Державний вищий навчальний заклад
“Прикарпатський національний університет імені Василя Стефаника”
Кафедра інформаційних технологій

УДК 004

ДИПЛОМНИЙ ПРОЕКТ

Тема Розробка фронтенд частини проекту «Покупка автомобілів з Європи»

Спеціальність 121 Інженерія програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДП.ІПЗ-11.ПЗ
(позначення)

Рецензент
к.т.н., доц. Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

Студент
ІПЗ-41 Маснюк Р.М.
(шифр групи) (підпис) (дата) (розшифровка підпису)

Нормоконтролер
к.т.н., доц. Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

Керівник дипломного проекту
к.ф.-м.н. Савка І.Я.
(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту

Завідувач кафедри

к.т.н., доц. Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

2020
(рік)

Державний вищий навчальний заклад

«Прикарпатський національний університет імені Василя Стефаника»
Факультет математики та інформатики Кафедра інформаційних технологій
Спеціальність Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Завідувач кафедри Козленко Микола Іванович

_____
„_____”_____20___р.

ЗАВДАННЯ
НА ВИКОНАННЯ ДИПЛОМНОГО ПРОЕКТУ

Студенту Маснюку Ростиславу Мироновичу
(прізвище, ім’я, по батькові студента)

1. Тема проекту Розробка фронтенд частини проекту «Покупка автомобілів з Європи»
затверджена наказом від „25” жовтня 2019 р.№7
2. Термін здачі студентом закінченого проекту 2020-05-22
3. Вихідні дані до дипломного проекту Стандарт кафедри Інформаційних Технологій ПНУ “Вимоги до змісту та оформлення”, Angular API, Bootstrap API
4. Зміст пояснювальної записки (перелік питань, що їх належить опрацювати):
    1. “Аналіз предметної області”;
    2. “Архітектура та дизайн проекту”;
    3. “Практична реалізація проекту”;
    4. “Бізнес план”;
5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень): “Мета проекту”, “Основні завдання проекту”, “Аналіз конкурентів”, “Архітектура та інструменти розробки”, “Діаграма класів”, “Варіанти взаємодії користувача з системою”, “Замовлення авто”, “Продаж авто”, “Реферальна система”, “Бізнес план”
6. Дата видачі завдання 2019.09.11

Керівник _____ Савка І. Я.
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Маснюк Р. М.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів дипломного проекту	Термін виконання етапів проекту	Примітка
Обґрунтування актуальності, формулювання мети, завдання, предмету та об'єкту дослідження проекту	10.10.2019	виконав
Опрацювання джерел з теми проекту	30.10.2019	виконав
Підготовка I розділу проекту	15.12.2019	виконав
Підготовка II розділу проекту	31.01.2020	виконав
Підготовка III розділу проекту	20.02.2020	виконав
Підготовка IV розділу проекту	20.03.2020	виконав
Виправлення зауважень попередніх звітів. Підготовка вступу та висновків	20.04.2020	виконав
Оформлення проекту згідно вимог	10.05.2020	виконав

Студент Маснюк Р. М.
(підпис) (розшифровка підпису)

Керівник проекту Савка І. Я.
(підпис) (розшифровка підпису)

РЕФЕРАТ

Пояснювальна записка: 115 сторінок (без додатків), 56 рисунків, 20 джерел, 1 додаток на 38 сторінках.

Ключові слова: CLI, SPA, SSR, CSR, REST, HTML, CSS.

Об`єктом дослідження є купівля та продаж автомобілів.

Мета роботи: створення веб-додатку, за допомогою якого користувачі зможуть купувати, продавати та замовляти автомобілі.

Стислий опис тексту пояснювальної записки:

Робота описує розробку фронтенд частини для веб-додатку. Основну увагу зосереджено на дизайні та максимально легкій взаємодії користувача з додатком. Досліджено актуальні методи веб-розробки та окреслено їх переваги і недоліки. Описано етапи, з яких складається процес розробки проекту.

ABSTRACT

Explanatory note: 115 pages (without appendix), 56 figures, 20 references, 1 appendix on 38 pages.

Key words: CLI, SPA, SSR, CSR, REST, HTML, CSS.

Object of study: buying and selling cars.

Brief description of the text of the explanatory note:

The work describes the development of a frontend part for a web application. The focus is on the design and ease of user interaction with the application. Current methods of web development are explored and their advantages and disadvantages are outlined. The stages that make up the project development process are described.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CSR	Client Side Rendering
SSR	Server Side Rendering
SPA	Simple Page Application
CLI	Command-line Interface
REST	Representational State Transfer
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
HTTP	HyperText Transfer Protocol
JS	JavaScript
UML	Unified Modeling Language

ВСТУП

Уже багато років автомобіль являється найпопулярнішим видом транспорту на нашій планеті, адже він комфортний, зручний , з його допомогою можна легко переміщатися та швидко добиратися до потрібного місця. Як тільки автомобілі з'явилися, люди зразу ж зрозуміли їхню важливість і сьогодні ні одна людина на Землі не зможе існувати без автомобіля. В даний час автомобіль став найпопулярнішим видом транспорту серед всіх існуючих. Розвиток автомобілебудування не стоїть на місці та з року в рік прогресує.

Дуже багато людей задумуються над покупкою свого власного автомобіля. Автомобілісти хочуть замінити свій автомобіль на більш новий або іншої моделі чи марки. Для одних людей автомобіль – це просто засіб для пересування, а для інших це джерело прибутку. І у всіх цих людей виникає питання, як швидко і вигідно купити транспорт з мінімальними витратами часу та коштів.

Є багато пропозицій щодо цього. Це можуть бути авторинки, рекламні буклети газет, інтернет. Зараз дуже вигідно купувати автомобіль з автомобільного сайту, адже це суттєво економить час. Не потрібно витрачати багато часу на перегляд газет чи оголошень. В інтернеті можна відразу знайти потрібну рубрику і побачити фотографії автомобіля, який сподобався, його ціну, рік випуску, двигун, та інші характеристики. Також можна зробити вибір на користь нових автомобілів або вживаних.

Тепер вже нікого не здивуєш ростом модельного ряду автомобілів. Кожен автовиробник прагне вдосконалити і добавити свою родзинку в автомобіль, тому модельний ряд з кожним роком росте.

Актуальність теми

Купівля автомобіля – досить-таки затратний процес, який сильно б'є по кишені покупця, тому пригін автомобіля з-за кордону буде ідеальним варіантом. Як відомо, в Європі досить великий вибір автомобілів різних марок, типів, а

найголовніше – відмінної якості і хорошого стану. В європейських країнах ідеальні дороги для автомобілів, а регулярне технічне обслуговування дозволяє зберігати автомобіль в ідеальному стані досить довгий час. Всі ці фактори та критерії притягують покупців з різних країн, в тому числі і з України. В такому випадку виникає необхідність у продукті, який б зміг покрити потреби людей, які хочуть придбати авто з-за кордону за невеликі кошти, швидко та без переживань. Потрібен продукт, за допомогою якого людина без всяких труднощів могла б замовити бажаний автомобіль та за короткий термін отримати замовлення. Звичайно, що на сьогоднішній день вже існують різні вирішення такої проблеми, але вони не повністю ефективні та не в повній мірі задовільняють потреби.

Об’єкт дослідження: купівля та продаж авто.

Предмет дослідження: задоволення потреби користувача в придбанні авто з Європи.

Методи дослідження: отримання даних від користувача, обробка цих даних.

Мета дипломного проекту: створення веб-додатку, який через взаємодію з користувачем зміг би покрити його необхідність в купівлі чи продажі авто.

Для досягнення мети дипломної роботи поставлено такі завдання:

- аналіз ринку автомобілів;
- аналіз характеристик автомобілів;
- аналіз потреб людей в придбанні автомобілів;
- створення веб-додатку для покриття потреби людей в придбанні авто.

Завдання проекту: розробити веб-додаток, за допомогою якого користувачі б змогли купувати авто легко та швидко, прикладаючи для цього мінімум зусиль.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз ринку продажу транспортних засобів

На даний час існує багато способів купівлі автомобілів, серед яких купівля автомобіля у знайомих, автобазар, автосалони, дошки оголошень, веб-сайти.

Покупка автомобіля у знайомих з однієї сторони привабливий варіант, адже ризик обману в такому випадку близький до нуля, але і вибір автомобіля дуже обмежений. Тому такий варіант розглядається зі сторони випадкового шансу.

Покупка автомобіля на автобазарі – уже застарілий спосіб. Плюс в покупці авто таким способом – це можливість збити ціну, тобто зробити так званий торг. На авторинках пропонуєтьсся певна «риночна» ціна, але тим не менш через широкий ряд пропозицій автовласники готові поступитися в досить непоганій сумі, заради того, щоб ви вибрали їх авто. В такому місці як правило торгують досвідчені люди віком від сорока років, які поки що погано розбираються в інтернеті, але точно не в автомобілях. Основний мінус автобазарів - це шахраї, тому потрібно бути уважними при купівлі авто таким способом.

Покупка авто в автосалоні також має свої плюси. Основною перевагою такого способу є юридична чистота оформлення угоди. Практично не буває випадків коли після покупки автомобіля в автосалоні з нею виникають труднощі в плані наявності арештів накладених на транспортний засіб чи кредитних заборгованостей. Мінус такого способу – комісійний відсоток, який беруть собі компанії-посередники і ризик потрапити на технічно-неідеальний транспортний засіб. Таку проблему можна вирішити, приїхавши на місце з спеціалістом, адже дуже мала ймовірність, що менеджери запропонують експерту сумнівну техніку.

Дуже цікаві пропозиції можна зустріти на дошках оголошень, таких як OLX чи інші. Переважно на таких площадках автомобілі продають не самі

власники, а їх посередники. Якщо немає ніякої конкретної інформації щодо експлуатації автомобіля, а вказані тільки характеристики авто та привабливий текст про те, яка хороша ця машина, то таке оголошення виклав дійсно посередник.

Найцікавішим зі всіх способів купівлі авто є спеціальні сайти в інтернеті. Вони дають можливість знайти відповідну марку автомобіля максимально швидко. На автомобільних сайтах можна підібрати модель будь-якого року випуску, з різними технічними характеристиками, нову чи таку, яка вже була в користуванні. Продаж автомобілів через інтернет з кожним роком стає все більш популярним. Крім зручності це також дозволяє охопити більшу кількість потенційних покупців. Купівля авто через веб-сайт дає значні перевати перед іншими варіантами купівлі. Це суттєво економить час, а також дає можливість детальніше ознайомитися з особливостями автомобіля, поспілкуватися з продавцем, а також побачити рейтинг продавця та зрозуміти чи потрібно довіряти такому продавцю, та купити в нього автомобіль чи все ж таки утриматися від покупки, адже досить багато продавців бувають нечесними. За допомогою таких веб-сайтів користувачі можуть в будь який момент часу знайти відповідний їм автомобіль, домовитися з продавцем та купити його. Людина ж яка хоче продати автомобіль з легкістю зможе опублікувати його на такому сайті внесши необхідні дані про транспортний засіб. Розмістивши оголошення на веб-сайті, можна спокійно працювати чи займатися своїми справами в той час як автомобіль буде продаватися.

Отже, проаналізувавши ринок продажу автомобілів, можна зрозуміти, що у сайтів продажу авто немає функції замовлення автомобіля, тому виникає потреба в розробці веб-сайту в якому буде така можливість. Основним завданням проекту буде створення веб-додатку, в якому буде функція замовлення автомобіля.

1.2 Аналоги проекту

Ніхто не буде сперечатися з тим, що шукати і вибирати собі автомобіль через інтернет досить зручно. Немає необхідності їздити по автосалонах чи авторинках, передивлятися кожен автомобіль і спілкуватися з набридливими продавцями. В інтернеті є вся інформація про авто. При цьому також можна додатково зв'язатися з власником або представником автосалона, щоб задати додаткові запитання або уточнити деякі деталі. Велика кількість сайтів з продажу авто в Україні в якійсь мірі ускладнює роботу потенційного покупця. Потрібно серед усього цього розмаїття вибрати ресурси, де пропонують досить великий асортимент, але при цьому є мінімальний ризик зіткнутися з шахраями. Від них нікуди не дінешся, але сайти з високим рівнем довіри користувачів найчастіше пропонують більш безпечні умови.

Лідери з продажу

Кількість нових автомобілів і машин з пробігом в Укрїіні стає все більше, через що регулярно з'являються нові сайти продаж. Але далеко не всі з них користуються популярністю. Деякі ресурси знаходяться лиш на початковій стадії свого розвитку, тому аудиторія у них невелика, та і автомобілів там представлено не так багато. Також є сайти, які розроблені виключно для заробітку на рекламі. Власники таких порталів сконцентровані більше на заробіток, ніж на покращенні умов користування своїм ресурсом. Тому їхня популярність умовна, оскільки через них мало хто наважується купити транспортний засіб.

Але є окремий перелік сайтів, які відносяться до списку найпопулярніших і найбільш затребуваних. Ці сайти дозволяють здійснювати продажі транспортних засобів в Україні, причому тут доступні як авто, які вже були в користуванні, так і нові автомобілі. У підсумковий рейтинг увійшли:

- Auto Ria;
- RST;

- InfoCar;
- Auto UA;
- Avto Bazar;
- Автопортал;
- UAvto.

Auto Ria

Будучи підрозділом великої торговельної площадки Ria.com, автопортал отримав власний піддомен, що формально дозволяє зарахувати його до самостійних авто-сайтів України.

