій. Це більш низький рівень CMS. Фреймворк (в інформаційних системах)- це структура програмної системи, що полегшує розробку і об'єднання різних компонентів великого програмного проекту. Можна також говорити про каркасний підхід як про підхід до побудови програм, де будь-яка конфігурація програми будується з двох частин: перша,

постійна частина - каркас, незалежний від конфігурації до конфігурації і несе в собі гнізда, в яких розміщується друга, змінна частина - змінні модулі (або точки розширення).Відомі такі види фреймворкних продуктів:

Фреймворк програмної системи - це каркас системи або підсистеми. Він може включати допоміжні програми, мови сценаріїв, все, що полегшує розробку і об'єднання різних компонентів. Від бібліотеки він відрізняється виконанням коду, який написаний для нього, але не виконується сам. До цього виду фреймворків відносяться і фреймворки для WEB.

Фреймворк додатку має стандартну структуру. З ростом необхідності в графічних інтерфейсах користувача з'явилася і необхідність у фреймворках додатків. З їх допомогою простіше створювати засоби для створення графічних інтерфейсів автоматично. Для створення фреймворку додатків використовують об'єктно-орієнтоване програмування. Перший такий Фреймворк написала компанія Apple для Macintosh. Спочатку він був створений за допомогою Паскаль, потім же перероблений в C ++.

Фреймворк концептуальної моделі - це абстрактне поняття даної структури для визначення способів вирішення конкретної проблеми.

WEB фреймворки - це каркас, призначений для створення динамічних веб-сайтів, мережевих додатків, сервісів або ресурсів. Він спрощує розробку і позбавляє від необхідності написання рутинного коду. Багато фреймворків спрощують доступ до баз даних, розробку інтерфейсу, а також зменшують дублювання коду .

Є п'ять типів веб-фреймворків:

Request-based: фреймворки, які безпосередньо обробляють вхідні запити. Збереження стану відбувається за рахунок серверних сесій. Приклади: Django, Ruby на Rails, Struts, Grails.

Component-based: фреймворки, які абстрагують обробку запитів всередині стандартних компонентів і самостійно стежать за станом. Своєю поведінкою дані

каркаси нагадують стандартні програмні графічні інтерфейси.. Приклади: JSF, Tapestry, Wicket.

Hybrid-based: фреймворки, які комбінують Request-based та Component-based фреймворки, беручи під свій контроль всі дані і логічний потік в заснованої на запиті моделі. Розробники мають повний контроль над URL, формами, параметрами, cookies і pathinfos. Однак замість того, щоб відобразити дії і контролери безпосередньо до запиту, гібридні фреймворки забезпечують об'єктну модель компонентів, яка поводиться тотожно в багатьох різних ситуаціях, таких як окремі сторінки, перервані запити, подібні порталу фрагменти сторінок і інтегровані віджети. Компоненти можуть розподілятися окремо і ефективно інтегруватися в інші проекти. Приклади: RIFE.

Meta -based: у фреймворків є ряд базових інтерфейсів для загального обслуговування і основу яка легко розширюється, для інтегрування компонентів і служб. Приклад: Keel. *RIA-based:* фреймворки для розробки Rich Internet Applications (RIA). Служать для розробки повноцінних додатків, що запускаються всередині браузера. Приклад: Flex.

5. SaaS-платформи для створення сайтів - це можливість запустити досить простий веб-проект дуже швидко і досить дешево (ще і на умовах оренди і без необхідності хостінгу). Щось подібне до Гугл сайтів, але зі значно більшими можливостями. Рішення підходить для простих сайтів, тимчасових проектів і для перевірки бізнес-ідей. SaaS-платформи, як і CMS, бувають специфічними (наприклад, тільки для інтернет-магазинів) і універсальними (для всіх типових видів сайтів).

Для розроблення Веб-орієнтованої системи автомобільної компанії нами використано ***Стек технології MEVN*** - це набір фреймворків та інструментів, які використовуються для розробки програмного продукту. Ці фреймворки та інструменти були спеціально вибрані для забезпечення функціональності програмного забезпечення найбільш оптимізованим способом. Стек MEVN — це стек програмного забезпечення JavaScript з відкритим кодом, який з’явився як

новий і розвивається спосіб створення потужних і динамічних веб-додатків. Його програмні компоненти можна використовувати для ефективного проектування розробки інтерфейсу та бекенда та покращення функціональності веб-сайту чи програми.

MEVN містить наступне забезпечення: MongoDB, Express.js, VueJS, Node.js.

MongoDB - документно-орієнтована база даних без SQL, яка використовується для зберігання даних програми.

ExpressJS - фреймворк, розміщений поверх NodeJS, який використовується для створення бекенда сайту за допомогою функцій і структур NodeJS. Оскільки NodeJS в першу чергу розроблявся для запуску JavaScript на машині замість створення веб-сайтів, ExpressJS був створений для останньої мети.

Vue JS− називають фреймворком на стороні клієнта і особливо використовується у веб-розробці інтерфейсу. Він має двостороннє прив'язування даних, що забезпечує безперебійну розробку інтерфейсу разом із можливостями MVC та інтерактивними додатками на стороні сервера.

NodeJS− середовище виконання JavaScript. Він використовується для запуску JavaScript на комп'ютері, а не в браузері.

1.3. Висновки до розділу 1

У першому розділі розглянуто характеристики веб-додатку та інтерфейсів. Здійснено аналіз предметної області, відзначені недоліки та переваги мов програмування, в результаті чого були виділені основні вимоги до роброблюваного веб-додатку для адміністрування баз даних. Основними методами розроблення веб-сисем визначено: за допомогою мови розітки гіпертексту(Html); за допомогою програмних елементів імплементації сайтів;за використанням комплесних движків таких як CMS; з використанням популярних на цей час фреймворків; на software-as-a-service -платформах у CLOUD**.**

Для створення ПЗ було вибрано методологію на основі використання популярних фреймворків *MEVN* – це стек програмного забезпечення *JavaScript* з

відкритим кодом, який з’явився як новий і розвивається спосіб створення потужних і динамічних веб-додатків. Його програмні компоненти дозволять проектувати розробку інтерфейсу та бекенда та покращити функціональність веб-сайту. Для зберігання даних програми документно-орієнтована база даних без *SQL* використано *MongoDB; ExpressJS* використовано для створення бекенда сайту за допомогою функцій і структур *NodeJS*; для веб-розробки клієнтського інтерфейсу використано *VueJS*; для запуску системи застосовано *Node.js.* Додаток буде розміщено на серверній платформі *Node.js локально*.

РОЗДІЛ 2
ЕТАПИ ТА ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

2.1. Етапи розробки веб-орієнтованого додатку

Цикл розробки ПЗ складається з процесів:

- Аналіз вимог, розроблення технічного завдання (ТЗ);
- Проектування. Створення макета веб-інтерфейсу, концепції веб-орієнтованого додатку
- Розробка та кодування;
- Функціональний та інтеграційний тестінг;
- Супроводження та підтримка роботи.

2.2. Технічні вимоги до веб-орієнтованого додатку

1. Тематика і призначення веб-ресурсу:

Мета веб-орієнтованого продукту– адміністрування веб-додатку на основі Mevn cnfre.

Веб-орієнтований додаток виконує такі функції:

- Підключіться до сервера баз даних за допомогою імені користувача та пароля;
- Можливість вибору існуючу бази даних або створення нової;
- Відображення користувачів, їх прав, можливість їх змінити;
- Створення, відображення, оновлення, видалення баз даних;
- Створення, відображення, оновлення, видалення таблиць;
- Додавання та видалення стовпців;

- Пошук даних в базах даних, таблицях та відображення результатів;
- Підтримка основних типив даних;
- Адміністрування декількох серверів.

Основні *цілі* створення ***web-*** додатку наступні:

- створення позитивного іміджу автомобільної компанії та її продуктів;
- забезпечення комерційного успіху підприємству за рахунок просування веб-додатку в, а також просування та продажу продуктів автомобільної компанії шляхом залучення відвідувачів, перетворення їх в постійних споживачів автомобільних продуктів;

Виходячи із зазначених цілей, основними завданнями, що стоять при розробці веб-додатку, є:

- визначення основних структурних і технологічних елементів сайту;
- визначення відповідності додатку цілям і задачам автомобільної компанії
- визначення цільової аудиторії, з урахуванням потреб і потреб якої розроблятиметься додаток;
- підготовка технічного опису web- додатку, в якому повинно міститися детальний опис цілей, завдань, структури, сервісів і додаткових можливостей, а також повинні бути визначені технічні вимоги до нього, необхідне обладнання для нього у контекстах забезпечення реалізації клієнтської частини та серверної.

Основними вимогами, що пред'являються до розробки веб-орієнтованого додатку автомобільної компанії, є наступні:

- структурні частини об'єднані загальною адресою, темою, логічною структурою, оформленням;
- дизайн орієнтований на клієнтів автомобільної компанії, на рішення цілей і завдань, поставлених розробником;
- розробка дизайну (***web-design***) включає створення графічних елементів для додатку, проектування його структури, навігації і іноді навіть

движків, потрібних для роботи, створення дизайну цілком;

- візуальне оформлення ***web-*** сторінок, яке спрямоване на вирішення багатьох завдань, включаючи підвищення продажів, зміцнення довіри, створення певного іміджу підприємству;
- дотримання принципів юзабіліті (*usability*), згідно з якими додаток має забезпечувати досягнення цілей користувачів з максимальним ступенем ефективності, продуктивності та задоволеності;
- забезпечення швидкої та зручної людино-комп'ютерного взаємодії ***(HCI human- computer interaction)*** з урахуванням вимог, включаючи елементи психології, ергономіки, інформатики, графічного дизайну, соціології та антропології;
- орієнтація на потреби і вимоги споживачів (UCD - user-center design);
- використання методів і прийомів для полегшення користувачами пошуку і перегляду інформації (IA information architecture, інформаційна архітектура).

З урахуванням цих вимог *додаток дозволить:*

- забезпечити доступність і зрозумілість змісту сайту;
- врахувати вимоги оформлення дизайну;
- розташовувати ключову інформацію на видимій частині додатку;
- використовувати відповідні кольори при оформленні посилань;
- намагатися зберегти традиційне розташування обов'язкових елементів додатку;
- забезпечити зручність і простоту контенту (наповнення) додатку для вивчення і пошуку інформації.

Ці*льова аудиторія(користувачі) веб-додатку включає наступні цільові групи людей:*

- Веб-програмісти;
- Користувачі хостингів;

- Клієнти автомобільної компанії;
- Операційний персонал та менеджмент автомодільної компанії.

Мови веб-ресурсу:

Одна мовна версія веб-сайту: Англійська. У подальшому можливе створення і украхномовної версії веб-додатку. 2. Структура та функціональність веб-ресурсу

Вимоги до вибору технологій платформи:

Frontend-частина

Використання еко-системи Vue та додаткових модулів і залежностей серед яких обовязкові наступні:

- Vue 3
- Vue-router 4
- Vuex 4
- Axios: 0.21.1
- Vee-validate 4
- Bootstrap 4
- vue-fontawesome 3

Backend-частина:

- Node.js
- MongoDB
- Express.js: 4.17.1
- Додаткові модулі та залежності Cors, moongose, jsonwebtoken, bcryptjs та ін.

Вимоги до дизайну користувацького інтерфейсу:

Загальний дизайн сторінок веб-додатку має базуватися на перевірених часом та практикою веб стандартах та не повинен використовувати функціонал, доступний тільки в невеликому колі браузерів.

Дизайн повинен слудівти усім сучасним тенденція та течіям у веб-дизайні, бути простим й водночам інтуїтивно зрозумілим, практичним та сучасним. Дизайн

необхідно виконати в спокійній кольоровій гамі. Дизайн повинен бути виконаний в стилі мінімалізму (нічого зайвого, відволікаючого увагу).

Контент це наповнення або інформаційний зміст автомобільного додатку, включаючи тексти, графічні документи, фотографії, доповнюючи програмне забезпечення і т.д. Важливі параметри контенту це його обсяг, актуальність, доступність, дизайн, привабливість. З урахуванням цих аспектів відповідний контент розміщується на сайті.

Інтерфейс користувача Розробляючи інтерфейс користувача для свого сайту, важливо враховувати очікування користувачів щодо доступності, візуальної естетики та простоти використання. Оптимальне поєднання ефективних візуальних елементів та ефективної швидкості реагування покращить коефіцієнт конверсії вашого сайту, оскільки він передбачає потреби користувача, а потім задовольняє ці потреби.

Результатом робіт цього етапу є структурована система навігації, яка може мати такі *блоки:*

- розробка базової навігації додатку (створення основних шляхів переходу для користувача по сторінках додатку);
- створення авторизаційних і реєстраційних форм продукту;
- розробка контекстного пошуку;
- створення модулів додатку;
- розробка додаткової навігації сайту, що представляє собою допоміжну структуру, що служить для підтримки переходу відвідувача на інші (сусідні) сторінки і до ближніх розділах;
- проектування карти сайту або службової навігації і т.д.

Дизайн веб-інтерфейсу має коректно відображатись в наступних основних браузерах: IE 8.0 і вище, останні версії Mozilla Firefox, Opera, Chrome.

2.3. Загальна концепція веб-додатку

На сьогоднішній день існує два принципових підходи до створення веб-додатків та веборієнтованих систем:

- традиційні веб-додатки- велика частина логіки яких виконується на сервері;

- односторінкові додатки, логіка призначеного для користувача інтерфейсу яких виконується переважно в веб-браузері, а взаємодія з веб-сервером здійснюється головним чином через веб-API;

- гібридний підхід, при якому в простому випадку в рамках великого традиційного веб-додатки розміщуються одне або кілька повнофункціональних підлеглих додатків, побудованих на основі односторінковий моделі.

Більшість користувачів багатьох веб-додатків можуть працювати тільки з функціями читання. Додатки, призначені виключно або переважно для читання, зазвичай набагато простіше тих, в яких реалізується управління власністю.

Наприклад, в пошуковій системі цілком достатньо реалізувати одну точку входу з текстовим полем і другу сторінку для відображення результатів пошуку. Запити можуть виконувати анонімні користувачі, в зв'язку з чим на стороні клієнта практично не потрібно реалізовувати логіку. Аналогічним чином, публічні додатки блогів або систем управління вмістом працюють переважно з вмістом і практично не мають функцій, що реалізуються на стороні клієнта.

Такі додатки легко створювати в форматі традиційних серверних веб-додатків, які виконують логіку на веб-сервері і перетворюють HTML-код для відображення в браузері. Той факт, що кожна унікальна сторінка сайту має власний URL-адресу, який може додаватися в закладки або індексуватися пошуковими системами (така поведінка реалізується за замовчуванням і не вимагає додавання в додаток окремої функції), також є очевидною перевагою такого сценарію.

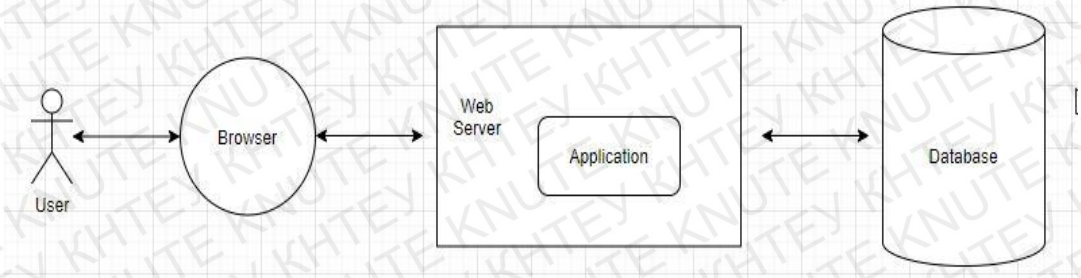


Рис. 2.1. Загальна концепція проектованого веб-додатку

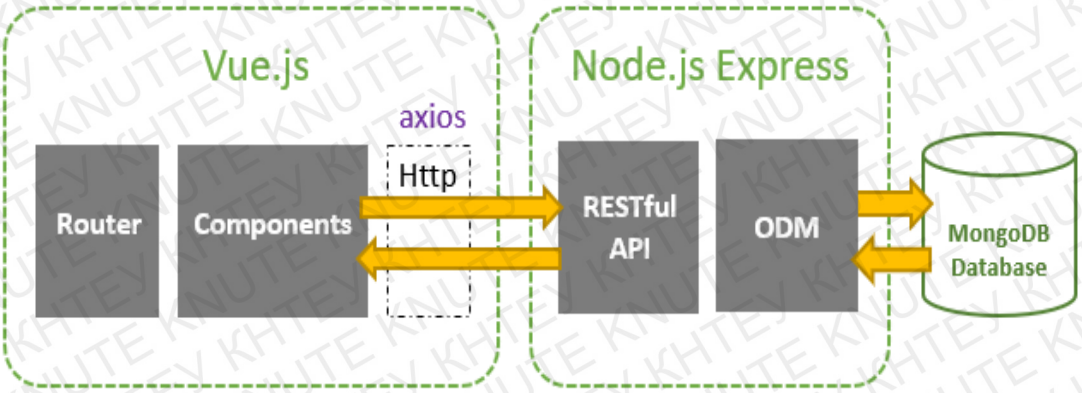


Рис. 2.2 Блок-схема моделі взаємодії між компонентами шаблону MEVN

Node.js Express експортує REST API та взаємодіє з базою даних MongoDB за допомогою Mongoose ODM. Vue клієнт надсилає HTTP-запити та отримує HTTP-відповіді за допомогою axios, споживає дані про компоненти. Vue Router використовується для переходу до сторінок.