В теперішній час він є безперечним лідером по частині продажу транспортних засобів усіх категорій, запчастин, автоаксесуарів. Володіє інтуїтивно зрозумілим інтерфейсом, який зберігається незмінним протягом декілька років. Але це не говорить про занедбаність сервісу. Швидше, навпаки, відданість традиціям дозволяє не тільки залучати нових відвідувачів, але і утримувати існуючих. На сайті також присутня реклама і для користувача, який вперше потрапив на портал вона може здатися набридливою та заважати перегляду. Крім автомобілів з пробігом, тут є можливість придбати нові автомобілі. Також присутній розділ автоновин, тест-драйвів, гарячих пропозицій. Дизайн сайту можна побачити на рисунку 1.1.

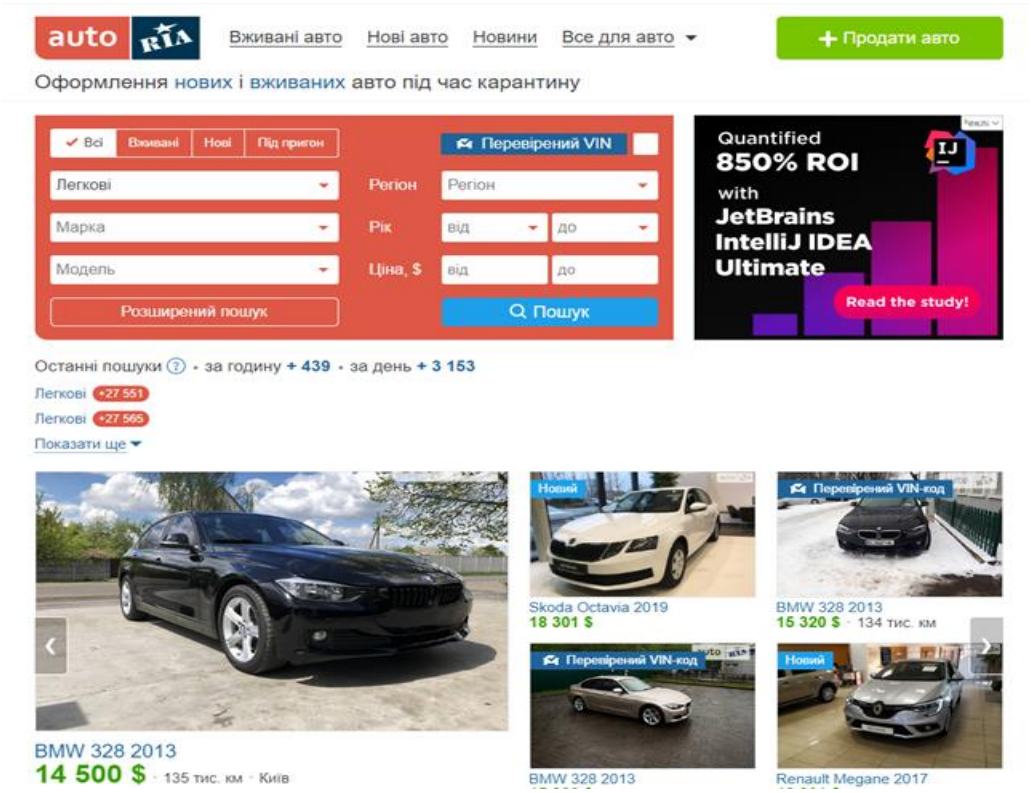

Рисунок 1.1 - Auto Ria

На початку блок фільтра має декілька позицій, але в його розширеній формі можна вказати навіть якісь екзотичні параметри. З кожним днем база даних наповнюється декількома тисячами записів, а кількість щоденних переглядів перевищує 12 мільйонів людей (також враховуються неунікальні відвідувачі).

По замовчуванню подача оголошень є безплатною, присутня можливість сортувати показ авто по п’ятьох параметрах. Також присутня і премуім можливість – за гроші можна купити так званий «преміум аккаунт» і ваш автомобіль з’явиться в топі на головній сторінці. Після деякого проміжку часу записи автоматично видаляються, продовження чи відновлення можна отримати безкоштовно.

Крім легкових автомобілів на сайті також присутні інші категорії транспортних засобів:

- грузовики;
- автобуси;
- мотоцикли;
- сільськогосподарська техніка;
- водний транспорт.

Портал пропонує різноманітність інших опцій, таких як безпечні угоди, а також телефонна форма спілкування з технічною підтримкою, що дозволяє економити час та отримати потрібну відповідь в досить короткий термін. Також в сайту є мобільна версія додатку.

RST

Недивлячись на наявність блоку з новинами на головній сторінці, це дошка оголошень в чистому виді, до того ж дуже популярна. Хоча і по кількості переглядів цей портал програє Auto Ria, але розмір бази даних користованих авто тут помітно більший, а за день може додатися до тридцяти тисяч оголошень. В такому випадку можна сказати, що цей сайт в Україні хоч і програє лідеру, але не настільки сильно. На порталі також можна дізнатися свіжі дані про автосалони і переглянути каталог недавно випущених з виробництва автомобілів. У сервісу присутній специфічний, унікальний дизайн який досить важко сплутати з іншими. Також присутня реклама, яку досить легко відрізнити від оголошень. Дизайн сайту можна побачити на рисунку 1.2.

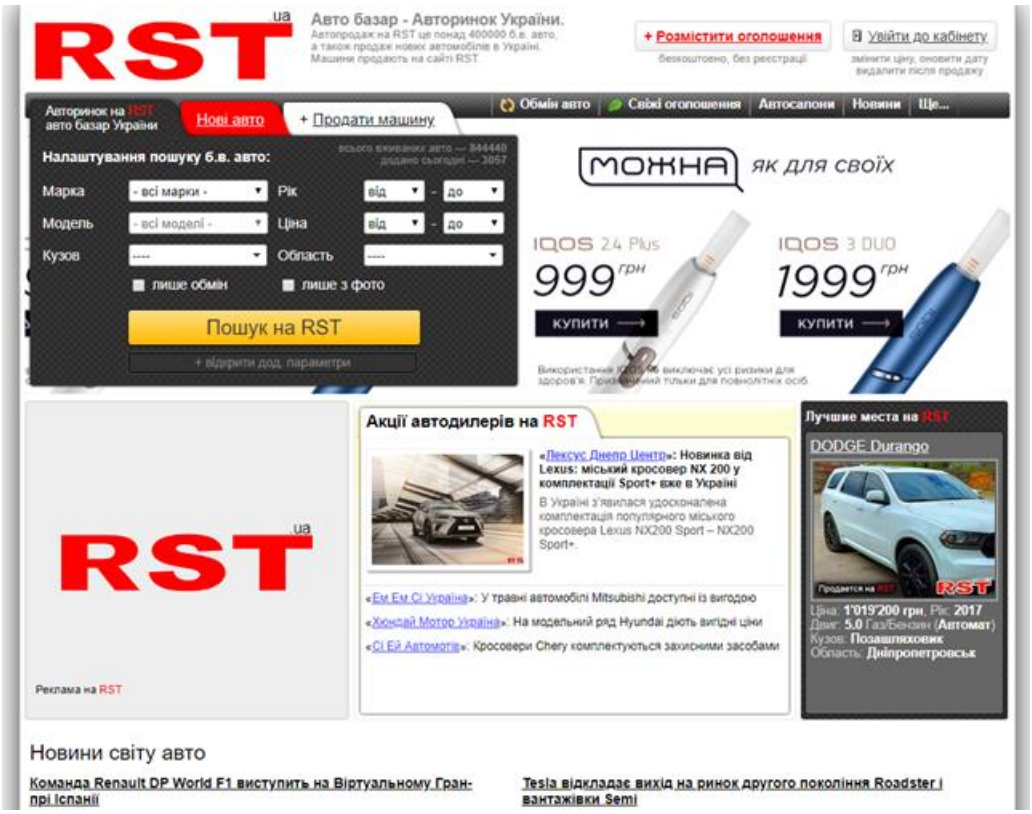

Рисунок 1.2 – RST

Щоб опублікувати дані про автомобіль потрібна реєстрація, але окремо такої опції не існує, тому всі необхідні реєстраційні дані, в тому числі логін і пароль подаються безпосередньо під час подачі оголошення про продаж авто. В результаті зареєстровані користувачі мають змогу вносити зміни в дані, які були опубліковані, а також переглядати статистику.

InfoCar.ua

Цей сервіс з продажу авто займає третю сходинку в топі кращих в Україні. На сайті досить непогане оформлення, а також великий об'єм інформаційно-довідкових розділів (відгуки, новини, тест-драйви, відеоматеріали, статті, автосалони). По кількості пропозицій цей сервіс значно поступається лідерам – в базі даних близько п'яти тисяч дописів, а повсякденна аудиторія складає близько триста п'ятдесят – чотириста тисяч осіб. Дизайн сайту можна побачити на рисунку 1.3.

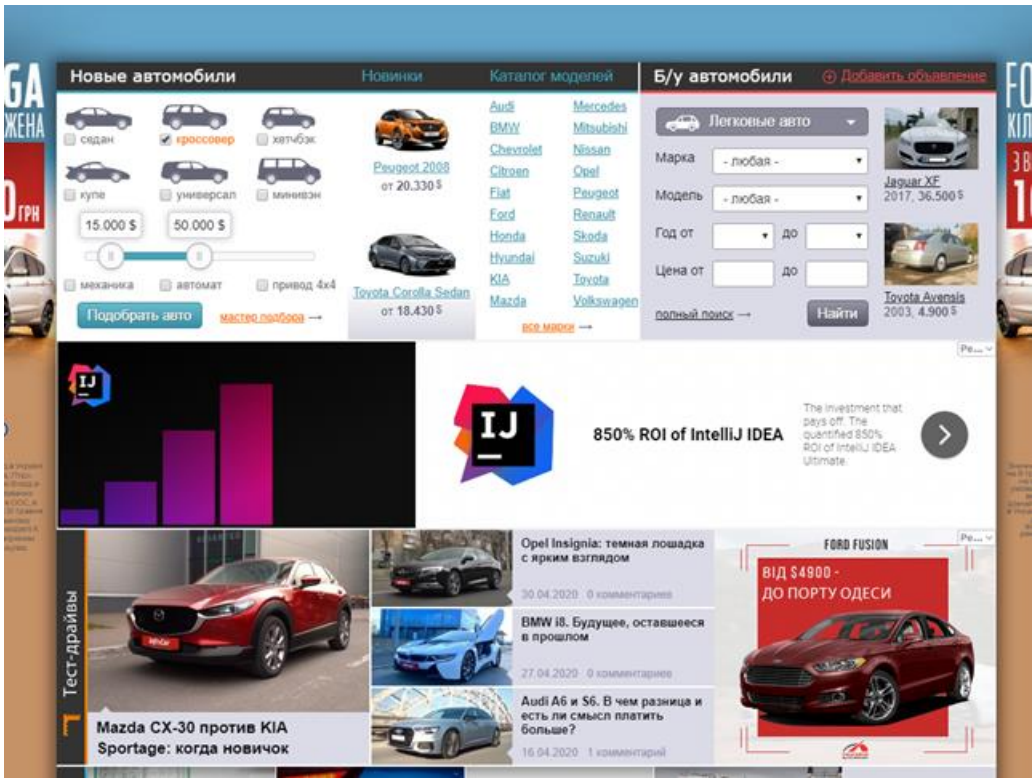

Рисунок 1.3 - InfoCar.ua

Для того, щоб додати автомобіль в базу не обов'язково реєструватися, але в такому випадку можна позбутися можливості виконувати такі дії:

- редаговувати дописи;

- видаляти дописи;

- видаляти коментарії;

- пропонувати обмін авто;

- створювати аукціон;

- отримувати сповіщення.

Сама процедура реєстрації досить примітивна і не займає багато часу та зусиль, але також існує опція реєстрації через соціальні мережі. До недоліків сервісу можна віднести досить слабкий алгоритм сортування, а також

перегруженість рекламою, при чому достатньо одного неправильного кліку, щоб відкрилося нове вікно з рекламованим продуктом.

Autoua

Серед найпопулярніших сайтів з продажу вживаної автомобільної техніки також розмістився портал Autoua. Якщо на початку головною перевагою ресурсу був автофорум, на якому користувачі ділилися своїм досвідом в різних аспектах, то з часом вони стали тими користувачами, які вирішили використати даний портал для продажу своїх автомобілів. Саме на цьому сайті можна задати досить заплутані питання і гарантовано отримати відповідь на них, а то і не одну. Тут можна подивитися тест-драйви нових чи недавно випущених з виробництва автомобілів, почитати відгуки від реальних власників, зрозуміти всю картину про автомобіль і купити його чи навпаки продати. Дизайн сайту можна побачити на рисунку 1.4.

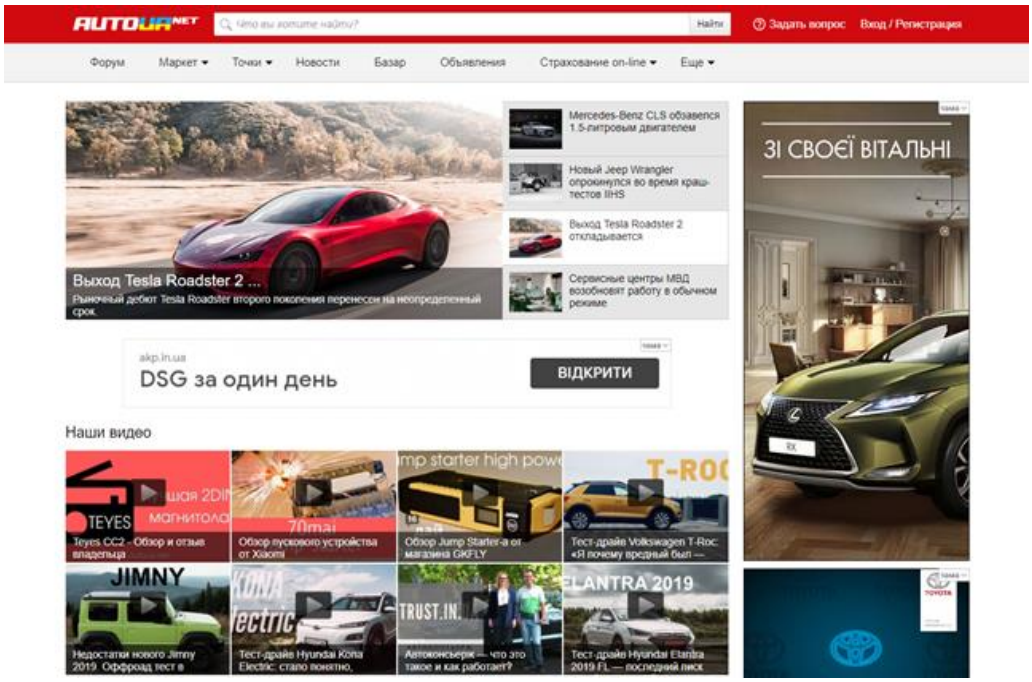

Рисунок 1.4 – Autoua

В розділі «Маркет» крім нових легкових автомобілів також представлена мототехніка і комерційні авто. Тут можна знайти найближчий автосалон і побачити відгуки тих користувачів, які вже являються власниками даної моделі.