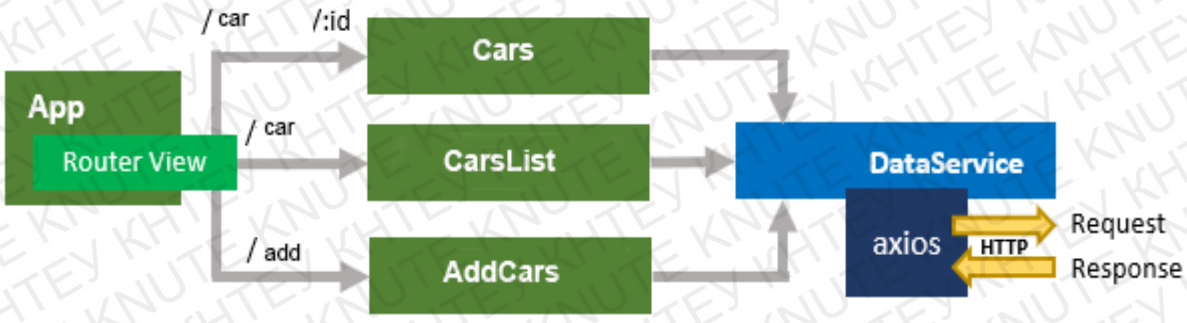


Рис. 2.3. Схема Frontend частини

App – це контейнер із переглядом маршрутизатора. Він має навігаційну панель, яка посилається на маршрути. Компонент CaesList отримує та відображає список автомобілів. Компонент Cars має форму для редагування деталей підручника на основі :id. Компонент AddCars має форму для подання нового підручника. Ці компоненти викликають методи DataService, які використовують axios для виконання запитів HTTP та отримання відповідей.

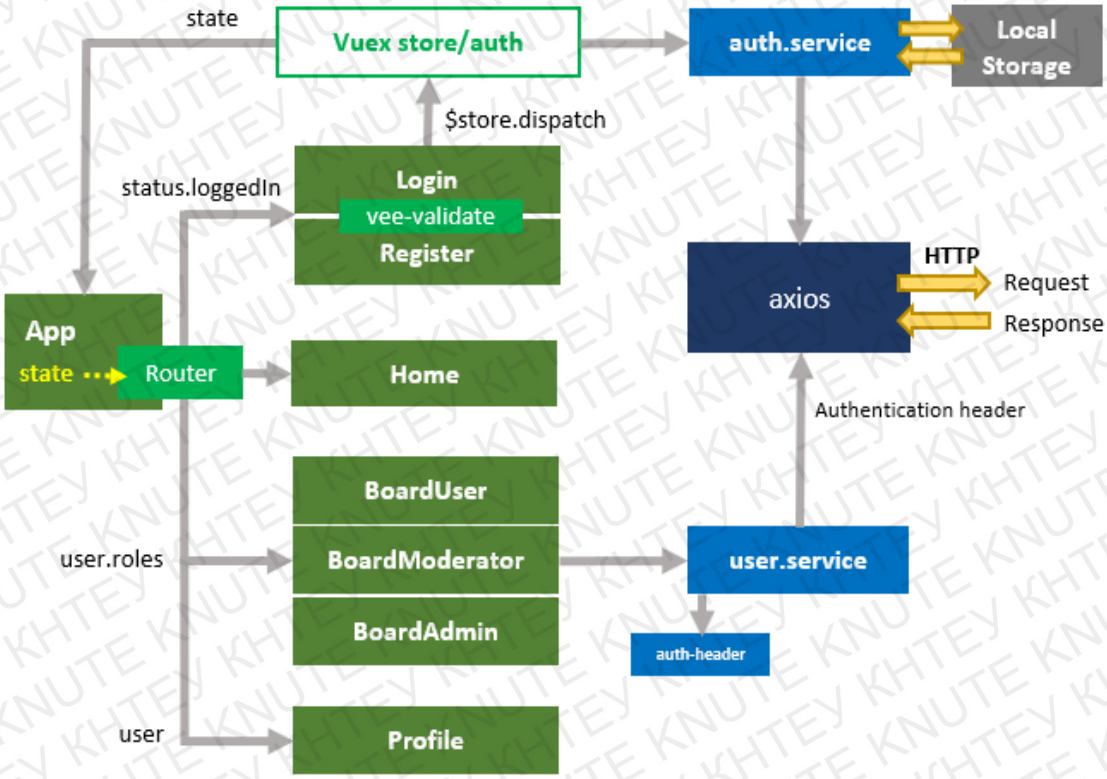


Рис. 2.4. Діаграма JWT авторизпції

Компонент App – це контейнер з маршрутизатором. Він отримує стан програми з Vuex store/auth. Панель навігації тепер може відображатися на основі стану. Компонент програми також передає стан своїм дочірнім компонентам. Компоненти «Login і Register» мають форму для подачі даних (з підтримкою vee-validate). Ми викликаємо функцію Vuex store dispatch для виконання дій входу/реєстрації.

Наші дії Vuex викликають методи auth.service, які використовують axios для виконання запитів HTTP. Ми також зберігаємо або отримуємо JWT з локального сховища браузера всередині цих методів. Home компонент є загальнодоступним для всіх відвідувачів. Компонент профілю отримує дані користувача зі свого батьківського компонента та відображає інформацію про користувача. Компоненти BoardUser, BoardModerator, BoardAdmin будуть відображатися в стані user.roles Vuex. У цих компонентах ми використовуємо user.service для отримання захищених ресурсів з API. user.service використовує допоміжну функцію auth-header(), щоб додати JWT до заголовка авторизації HTTP. auth-header() повертає об'єкт, що містить JWT поточного користувача, який увійшов у систему з локального сховища.

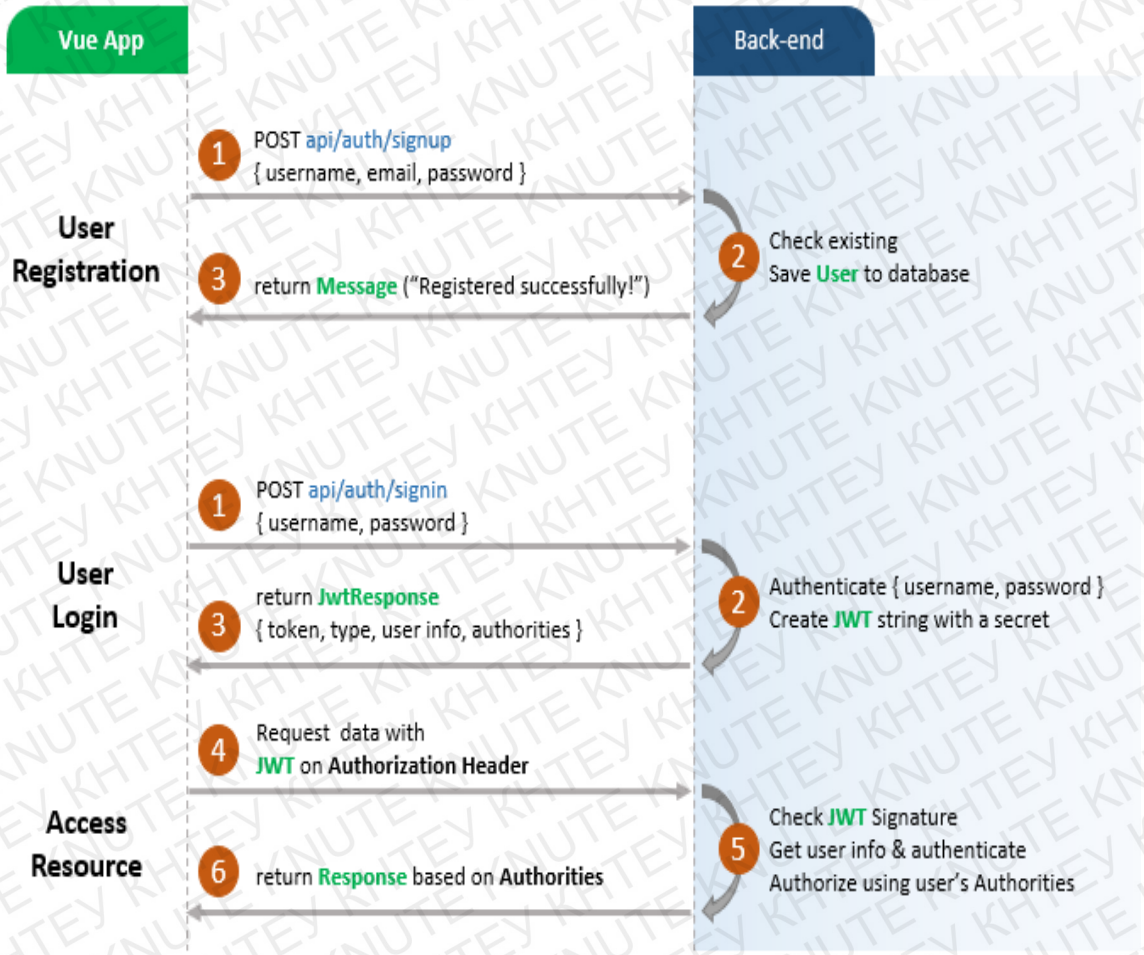


Рис. 2.6. Блок-схема автентифікації JWT

Для автентифікації JWT (рис.2.6) ми будемо викликати 2 кінцеві точки: POST api/auth/signup для реєстрації користувача POST api/auth/signin для входу користувача можна переглянути наведений нижче потік, щоб отримати огляд запитів та відповідей, які буде робити або отримувати Vue Client.

Всі етапи створення сайту важливі для якісного функціонування ресурсу(рис. 2.7, рис. 2.8)., але розробка дизайну - один з головних. Адже дизайн - це те, що бачить відвідувач в першу чергу, оцінює його і приймає рішення залишитися на сторінці або закрити вкладку браузера. Використовуючи сайт WireFrame був створений візуальний начерк користувацького інтерфейсу (рис.2.9)