В розділі «Базар» знаходять автомобілі, які вже були в користуванні, при цьому загальна кількість оголошень з продажу перевищує сто тисяч записів, а повсякденний трафік близько двісті тисяч осіб.

Цей портал являється одним з тих, на яких кожен відвідувач може висловити свою думку про конкретну модель і до того ж бажаючих поділитися інформацією завжди багато.

AvtoBazar

Цей ресурс вважався одним з найпопулярніших автосайтів в Україні на протязі довгого часу і був однойменною копією газети. З часом паперова версія перестала бути настільки затребуваною, як і раніше звичайно ж через широке поширення інтернету. Цей портал з часом почав втрачати свої позиції на ринку, але сьогодні він з впевненістю входить в топ автосайтів з повсякденною аудиторією в сто п’ятдесят тисяч відвідувачів.

Можна стверджувати, що початкова орієнтація на інформаційну складову сьогодні поступилася комерційній і тому на головній сторінці користувачі можуть переглядати базу б/у автомобілів. Дизайн сайту можна побачити на рисунку 1.5.

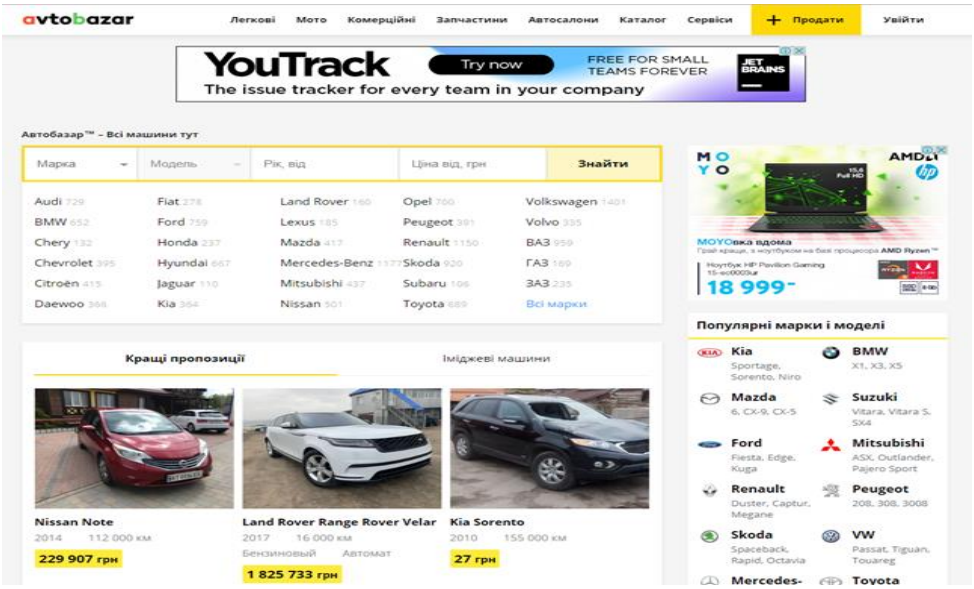

Рисунок 1.5 – AvtoBazar

По розміру бази даних «AutoBazar» поступається всім вищезгаданим сайтам. Крім легвових автомобілів тут можна придбати чи продати мототехніку, комерційні авто, автобуси, запчастини до автомобілів, побувати на авторозбиранні, також почитати відгуки, подивитися тест-драйви. Просто кажучи, функцій тут дійсно багато.

Для того, щоб запостити автомобіль для продажу, буде потрібно зареєструватися і підтвердити адрес електроної пошти або телефон. Процес подачі оголошення нестандартний, через те, що автомобілі продаються кожного дня, тому пройти цю процедуру один раз буде потрібно точно.

Autoportal.ua

Це один з тих небагатьох платформ для продажу та купівлі транспортних засобів, де список пропозицій обмежується тільки легковими автомобілями та мікроавтобусами, тобто у списку товарів тут немає ні грузовиків, ні мототехніки і тим більше спецтехніки. Але ні в якому разі не потрібно думати, що такий портал не потрібний і бідний на пропозиції.

Також тут присутній досить цікавий розділ «Читальня», де можна переглянути останні новини в світі автомобілів, ознайомитися з найновішими тест-драйвами та оглядами автомобілів, а також почитати аналітику. Дизайн сайту можна побачити на рисунку 1.6.

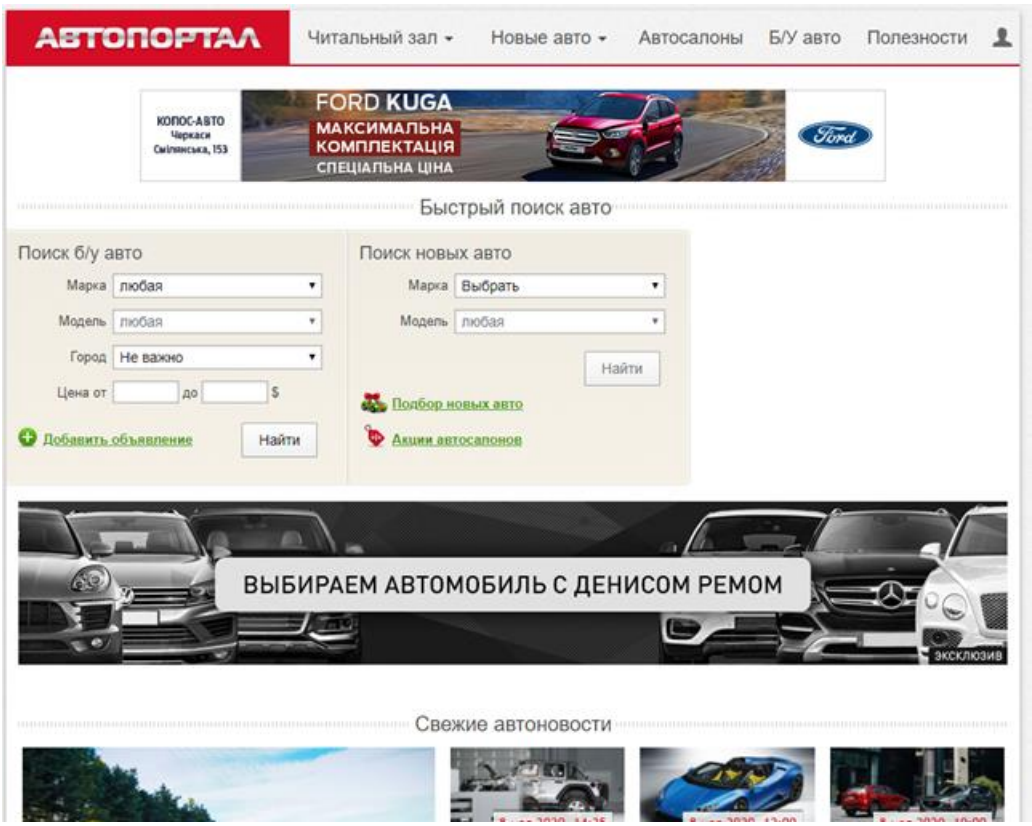

Рисунок 1.6 - Autoportal.ua

Увійшовши на сайт, можна побачити, що на ньому досить багато реклами, і це говорить, що портал втрачає популярність, проте виглядає він привабливо, акуратно, а що найголовніше – шахрайських пропозицій тут дуже мало, адже адміністрація сайту робить все можливе, щоб запобігти обманів користувачів. Фільтр тут звичайний, а сортування здійснюється по двох критеріях. Втім, кількість варіантів, які задовільнять потреби користувачів буде невелика, тому останній недолік неважливий.

Uavto

Це один з тих сайтів, де можна купити машину, вибравши відповідний регіон. За кількістю оголошень ресурс досить непоганий - понад сто тисяч дописів, які щодня поповнюються на три – п’ять тисяч нових пропозицій. Але таке наповнення - не стільки результат діяльності продавців, скільки наслідок роботи скриптів, які моніторять інші популярні ресурси. В великій мірі Uavto можна назвати генератором безкоштовних оголошень. Для користувачів це зручно, але з точки зору достовірності наданої інформації не все так гладко, кількість шахрайських, фейковий або прострочених пропозицій велика, бо не контролюється. Дизайн платформи можна побачити на рисунку 1.7.

Рисунок 1.7 – Uavto

На цьому сайті авто для продажу можна розміщувати і без реєстрації. Для того, щоб здійснити основні операції з оголошенням порібно використовувати код, який прийде на мобільний телефон. Тут є розділ з новими автомобілями, а вживаний транспорт розбитий на категорії. Також можна ознайомитися з списком регіональних автосалонів та магазинів. Реклама присутня, але її межі чітко видно, тому можна з легкістю переміщатися по сайту.

1.3 Актуальність проекту

Не кожна людина може собі дозволити купити престижну іномарку чи навіть вітчизняний новий автомобіль через досить велику ціну. Цей факт зумовив масову зацікавленість громадян України в покупці і привозі б/у автомобілів з Європи.

Покупка автомобіля, який вже був у користуванні за кордоном має дуже багато так званих «підводних каменів», про які повинна знати людина, яка бажає купити такий автомобіль в Європі самостійно. В першу чергу це стосується витрати коштів. Перед тим як відправитися за покупкою бажаного авто, потрібно добре подумати, чи буде це максимально вигідно з фінансової точки зору, чи може простіше буде підібрати схожий автомобіль на Українському ринку. Затрати на проїзд, проживання в тій чи іншій Європейській країні, оплата державного мита, а також брокерських послуг обійдеться в 50% від вартості автомобіля. Як правило, потенційний покупець перед тим як купити автомобіль звертається до спеціалізованих веб-сайтів, які продають автомобілі в тій чи іншій країні. На таких сайтах можна переглянути різні варіанти автомобілів, які були виставлені на продаж та порівняти ціни з аналогами в Україні та після цього домовитися про покупку. В подальшому, покупець самостійно їде в країну продавця. Після знайомства з продавцем і огляду автомобіля самостійно, він

може скористатися послугами станцій технічного обслуговування, та здійснити перевірку ще раз. Таку перевірку здійснюють професіонали, тому це дасть більше гарантії про добрий технічний стан автомобіля. Єдиний мінус такої процедури це те, що вона звичайно ж платна і коштує не зовсім то й мало. Також можна запропонувати продавцю перевірити автомобіль на те чи він був украдений. Здійснити таке можна в поліцейському відділку. Але знову ж таки, за це потрібно заплатити гроші. Після цього можна буде укласти договір про купівлю-продаж, зняття автомобіля з реєстрації, після чого можна буде забрати автомобіль та поїхати додому. Такий процес звичайно можна пришвидшити, якщо купувати автомобіль з салону, де всі ці дії будуть проводитися в самому салоні. В кінці після перетину Українського кордону можна зайнятися розмитненням автомобіля.

Дуже часто люди не хочуть займатися вище згаданими процедурами власноруч, через необізнаність та некомпетентність у цій сфері, чи може не хочуть тратити власний час на здійснення такої роботи. Дуже багато людей звикли отримувати готове і доволі часто людині буде простіше замовити послуги ніж робити це самому. В Україні є дуже багато осіб чи компаній, які займаються таким видом діяльності. Такі люди чи компанії займаються пригоном автомобілів з Європи та їх розмитненням не малий час і можна сказати, що вони є професіоналами у цій справі, тобто знають всі нюанси. Вони допоможуть підібрати автомобіль, який ви забажаєте, перевірити його технічний стан, історію автомобіля. Від імені замовника можуть купити автомобіль в будь-якій Європейській країні відповідно до бюджету, який надасть замовник. Після цього оформлять всі документи для транспортування авто і доставлять його замовнику. Тобто всього за декілька днів, нічого не роблячи, сидячи на дивані, і не прикладаючи ніяких зусиль, можна отримати бажаний автомобіль.

Отже виникає питання, де замовити потрібний автомобіль. Для цього і потрібно створити сайт, на якому буде така можливість. Потрібно створити такий сайт, на який можна зайти, вказати всі деталі про автомобіль який хочеться

придбати, та в короткий термін отримати пропозицію про бажання виконати таке замовлення. Такий вебсайт просто перекриє потребу людей в пошуку знайомих чи інших людей, які займаються пригоном автомобілів з Європейських країн, а головне це економить час та ресурси. Такий сайт буде ідеально підходити як для осіб, які хочуть купити чи замовити авто, так і для людей, які займаються виконанням замовлень з пригону авто, адже не потрібно буде більше робити реклами своїх послуг, а просто, зайшовши на сайт, вибрати замовлення яке можна здійснити та виконати його в максимально короткий термін, щоб підвищити свою репутацію і відповідно збільшити свій дохід.