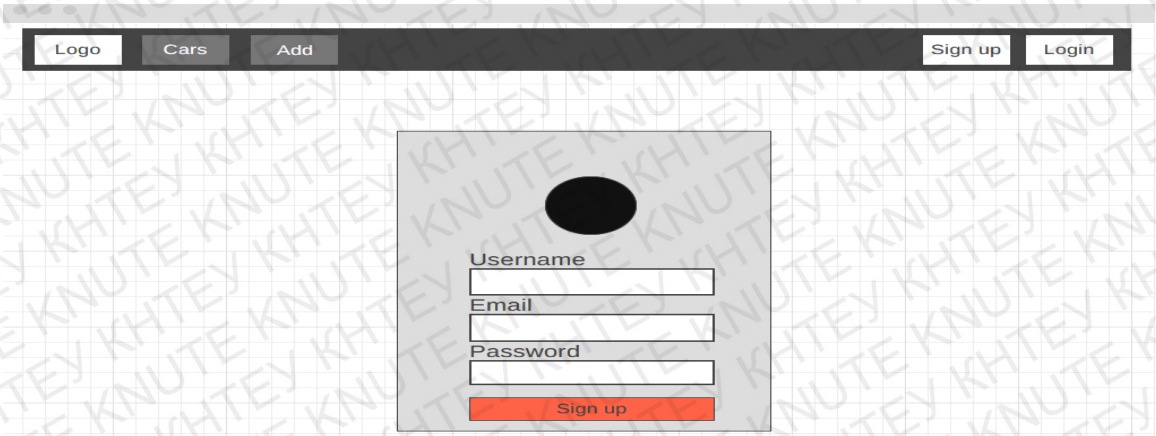


Рис. 2.7. Візуальний начерк сторінки регістрації

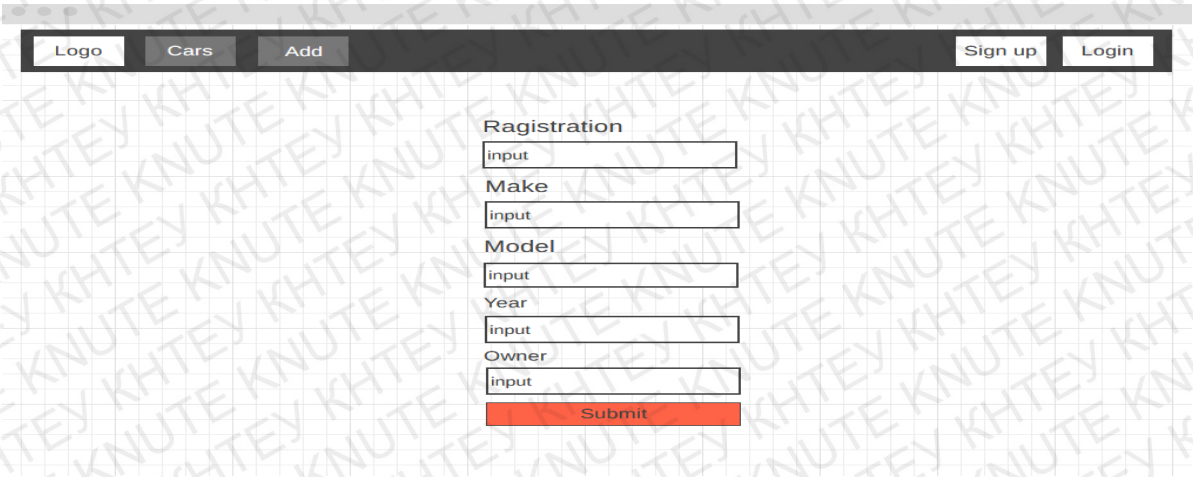


Рис. 2.8. Сторінка додавання нових данних

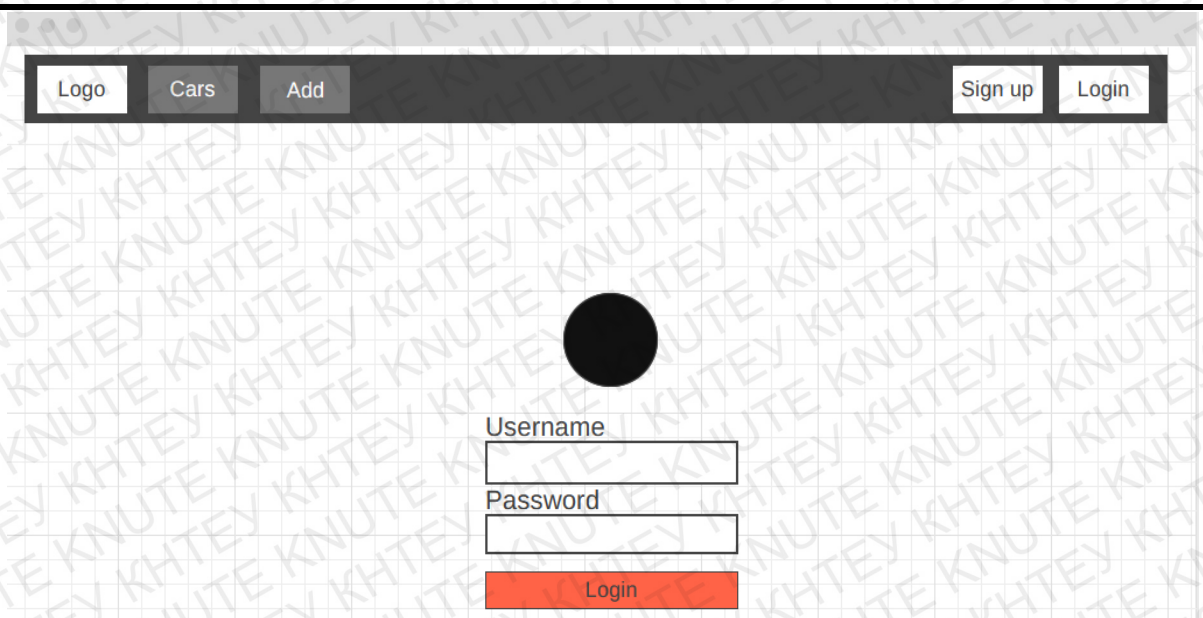


Рис. 2.9. Блок-схема авторизації користувача

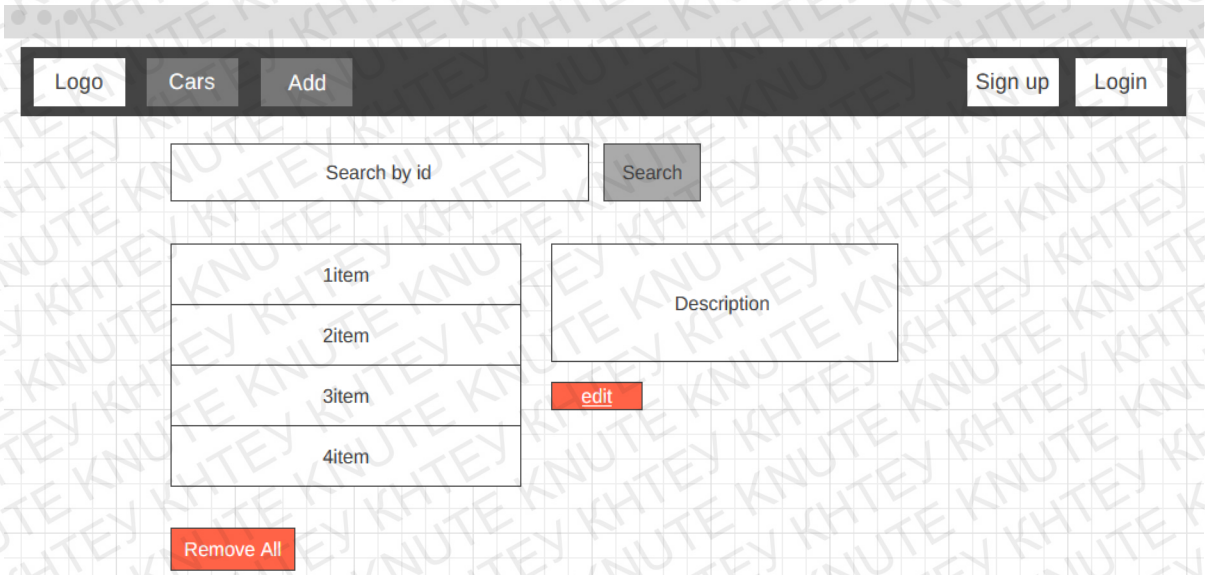


Рис. 2.10. Візуальний начерк сторінки додавання нових данних

Ґрунтуючись на технічному завданні, створюю кнопки відповідно до зазначених функцій, списки для відображення даних , та інші графічні елементи.

Іншими словами, той прототип, який був створений на першому етапі розробки сайту, отримує зовнішній вигляд користувацького інтерфейсу.
Продемонструємо кілька зображень та функціонування веб-додатку.

2.4. Верстка інтерфейсу та кодування логіки додатку

За допомогою відкритого фреймворку Vue.js (веб-фреймворк для розробки користувацьких інтерфейсів на мові програмування JavaScript. Vue створений для поступового впровадження в існуючий додаток. Він дозволяє вирішувати різні задачі рівня представлення (view), спрощує роботу з іншими бібліотеками та допомагає створювати складні одностраничні додатки (SPA, Single-Page Applications). Це фреймворк, котрий використовують веб-програмісти для швидкої та ефективної розробки сайтів та веб-додатків.) та його екосистеми з встановленими компонентами та залежностями(Bootstrap,vue router,vuex і тд),оговореними в розділі раніше,розробляємо зовнішній вигляд додатку. Ресурс відображається у всіх існуючих браузерах. Все це доповниться backend кодом за допомогою Node.js Express.js з встановленими доповненнями.

За допомогою середовища розробки Vscode було написано зовшішня частина частина веб-додатку (рис.2.11).

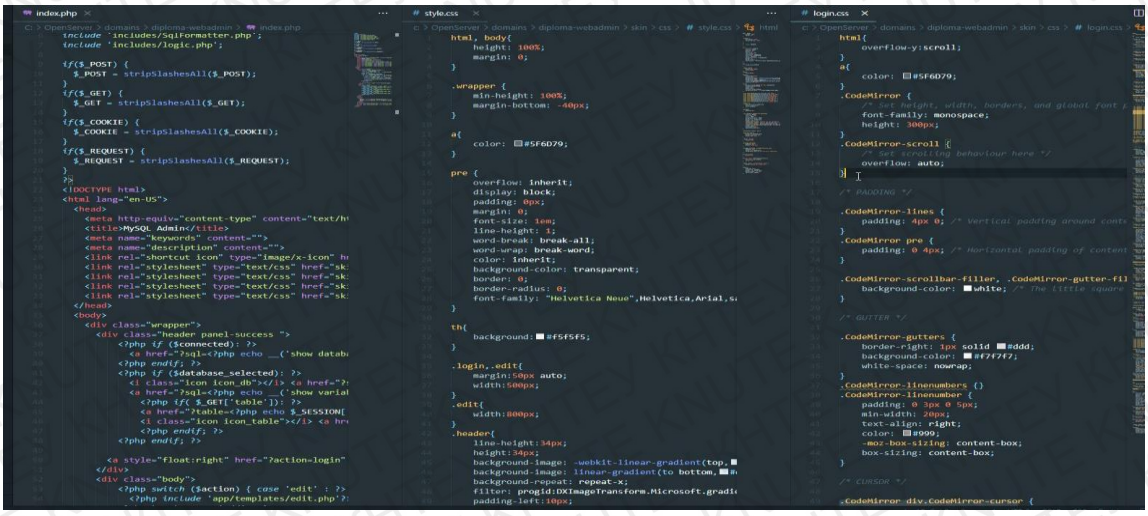


Рис. 2.11. VScode та код MEVN

Для динамічного відображення,зберігання та обробки даних та іншого – було використано констркції фреймворку.

2.5. Встановлення MongoDB та налаштування проекту

В першу чергу нам необхідно перейти на офіційний сайт mongoDB та встановити MongoDB Community Edition (рис. 2.12). Яка відповідає нашим потребам.

Виконуэмо кроки які написані в офіційному туторіалі.

Рис. 2.12. MongoDB Community Edition

Всі необхідні нам модули є в складі базової версії цього продукту. Тому завантажуємо та розгортаємо її локально на компьютері. Тепер маємо портативну серверну платформу.

Щоб успішно запустити наш проект необіхідно налагодити перевірити правильність виконання налаштувань.Зробити це можна виконавши команду mongod в командній commant prompt: mongod

Результат:

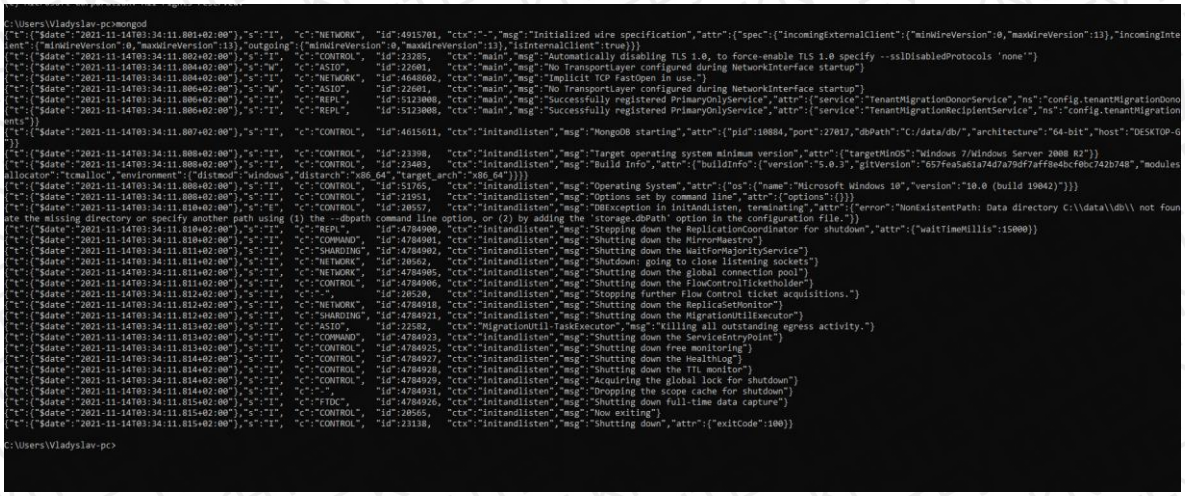


Рис. 2.13. MongoDB налаштування

На вкладці «Кодування» встановлюємо стандартне кодування «UTF-8_GENERAL_CI».

Необхідно також встановити node.js і vue cli та перевірити налаштування:

Запустити команди node та vue –version:

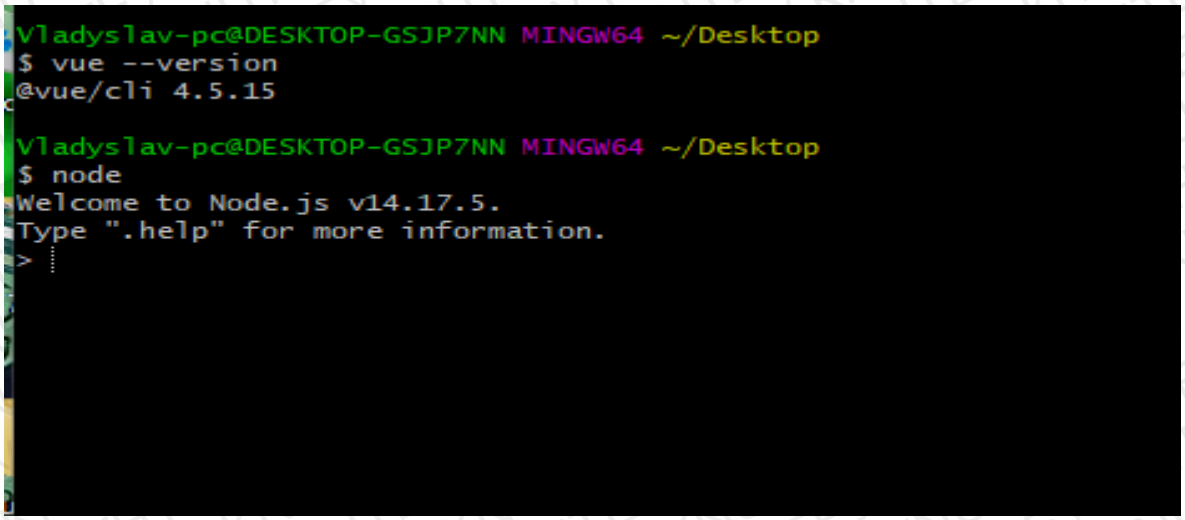


Рис. 2.14. Запуск команди node та vue –version

На вкладці «Домени» додаємо новий локальний домен який будемо використовувати для запуску проекту.

На цьому наши налаштування завершуються. Але щоб запустити проект, необхідно запускати одночано сервер бекенда фронтеда і MongoDB(npm run serve/node server.js).Після цього, коли всі необхідні файли будуть завантажені - можна починати роботу.

2.6. Висновки до розділу 2

Отже, в цьому розділі мною досліджено особливості веб-додатку, його специфікація та було сформована загальна концепція веб-додатку.

Були представлені етапи розробки веб-додатку. Згідно яких буде проводитися розробка ПЗ. Візуальний нарис веб-додатку був розроблений за допомогою веб-ресурсу Wireframe.

Як середовище розробки була обрана VScode.

Для реалізації поставленної мети була розгорнута та налаштована серверна платформа node js та MongoDB, для подальшої відладки проекту.

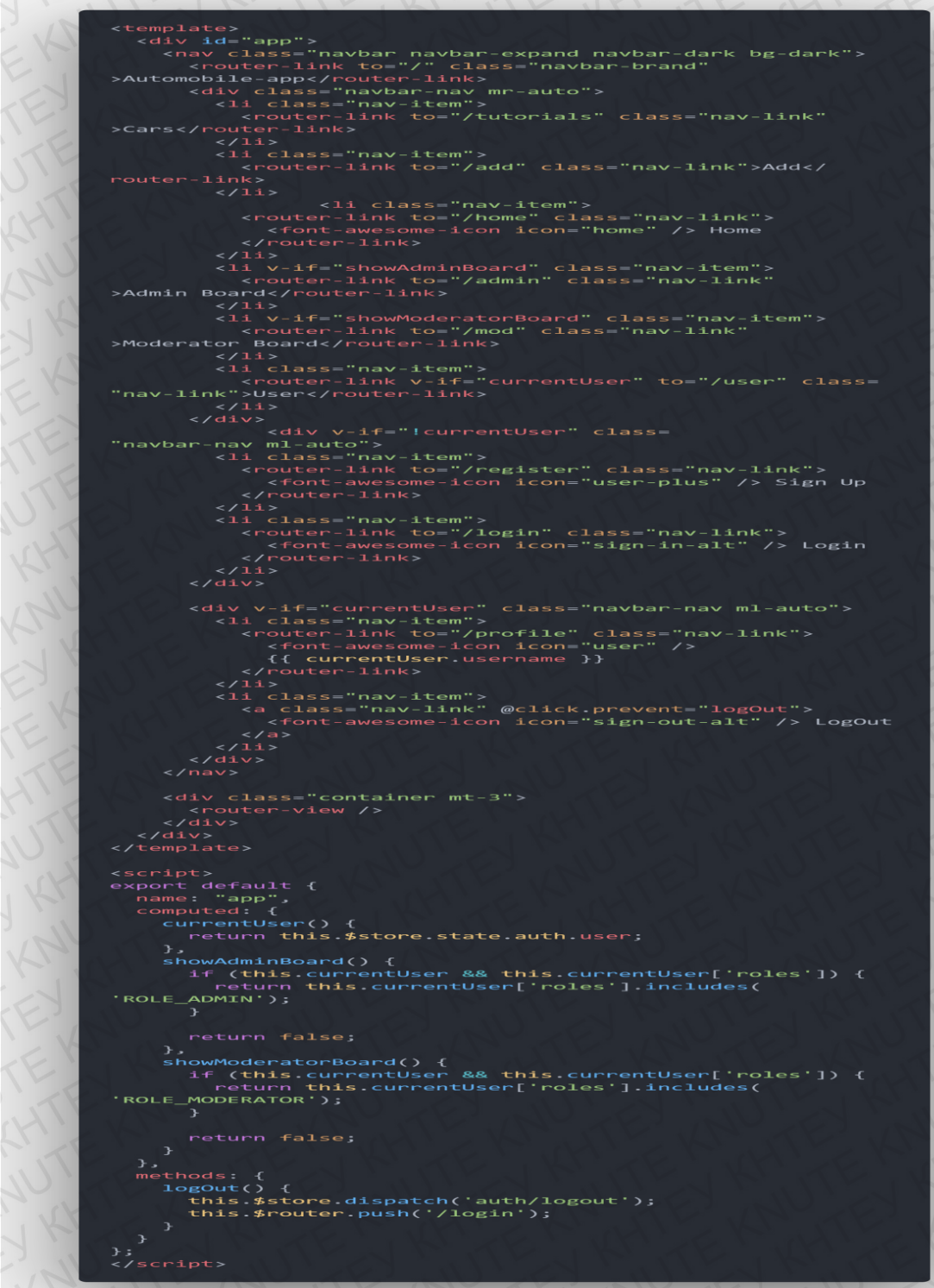
РОЗДІЛ 3
РЕАЛІЗАЦІЯ ЗАСОБУ АДМІНІСТРУВАННЯ БАЗИ ДАНИХ ЧЕРЕЗ ВЕБ-ІНТЕРФЕЙС

3.1. Елементи користувацького інтерфейсу

Веб-додаток складається з 4х темплейтів та однієї головної сторінки яка переключає ці темплейти:

- Сторінка логіну;
- Сторінка регістрації;
- Основна сторінка;
- Сторінка інформації про аккаунт;
- Сторінка загальної інофрмації;
- Сторінка редагування запису;
- Сторінка додавання запису.