1.4 Постановка задачі

Об’єкт проекту: Веб-додаток для купівлі, продажу та замовлення авто.

Предмет проекту: Розробка веб-додатку.

Мета проекту полягає в створені веб-сайту, за допомогою якого користувачі зможуть купувати, продавати чи замовляти автомобілі.

Завдання проекту: Для досягнення мети дипломної роботи поставлено такі завдання:

- проаналізувати ринок автомобілів;
- проаналізувати характеристики автомобілів;
- проаналізувати потреби людей в придбанні автомобілів;
- створити веб-додаток, який задовільнить потребу людей в придбанні авто.

2 АРХІТЕКТУРА ТА ДИЗАЙН ПРОЕКТУ

2.1 Основні завдання фронтенду

Ми живемо в час інформаційних технологій. Зараз народжуються, розвиваються і трансформуются в щось велике не тільки технології, але і цілі професії. Така ситуація трапилася з так званими «сисадмінами». Їм на зміну прийшли DevOps-інженери (Development Operations engineers) – люди, які займаються автоматизацією завдань пов’язаних з налаштуванням та розгортанням проектів. А от з фронтенд розробкою зовсім інша ситуація. Вона зараз в розквіті.

В той час, люди ще дуже мало знали про інтернет і не розуміли, що в ньому можна робити. Тоді майже всі сайти були доволі прості: просто текст з інформацією, декілька посилань на аналогічну сторінку і все. Браузери теж були дуже прості, а часто навіть чисто текстові (наприклад, консольні), тому і сторінки складалися з базових тегів HTML[6], які розмічали структуру текстового документу. Проаналізувавши все, що відбувається зараз, можна сказати, що саме тоді і був зоряний час семантики і html-теги використовувалися за прямим призначенням. Проте, такий час інтернету швидко пройшов і з'явилися перші великі сайти з високими вимогами до зовнішнього вигляду, а отже, і до верстки.

Оскільки єдиним універсальним способом точного розміщення елементів дизайну були таблиці, то настав час табличної верстки. CSS тоді ще не розглядався більшістю як інструмент позиціонування елементів на сторінці, в основному через недостатню підтримку відповідних властивостей в браузерах. Найкраще, на що міг розраховувати CSS, - це управління кольором, вирівнюванням і положенням шрифту, і то для цього було безліч атрибутів у тегів.

Як наслідок, не було ніяких «фронтенд-розробників», оскільки не було самого фронтенда. Були верстальники, і були програмісти. Програмісти робили всі круті речі, а верстальники відчували себе приниженими і чекали IE6, в якому буде підтримуватися багато крутих речей.

Початком народження фронтенду, який ми зараз бачимо, була поява можливості асинхронних запитів через XHR. Сайтам потрібен був AJAX, щоб бути швидкими та динамічними, а верстальники зрозуміли, що за допомогою JavaScript[1] можна не тільки писати атрибут onclick, а й взагалі часто його використовувати для оживлення сторінок.

У міру зростання продуктивності браузерів на клієнт, тобто в браузер, який працює на машині користувача, стали переносити багато паттернів з «великого» програмування. З'явилися перші MVC-фреймворки, які керували даними, що завантажувалися через AJAX запити, правильно їх відображали на веб-сторінці і робили інші запити за потреби. Все це працювало не так швидко, як зараз, просто тому, що JavaScript-рушії були повільні і браузер міг зависнути від доволі простих задач.

Приблизно в цей період все і почалося. Хоча верстальникам і раніше доводилося «оживляти» власні макети, накладаючи на них реальні дані, але з появою динаміки на клієнті стало ясно, що займатися формуванням HTML на сервері і подальшою його зміною на клієнті повинна одна людина. Після цього було чітко відділено бекенд, як деяка обгортка навколо бази даних, що приховує її внутрішній устрій і реалізує деякі методи отримання даних і фронтенд, який займався показом необхідної HTML сторінки за запитами користувачів. Приблизно в цей час і встановилися основні завдання фронтенду: агрегація, шаблонізація і кешування.

Агрегація - це обробка даних, що надійшли з бекенду, в той вид, який найбільш зручний для подальшої шаблонізації. Наприклад, якщо бекенд видає необхідні дані двома окремими методами, а в результаті потрібен один список

на сторінці, буває зручно заздалегідь злити результати двох методів, щоб спростити шаблон. Або, наприклад, якщо бекенд не вміє сортувати в потрібному порядку або таких порядків відразу кілька, то на етап агрегації може лягти завдання самостійного сортування отриманих даних.

Шаблонізація - це створення результуючого HTML з отриманих даних. У різних архітектурах шаблони бувають різного ступеня складності: від найпростіших шаблонів, які просто відображають дані, які прийшли з бази даних, до складних систем, що включають в себе велику частину бізнес-логіки.

Кешування - це запам'ятовування отриманої за конкретним запитом відповіді, щоб в подальшому уникнути запитів до бази даних і подальшої обробки. Кешування - це не завжди завдання фронтенду. Часто його перекладають на HTTP-сервер, проте зовсім позбутися від цієї частини не вийде: все одно ніхто краще від фронтенду не знає, як саме треба кешувати певні розділи сайту.

Ну і звичайно ж, величезним поштовхом для розвитку фронтенд-розробки стала поява Node.js. Якщо говорити точніше, то сама поява Node.js стала наслідком підвищення значення JavaScript. З появою JavaScript фронтенд-розробники отримали багато можливостей.

Якщо раніше JavaScript сприймався як суто мова для браузерів, то тепер з'явилася можливість запускати його на серверній стороні. Фронтенд-розробники почали самостійно збирати свої проекти, тобто приводити їх з того виду, в якому зручно розробляти, в той вид, в якому все повинно показуватися користувачеві (мініфікувати код, що завантажується на клієнт, мінімізувати картинки, компілювати шаблони за потреби і так далі).

2.2 Серверний та клієнтський рендеринг

Дискусія про візуалізацію веб-сторінок з'явилася на світ досить недавно. Раніше веб-сайти та веб-додатки мали спільну стратегію. Вони готували вміст HTML для надсилання у браузери на стороні сервера. Потім цей вміст виводився у браузері у форматі HTML із CSS[7]. З появою фреймворків JavaScript прийшов до зовсім іншого підходу з веб-розробки. Фреймворки JavaScript надали можливість скидання тягаря з сервера. Завдяки потужності фреймворків JavaScript[16] стало можливим візуалізувати динамічний вміст прямо з браузера, запитуючи лише необхідний вміст. Сервер у цьому сценарії обслуговував лише необхідну базову HTML-обгортку. Ця трансформація надала користувачам безперебійний досвід, оскільки для завантаження веб-сторінки витрачається дуже мало часу. Більше того, після завантаження веб-сторінка не завантажується знову.

Візуалізація на стороні сервера або SSR (Server Side Rendering) - це звичайний спосіб рендерингу веб-сторінок у веб-браузері. Традиційний шлях рендерингу динамічного веб-контенту складається з таких кроків:

1. користувач надсилає запит на веб-сайт (зазвичай через браузер);
2. сервер перевіряє ресурс, збирає та готує вміст HTML;
3. скомпільований HTML відправляється в браузер клієнта для подальшого рендерингу та відображення;
4. браузер завантажує HTML і робить сайт видимий для користувача;
5. браузер завантажує всі необхідні JavaScript[17] файли і виконує їх, що робить веб-сторінку інтерактивною.

Весь хід SSR подій зображено на рисунку 2.1.

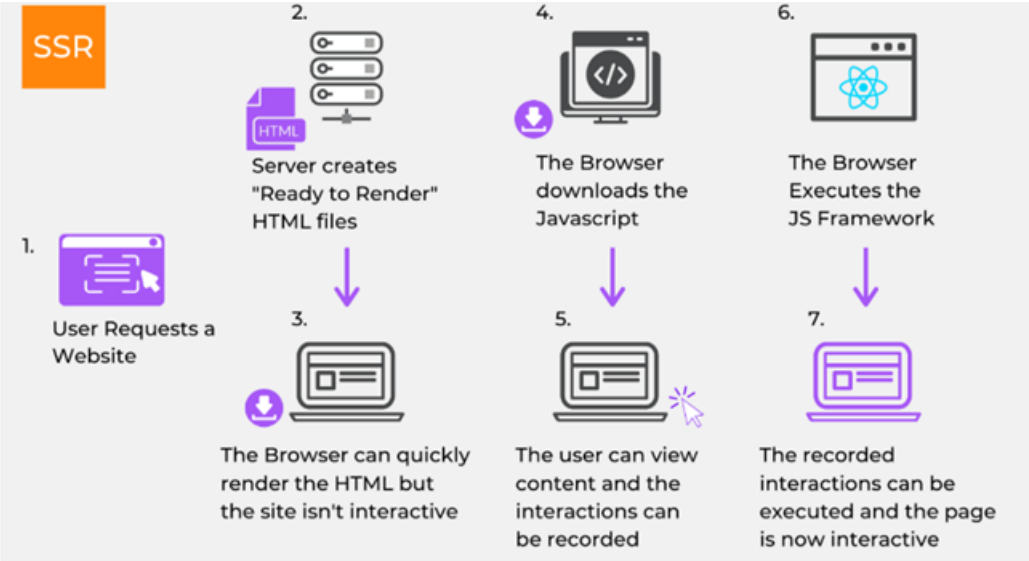

Рисунок 2.1 – SSR

У цьому процесі всі дії з отримання динамічного контенту, перетворення його в HTML і відправлення його в браузер відбуваються на сервері. Отже, цей процес називається візуалізацією чи рендерингом на стороні сервера (SSR). Такий процес збільшує час завантаження сторінки порівняно з часом завантаження сторінки для статичного сайту, де немає динамічного вмісту для відображення.

Клієнтська візуалізація або CSR (Client Side Rendering) - це інший випадок того, як може оброблятися веб-сторінка для відображення в браузері. У CSR компіляцію динамічного контенту та генерування HTML виконує сам браузер. Такий підхід підтримується фреймворками JavaScript та бібліотеками, такими як ReactJS, VueJS та Angular. CSR здійснюється так:

1. користувач надсилає запит на веб-сайт (зазвичай через браузер);
2. замість сервера може використовуватися CDN (Content Delivery Network) для подачі статичного HTML, CSS та підтримуючих файлів користувачеві;
3. браузер завантажує HTML, а потім JS, в той час як користувач бачить символ завантаження;

4. після того, як браузер отримує JS, він робить API запити через AJAX для отримання динамічного вмісту та обробляє його для надання кінцевого вмісту;
5. після того, як сервер відповідає, кінцевий вміст виводиться у браузері.

Весь хід CSR подій можна побачити на рисунку 2.2.

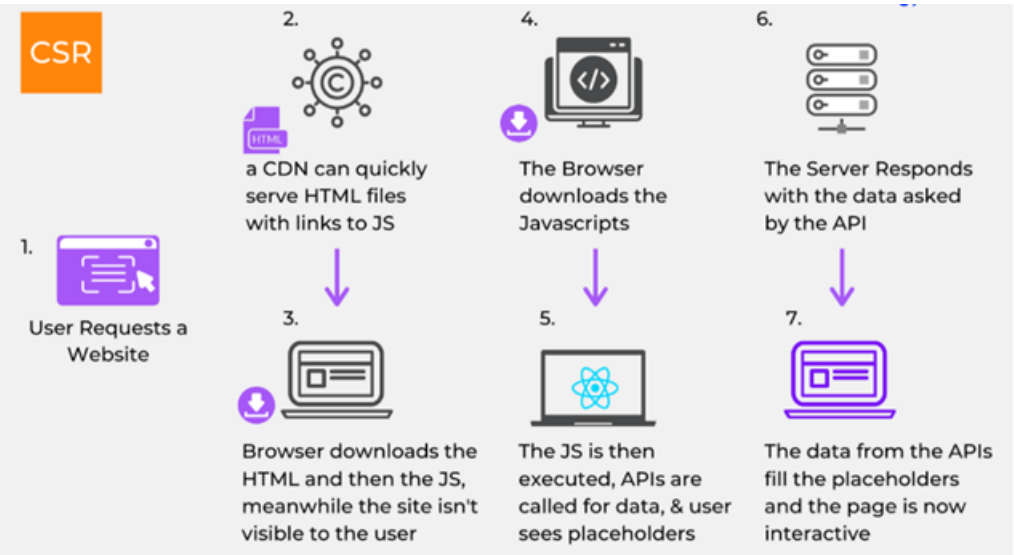

Рисунок 2.2 – CSR

Оскільки цей процес включає отримання та обробку даних на частині клієнта то він називається візуалізацією чи рендерингом на стороні клієнта.

Обидва підходи відрізняються за способом обробки контенту, тому кожен метод має свої переваги, що ускладнює зробити рішення, що вибрати - CSR або SSR. Основні відмінності між такими видами рендерингу полягають в швидкості завантаження сторінки, впливу на кешування а також впливу на SEO оптимізацію.

Час завантаження веб-сторінки - це час, який проходить між тим, коли запит надсилається на сервер, і коли дані відображаються в браузері. Це важливий аспект, коли мова йде про «user experience» для веб-сайту чи веб-

додатку. Час завантаження веб-сторінки для CSR та SSR відрізняється у двох сценаріях: загрузка першої сторінки, та загрузка другої сторінки і наступних.

Час завантаження першої сторінки - це середній час, який витрачається під час першого завантаження користувачем веб-сайту. Під час першого завантаження у CSR браузер завантажує базовий HTML, CSS та всі необхідні сценарії одразу, а потім компілює HTML. Загрузка першої сторінки у SSR відбувається швидше ніж у CSR.

Час завантаження другої сторінки - це середній час, необхідний для навігації з однієї сторінки на іншу. Оскільки у CSR всі скрипти завантажуються заздалегідь то відповідно час завантаження для CSR менший (і, таким чином, краща продуктивність).

В теперішній час кешування є дуже важливим. Щоб пришвидшити великі веб-додатки, кожен браузер, а також веб-сервер використовує механізми кешування для кешування скриптів багаторазового використання на стороні клієнта. Це покращує загальний час завантаження для CSR, а також SSR. Однак більшу вигоду від цього отримує звичайно ж CSR. Якщо у CSR не використовується лінива загрузка модулів, то веб-додаток може працювати без інтернету. На відміну від CSR, у SSR запити на сервер завжди надсилаються, а отже і час завантаження сторінки більший. Кешування збільшує швидкість рендерингу вмісту веб-сторінки, оскільки скрипти витягуються браузером з кешу. На рисунку 2.3 зображено як браузер обробляє запит на скрипт, який знаходиться у кеші.

Рисунок 2.3 - Обробка запиту на кешований скрипт

Більшість скриптів завантажуються з пам'яті або дискового кешу. Це значно покращує час завантаження, а також запобігає надмірному навантаженню на сервер.

Для бізнес-сайтів SEO оптимізація є надзвичайно важливою. Пошукові системи читають і розуміють веб-сайти, використовуючи автоматизовані боти, які називаються сканерами. Цих сканерів більше цікавлять метадані веб-сайту, ніж власне вміст. Отже, важливо, щоб веб-сторінка відображала правильні метадані для пошукових систем. Завдяки CSR вміст веб-сторінки динамічно генерується за допомогою JavaScript. Це означає, що зміна метаданих з однієї сторінки на іншу покладається на виконання JavaScript. В SSR веб-сторінка складається з правильних метаданих та надсилається на фронтенд лише після отримання остаточного вмісту HTML. Це забезпечує те, що метадані сторінки завжди будуть точними, незалежно від того, чи дозволяє сканер використовувати JavaScript чи ні. Це робить SSR кращим рішенням для сторінок, оптимізованих під пошукові системи, порівняно з CSR. Оскільки проект розроблявся на фреймворку Angular, то в ньому застосовується технологія CSR. SEO оптимізацією в такому випадку було знехтувано, та зроблено вибір на користь швидкодії веб-додатку.

2.3 Single page application як тип веб-додатку

Односторінковий веб-додаток (SPA) - це підхід до розробки веб-сайтів, де вміст кожної нової сторінки подається не з завантаження нових сторінок HTML, а генерується динамічно завдяки можливості JavaScript маніпулювати елементами DOM на самій існуючій сторінці. У більш традиційній архітектурі веб-сторінок файл index.html може посилатися на інші HTML-сторінки на сервері, які браузер завантажуватиме та відображатиме з нуля. SPA-підхід

дозволяє користувачеві продовжувати користуватися сторінкою та взаємодіяти з нею, коли нові елементи оновлюються чи отримуються, і це може призвести до набагато швидшої взаємодії та перезавантаження вмісту. Крім того, API History HTML5 дозволяє змінювати URL-адресу сторінки без перезавантаження самої сторінки, що дає можливість створювати окремі URL-адреси (роути) для різних представлень даних. Додаток, розроблений за допомогою SPA може динамічно отримувати вміст із сервера через AJAX-запити або веб-сокети. Це дозволяє браузеру тримати поточну сторінку відкритою під час подання запитів на сервер у фоновому режимі, щоб отримати додатковий вміст або нові сторінки взагалі. Насправді, серверні запити можуть отримувати будь-які дані, часто приймаючи форму JSON[8], строк або навіть HTML-елементів які вже підготовлені для візуалізації. Однією з найбільших переваг SPA є досвід користувачів, тобто коли користувач може без проблем користуватися додатком без необхідності чекати перезавантаження сторінки та інших речей.

Основною перевагою SPA-додатків є їх швидкість. Більшість необхідних ресурсів для роботи програми в SPA (HTML, CSS, скрипти) завантажуються при запуску програми і потім не потрібно буде їх завантажувати знову під час використання самої програми. Єдине, що змінюється – це дані, які передаються на сервер та отримуються з нього. Як результат, додаток дуже швидко реагує на запити користувача, що викликає ще більше задоволення від користування. Численні дослідження Google та ключові висновки таких великих компаній, як Amazon, WalMart показують, що якщо на завантаження сторінки потрібно більше 200 мілісекунд то це може зруйнувати бізнес, або, щонайменше, коштувати великих грошей. Наприклад для Amazon 1 секунда додаткової затримки буде коштувати 1% від продажів (що, зважаючи на кількість продажів Amazon, становить 1,6 мільярда доларів на рік). Також одним з найважливіших переваг SPA є їх універсальність. Вони однаково добре функціонують як на персональних комп'ютерах, так і на мобільних пристроях. Планшети, смартфони і навіть звичайні мобільні телефони без проблем працюють з проектами,

розробленими по принципу SPA. Отже великим плюсом односторінкових застосунків можна вважати широке охоплення пристроїв, на яких вони можуть працювати. Відповідно розробка односторінкового застосунку дозволяє розраховувати на значно більшу цільову аудиторію, ніж при використанні звичайних методів веб-розробки.

Іншою немаловажливою перевагою SPA є багатий інтерфейс. Цей плюс зумовлений тим, що на одній веб-сторінці набагато простіше створити насичений інтерфейс. Такий підхід дуже сильно спрощує процеси зберігання інформації, а також керування станом представлення та анімацією.

Ще одна перевага одосторінкових застосунків полягає в спрощенні процедури загрузки контенту. Якщо веб-сайт працює з шаблонами, то разом з загрузкою будь-якої сторінки користувач завантажує також розмітку шаблону. Звичайно кешування зараз досягло дуже великих результатів, проте в SPA немає що кешувати, а це означає що відбувається дуже сильна економія як часу, так і ресурсів.

Принцип роботи односторінкових застосунків зображено на рисунку 2.4.

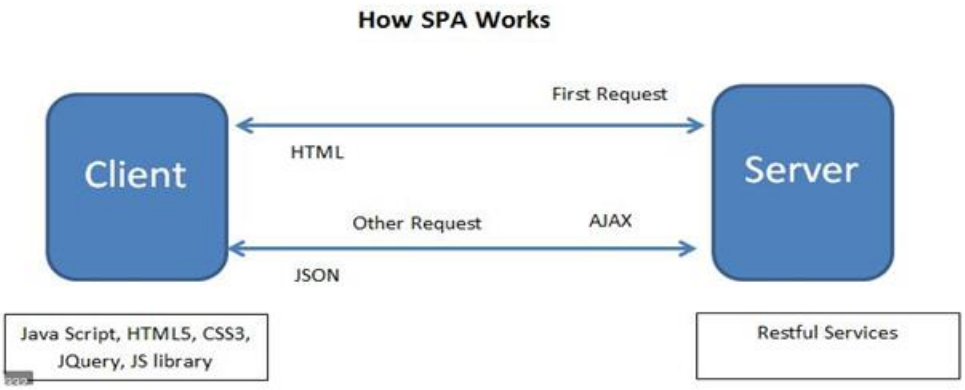

Рисунок 2.4 - Принцип роботи SPA

Можна зробити висновок, що SPA – це сайт або веб-додаток, який розміщений на одній веб-сторінці. Для SPA характерний більш гнучкий і зручний інтерфейс користувача, який схожий з інтерфейсом десктопного додатку. Крім того односторінковий застосунок має більш високу продуктивність. SPA забезпечує більш природній і контрольований досвід взаємодії (UX), приховуючи складні переходи (запити, відповіді і т.д.). SPA відображає тільки той контент, який запитує користувач, завантажуючи HTML-дані частинами. Отже існує два види SPA-додатків: додаток, який вбудовує контент декількох сторінок в одну сторінку-оболонку, і додаток, який використовує Ajax-загрузку контенту, коли користувач здійснює клік по посиланні. Таким чином на сторінці оновлюється та частина вмісту, яка змінюється, а інша частина сторінки залишається без змін та перезагрузок. SPA-підхід для розробки веб-додатків слід використовувати, якщо під метою стоїть створення повномасштабного застосунку з складним інтерфейсом користувача та сотнями сторінок, а також якщо такий додаток повинен задовільняти потреби численної та різноманітної аудиторії користувачів, які використовують різні пристрої. Також SPA-додатки слід використовувати, якщо потрібен супер-швидкий сайт з динамічною загрузкою даних та покращеним UX. JavaScript-фреймворк Angular – яскравий представник технології SPA, в якому присутні всі яскраві риси такого виду створення веб-додатків. Дипломний проект задумувався як швидкий, зручний та легкий в користуванні веб-застосунок і за допомогою SPA та Angular можна таке реалізувати.

2.4 Можливості Typescript

JavaScript був представлений як мова програмування для сторони клієнта. Але з ростом популярності цієї мови, та розвитку коду також росла важкість в

налагодженні та перевикористанні коду JavaScript. Код ставав заплутанішим. JavaScript – це нетипізована мова програмування і саме в типізації і перевірці на наявність помилок відображається її проблема. Для цього і був створений TypeScript[2]. Angular[3,4] написаний на TypeScript тому є невід'ємною його частиною, а отже і сам TypeScript є невід'ємною частиною архітектури проекту.

Typescript володіє багатьма перевагами перед Javascript. Завдяки строгій типізації код, написаний на Typescript є більш передбачуваним і як правило простіший в налагодженні. Він спрощує організацію базового коду для дуже великих і складних програм завдяки модулям, просторам імен і потужній підтримці ООП. Компіляція в Typescript включає в себе етап переходу до Javascript, який виявляє всі помилки до того, як код, в якому є помилки виконається.

При створенні великомасштабних додатків, об'єктно-орієнтовний стиль програмування є більш привабливим для розробників, особливо в таких мовах програмування як C# або Java. TypeScript пропонує систему класів, яка дуже схожа на ту, яка використовується в цих мовах програмування, в тому числі і наслідування, абстрактні класи, реалізації інтерфейсу, сеттери, геттери та інше.

Generics – це шаблони, які дозволяють в одній і ті ж функції приймати аргументи різних типів. Краще створювати компоненти в Generics, ніж використовувати тип даних any, тому, що Generics зберігає типи даних, які проходять через нього.

Іншим важливим поняттям при роботі з великими проектами є модульність. Розділення коду на велику кількість малих, повторно використовуваних компонентів допомагає проекту залишатися організованим і зрозумілим, порівняно з файлом, що складається з тисячі рядків коду. Typescript вводить синтаксис для експорту і імпорту модулів, але не може опрацьовувати зв'язки між файлами.

Звичайно, на початку ознайомлення з TypeScript можна і не зрозуміти, чому варто використовувати його для розробки додатків. Реальне розуміння проблеми приходить тільки тоді коли в коді почнуть з'являтися нерозумні і серйозні помилки. Більше того після використання TypeScript код стає більш структурованим і самозадокументованим, що дуже спрощує розробку як самому програмісту, так і його колегам. Якщо «кодити» на тайпскрипті в професіональному середовищі розробки, то можна одразу ж побачити дуже круте автодоповнення, яке дуже сильно допомагає в розробці, а що найголовніше – це дуже впливає на час розробки. Просто ввівши першу букву методу, який хочеться викликати, одразу можна побачити його в списку рекомендованих методів для автозаповнення.

Працюючи з TypeScript, можна спостерігати дуже сильний прогрес в продуктивності. Розробницький досвід також збільшується. Дійсно, TypeScript – це дуже потужний інструмент, який допомагає створювати масштабовані, стійкі та надійні програми. TypeScript, як і JavaScript все більше розвається, але потрібно розуміти що TypeScript це просто доповнення JavaScript і він, скоріш за все, ніколи не замінить його.

2.5 Фреймворк Angular - каркас веб-додатку

Angular – це фронт-енд фреймворк, який був створений компанією Google, щоб полегшити процес створення сучасних веб-додатків. Можливості фронт-енд розробки значно покращилися за останні декілька років. Теперішні користувачі веб-додатків хочуть бачити красивий інтерфейс та отримувати чудовий «user experience», тобто з легкістю проводити навігацію по сайту. Фундамент Angular побудований навколо набору основних концепцій, які надають міцності особливостям фреймворку. Крім багатого інтерфейсу, Angular[3,4] пропонує різні способи структурування коду програми. Фреймворк призначений для

побудови додатків з використанням ряду блоків з кодом, розділених на окремі модулі. Такий підхід допомагає розробникам писати чистий та легкий у підтримці код.

Веб-додаток, розроблений за допомогою Angular можна розглядати як дерево компонентів. Такий додаток при запуску спочатку завантажує головний компонент, а потім решту інших компонентів, які вбудовані в головний. На рисунку 2.5 можна побачити поділ додатку на компоненти.