Коли користувач взаємодіє з веб-додатком через веб-інтерфейс він виконує деякі дії і в залежності від дії, яку виконує користувач, перемикаються темплейти інтерфейсу. (рис. 3.1.)



```
<template>
  <div id="app">
    <nav class="navbar navbar-expand navbar-dark bg-dark">
      <router-link to="/" class="navbar-brand"
>Automobile-app</router-link>
      <div class="navbar-nav mr-auto">
        <li class="nav-item">
          <router-link to="/tutorials" class="nav-link"
>Cars</router-link>
        </li>
        <li class="nav-item">
          <router-link to="/add" class="nav-link">Add</
router-link>
        </li>
              <li class="nav-item">
          <router-link to="/home" class="nav-link">
            <font-awesome-icon icon="home" /> Home
          </router-link>
        </li>
        <li v-if="showAdminBoard" class="nav-item">
          <router-link to="/admin" class="nav-link"
>Admin Board</router-link>
        </li>
        <li v-if="showModeratorBoard" class="nav-item">
          <router-link to="/mod" class="nav-link"
>Moderator Board</router-link>
        </li>
        <li class="nav-item">
          <router-link v-if="currentUser" to="/user" class=
"nav-link">User</router-link>
        </li>
      </div>
            <div v-if="!currentUser" class=
"navbar-nav ml-auto">
        <li class="nav-item">
          <router-link to="/register" class="nav-link">
            <font-awesome-icon icon="user-plus" /> Sign Up
          </router-link>
        </li>
        <li class="nav-item">
          <router-link to="/login" class="nav-link">
            <font-awesome-icon icon="sign-in-alt" /> Login
          </router-link>
        </li>
      </div>

      <div v-if="currentUser" class="navbar-nav ml-auto">
        <li class="nav-item">
          <router-link to="/profile" class="nav-link">
            <font-awesome-icon icon="user" />
            {{ currentUser.username }}
          </router-link>
        </li>
        <li class="nav-item">
          <a class="nav-link" @click.prevent="logOut">
            <font-awesome-icon icon="sign-out-alt" /> LogOut
          </a>
        </li>
      </div>
    </nav>

    <div class="container mt-3">
      <router-view />
    </div>
  </div>
</template>

<script>
export default {
  name: "app",
  computed: {
    currentUser() {
      return this.$store.state.auth.user;
    },
    showAdminBoard() {
      if (this.currentUser && this.currentUser['roles']) {
        return this.currentUser['roles'].includes(
'ROLE_ADMIN');
      }

      return false;
    },
    showModeratorBoard() {
      if (this.currentUser && this.currentUser['roles']) {
        return this.currentUser['roles'].includes(
'ROLE_MODERATOR');
      }

      return false;
    }
  },
  methods: {
    logOut() {
      this.$store.dispatch('auth/logout');
      this.$router.push('/login');
    }
  }
};
</script>
```

Рис. 3.1. Фрагмент програмного коду «Переключення темплейтів інтерфейсу»

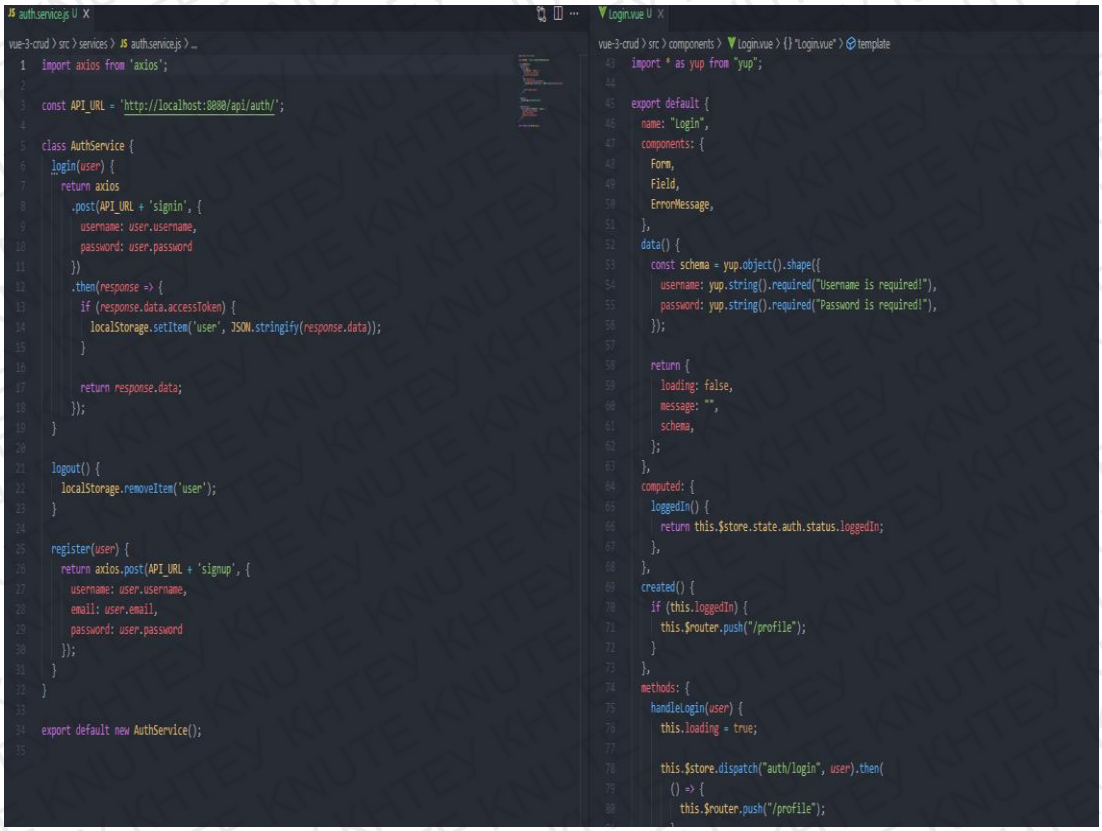
3.2. Сторінка логіну

При відкритті веб-додатку відображається темплейт інтерфейсу логіну відповідно до дії користувача «login». Ініціалізується сессія та створюється cookie запис «PHPSESSID», що містить ідентифікатор сеансу.

На сторінці логіну знаходяться такі елементи:

- Поле вводу для користувача бази даних
- Поле вводу для паролю копистувача
- Кнопка «Login»

Після того як користувач заповнив всі неодхідні поля, то виконується валідація введених даних за допомомою компонента vee-validate 4 та перевірка на наявність такого логіна і пароля в системі. Також є можливість зберегти в кукі, тобто, користувач матиме змогу автоматичного заповнення полів. Якщо валідація успішна, то виконується авторизація(рис. 3.2. 3.3)



```
// auth.service.js
import axios from 'axios';

const API_URL = 'http://localhost:8080/api/auth/';

class AuthService {
  login(user) {
    return axios
      .post(API_URL + 'signin', {
        username: user.username,
        password: user.password
      })
      .then(response => {
        if (response.data.accessToken) {
          localStorage.setItem('user', JSON.stringify(response.data));
        }

        return response.data;
      });
  }

  logout() {
    localStorage.removeItem('user');
  }

  register(user) {
    return axios.post(API_URL + 'signup', {
      username: user.username,
      email: user.email,
      password: user.password
    });
  }
}

export default new AuthService();
```

```
// Login.vue
import * as yup from "yup";

export default {
  name: "Login",
  components: {
    Form,
    Field,
    ErrorMessage,
  },
  data() {
    const schema = yup.object().shape({
      username: yup.string().required("Username is required!"),
      password: yup.string().required("Password is required!"),
    });

    return {
      loading: false,
      message: "",
      schema,
    };
  },
  computed: {
    loggedIn() {
      return this.$store.state.auth.status.loggedIn;
    },
  },
  created() {
    if (this.loggedIn) {
      this.$router.push("/profile");
    }
  },
  methods: {
    handleLogin(user) {
      this.loading = true;

      this.$store.dispatch("auth/login", user).then(
        () => {
          this.$router.push("/profile");
```

Рис. 3.2. Фрагмент програмного поля автоизації

auth.module.js - Final diplom - Visual Studio Code

auth.module.js U

vue-3-crud > src > store > auth.module.js > ...

```
import AuthService from '../services/auth.service';

const user = JSON.parse(localStorage.getItem('user'));
const initialState = user
  ? { status: { loggedIn: true }, user }
  : { status: { loggedIn: false }, user: null };

export const auth = {
  namespaced: true,
  state: initialState,
  actions: {
    login({ commit }, user) {
      return AuthService.login(user).then(
        user => {
          commit('loginSuccess', user);
          return Promise.resolve(user);
        },
        error => {
          commit('loginFailure');
          return Promise.reject(error);
        }
      );
    },
    logout({ commit }) {
      AuthService.logout();
      commit('logout');
    },
    register({ commit }, user) {
      return AuthService.register(user).then(
        response => {
          commit('registerSuccess');
          return Promise.resolve(response.data);
        },
        error => {
          commit('registerFailure');
          return Promise.reject(error);
        }
      );
    }
  },
  mutations: {
    loginSuccess(state, user) {
      state.status.loggedIn = true;
      state.user = user;
    },
    loginFailure(state) {
      state.status.loggedIn = false;
      state.user = null;
```

Рис. 3.3. Фрагмент програмного поля логіна

3.3. Інформація про користувача та основна сторінка

Після авторизації виповнюється перехід на сторінка інофрмація про аккаунт та відповідно відкривається темплейт необхідної сторінки . Відбувається перехід на /home і генеруються відповідні дані про користувача. Створюються візуальні елементи: (рис 3.4.)

- Token
- Id
- Email
- Authorities
- Profile
- Таблиця повернутими в результаті get виклику

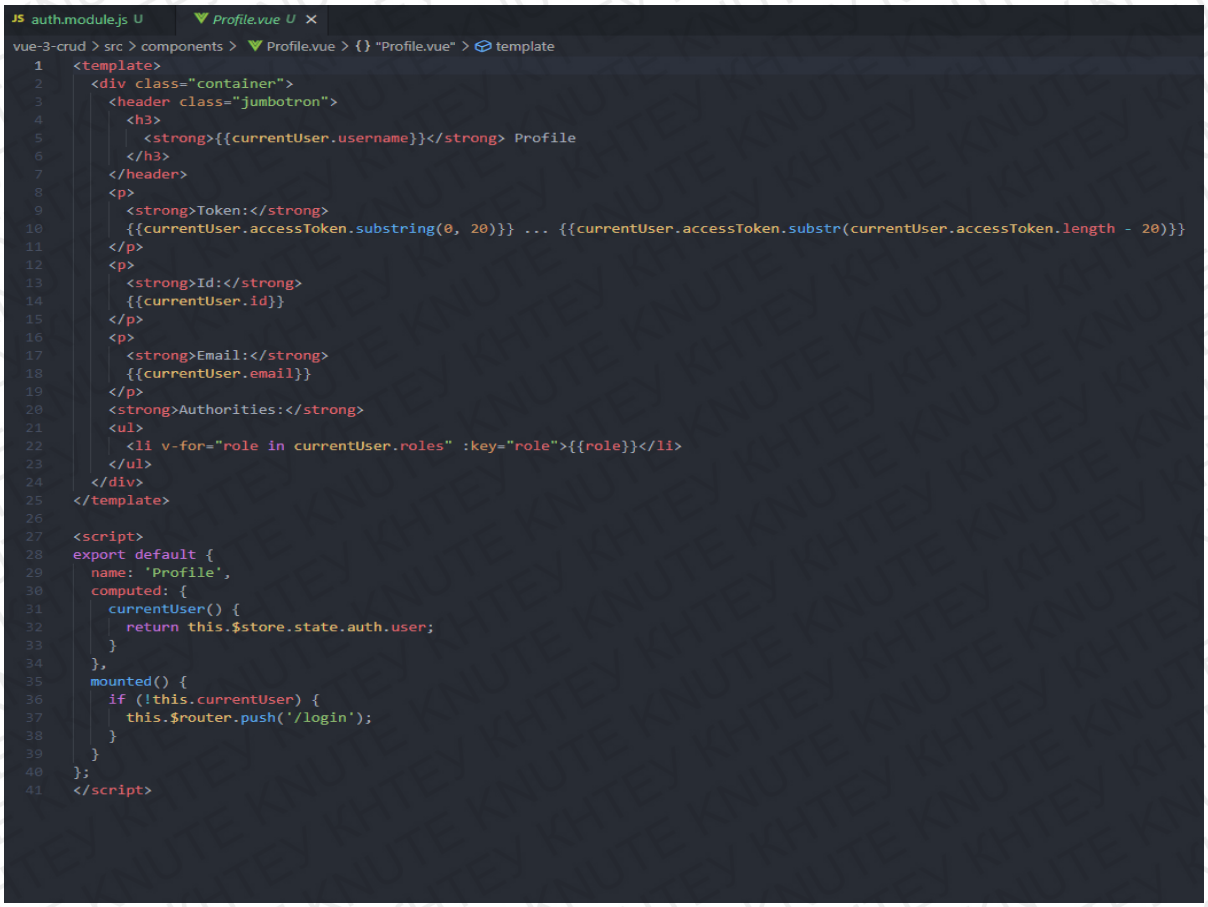


Рис. 3.4. Створення сторінки на основі отриманих данних

Ця таблиця наповнюється даними які було отримано в результаті get виклику. Записи у таблиці створюються у вигляді колекцій, з відповідною для бази даних схемою та іншими характеристиками.

Створення цих елементів виконужться за допомогою викликаних методів (рис. 3.5.)

```
import axios from 'axios';
import authHeader from './auth-header';

const API_URL = 'http://localhost:8080/api/test/';

class UserService {
  getPublicContent() {
    return axios.get(API_URL + 'all');
  }

  getUserBoard() {
    return axios.get(API_URL + 'user', { headers: authHeader() });
  }

  getModeratorBoard() {
    return axios.get(API_URL + 'mod', { headers: authHeader() });
  }

  getAdminBoard() {
    return axios.get(API_URL + 'admin', { headers: authHeader() });
  }
}

export default new UserService();
```

Рис. 3.5. Методи які надають повернутим даним відповідні візуальні характеристики.

3.4. Основна сторінка

Загалом на цій сторінці знаходяться такі елементи (рис. 3.6.):

- Поле, для пошуку конкретного запису
- Список всіх записів

- Кнопка”remove all” видалення всіх записів
- Колонка інформації про кожний запис
- Кнопка “edit” для редагування,оновлення,публікації і відправки запису в MongoDB
- Меню переходу на інші вкладки та функції, яке закріплено на всіх Сторінках