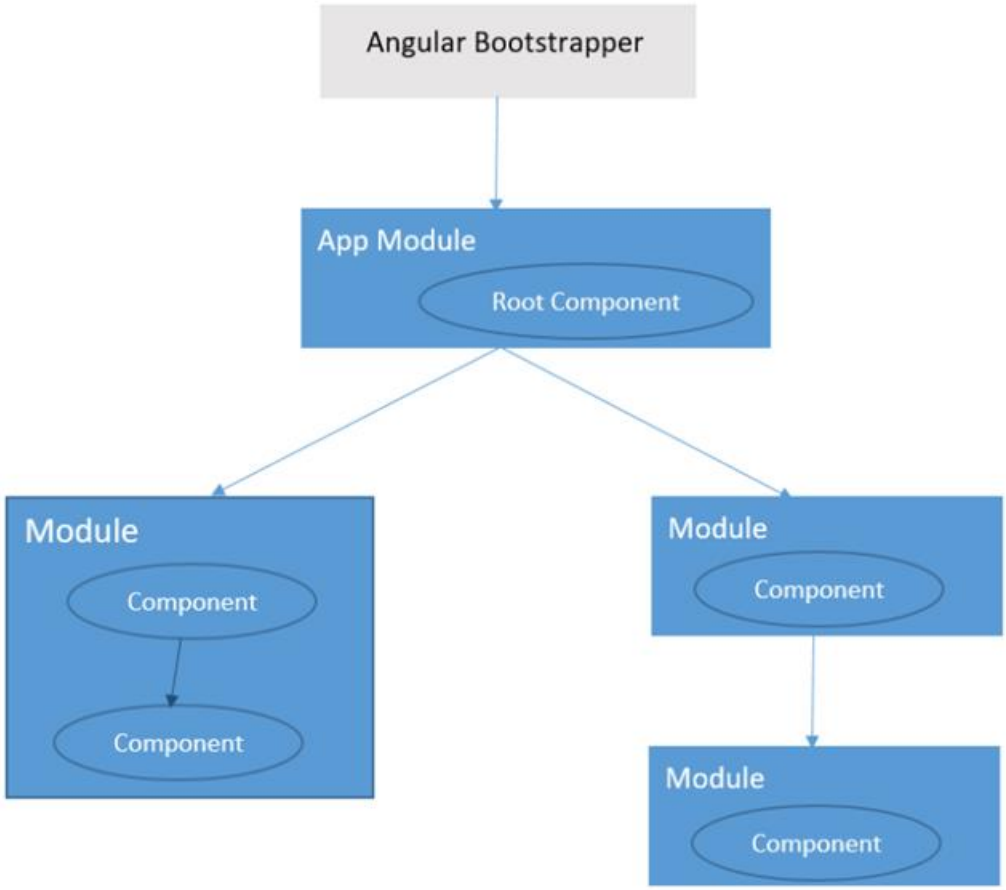

Рисунок 2.5 - Архітектура Angular додатка

Компонент у Angular вміщує функціонування HTML-шаблону, а також CSS-стилів, які використовуються в цьому шаблоні. Як видно на рисунку 2.5, кожен компонент живе в модулі і завантажується також з модуля. Компонент

використовує сервіси для отримання даних і представлення їх у шаблоні або для запуску частини логіки, яку можна повторно використовувати в додатку та не включати в DOM. Компоненти отримують об'єкти сервісів через спеціальну технологію, яка була розроблена для фреймворку та називається «Dependency injection». Директиви використовуються в шаблоні компонента з метою розширення поведінки елементів HTML. Дані в шаблоні використовують технологію прив'язки даних, тобто прив’язуються до даних, які знаходяться в файлі з розширенням «.ts». Будь-яка зміна даних обробляється системою виявлення змін, вбудованою в Angular, і інтерфейс користувача оновлюється для відображення останніх даних. Зони відстежують події, що відбуваються на рівні браузера, щоб почати виявляти зміни коли щось відбувається поза контекстом Angular. Ось так Angular фреймворк використовує свої основні концепції для обробки веб-додатків, побудованих на ньому. На рисунку 2.6 показано, як різні частини програми зв'язані між собою.

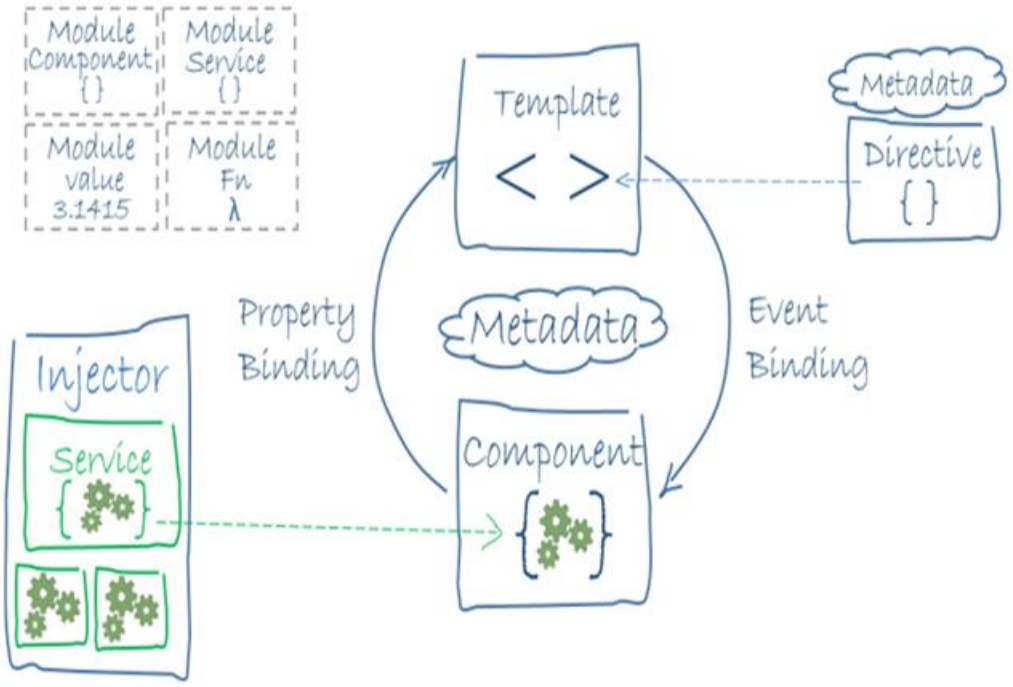

Рисунок 2.6 - Взаємодія частин Angular між собою

Директиви використовуються для розширення HTML, створюючи власні елементи HTML та розширюючи існуючі елементи. Фреймворк підтримує три типи директив. Директиви, що використовуються для створення користувацьких елементів, називаються компонентами. Директиви, які використовуються для розширення елемента HTML через новий атрибут, називаються атрибутними. А директиви, які взаємодіють з DOM та маніпулюють цільовим елементом, називаються структурними директивами. Компоненти створюються за допомогою декоратора @Component. Атрибутні та структурні директиви створюються за допомогою декоратора @Directive. Директиви не мають шаблону, оскільки вони діють, як доповнення до елемента.

Взагальному, компонент є ніби частиною тіла фреймворку. Додаток, створений за допомогою Angular складається з різних компонентів з різними обов'язками. Як показано на рисунку 2.5, додаток запускається з компонента, і цей компонент використовується для завантаження інших компонентів. Компонентом може бути що завгодно, починаючи з елемента, який запускає додаток, елемента, який завантажує вміст сторінки, елемента, який завантажує частину сторінки, або все, що може бути відокремлено в самостійний фрагмент сторінки. Компоненти Angular - це користувацькі елементи HTML, які роблять додаток більш декларативним, а шаблон програми - більш читабельним. Це пояснюється тим, що більшість компонентів мають імена, які можна легко читати. Якщо використовувати неправильні та нечитабельні імена для компонентів, то іншим розробникам буде важко зрозуміти який сенс несе такий компонент.

Отже, компонент можна розглядати як поєднання HTML, CSS та JavaScript, які працюють разом для виконання завдань. Крім того, що код стає більш читабельним, до компонентів також можна додавати власні стилі CSS. Стилі CSS, додані в компонент, залишаються ізольованими всередині компонента, і вони не впливають на інші компоненти, тобто кожен компонент ніби має свою власну таблицю стилів. Така особливість робить поведінку будь-

якого компонента більш передбачуваною та економить багато часу для розробників, оскільки їм не потрібно робити імена класів CSS унікальними. CSS стилі, які належать до певного компоненту будуть доступні тільки в межах цього компоненту.

Angular використовує Shadow DOM або емулятор, який залежить від метаданих компонента, щоб ізолювати стилі компонента і робити їх доступними тільки в межах цього компонента. Для того щоб зробити стилі загальними для всього додатку, можна використати селектор ng-deep.

Компонент визначається в Angular за допомогою класу TypeScript з декоратором компонента (@Component). Для того, щоб вставити один компонент в інший потрібно записати ім'я класу компонента у вигляді тега в шаблоні. Усі поля та методи класу компонента також доступні в шаблоні.

Dependency Injection (DI) - це механізм, який вирішує проблему обробки залежностей, необхідних у блоці з кодом. Цей механізм говорить про те, що залежності, необхідні компоненту, повинні бути створені зовнішнім агентом, і вони повинні бути введені в конструктор класу компонента. Оскільки логіка створення об'єктів зберігається в одному місці, це зменшує можливість повторення коду. Код, що використовує введену залежність, не знає, як створюється об'єкт, єдине, що він знає - це структура об'єкта. Це означає, що об’єкт можна легко замінити будь-яким іншим об’єктом, який має подібну структуру. Ми можемо створити інтерфейс TypeScript для об’єкта і мати різні реалізації інтерфейсу для різних середовищ. Скажімо, наприклад, компонент використовує об'єкт для реєстрації повідомлень на сервері. Якщо код ще розробляється, і розробник не хоче розміщувати непотрібні повідомлення на сервері, об’єкт, відповідальний за реєстрацію повідомлення, може бути легко замінений об’єктом, який записує повідомлення в консоль браузера методами подібних підписів. DI досить популярний серед мов програмування на стороні сервера, таких як C # і Java. AngularJS приніс цю функцію в JavaScript, а Angular

реалізовує її вдосконалено. Angular використовує DI для введення залежностей в компоненти, директиви, пайпи і навіть у сервіси. Сервіс у Angular - це звичайний клас ES6, який можна використовувати для виконання таких завдань, як взаємодія з HTTP API, бізнес-розрахунки або все, що не залежить від DOM. Взагалі, сервіси не залежать від того, де вони будуть використовуватися, тому логіка сервісу може використовуватися в будь-якому місці програми, де є необхідність такої логіки. Декоратор «injectable» дозволяє робити DI в інші сервіси.

Модулі в Angular відіграють дуже важливу роль. Кожен веб-додаток, створений за допомогою цього фреймворку, має як мінімум один модуль, який називається кореневим модулем. Модулі використовуються для групування компонентів, директив чи сервісів. Можна сказати, що модуль – це набір блоків, які відповідають за певний функціонал. Модуль також можна імпортувати в інший модуль. В Angular прийнято розбивати програму на модулі, для того, щоб відокремити різні за функціоналом частини додатку. Модуль – це тайпскрипт клас, до якого застосовано декоратор @NgModule. Також дуже важливою характеристикою модулів є те, що вони можуть бути ліниво загружені, тобто будуть завантажуватися не на старті програми, а тоді коли в адресній строці з’явиться адреса, до якої прив’язаний модуль.

Отже, можна дійти до висновку, що Angular – це дуже потужний інструмент в розробці веб-додатків. Його компонентний підхід та дуже велике різноманіття функціоналу дуже спрощує та пришвидшує розробку. Саме тому його було вибрано в якості фреймворка для створення дипломного проекту.

2.6 Сутності додатку та взаємодія в системі

Для того, щоб змоделювати діаграму сутностей, було використано UML[10]. UML – це абревіатура, яка розшифровується, як «Уніфікована мова моделювання». Просто кажучи, UML – це сучасний підхід до моделювання та документування програмного забезпечення. Це один з найпопулярніших видів бізнес-процесів. Він базується на схематичному зображенні компонентів програмного забезпечення. Тому використовуючи UML ми можемо краще зрозуміти та побачити важливі недоліки з програмної сторони чи сторони бізнес процесу. В основному UML використовується як мова моделювання в сфері інженерії програмного забезпечення, але зараз вона використовується також і в багатьох інших сферах. В UML є декілька типів діаграм:

- діаграма класів;
- діаграма кооперацій;
- діаграма сутностей;
- діаграма компонентів;
- діаграма композитів;
- діаграма розгортання;
- діаграма пакетів.

На рисунку 2.7 зображено діаграму сутностей проекту.

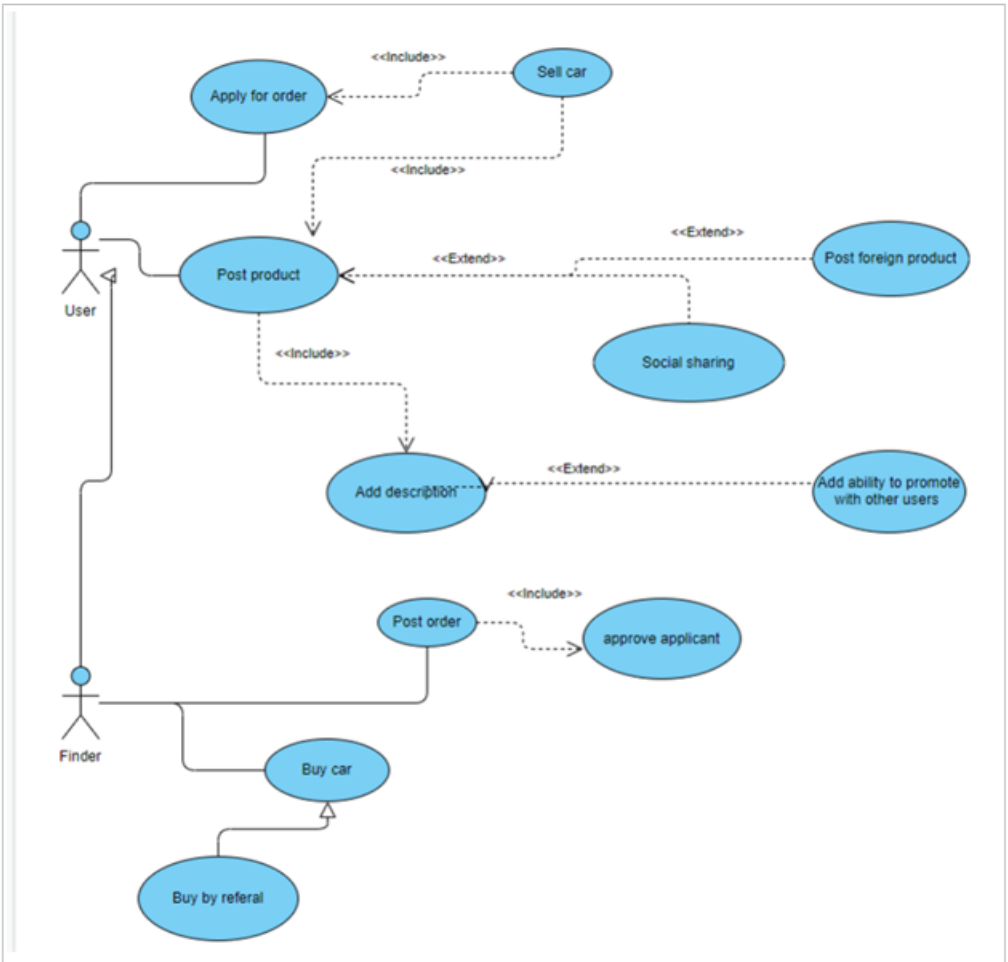

Рисунок 2.7 - Діаграма сутностей

Діаграма сутностей – це діаграма поведінки, яка відображає взаємодію між акторами та системою. Діаграма складається з системи, різних сценаріїв та діючих осіб і пов'язує їх між собою. Така діаграма не описує порядок здійснення сценаріїв.