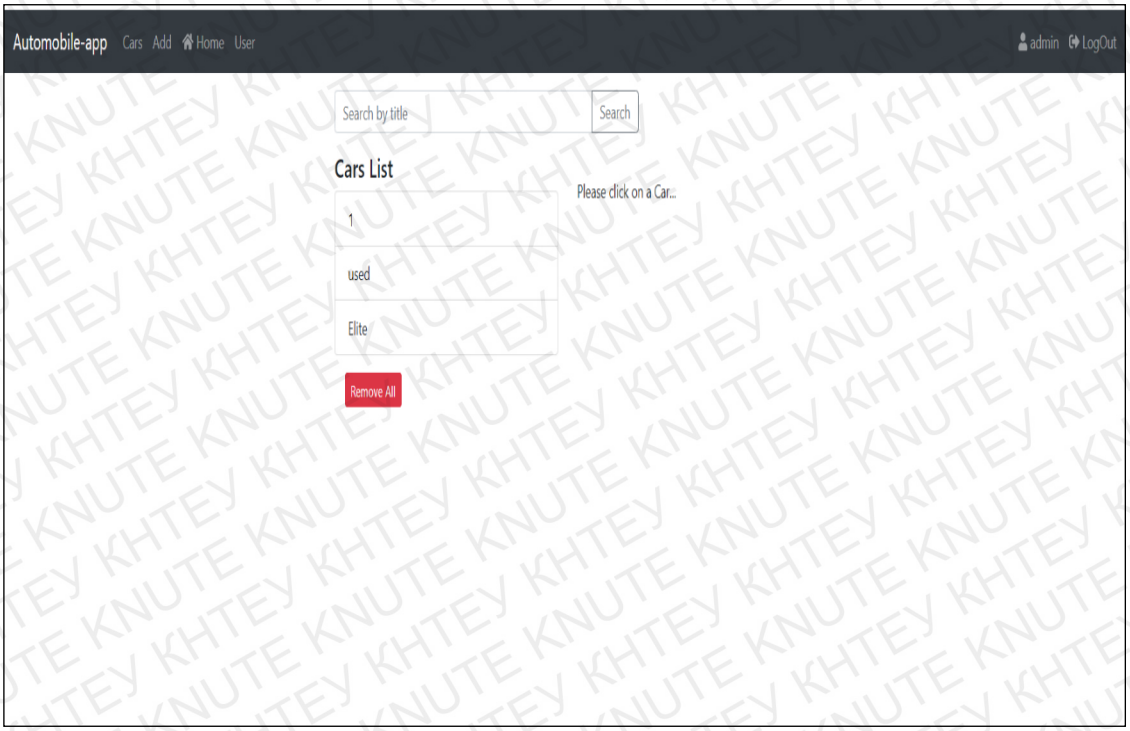
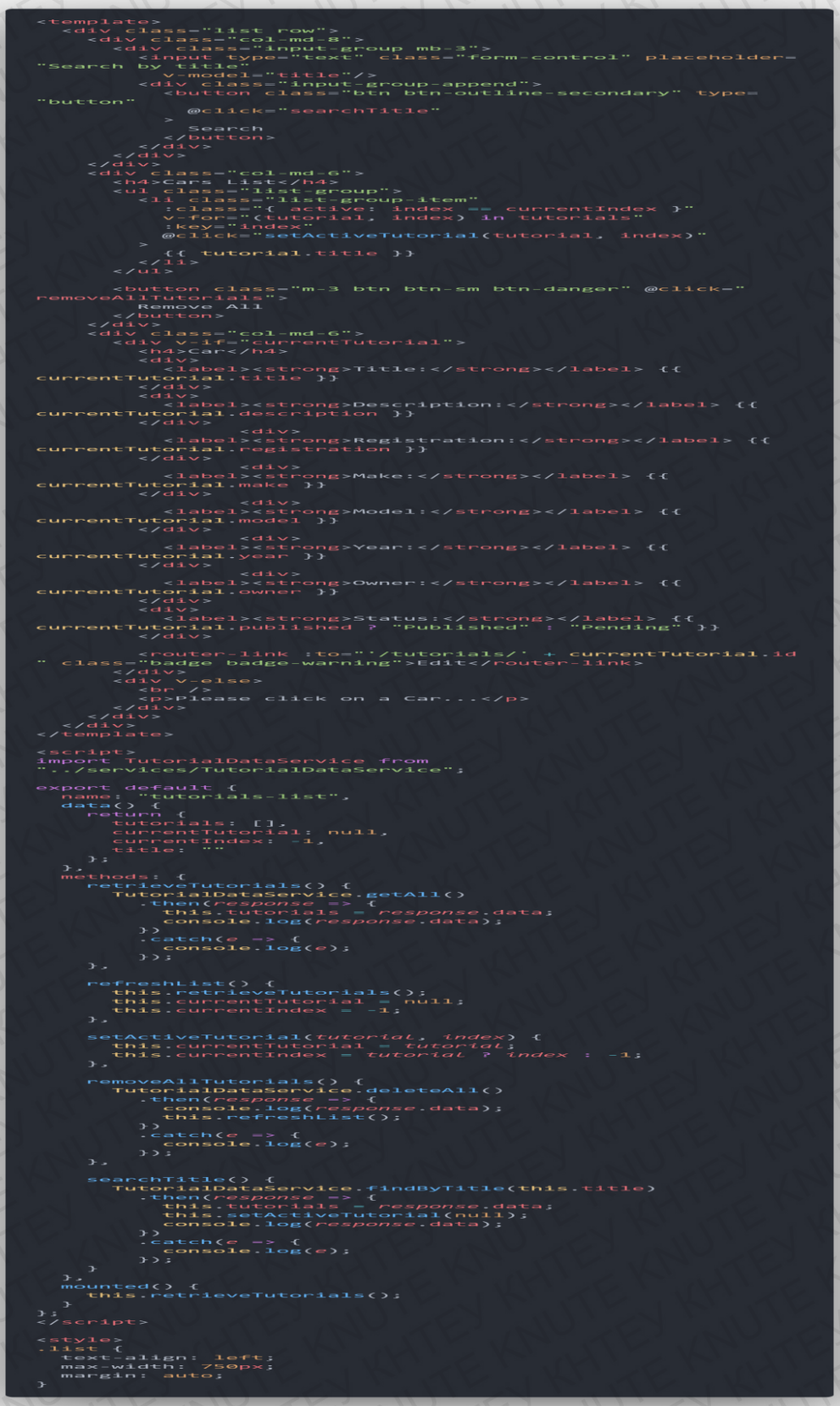


Рис. 3.6. Основна сторінка

Фрагмент реалізації представлено на рис.3.7.



```
<template>
  <div class="list row">
    <div class="col-md-8">
      <div class="input-group mb-3">
        <input type="text" class="form-control" placeholder=
"Search by title"
          v-model="title"/>
        <div class="input-group-append">
          <button class="btn btn-outline-secondary" type=
"button"
            @click="searchTitle"
          >
            Search
          </button>
        </div>
      </div>
    </div>
    <div class="col-md-6">
      <h4>Cars List</h4>
      <ul class="list-group">
        <li  class="list-group-item"
          :class="{ active: index == currentIndex }"
          v-for="(tutorial, index) in tutorials"
          :key="index"
          @click="setActiveTutorial(tutorial, index)"
        >
          {{ tutorial.title }}
        </li>
      </ul>

      <button class="m-3 btn btn-sm btn-danger" @click="
removeAllTutorials">
        Remove All
      </button>
    </div>
    <div class="col-md-6">
      <div  v-if="currentTutorial">
        <h4>Car</h4>
        <div>
          <label><strong>Title:</strong></label> {{
currentTutorial.title }}
        </div>
        <div>
          <label><strong>Description:</strong></label> {{
currentTutorial.description }}
        </div>
                <div>
          <label><strong>Registration:</strong></label> {{
currentTutorial.registration }}
        </div>
                <div>
          <label><strong>Make:</strong></label> {{
currentTutorial.make }}
        </div>
                <div>
          <label><strong>Model:</strong></label> {{
currentTutorial.model }}
        </div>
                <div>
          <label><strong>Year:</strong></label> {{
currentTutorial.year }}
        </div>
                <div>
          <label><strong>Owner:</strong></label> {{
currentTutorial.owner }}
        </div>
        <div>
          <label><strong>Status:</strong></label> {{
currentTutorial.published ? "Published" : "Pending" }}
        </div>

        <router-link :to="'/tutorials/' + currentTutorial.id
" class="badge badge-warning">Edit</router-link>
      </div>
      <div v-else>
        <br />
        <p>Please click on a Car...</p>
      </div>
    </div>
  </div>
</template>

<script>
import TutorialDataService from
"../services/TutorialDataService";

export default {
  name: "tutorials-list",
  data() {
    return {
      tutorials: [],
      currentTutorial: null,
      currentIndex: -1,
      title: ""
    };
  },
  methods: {
    retrieveTutorials() {
      TutorialDataService.getAll()
        .then(response => {
          this.tutorials = response.data;
          console.log(response.data);
        })
        .catch(e => {
          console.log(e);
        });
    },

    refreshList() {
      this.retrieveTutorials();
      this.currentTutorial = null;
      this.currentIndex = -1;
    },

    setActiveTutorial(tutorial, index) {
      this.currentTutorial = tutorial;
      this.currentIndex = tutorial ? index : -1;
    },

    removeAllTutorials() {
      TutorialDataService.deleteAll()
        .then(response => {
          console.log(response.data);
          this.refreshList();
        })
        .catch(e => {
          console.log(e);
        });
    },

    searchTitle() {
      TutorialDataService.findByTitle(this.title)
        .then(response => {
          this.tutorials = response.data;
          this.setActiveTutorial(null);
          console.log(response.data);
        })
        .catch(e => {
          console.log(e);
        });
    }
  },
  mounted() {
    this.retrieveTutorials();
  }
};
</script>

<style>
.list {
  text-align: left;
  max-width: 750px;
  margin: auto;
}
```

Рис. 3.7. Фрагмент реалізації

3.5. Вигляд сторінки додавання записів

Після вибору ‘add’. Виконується get виклик вибраної бази даних , дія index, створення візуальних елементів згідно з повернутими результатами та відображаються такі елементи (рис. 3.8.):

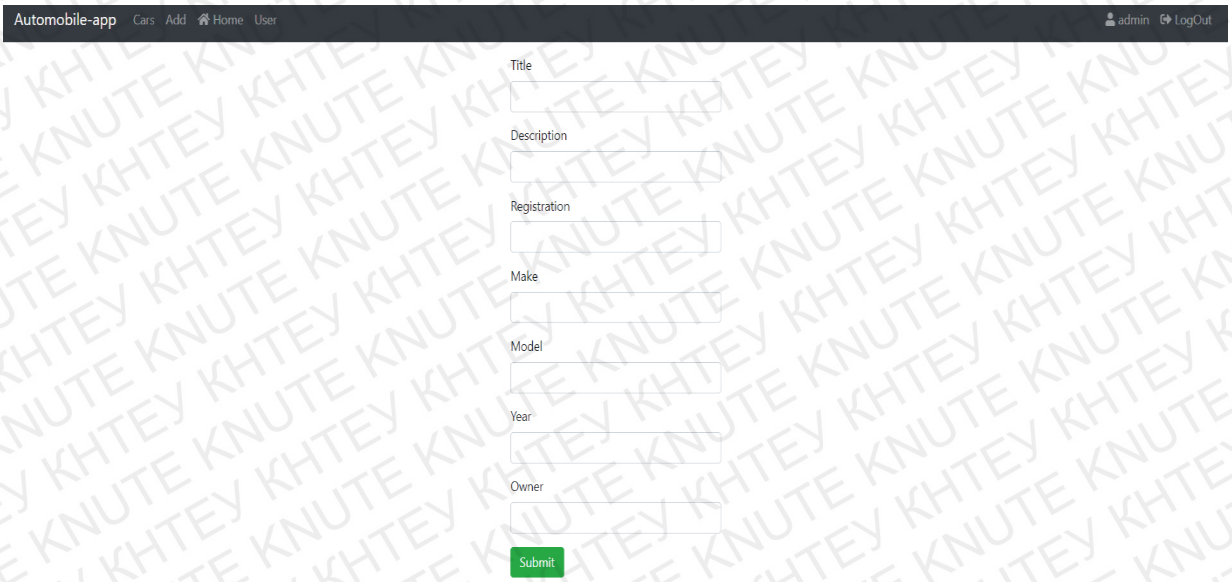


Рис. 3.8. Вигляд сторінки додавання інформації

- Поле title
- Полу Description
- Полу Registration
- Полу Make
- Полу Model
- Поле year
- Поле owner
- Кнопка ‘submit’ яка виконує підтвердження данних

Після заповнення елементів та натискання submit відбувається запис введених данних в базу данних та наступне графічне відображення цих данних на вкладці Cars.

Фрагмент реалізаціїї логіки (рис. 3.9):

```
<template>
  <div class="submit-form">
    <div v-if="!submitted">
      <div class="form-group">
        <label for="title">Title</label>
        <input
          type="text"
          class="form-control"
          id="title"
          required
          v-model="tutorial.title"
          name="title"
        />
      </div>

      <div class="form-group">
        <label for="description">Description
        <input
          class="form-control"
          id="description"
          required
          v-model="tutorial.description"
          name="description"
        />
      </div>
          <div class="form-group">
        <label for="registration">Registrati
        <input
          class="form-control"
          id="registration"
          required
          v-model="tutorial.registration"
          name="registration"
        />
      </div>
          <div class="form-group">
        <label for="make">Make</label>
```

```
const db = require("../models");
const Tutorial = db.tutorials;

// Create and Save a new Tutorial
exports.create = (req, res) => {
  // Validate request
  if (!req.body.title) {
    res.status(400).send({ message: "Content
    return;
  }

  // Create a Tutorial
  const tutorial = new Tutorial({
    title: req.body.title,
    description: req.body.description,
    registration: req.body.registration,
    model: req.body.model,
    make: req.body.make,
    year: req.body.year,
    owner: req.body.owner,
    published: req.body.published ? req.body
  });

  // Save Tutorial in the database
  tutorial
    .save(tutorial)
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
        message:
          err.message || "Some error occurre
      });
    });
};
```

Рис. 3.9. Фрагмент реалізації данних на вкладці Cars

3.6. Вигляд сторінки редагування запису

Після натискання кнопки «edit необхідного записку відобразяться такі елементи: (рис. 3.10.)

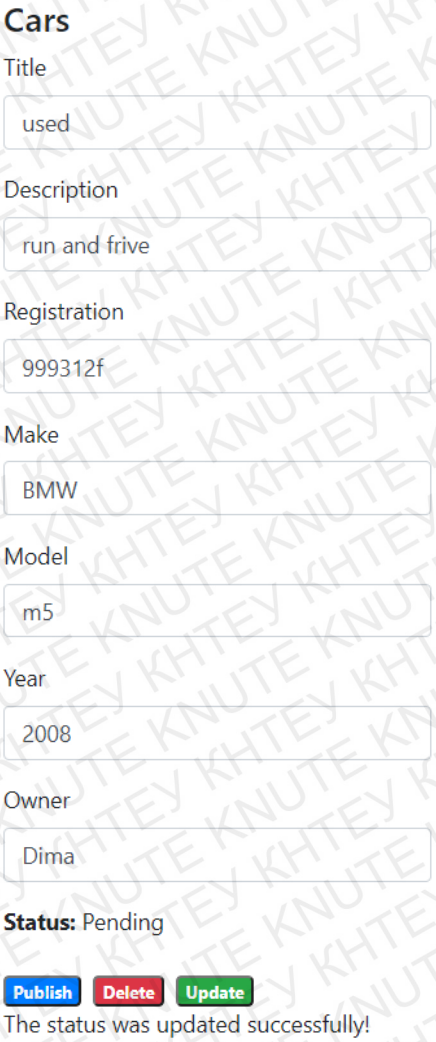


Рис. 3.10. Вигляд редагування запису

- Можливість ввести нову інформацію в кожне поле окрема
- Відображається актуальні дані про запис
- Можливість змінювати статус запусу Кількість елементів, до яких було застосована виконана команда
- Можливість видалленя
- Можливість оновлення
- Після натискання на потрібну фінкцію інформація про запис перезаписується і рнедериться на вкладці всіх записів

Фрагмент реалізація представлено на рис. 3. 11

```
updatePublished(status) {
  var data = {
    id: this.currentTutorial.id,
    title: this.currentTutorial.title,
    description: this.currentTutorial.description,
    registration: this.currentTutorial.registration,
    make: this.currentTutorial.make,
    model: this.currentTutorial.model,
    year: this.currentTutorial.year,
    owner: this.currentTutorial.owner,
    published: status
  };

  TutorialDataService.update(this.currentTutorial.id, data)
    .then(response => {
      console.log(response.data);
      this.currentTutorial.published = status;
      this.message = 'The status was updated successfully!';
    })
    .catch(e => {
      console.log(e);
    });
},

updateTutorial() {
  TutorialDataService.update(this.currentTutorial.id, this.current
    .then(response => {
      console.log(response.data);
      this.message = 'The car was updated successfully!';
    })
    .catch(e => {
      console.log(e);
```

Рис. 3.11. Фрагмент реалізації сторінки редагування запису

3.7. Вигляд колекції запису у форматі MongoDB

Після натискання кнопки створення запису в MongoDB необхідної itema вона має наступний вигляд: (рис. 3.12.)

[{"title":"1","description":"new
brand","registration":"123rq341","model":"q7","make":"Audi","year":"2012","owner":"alex","published":false,"createdAt":"2021-11-14T18:21:27.067Z","updatedAt":"2021-11-14T18:21:27.067Z","id":"619153a77982f34b68b559ef"},{"title":"used","description":"run and frive","registration":"999312f","model":"m5","make":"BMW","year":"2008","owner":"Dima","published":false,"createdAt":"2021-11-14T18:23:35.745Z","updatedAt":"2021-11-14T18:23:35.745Z","id":"619154277982f34b68b559f2"},{"title":"Elite","description":"Elite car","registration":"111109","model":"continental
gt","make":"Bentley","year":"1018","owner":"kirz","published":false,"createdAt":"2021-11-14T18:24:32.244Z","updatedAt":"2021-11-14T18:24:32.244Z","id":"619154607982f34b68b559f4"}]

Рис. 3.12. Вигляд створених записів MongoBB

MongoDB зберігає **дані** у вигляді рядка JSON незалежно від кількості атрибутів або імені атрибутів у певному стовпці.

3.8. Вигляд запису в графічному інтерфейсі

Після натискання на назву запису відобразяться такі елементи: (рис. 3.13.)

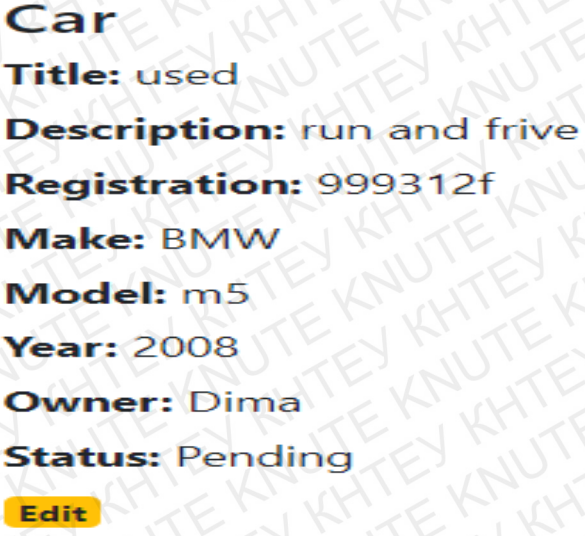


Рис. 3.13. Вигляд в графічному інтерфейсі

- Дані таблиці відповідно до колонок
- Можливість вибирати(чекбокси навпроти запису), редагувати або видалята записи
- Кнопки додавання та видалення запису.

3.9. Висновки до розділу 3

Отже, було розроблено та продемонстровано роботу засобу для адміністрування бази даних через веб-інтерфейс згідно з технічним завданням та на основі спроектованої архітектури програмного продукту було написано програмний код логіки серверної частини веб-додатку з використанням Node,js та Express js і MEVN платформи розробки.

За допомогою Vue.js да додаткових компонентів екосистеми та бібліотек, мови описання зовнішнього виду документа CSS була реалізована візуальна частина веб-додатку.

Поточний веб-додаток містить в собі всі необхідні компоненти, для того щоб виконувати різноманітні дії над mongoDB. Було проведено функціональне та інтеграційне тестування. Виходячи з результатів тестування можно зробити висновок, що веб-додаток відповідає ТЗ, виконуючи наведені функції.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Отже, було розглянуто характеристики веб-додатку та інтерфейсів. Був проведений аналіз предметної області,, відзначені недоліки та переваги мов програмування, в результаті чого були виділені основні вимоги до роброблюваного веб-додатку для адміністрування баз даних.

Для створення ПЗ було вибрано Mevn - для опису дій, які будуть виконуватися на клієнтській частині веб-додатку та серверної частини веб-додатку. Для зберігання даних програми NO-*SQL* використано *MongoDB;* для створення бекенда сайту за допомогою функцій і структур *NodeJS* використовано *ExpressJS*; для веб-розробки клієнтського інтерфейсу використано *VueJS*; для запуску системи застосовано *Node.js.* Додаток розміщено на серверній платформі *Node.js локально.*

Візуальний нарис веб-додатку розроблено за допомогою веб-ресурсу WireFrame.

Написання веб-додатку було здійснено за допомогою середовища розробки Vscode. Для реалізації поставленної мети була розгорнута та налаштована серверна платформа Node.js.

Виходячи з результатів програми методики випробувань, можна зробити висновок, що веб-додаток відповідає ТЗ, виконуючи наведені функції. Даний веб-додаток не є кінцевим продуктом. Використання повного функціоналу стане доступним у наступній версії веб-додатку.

					КНТЕУ 121 02м-15.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		*01.11.21*	*Проектування web-орієнтованої системи автомобільної компанії*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Рассамакін В.Я.		*01.11.21*		*ВП*	*47*	*49*
Гарант		Токар В.В.		*01.11.21*		*Факультет інформаційних технологій 2м курс, 2м група*		
Розробив		Новак В.В.		*01.11.21*	*Висновки та пропозиції*			

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Recruiting & Staffing Software and Tracking System. Recruiting & Staffing Software. [Електронний ресурс]. – Режим доступу: http://www.pcrecruiter.net
2. Балик Н.Б. Методичні рекомендації для проведення практики з Web-програмування. / Н.Б. Балик, В.П. Олексюк, В.І. Мандзюк – Тернопільський національний педагогічний університетімені В.Гнатюка, 2015. – 56 с.
3. Петренко О.Я. Використання комунікаційних мобільних пристроїв: Навчальний посібник. / О.Я. Петренко – К. ІПДО, 2018. – 42с.
4. Дегтярьова Л.М - Навчальний посібник з дисципліни «Технології розробки програмного забезпечення» для студентів спеціальності 123 «Комп'ютерна інженерія». – Полтава: ПолтНТУ, 2017 – 70-72.
5. Simpson K. You Don't Know JS: Async & Performance / Kyle Simpson. – O'Reilly Media, Inc, 2015 – 247 p.
6. Cross-Platform Desktop Applications: Using Node, Electron, and NW.js / Paul Jensen. – Paul B. Jensen. – Manning Publications, 2017 – 312 p.
7. Е. Фріман, Head First HTML with CSS&XHTML, 2010 – 656с.
8. Е. Тіттел, HTML, XHTML & CSS For Dummies,7th Edition, 2011. – 400 с.
9. П. Лаббертс, Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development, 2011. – 272 с.
10. Д. Роббінс – HTML5, CSS3 і JavaScript, 2014. – 528 с.

Електронні ресурси

11. Документація Vue.js[Електронний ресурс]. - Режим доступу: https://ru.vuejs.org/

					КНТЕУ 121 02м-15.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		*27.02.21*	*Проектування web-орієнтованої системи автомобільної компанії*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Рассамакін В.Я.		*27.02.21*		*СВД*	*48*	*49*
Гарант		Токар В.В.		*27.02.21*		*Факультет інформаційних технологій 2м курс, 2м група*		
Розробив		Новак В.В.		*27.02.21*	*Список використаних джерел*			

12. Документація Vue.cli[Електронний ресурс]. - Режим доступу: https://cli.vuejs.org/
13. Документація Vuex[Електронний ресурс]. - Режим доступу: https://vuex.vuejs.org/
14. Документація Router [Електронний ресурс]. - Режим доступу: https://router.vuejs.org/
15. Документація expressjs[Електронний ресурс]. - Режим доступу: https://expressjs.com/
16. Документація mongodb[Електронний ресурс]. - Режим доступу: https://docs.mongodb.com/
17. Документація Node,js [Електронний ресурс]. - Режим доступу: https://nodejs.org/
18. Веб-інтерфейс [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/Вебінтерфейс
19. Документація vue vrud [Електронний ресурс]. - Режим доступу:https://vue-crud.github.io

ДОДАТКИ

Додаток А

auth.controller.js

```
const config = require("../config/auth.config");
const db = require("../models");
const User = db.user;
const Role = db.role;

var jwt = require("jsonwebtoken");
var bcrypt = require("bcryptjs");

exports.signup = (req, res) => {
 const user = new User({
  username: req.body.username,
  email: req.body.email,
  password: bcrypt.hashSync(req.body.password, 8)
 });

 user.save((err, user) => {
  if (err) {
   res.status(500).send({ message: err });
   return;
  }

  if (req.body.roles) {
   Role.find(
    {
     name: { $in: req.body.roles }
    },
    (err, roles) => {
     if (err) {
      res.status(500).send({ message: err });
      return;
     }

     user.roles = roles.map(role => role._id);
     user.save(err => {
      if (err) {
       res.status(500).send({ message: err });
       return;
      }
```

```
            res.send({ message: "User was registered successfully!" });
          });
        }
      );
    } else {
      Role.findOne({ name: "user" }, (err, role) => {
        if (err) {
          res.status(500).send({ message: err });
          return;
        }

        user.roles = [role._id];
        user.save(err => {
          if (err) {
            res.status(500).send({ message: err });
            return;
          }

          res.send({ message: "User was registered successfully!" });
        });
      });
    }
  });
};

exports.signin = (req, res) => {
  User.findOne({
    username: req.body.username
  })
    .populate("roles", "-__v")
    .exec((err, user) => {
      if (err) {
        res.status(500).send({ message: err });
        return;
      }

      if (!user) {
        return res.status(404).send({ message: "User Not found." });
      }

      var passwordIsValid = bcrypt.compareSync(
        req.body.password,
        user.password
      );
```

```
        if (!passwordIsValid) {
          return res.status(401).send({
            accessToken: null,
            message: "Invalid Password!"
          });
        }

        var token = jwt.sign({ id: user.id }, config.secret, {
          expiresIn: 86400 // 24 hours
        });

        var authorities = [];

        for (let i = 0; i < user.roles.length; i++) {
          authorities.push("ROLE_" + user.roles[i].name.toUpperCase());
        }
        res.status(200).send({
          id: user._id,
          username: user.username,
          email: user.email,
          roles: authorities,
          accessToken: token
        });
      });
  };
```

db.config.js

```
module.exports = {
  url: "mongodb://localhost:27017/auto_db",
  HOST: "localhost",
  PORT: 27017,
  DB: "auto_db"
};
```

tuturial.controller.js

```
const db = require("../models");
const Tutorial = db.tutorials;

// Create and Save a new Tutorial
```

```
exports.create = (req, res) => {
  // Validate request
  if (!req.body.title) {
    res.status(400).send({ message: "Content can not be empty!" });
    return;
  }

  // Create a Tutorial
  const tutorial = new Tutorial({
    title: req.body.title,
    description: req.body.description,
    registration: req.body.registration,
    model: req.body.model,
    make: req.body.make,
    year: req.body.year,
    owner: req.body.owner,
    published: req.body.published ? req.body.published : false
  });

  // Save Tutorial in the database
  tutorial
    .save(tutorial)
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
        message:
          err.message || "Some error occurred while creating the Cars."
      });
    });
};

// Retrieve all Tutorials from the database.
exports.findAll = (req, res) => {
  const title = req.query.title;
  var condition = title ? { title: { $regex: new RegExp(title), $options: "i" } } : {};

  Tutorial.find(condition)
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
```

```
        message:
          err.message || "Some error occurred while retrieving cars."
        });
      });
};

// Find a single Cars with an id
exports.findOne = (req, res) => {
  const id = req.params.id;

  Tutorial.findById(id)
    .then(data => {
      if (!data)
        res.status(404).send({ message: "Not found Cars with id " + id });
      else res.send(data);
    })
    .catch(err => {
      res
        .status(500)
        .send({ message: "Error retrieving Cars with id=" + id });
    });
};

// Update a Cars by the id in the request
exports.update = (req, res) => {
  if (!req.body) {
    return res.status(400).send({
      message: "Data to update can not be empty!"
    });
  }

  const id = req.params.id;

  Tutorial.findByIdAndUpdate(id, req.body, { useFindAndModify: false })
    .then(data => {
      if (!data) {
        res.status(404).send({
          message: `Cannot update Tutorial with id=${id}. Maybe Tutorial was not found!`
        });
      } else res.send({ message: "Cars was updated successfully." });
    })
    .catch(err => {
      res.status(500).send({
        message: "Error updating Cars with id=" + id
```

```
      });
    });
};

// Delete a Cars with the specified id in the request
exports.delete = (req, res) => {
  const id = req.params.id;

  Tutorial.findByIdAndRemove(id, { useFindAndModify: false })
    .then(data => {
      if (!data) {
        res.status(404).send({
          message: `Cannot delete Cars with id=${id}. Maybe Cars was not found!`
        });
      } else {
        res.send({
          message: "Cars was deleted successfully!"
        });
      }
    })
    .catch(err => {
      res.status(500).send({
        message: "Could not delete Cars with id=" + id
      });
    });
};

// Delete all Cars from the database.
exports.deleteAll = (req, res) => {
  Tutorial.deleteMany({})
    .then(data => {
      res.send({
        message: `${data.deletedCount} Cars were deleted successfully!`
      });
    })
    .catch(err => {
      res.status(500).send({
        message:
          err.message || "Some error occurred while removing all Cars."
      });
    });
};

// Find all published Cars
```

```
exports.findAllPublished = (req, res) => {
  Tutorial.find({ published: true })
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
        message:
          err.message || "Some error occurred while retrieving Cars."
      });
    });
};
```

user.controller.js

```
exports.allAccess = (req, res) => {
  res.status(200).send("Public Content.");
};

exports.userBoard = (req, res) => {
  res.status(200).send("User Content.");
};

exports.adminBoard = (req, res) => {
  res.status(200).send("Admin Content.");
};

exports.moderatorBoard = (req, res) => {
  res.status(200).send("Moderator Content.");
};
```

authJwt.js

```
const jwt = require("jsonwebtoken");
const config = require("../config/auth.config.js");
const db = require("../models");
const User = db.user;
const Role = db.role;

verifyToken = (req, res, next) => {
```

```
  let token = req.headers["x-access-token"];

  if (!token) {
    return res.status(403).send({ message: "No token provided!" });
  }

  jwt.verify(token, config.secret, (err, decoded) => {
    if (err) {
      return res.status(401).send({ message: "Unauthorized!" });
    }
    req.userId = decoded.id;
    next();
  });
};

isAdmin = (req, res, next) => {
  User.findById(req.userId).exec((err, user) => {
    if (err) {
      res.status(500).send({ message: err });
      return;
    }

    Role.find(
      {
        _id: { $in: user.roles }
      },
      (err, roles) => {
        if (err) {
          res.status(500).send({ message: err });
          return;
        }

        for (let i = 0; i < roles.length; i++) {
          if (roles[i].name === "admin") {
            next();
            return;
          }
        }

        res.status(403).send({ message: "Require Admin Role!" });
        return;
      }
    );
  });
```

```
};

isModerator = (req, res, next) => {
  User.findById(req.userId).exec((err, user) => {
    if (err) {
      res.status(500).send({ message: err });
      return;
    }

    Role.find(
      {
        _id: { $in: user.roles }
      },
      (err, roles) => {
        if (err) {
          res.status(500).send({ message: err });
          return;
        }

        for (let i = 0; i < roles.length; i++) {
          if (roles[i].name === "moderator") {
            next();
            return;
          }
        }

        res.status(403).send({ message: "Require Moderator Role!" });
        return;
      }
    );
  });
};

const authJwt = {
  verifyToken,
  isAdmin,
  isModerator
};
module.exports = authJwt;
```

index.js

```
const authJwt = require("./authJwt");
const verifySignUp = require("./verifySignUp");

module.exports = {
  authJwt,
  verifySignUp
};
```

verifySignUp.js

```
const db = require("../models");
const ROLES = db.ROLES;
const User = db.user;

checkDuplicateUsernameOrEmail = (req, res, next) => {
  // Username
  User.findOne({
    username: req.body.username
  }).exec((err, user) => {
    if (err) {
      res.status(500).send({ message: err });
      return;
    }

    if (user) {
      res.status(400).send({ message: "Failed! Username is already in use!" });
      return;
    }

    // Email
    User.findOne({
      email: req.body.email
    }).exec((err, user) => {
      if (err) {
        res.status(500).send({ message: err });
        return;
      }

      if (user) {
```

```
        res.status(400).send({ message: "Failed! Email is already in use!" });
        return;
      }

      next();
    });
  });
};

checkRolesExisted = (req, res, next) => {
  if (req.body.roles) {
    for (let i = 0; i < req.body.roles.length; i++) {
      if (!ROLES.includes(req.body.roles[i])) {
        res.status(400).send({
          message: `Failed! Role ${req.body.roles[i]} does not exist!`
        });
        return;
      }
    }
  }

  next();
};

const verifySignUp = {
  checkDuplicateUsernameOrEmail,
  checkRolesExisted
};

module.exports = verifySignUp;
```

index.js

```
const dbConfig = require("../config/db.config.js");

const mongoose = require("mongoose");
mongoose.Promise = global.Promise;

const db = {};
db.mongoose = mongoose;
db.user = require("./user.model");
db.role = require("./role.model");
```

```
db.ROLES = ["user", "admin", "moderator"];
db.url = dbConfig.url;
db.tutorials = require("./tutorial.model.js")(mongoose);

module.exports = db
```

role.model.js

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

const Role = mongoose.model(
  "Role",
  new mongoose.Schema({
    name: String
  })
);

module.exports = Role;
```

tutorial.model.js

```
module.exports = mongoose => {
  var schema = mongoose.Schema(
    {
      title: String,
      description: String,
      registration: String,
      model: String,
      make: String,
      year: String,
      owner: String,
      published: Boolean
    },
    { timestamps: true }
  );

  schema.method("toJSON", function() {
    const { __v, _id, ...object } = this.toObject();
    object.id = _id;
```