Отже на рисунку 2.7 можна побачити діаграму сутностей, яку було розроблено для проекту, щоб візуалізувати бізнес-процеси, які будуть присутні у додатку. На діаграмі можна побачити фігури у вигляді «чоловічків». Ці фігури являються акторами в даній діаграмі. Актори у UML – це користувачі, які взаємодіють з системою. Актором може бути особа, організація, або взагалі зовнішня чи внутрішня система, яка взаємодіє з даною системою. На діаграмі видно, що в проекті є два типи акторів: користувач (User) та користувач який

бажає замовити автомобіль (Finder). По факту User та Finder це один і той же самий користувач, але для того щоб наочніше побачити функції, які може виконувати користувач, його було розділено на два типи. На рисунку 2.8 зображено два типи користувачів.

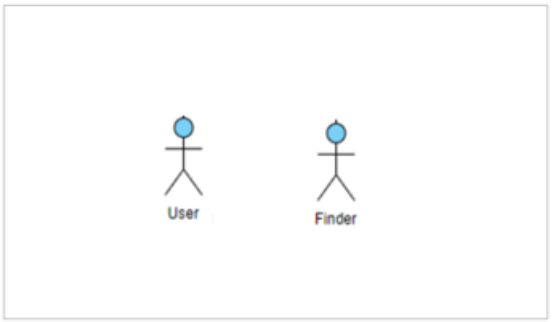

Рисунок 2.8 - Типи користувачів у системі

Оскільки у системі є користувачі, то значить вони можуть якось взаємодіяти з нею та робити якісь певні дії. Такі дії у UML ще називаються прецедентами. За допомогою прецедентів можна змоделювати взаємодію між актором та системою і вони визначають можливості, які надає система для користувача. Набір всіх прецедентів в системі і визначає те, як буде використовуватися така система. На рисунку 2.9 зображено один з прецедентів системи.

Рисунок 2.9 - Прецедент системи

Отже в системі для користувача (User) доступні такі дії: Зробити пост автомобіля, який він бажає продати, подати заявку на те, що готовий виконати замовлення автомобіля, в якій звичайно ж повинні бути дані про автомобіль, який користувач бажає замовити, і ще одна можливість – це купити автомобіль. Для того, щоб зробити пост автомобіля для продажу, користувачу потрібно буде заповнити форму, де буде вказано всі дані про його автомобіль, а саме: рік випуску, об'єм двигуна, тип коробки передач, привід (передній, задній чи повний), колір автомобіля, і звичайно ж марку та модель. Після того як такі дії будуть виконані, то пост з цим автомобілем з'явиться у розділі з автомобілями, які виставлено на продаж. В цьому розділі користувач також зможе побачити інші автомобілі, які продаються. Можливо серед цих автомобілів буде такий, який користувач хотів би купити. Для цього йому потрібно буде перейти на строінку цього автомобіля, та зв'язатися з його власником. Дані про власника будуть доступні на цій сторінці. Ще однією можливістю для користувача в системі буде замовлення автомобіля. Можуть бути такі випадки коли людина, зайшовши на сайт з метою купити авто, не знайде автомобіль бажаної комплектації. Бувають коли, людина хоче купити автомобіль, але їй не потрібні всі можливості комплектації такого автомобіля. Для прикладу, користувач вибрав для себе автомобіль та захотів його купити. В комплектації такого автомобіля присутній кондиціонер, і від цього він коштує дорожче ніж аналоги без кондиціонера. Користувач не готовий переплачувати за кондиціонер, тому

що він йому не потрібний. Але виявляється що в списку автомобілів для продажу є тільки такий автомобіль але з кондиціонером. Тоді виникає питання: що робити? Не проблема, адже в користувача буде можливість замовити автомобіль який він бажає. Все що йому для цього потрібно буде зробити – це заповнити форму, де вказати всі характеристики автомобіля, які повинні бути присутні в ньому. Після цього оголошення про замовлення буде доступне в списку замовлень на сайті. Цей список будуть бачити всі користувачі, а отже будь-який з них може відгукнутися на це замовлення і запропонувати свої послуги та відповідно плату за них. Після того як угода буде укладена, користувачу залишиться тільки чекати своє замовлення. Після здійснення замовлення, щасливий користувач залишить позитивну оцінку тому користувачу, який здійснив його замовлення. Чим більший рейтинг в користувача, який надає послуги з замовлення авто, тим більше людей будуть в нього замовляти автомобіль, а отже дохід такого користувача буде рости. Оскільки дуже багато людей в Україні займається таким бізнесом, то такий сайт буде чудовим рішенням для продовження свого бізнесу в ньому. Не потрібно буде шукати клієнтів власноруч, обдзвонюючи знайомих, чи розміщувати рекламу в соцмережах і витрачати на це кошти. Можливість користувача замовити авто зображена на рисунку 2.10.

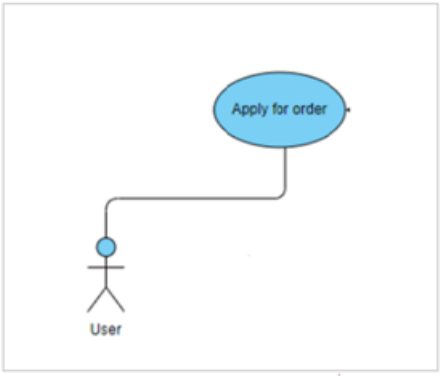

Рисунок 2.10 - Функція замовлення авто

Можливий такий варіант, що користувач, який займається продажем автомобіля, захоче продати автомобіль, який не зареєстрований в Україні. Це також не проблема, адже користувач може викласти автомобіль для продажу, вказавши його характеристики та фото, на якому буде зображено номер автомобіля, який встановить той факт, що такий автомобіль знаходиться за кордоном. Як правило автомобілі закордоном значно дешевші ніж аналогічні автомобілі в Україні. В нашій країні зараз дуже популярний ввіз авто з-за кордону. І дуже багато людей займається таким бізнесом. Дороги закордоном значно виграють у якості українським дорогам і це означає, що автомобілі там набагато кращої якості ніж в нас. Одна проблема, що такі автомобілі, як правило, розміщенні на веб-сайтах компаній чи майданчиків, які їх продають. Користувач в системі зможе зробити пост такого автомобіля. Це буде означати що такий користувач зможе зробити все для того, щоб привезти такий автомобіль на територію України, розмитнити його та зробити всі інші необхідні операції. При цьому користувачу який замовить таке авто не потрібно буде нічого робити власноруч, адже за нього все зроблять самі. І в результаті він отримає готовий автомобіль.

Ще однією важливою функцією у користувача буде можливість зробити шеринг автомобілів інших користувачів. Тобто будь-який користувач зможе зайти на сторінку з постом автомобіля іншого користувача і поділитися цим постом вже в себе в профілі. Це можна назвати невеликою рекламою. Оскільки безплатних реклам небуває, то користувач, який поділився таким постом, зможе отримати нагороду від того, що автомобіль був проданий за його посиланням. Тобто таку функцію можна назвати реферальною програмою. В програмі можна буде відслідковувати чи був автомобіль зашарений за допомогою токена. Функція шерингу зображена на рисунку 2.11.

Рисунок 2.11 - Шеринг автомобіля

Отже, функції, описані вище і будуть доступні користувачу User. Для користувача Finder будуть доступні такі ж функції, як і для користувача User, оскільки User та Finder – це, по суті, один і той же користувач. Можна розглядати користувача Finder, як користувача, який зайшов на сайт з метою купити автомобіль, тому в нього будуть також такі функції, як замовити автомобіль, прийняти замовлення і відповідно після цього купити цей автомобіль. Можливості користувача Finder зображені на рисунку 2.12.

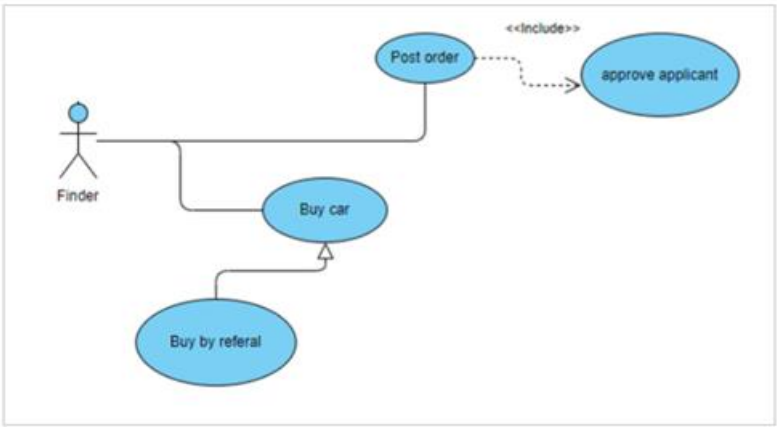

Рисунок 2.12 - Функції користувача Finder

2.7 Дизайн додатку

Макет додатку було спроектовано за допомогою сервісу «Moqups». Цей сервіс можна назвати конструктором для сайтів, адже за його допомоги можна легко зробити прототип майбутнього веб-додатку. В ньому є дуже багато готових компонентів, що дозволяє суттєво зекономити час при проектуванні. Прототипи це дуже важлива частина створення сайту, адже за допомоги прототипів можна швидко накидати дизайн для сайту і почати розробляти його. На прототипах зображуються основні елементи сайту у вигляді схеми.

Для кожної сторіки додатку було вирішено створити прототип, оскільки це суттєво збільшить швидкість розробки. В додатку будуть такі сторінки: домашня сторінка, сторінка зі списком продуктів(автомобілів), сторінка зі списком всіх замовлень, сторінка поста з авто, сторінка поста з авто для замовлення, сторінка з формою для подання оголошення про продаж авто, сторінка з формою для подання оголошення про замовлення авто, а також сторінка користувача.

В хедері будуть розміщені такі CTA кнопки, як «Купити авто», «Продати авто», «Замовити авто», а також кнопка логіну, і логотип.

На головній сторінці буде фільтр, за допомогою якого можна буде ввести дані про потрібний автомобіль, і перейти на сторінку з автомобілями, які підходять під критерій. Також буде список з автомобілями, які були недавно додані на сайт. Прототип головної сторінки можна побачити на рисунку 2.13.

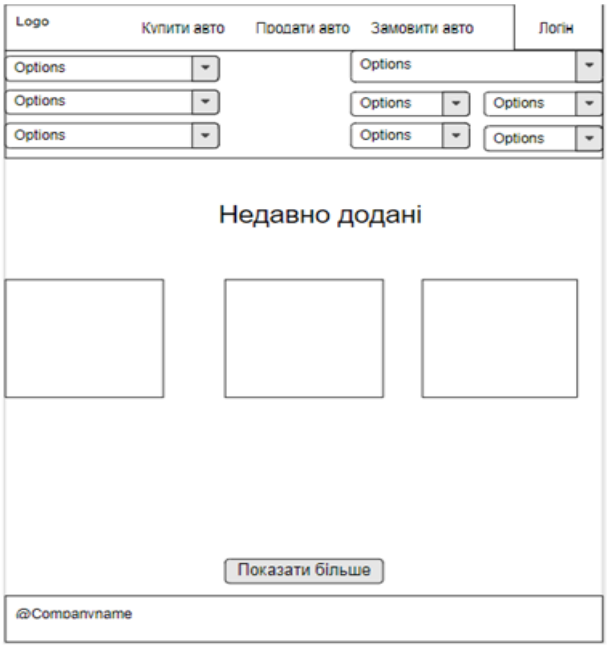

Рисунок 2.13 - Прототип головної сторінки

На сторінці зі списком автомобілів буде присутній фільтр, за допомогою якого можна буде відфільтрувати всі автомобілі за критерієм, а також сам список автомобілів. На рисунку 2.14 зображено прототип сторінки зі списком автомобілів для продажу.