```
  return object;
 });

 const Tutorial = mongoose.model("tutorial", schema);
 return Tutorial;
};
```

user.model.js

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

const User = mongoose.model(
 "User",
 new mongoose.Schema({
  username: String,
  email: String,
  password: String,
  roles: [
   {
    type: mongoose.Schema.Types.ObjectId,
    ref: "Role"
   }
  ]
 })
);

module.exports = User;
```

auth.routes.js

```
const { verifySignUp } = require("../middlewares");
const controller = require("../controllers/auth.controller");

module.exports = function(app) {
 app.use(function(req, res, next) {
  res.header(
   "Access-Control-Allow-Headers",
   "x-access-token, Origin, Content-Type, Accept"
  );
  next();
 });
```

```
  app.post(
    "/api/auth/signup",
    [
      verifySignUp.checkDuplicateUsernameOrEmail,
      verifySignUp.checkRolesExisted
    ],
    controller.signup
  );

  app.post("/api/auth/signin", controller.signin);
};
```

turorial.routes.js

```
module.exports = app => {
  const tutorials = require("../controllers/tutorial.controller.js");

  var router = require("express").Router();

  // Create a new Car
  router.post("/", tutorials.create);

  // Retrieve all Cars
  router.get("/", tutorials.findAll);

  // Retrieve all published Cars
  router.get("/published", tutorials.findAllPublished);

  // Retrieve a single Car with id
  router.get("/:id", tutorials.findOne);

  // Update a Car with id
  router.put("/:id", tutorials.update);

  // Delete a Car with id
  router.delete("/:id", tutorials.delete);

  // Create a new Car
  router.delete("/", tutorials.deleteAll);

  app.use("/api/tutorials", router);
};
```

user.routes.js

```
const { authJwt } = require("../middlewares");
const controller = require("../controllers/user.controller");

module.exports = function(app) {
  app.use(function(req, res, next) {
    res.header(
      "Access-Control-Allow-Headers",
      "x-access-token, Origin, Content-Type, Accept"
    );
    next();
  });

  app.get("/api/test/all", controller.allAccess);

  app.get("/api/test/user", [authJwt.verifyToken], controller.userBoard);

  app.get(
    "/api/test/mod",
    [authJwt.verifyToken, authJwt.isModerator],
    controller.moderatorBoard
  );

  app.get(
    "/api/test/admin",
    [authJwt.verifyToken, authJwt.isAdmin],
    controller.adminBoard
  );
};
```

server.js

```
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");
const dbConfig = require("./app/config/db.config");

const app = express();
```

```
var corsOptions = {
  origin: "http://localhost:8081"
};

app.use(cors(corsOptions));
// parse requests of content-type - application/json
app.use(bodyParser.json());

// parse requests of content-type - application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: true }));

// parse requests of content-type - application/json
app.use(express.json());  /* bodyParser.json() is deprecated */

// parse requests of content-type - application/x-www-form-urlencoded
app.use(express.urlencoded({ extended: true }));   /* bodyParser.urlencoded() is
deprecated */

const db = require("./app/models");
const Role = db.role;

db.mongoose
  .connect(`mongodb://${dbConfig.HOST}:${dbConfig.PORT}/${dbConfig.DB}`, {
    useNewUrlParser: true,
    useUnifiedTopology: true
  })
db.mongoose
  .connect(db.url, {
    useNewUrlParser: true,
    useUnifiedTopology: true
  })
  .then(() => {
    console.log("Connected to the database!");
  })
  .catch(err => {
    console.log("Cannot connect to the database!", err);
    process.exit();
  });

// simple route
app.get("/", (req, res) => {
  res.json({ message: "Welcome to bezkoder application." });
});
// routes
```

```
require("./app/routes/auth.routes")(app);
require("./app/routes/user.routes")(app);

require("./app/routes/turorial.routes")(app);

// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
function initial() {
  Role.estimatedDocumentCount((err, count) => {
    if (!err && count === 0) {
      new Role({
        name: "user"
      }).save(err => {
        if (err) {
          console.log("error", err);
        }

        console.log("added 'user' to roles collection");
      });

      new Role({
        name: "moderator"
      }).save(err => {
        if (err) {
          console.log("error", err);
        }

        console.log("added 'moderator' to roles collection");
      });

      new Role({
        name: "admin"
      }).save(err => {
        if (err) {
          console.log("error", err);
        }

        console.log("added 'admin' to roles collection");
      });
    }
  });
}
```

index.html

```
<!DOCTYPE html>
<html lang="">
 <head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width,initial-scale=1.0">
  <link rel="icon" href="<%= BASE_URL %>favicon.ico">
  <title><%= htmlWebpackPlugin.options.title %></title>
 </head>
 <body>
  <noscript>
   <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work
properly without JavaScript enabled. Please enable it to continue.</strong>
  </noscript>
  <div id="app"></div>
  <!-- built files will be auto injected -->
 </body>
</html>
```

AddTutorial.vue

```
<template>
 <div class="submit-form">
  <div v-if="!submitted">
   <div class="form-group">
    <label for="title">Title</label>
    <input
     type="text"
     class="form-control"
     id="title"
     required
     v-model="tutorial.title"
     name="title"
    />
   </div>

   <div class="form-group">
    <label for="description">Description</label>
```

```
  <input
   class="form-control"
   id="description"
   required
   v-model="tutorial.description"
   name="description"
  />
 </div>
    <div class="form-group">
  <label for="registration">Registration</label>
  <input
   class="form-control"
   id="registration"
   required
   v-model="tutorial.registration"
   name="registration"
  />
 </div>
    <div class="form-group">
  <label for="make">Make</label>
  <input
   class="form-control"
   id="make"
   required
   v-model="tutorial.make"
   name="make"
  />
 </div>
    <div class="form-group">
  <label for="model">Model</label>
  <input
   class="form-control"
   id="model"
   required
   v-model="tutorial.model"
   name="model"
  />
 </div>
    <div class="form-group">
  <label for="year">Year</label>
  <input
   class="form-control"
   id="year"
   required
```

```
        v-model="tutorial.year"
        name="year"
       />
      </div>
          <div class="form-group">
       <label for="owner">Owner</label>
       <input
        class="form-control"
        id="owner"
        required
        v-model="tutorial.owner"
        name="owner"
       />
      </div>

     <button @click="saveTutorial" class="btn btn-success">Submit</button>
    </div>

    <div v-else>
     <h4>You submitted successfully!</h4>
     <button class="btn btn-success" @click="newTutorial">Add</button>
    </div>
   </div>
</template>

<script>
import TutorialDataService from "../services/TutorialDataService";

export default {
  name: "add-tutorial",
  data() {
    return {
      tutorial: {
        id: null,
        title: "",
        description: "",
        registration: "",
        make: "",
        model: "",
        year: "",
        owner: "",
        published: false
      },
      submitted: false
```

```
    };
  },
  methods: {
   saveTutorial() {
     var data = {
       title: this.tutorial.title,
       description: this.tutorial.description,
       make: this.tutorial.make,
       registration: this.tutorial.registration,
       model: this.tutorial.model,
       year: this.tutorial.year,
       owner: this.tutorial.owner
     };

     TutorialDataService.create(data)
       .then(response => {
         this.tutorial.id = response.data.id;
         console.log(response.data);
         this.submitted = true;
       })
       .catch(e => {
         console.log(e);
       });
   },

   newTutorial() {
    this.submitted = false;
    this.tutorial = {};
   }
  }
};
</script>

<style>
.submit-form {
  max-width: 300px;
  margin: auto;
}
</style>
```

BoardAdmin.vue

```
<template>
  <div class="container">
    <header class="jumbotron">
      <h3>{{ content }}</h3>
    </header>
  </div>
</template>

<script>
import UserService from "../services/user.service";

export default {
  name: "Admin",
  data() {
    return {
      content: "",
    };
  },
  mounted() {
    UserService.getAdminBoard().then(
      (response) => {
        this.content = response.data;
      },
      (error) => {
        this.content =
          (error.response &&
            error.response.data &&
            error.response.data.message) ||
          error.message ||
          error.toString();
      }
    );
  },
};
</script>
```

BoardModerator.vue

```
<template>
  <div class="container">
```

```
    <header class="jumbotron">
      <h3>{{ content }}</h3>
    </header>
  </div>
</template>

<script>
import UserService from "../services/user.service";

export default {
  name: "Moderator",
  data() {
    return {
      content: "",
    };
  },
  mounted() {
    UserService.getModeratorBoard().then(
      (response) => {
        this.content = response.data;
      },
      (error) => {
        this.content =
          (error.response &&
            error.response.data &&
            error.response.data.message) ||
          error.message ||
          error.toString();
      }
    );
  },
};
</script>
```

BoardUser.vue

```
<template>
  <div class="container">
    <header class="jumbotron">
      <h3>{{ content }}</h3>
    </header>
  </div>
```

```
</template>

<script>
import UserService from "../services/user.service";

export default {
  name: "User",
  data() {
    return {
      content: "",
    };
  },
  mounted() {
    UserService.getUserBoard().then(
      (response) => {
        this.content = response.data;
      },
      (error) => {
        this.content =
          (error.response &&
            error.response.data &&
            error.response.data.message) ||
          error.message ||
          error.toString();
      }
    );
  },
};
</script>
```

Home.vue

```
<template>
  <div class="container">
    <header class="jumbotron">
      <h3>{{ content }}</h3>
    </header>
  </div>
</template>

<script>
import UserService from "../services/user.service";
```

```
export default {
  name: "Home",
  data() {
    return {
      content: "",
    };
  },
  mounted() {
    UserService.getPublicContent().then(
      (response) => {
        this.content = response.data;
      },
      (error) => {
        this.content =
          (error.response &&
            error.response.data &&
            error.response.data.message) ||
          error.message ||
          error.toString();
      }
    );
  },
};
</script>
```

Login.vue

```
<template>
  <div class="col-md-12">
    <div class="card card-container">
      <img
        id="profile-img"
        src="//ssl.gstatic.com/accounts/ui/avatar_2x.png"
        class="profile-img-card"
      />
      <Form @submit="handleLogin" :validation-schema="schema">
        <div class="form-group">
          <label for="username">Username</label>
          <Field name="username" type="text" class="form-control" />
          <ErrorMessage name="username" class="error-feedback" />
        </div>
```

```
      <div class="form-group">
       <label for="password">Password</label>
       <Field name="password" type="password" class="form-control" />
       <ErrorMessage name="password" class="error-feedback" />
      </div>

      <div class="form-group">
       <button class="btn btn-primary btn-block" :disabled="loading">
        <span
         v-show="loading"
         class="spinner-border spinner-border-sm"
        ></span>
        <span>Login</span>
       </button>
      </div>

      <div class="form-group">
       <div v-if="message" class="alert alert-danger" role="alert">
        {{ message }}
       </div>
      </div>
     </Form>
    </div>
   </div>
  </template>

  <script>
  import { Form, Field, ErrorMessage } from "vee-validate";
  import * as yup from "yup";

  export default {
   name: "Login",
   components: {
    Form,
    Field,
    ErrorMessage,
   },
   data() {
    const schema = yup.object().shape({
     username: yup.string().required("Username is required!"),
     password: yup.string().required("Password is required!"),
    });

    return {
```

```
      loading: false,
      message: "",
      schema,
    };
  },
  computed: {
    loggedIn() {
      return this.$store.state.auth.status.loggedIn;
    },
  },
  created() {
    if (this.loggedIn) {
      this.$router.push("/profile");
    }
  },
  methods: {
    handleLogin(user) {
      this.loading = true;

      this.$store.dispatch("auth/login", user).then(
        () => {
          this.$router.push("/profile");
        },
        (error) => {
          this.loading = false;
          this.message =
            (error.response &&
              error.response.data &&
              error.response.data.message) ||
            error.message ||
            error.toString();
        }
      );
    },
  },
};
</script>

<style scoped>
label {
  display: block;
  margin-top: 10px;
}
```

```
.card-container.card {
  max-width: 350px !important;
  padding: 40px 40px;
}

.card {
  background-color: #f7f7f7;
  padding: 20px 25px 30px;
  margin: 0 auto 25px;
  margin-top: 50px;
  -moz-border-radius: 2px;
  -webkit-border-radius: 2px;
  border-radius: 2px;
  -moz-box-shadow: 0px 2px 2px rgba(0, 0, 0, 0.3);
  -webkit-box-shadow: 0px 2px 2px rgba(0, 0, 0, 0.3);
  box-shadow: 0px 2px 2px rgba(0, 0, 0, 0.3);
}

.profile-img-card {
  width: 96px;
  height: 96px;
  margin: 0 auto 10px;
  display: block;
  -moz-border-radius: 50%;
  -webkit-border-radius: 50%;
  border-radius: 50%;
}

.error-feedback {
  color: red;
}
</style>
```

Profile.vue

```
<template>
  <div class="container">
    <header class="jumbotron">
      <h3>
        <strong>{{currentUser.username}}</strong> Profile
      </h3>
    </header>
```

```
    <p>
      <strong>Token:</strong>
      {{currentUser.accessToken.substring(0, 20)}} ...
{{currentUser.accessToken.substr(currentUser.accessToken.length - 20)}}
    </p>
    <p>
      <strong>Id:</strong>
      {{currentUser.id}}
    </p>
    <p>
      <strong>Email:</strong>
      {{currentUser.email}}
    </p>
    <strong>Authorities:</strong>
    <ul>
      <li v-for="role in currentUser.roles" :key="role">{{role}}</li>
    </ul>
  </div>
</template>

<script>
export default {
  name: 'Profile',
  computed: {
    currentUser() {
      return this.$store.state.auth.user;
    }
  },
  mounted() {
    if (!this.currentUser) {
      this.$router.push('/login');
    }
  }
};
</script>
```

Register.vue

```
<template>
  <div class="col-md-12">
    <div class="card card-container">
      <img
```

```
      id="profile-img"
      src="//ssl.gstatic.com/accounts/ui/avatar_2x.png"
      class="profile-img-card"
     />
     <Form @submit="handleRegister" :validation-schema="schema">
      <div v-if="!successful">
       <div class="form-group">
        <label for="username">Username</label>
        <Field name="username" type="text" class="form-control" />
        <ErrorMessage name="username" class="error-feedback" />
       </div>
       <div class="form-group">
        <label for="email">Email</label>
        <Field name="email" type="email" class="form-control" />
        <ErrorMessage name="email" class="error-feedback" />
       </div>
       <div class="form-group">
        <label for="password">Password</label>
        <Field name="password" type="password" class="form-control" />
        <ErrorMessage name="password" class="error-feedback" />
       </div>

       <div class="form-group">
        <button class="btn btn-primary btn-block" :disabled="loading">
         <span
          v-show="loading"
          class="spinner-border spinner-border-sm"
         ></span>
         Sign Up
        </button>
       </div>
      </div>
     </Form>

     <div
      v-if="message"
      class="alert"
      :class="successful ? 'alert-success' : 'alert-danger'"
     >
      {{ message }}
     </div>
    </div>
   </div>
  </template>
```

```
<script>
import { Form, Field, ErrorMessage } from "vee-validate";
import * as yup from "yup";

export default {
  name: "Register",
  components: {
    Form,
    Field,
    ErrorMessage,
  },
  data() {
    const schema = yup.object().shape({
      username: yup
        .string()
        .required("Username is required!")
        .min(3, "Must be at least 3 characters!")
        .max(20, "Must be maximum 20 characters!"),
      email: yup
        .string()
        .required("Email is required!")
        .email("Email is invalid!")
        .max(50, "Must be maximum 50 characters!"),
      password: yup
        .string()
        .required("Password is required!")
        .min(6, "Must be at least 6 characters!")
        .max(40, "Must be maximum 40 characters!"),
    });

    return {
      successful: false,
      loading: false,
      message: "",
      schema,
    };
  },
  computed: {
    loggedIn() {
      return this.$store.state.auth.status.loggedIn;
    },
  },
  mounted() {
```

```
    if (this.loggedIn) {
      this.$router.push("/profile");
    }
  },
  methods: {
    handleRegister(user) {
      this.message = "";
      this.successful = false;
      this.loading = true;

      this.$store.dispatch("auth/register", user).then(
        (data) => {
          this.message = data.message;
          this.successful = true;
          this.loading = false;
        },
        (error) => {
          this.message =
            (error.response &&
              error.response.data &&
              error.response.data.message) ||
            error.message ||
            error.toString();
          this.successful = false;
          this.loading = false;
        }
      );
    },
  },
};
</script>

<style scoped>
label {
  display: block;
  margin-top: 10px;
}

.card-container.card {
  max-width: 350px !important;
  padding: 40px 40px;
}

.card {
```

```
  background-color: #f7f7f7;
  padding: 20px 25px 30px;
  margin: 0 auto 25px;
  margin-top: 50px;
  -moz-border-radius: 2px;
  -webkit-border-radius: 2px;
  border-radius: 2px;
  -moz-box-shadow: 0px 2px 2px rgba(0, 0, 0, 0.3);
  -webkit-box-shadow: 0px 2px 2px rgba(0, 0, 0, 0.3);
  box-shadow: 0px 2px 2px rgba(0, 0, 0, 0.3);
}

.profile-img-card {
  width: 96px;
  height: 96px;
  margin: 0 auto 10px;
  display: block;
  -moz-border-radius: 50%;
  -webkit-border-radius: 50%;
  border-radius: 50%;
}

.error-feedback {
  color: red;
}
</style>
```

Tutorial.vue

```
<template>
  <div v-if="currentTutorial" class="edit-form">
    <h4>Cars</h4>
    <form>
      <div class="form-group">
        <label for="title">Title</label>
        <input type="text" class="form-control" id="title"
          v-model="currentTutorial.title"
        />
      </div>
      <div class="form-group">
        <label for="description">Description</label>
        <input type="text" class="form-control" id="description"
          v-model="currentTutorial.description"
```

```
      />
    </div>
        <div class="form-group">
      <label for="registration">Registration</label>
      <input type="text" class="form-control" id="registration"
       v-model="currentTutorial.registration"
      />
    </div>
        <div class="form-group">
      <label for="make">Make</label>
      <input type="text" class="form-control" id="make"
       v-model="currentTutorial.make"
      />
    </div>
        <div class="form-group">
      <label for="model">Model</label>
      <input type="text" class="form-control" id="model"
       v-model="currentTutorial.model"
      />
    </div>
        <div class="form-group">
      <label for="year">Year</label>
      <input type="text" class="form-control" id="year"
       v-model="currentTutorial.year"
      />
    </div>
        <div class="form-group">
      <label for="owner">Owner</label>
      <input type="text" class="form-control" id="owner"
       v-model="currentTutorial.owner"
      />
    </div>

    <div class="form-group">
      <label><strong>Status:</strong></label>
      {{ currentTutorial.published ? "Published" : "Pending" }}
    </div>
  </form>

  <button class="badge badge-primary mr-2"
    v-if="currentTutorial.published"
    @click="updatePublished(false)"
  >
    UnPublish
```

```
    </button>
    <button v-else class="badge badge-primary mr-2"
      @click="updatePublished(true)"
    >
      Publish
    </button>

    <button class="badge badge-danger mr-2"
      @click="deleteTutorial"
    >
      Delete
    </button>

    <button type="submit" class="badge badge-success"
      @click="updateTutorial"
    >
      Update
    </button>
    <p>{{ message }}</p>
  </div>

  <div v-else>
    <br />
    <p>Please click on a Tutorial...</p>
  </div>
</template>

<script>
import TutorialDataService from "../services/TutorialDataService";

export default {
  name: "tutorial",
  data() {
    return {
      currentTutorial: null,
      message: ''
    };
  },
  methods: {
    getTutorial(id) {
      TutorialDataService.get(id)
        .then(response => {
          this.currentTutorial = response.data;
          console.log(response.data);
```

```
    })
    .catch(e => {
      console.log(e);
    });
},

updatePublished(status) {
  var data = {
    id: this.currentTutorial.id,
    title: this.currentTutorial.title,
    description: this.currentTutorial.description,
    registration: this.currentTutorial.registration,
    make: this.currentTutorial.make,
    model: this.currentTutorial.model,
    year: this.currentTutorial.year,
    owner: this.currentTutorial.owner,
    published: status
  };

  TutorialDataService.update(this.currentTutorial.id, data)
    .then(response => {
      console.log(response.data);
      this.currentTutorial.published = status;
      this.message = 'The status was updated successfully!';
    })
    .catch(e => {
      console.log(e);
    });
},

updateTutorial() {
  TutorialDataService.update(this.currentTutorial.id, this.currentTutorial)
    .then(response => {
      console.log(response.data);
      this.message = 'The car was updated successfully!';
    })
    .catch(e => {
      console.log(e);
    });
},

deleteTutorial() {
  TutorialDataService.delete(this.currentTutorial.id)
    .then(response => {
```

```
        console.log(response.data);
        this.$router.push({ name: "tutorials" });
       })
       .catch(e => {
        console.log(e);
       });
    }
  },
  mounted() {
   this.message = '';
   this.getTutorial(this.$route.params.id);
  }
};
</script>

<style>
.edit-form {
  max-width: 300px;
  margin: auto;
}
</style>
```

TutorialList.vue

```
<template>
  <div class="list row">
    <div class="col-md-8">
      <div class="input-group mb-3">
        <input type="text" class="form-control" placeholder="Search by title"
        v-model="title"/>
        <div class="input-group-append">
         <button class="btn btn-outline-secondary" type="button"
          @click="searchTitle"
         >
          Search
         </button>
        </div>
      </div>
    </div>
    <div class="col-md-6">
      <h4>Cars List</h4>
      <ul class="list-group">
```

```
      <li class="list-group-item"
       :class="{ active: index == currentIndex }"
       v-for="(tutorial, index) in tutorials"
       :key="index"
       @click="setActiveTutorial(tutorial, index)"
      >
       {{ tutorial.title }}
      </li>
     </ul>

     <button class="m-3 btn btn-sm btn-danger" @click="removeAllTutorials">
      Remove All
     </button>
    </div>
    <div class="col-md-6">
     <div v-if="currentTutorial">
      <h4>Car</h4>
      <div>
       <label><strong>Title:</strong></label> {{ currentTutorial.title }}
      </div>
      <div>
       <label><strong>Description:</strong></label> {{ currentTutorial.description }}
      </div>
          <div>
      <label><strong>Registration:</strong></label> {{ currentTutorial.registration }}
      </div>
          <div>
      <label><strong>Make:</strong></label> {{ currentTutorial.make }}
      </div>
          <div>
      <label><strong>Model:</strong></label> {{ currentTutorial.model }}
      </div>
          <div>
      <label><strong>Year:</strong></label> {{ currentTutorial.year }}
      </div>
          <div>
      <label><strong>Owner:</strong></label> {{ currentTutorial.owner }}
      </div>
      <div>
       <label><strong>Status:</strong></label> {{ currentTutorial.published ?
"Published" : "Pending" }}
      </div>
```

```
      <router-link :to="'/tutorials/' + currentTutorial.id" class="badge badge-
warning">Edit</router-link>
    </div>
    <div v-else>
     <br />
     <p>Please click on a Car...</p>
    </div>
   </div>
  </div>
</template>

<script>
import TutorialDataService from "../services/TutorialDataService";

export default {
  name: "tutorials-list",
  data() {
   return {
    tutorials: [],
    currentTutorial: null,
    currentIndex: -1,
    title: ""
   };
  },
  methods: {
   retrieveTutorials() {
    TutorialDataService.getAll()
     .then(response => {
      this.tutorials = response.data;
      console.log(response.data);
     })
     .catch(e => {
      console.log(e);
     });
   },

   refreshList() {
    this.retrieveTutorials();
    this.currentTutorial = null;
    this.currentIndex = -1;
   },

   setActiveTutorial(tutorial, index) {
    this.currentTutorial = tutorial;
```

```
    this.currentIndex = tutorial ? index : -1;
   },

   removeAllTutorials() {
    TutorialDataService.deleteAll()
     .then(response => {
      console.log(response.data);
      this.refreshList();
     })
     .catch(e => {
      console.log(e);
     });
   },

   searchTitle() {
    TutorialDataService.findByTitle(this.title)
     .then(response => {
      this.tutorials = response.data;
      this.setActiveTutorial(null);
      console.log(response.data);
     })
     .catch(e => {
      console.log(e);
     });
   }
  },
  mounted() {
   this.retrieveTutorials();
  }
};
</script>

<style>
.list {
 text-align: left;
 max-width: 750px;
 margin: auto;
}
</style>
```

Font-awesome.js

```
import { library } from "@fortawesome/fontawesome-svg-core";
import { FontAwesomeIcon } from "@fortawesome/vue-fontawesome";
```

```
import {
  faHome,
  faUser,
  faUserPlus,
  faSignInAlt,
  faSignOutAlt,
} from "@fortawesome/free-solid-svg-icons";

library.add(faHome, faUser, faUserPlus, faSignInAlt, faSignOutAlt);

export { FontAwesomeIcon };
```

auth-header.js

```
export default function authHeader() {
  let user = JSON.parse(localStorage.getItem('user'));

  if (user && user.accessToken) {
    return { Authorization: 'Bearer ' + user.accessToken }; // for Spring Boot back-end
    // return { 'x-access-token': user.accessToken };       // for Node.js Express back-end
  } else {
    return {};
  }
}
```

auth.service.js

```
import axios from 'axios';

const API_URL = 'http://localhost:8080/api/auth/';

class AuthService {
  login(user) {
    return axios
      .post(API_URL + 'signin', {
        username: user.username,
        password: user.password
      })
      .then(response => {
        if (response.data.accessToken) {
          localStorage.setItem('user', JSON.stringify(response.data));
```

```
      }

      return response.data;
    });
  }

  logout() {
    localStorage.removeItem('user');
  }

  register(user) {
    return axios.post(API_URL + 'signup', {
      username: user.username,
      email: user.email,
      password: user.password
    });
  }
}

export default new AuthService();
```

TutorialDataService.js

```
import http from "../http-common";

class TutorialDataService {
  getAll() {
    return http.get("/tutorials");
  }

  get(id) {
    return http.get(`/tutorials/${id}`);
  }

  create(data) {
    return http.post("/tutorials", data);
  }

  update(id, data) {
    return http.put(`/tutorials/${id}`, data);
  }
```

```
  delete(id) {
    return http.delete(`/tutorials/${id}`);
  }

  deleteAll() {
    return http.delete(`/tutorials`);
  }

  findByTitle(title) {
    return http.get(`/tutorials?title=${title}`);
  }
}

export default new TutorialDataService();
```

user.service.js

```
import axios from 'axios';
import authHeader from './auth-header';

const API_URL = 'http://localhost:8080/api/test/';

class UserService {
  getPublicContent() {
    return axios.get(API_URL + 'all');
  }

  getUserBoard() {
    return axios.get(API_URL + 'user', { headers: authHeader() });
  }

  getModeratorBoard() {
    return axios.get(API_URL + 'mod', { headers: authHeader() });
  }

  getAdminBoard() {
    return axios.get(API_URL + 'admin', { headers: authHeader() });
  }
}

export default new UserService();
```

auth.module.js

```
import AuthService from '../services/auth.service';

const user = JSON.parse(localStorage.getItem('user'));
const initialState = user
 ? { status: { loggedIn: true }, user }
 : { status: { loggedIn: false }, user: null };

export const auth = {
 namespaced: true,
 state: initialState,
 actions: {
  login({ commit }, user) {
   return AuthService.login(user).then(
    user => {
     commit('loginSuccess', user);
     return Promise.resolve(user);
    },
    error => {
     commit('loginFailure');
     return Promise.reject(error);
    }
   );
  },
  logout({ commit }) {
   AuthService.logout();
   commit('logout');
  },
  register({ commit }, user) {
   return AuthService.register(user).then(
    response => {
     commit('registerSuccess');
     return Promise.resolve(response.data);
    },
    error => {
     commit('registerFailure');
     return Promise.reject(error);
    }
   );
  }
 },
```

```
  mutations: {
    loginSuccess(state, user) {
      state.status.loggedIn = true;
      state.user = user;
    },
    loginFailure(state) {
      state.status.loggedIn = false;
      state.user = null;
    },
    logout(state) {
      state.status.loggedIn = false;
      state.user = null;
    },
    registerSuccess(state) {
      state.status.loggedIn = false;
    },
    registerFailure(state) {
      state.status.loggedIn = false;
    }
  }
};
```

index.js

```
import { createStore } from "vuex";
import { auth } from "./auth.module";

const store = createStore({
  modules: {
    auth,
  },
});

export default store;
```

App.vue

```
<template>
  <div id="app">
    <nav class="navbar navbar-expand navbar-dark bg-dark">
```

```
<router-link to="/" class="navbar-brand">Automobile-app</router-link>
<div class="navbar-nav mr-auto">
  <li class="nav-item">
    <router-link to="/tutorials" class="nav-link">Cars</router-link>
  </li>
  <li class="nav-item">
    <router-link to="/add" class="nav-link">Add</router-link>
  </li>
      <li class="nav-item">
    <router-link to="/home" class="nav-link">
      <font-awesome-icon icon="home" /> Home
    </router-link>
  </li>
  <li v-if="showAdminBoard" class="nav-item">
    <router-link to="/admin" class="nav-link">Admin Board</router-link>
  </li>
  <li v-if="showModeratorBoard" class="nav-item">
    <router-link to="/mod" class="nav-link">Moderator Board</router-link>
  </li>
  <li class="nav-item">
    <router-link v-if="currentUser" to="/user" class="nav-link">User</router-link>
  </li>
</div>
    <div v-if="!currentUser" class="navbar-nav ml-auto">
  <li class="nav-item">
    <router-link to="/register" class="nav-link">
      <font-awesome-icon icon="user-plus" /> Sign Up
    </router-link>
  </li>
  <li class="nav-item">
    <router-link to="/login" class="nav-link">
      <font-awesome-icon icon="sign-in-alt" /> Login
    </router-link>
  </li>
</div>

<div v-if="currentUser" class="navbar-nav ml-auto">
  <li class="nav-item">
    <router-link to="/profile" class="nav-link">
      <font-awesome-icon icon="user" />
      {{ currentUser.username }}
    </router-link>
  </li>
  <li class="nav-item">
```

```
        <a class="nav-link" @click.prevent="logOut">
          <font-awesome-icon icon="sign-out-alt" /> LogOut
        </a>
      </li>
    </div>
  </nav>

  <div class="container mt-3">
    <router-view />
  </div>
 </div>
</template>

<script>
export default {
 name: "app",
 computed: {
  currentUser() {
   return this.$store.state.auth.user;
  },
  showAdminBoard() {
   if (this.currentUser && this.currentUser['roles']) {
    return this.currentUser['roles'].includes('ROLE_ADMIN');
   }

   return false;
  },
  showModeratorBoard() {
   if (this.currentUser && this.currentUser['roles']) {
    return this.currentUser['roles'].includes('ROLE_MODERATOR');
   }

   return false;
  }
 },
 methods: {
  logOut() {
   this.$store.dispatch('auth/logout');
   this.$router.push('/login');
  }
 }
};
</script>
```

http-common.js

```
import axios from "axios";

export default axios.create({
  baseURL: "http://localhost:8080/api",
  headers: {
    "Content-type": "application/json"
  }
});
```

main.js

```
import { createApp } from 'vue'
import App from './App.vue'
import router from "./router";
import store from "./store";
import 'bootstrap'
import 'bootstrap/dist/css/bootstrap.min.css'
import { FontAwesomeIcon } from './plugins/font-awesome'

createApp(App)
  .use(router)
  .use(store)
  .component("font-awesome-icon", FontAwesomeIcon)
  .mount("#app");
```

router.js

```
import { createWebHistory, createRouter } from "vue-router";
import Home from "./components/Home.vue";
import Login from "./components/Login.vue";
import Register from "./components/Register.vue";
// lazy-loaded
const Profile = () => import("./components/Profile.vue")
const BoardAdmin = () => import("./components/BoardAdmin.vue")
const BoardModerator = () => import("./components/BoardModerator.vue")
const BoardUser = () => import("./components/BoardUser.vue")
```

```
const routes =  [
  {
    path: "/",
    name: "home",
    component: Home,
  },
  {
    path: "/",
    alias: "/tutorials",
    name: "tutorials",
    component: () => import("./components/TutorialsList")
  },
  {
    path: "/tutorials/:id",
    name: "tutorial-details",
    component: () => import("./components/Tutorial")
  },
  {
    path: "/add",
    name: "add",
    component: () => import("./components/AddTutorial")
  },
  {
    path: "/home",
    component: Home,
  },
  {
    path: "/login",
    component: Login,
  },
  {
    path: "/register",
    component: Register,
  },
  {
    path: "/profile",
    name: "profile",
    // lazy-loaded
    component: Profile,
  },
  {
    path: "/admin",
    name: "admin",
```

```
    // lazy-loaded
    component: BoardAdmin,
  },
  {
    path: "/mod",
    name: "moderator",
    // lazy-loaded
    component: BoardModerator,
  },
  {
    path: "/user",
    name: "user",
    // lazy-loaded
    component: BoardUser,
  },
];

const router = createRouter({
  history: createWebHistory(),
  routes,
});

export default router;
```

vue.config.js

```
module.exports = {
  devServer: {
    port: 8081
  }
}
```