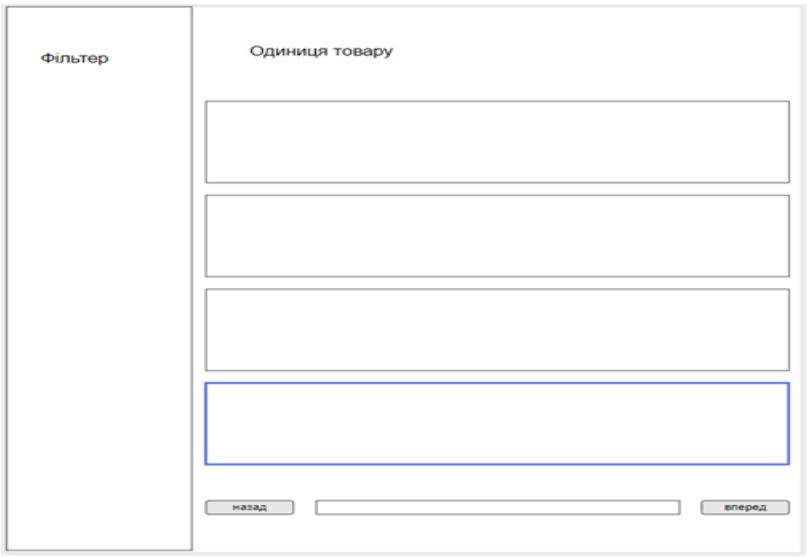

Рисунок 2.14 - Прототип сторінки зі списком автомобілів для продажу

На сторінці списку автомобілів які було замовлено, буде список цих автомобілів. На рисунку 2.15 зображено прототип сторінки зі списком автомобілів, які було замовлено.

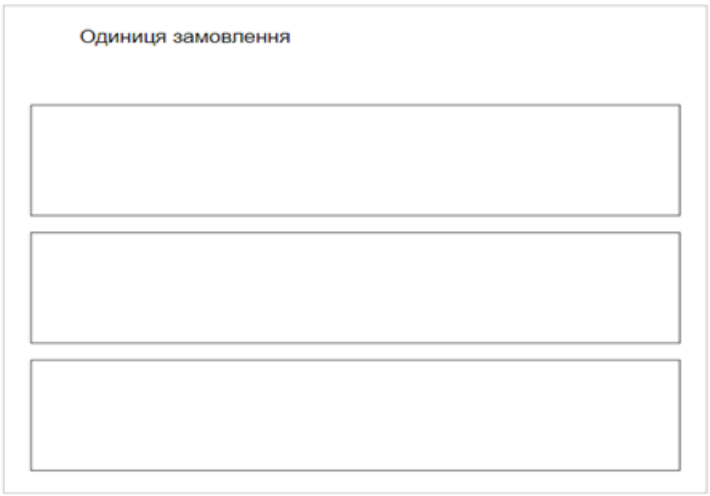

Рисунок 2.15 - Прототип сторінки зі списком автомобілів, які було замовлено

На сторінці Автомобіля, який було виставлено на продаж буде інформація про цей автомобіль, декілька фото автомобіля, його опис, а також деталі власника цього автомобіля. Прототип сторінки автомобіля, який було виставлено на продаж зображено на рисунку 2.16.

Рисунок 2.16 - Прототип сторінки автомобіля для продажу

На сторінці замовленого авто буде інформація про автомобіль, ціна, опис, додаткова інформація, інформація про продавця, а також список користувачів, які готові виконати замовлення. На рисунку 2.17 зображено прототип сторінки замовленого авто.

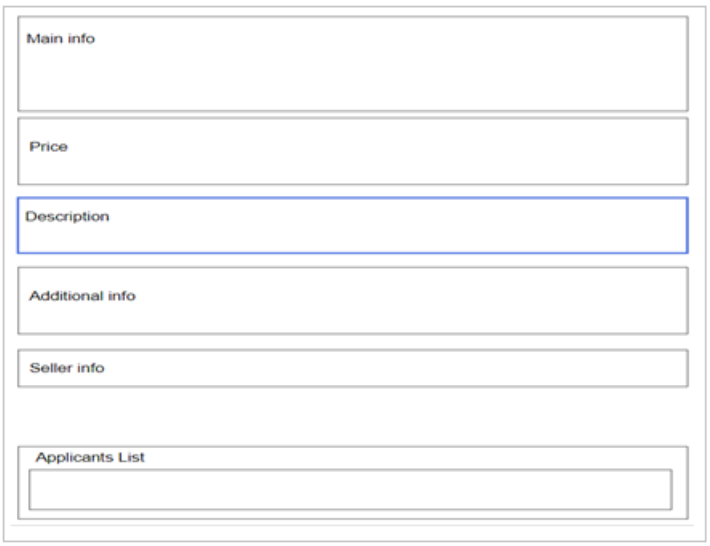

Рисунок 2.17 - Прототип сторінки замовленого авто

На сторінках замовлення авто та продажу авто буде форма, в якій потрібно буде вказати такі дані про автомобіль, як основна інформація, ціна, опис, додаткова інформація, а також потрібно буде додати фото автомобіля, але тільки в формі для продажу авто. На рисунку 2.18 зображено прототипи сторінок для замовлення авто та продажу авто.

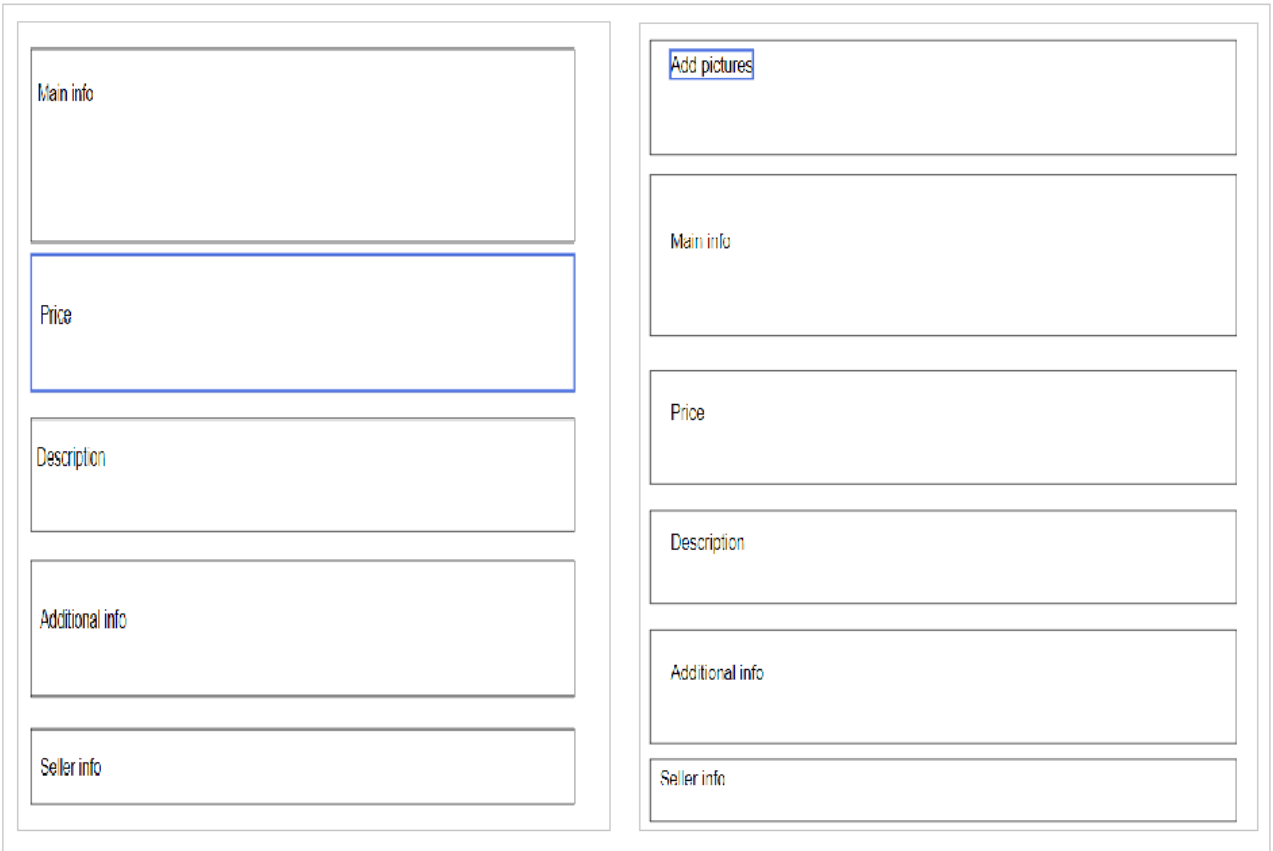

Рисунок 2.18 - Прототипи сторінок для замовлення та продажу авто

На сторінці користувача будуть відображатися основні дані про користувача такі, як ім'я, електронна пошта, локація, номер телефону а також буде присутнє фото користувача та список його товарів. На рисунку 2.19 зображено прототип сторінки користувача.

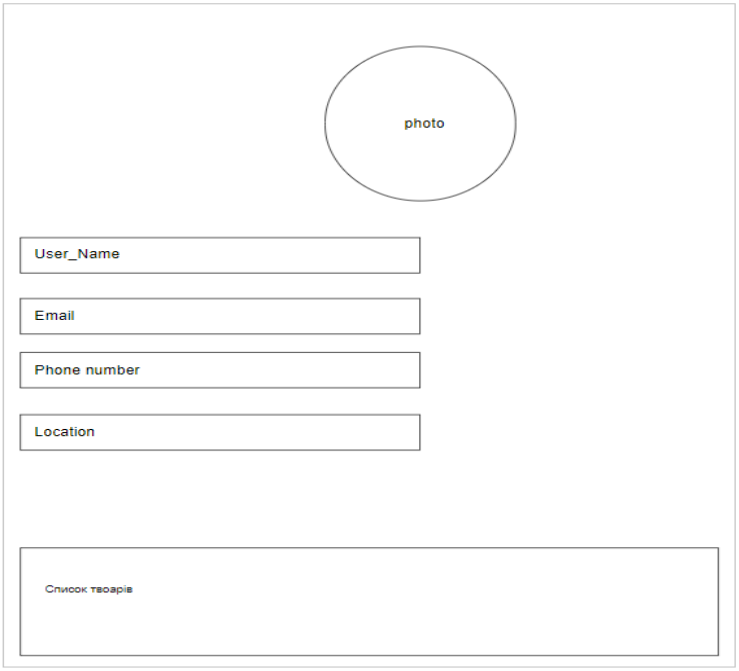

Рисунок 2.19 - Прототип сторінки користувача

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Генерація та настройка проекту

Для розробки веб-додатку було використано JavaScript фреймворк Angular. У цього фреймворка є дуже крутий інструмент, який називається «Angular CLI tool». Такий інструмент можна встановити для того, щоб комфортно працювати з Angular. Перед тим, як використовувати цей інструмент, спочатку потрібно встановити пакетний менеджер для Javascript, який називається «npm». Для того, щоб встановити npm, потрібно у командному рядку для Windows, або в терміналі Git Bash ввести команду «npm install». Після того як було встановленно пакетний менеджер npm, можна переконатися в тому, що він дійсно був встановлений за допомогою команди «npm --version». На рисунку 3.1 зображено успішну установку пакетного менеджера, виведенням в консоль його версії.

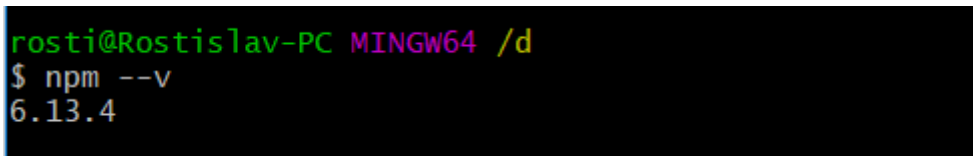

Рисунок 3.1 - Успішне встановлення npm

Після того, як було встановлено npm, потрібно до нього звернутися та установити інший пакет, який називається «@angular/cli». Для цього в термінал потрібно ввести команду «npm install –g @angular/cli», дочекатися, поки npm завантажить цей пакет, розархівує його і в результаті цей пакет буде доступний в терміналі. Після цього можна створити новий проект. Для того, щоб створити новий проект, можна звернутися до інструмента Angular CLI і прописати команду «ng help». Після виконання цієї команди ми побачимо, великий список

команд, які підтримує CLI: різноманітні добавлення бібліотек, аналітика, білд команди, і найважливіша команда на даний момент – це команда «new», яка дозволяє створити новий робочий простір «workspace» та проініціалізувати новий Angular додаток. Список з доступними командами Angular CLI зображено на рисунку 3.2.

```
rosti@Rostislav-PC MINGW64 /d
$ ng help
Available Commands:
  add Adds support for an external library to your project.
  analytics Configures the gathering of Angular CLI usage metrics. See https://v
8.angular.io/cli/usage-analytics-gathering.
  build (b) Compiles an Angular app into an output directory named dist/ at the
given output path. Must be executed from within a workspace directory.
  deploy Invokes the deploy builder for a specified project or for the default p
roject in the workspace.
  config Retrieves or sets Angular configuration values in the angular.json file
 for the workspace.
  doc (d) Opens the official Angular documentation (angular.io) in a browser, an
d searches for a given keyword.
  e2e (e) Builds and serves an Angular app, then runs end-to-end tests using Pro
tractor.
  generate (g) Generates and/or modifies files based on a schematic.
  help Lists available commands and their short descriptions.
  lint (l) Runs linting tools on Angular app code in a given project folder.
  new (n) Creates a new workspace and an initial Angular app.
  run Runs an Architect target with an optional custom builder configuration def
ined in your project.
  serve (s) Builds and serves your app, rebuilding on file changes.
  test (t) Runs unit tests in a project.
  update Updates your application and its dependencies. See https://update.angul
ar.io/
  version (v) Outputs Angular CLI version.
  xi18n (i18n-extract) Extracts i18n messages from source code.
```

Рисунок 3.2 - Доступні команди в Angular CLI

Для того, щоб створити новий проект, потрібно ввести команду «ng new» та додати назву проекту. Оскільки проект називається «JoinAuto», то потрібно зробити «ng new JoinAuto». Після того, як натиснути клавішу «Enter», можна побачити, ще додаткові кроки, які пропонує Angular CLI, для того щоб згенерувати новий проект. В першу чергу він запитує чи хочемо ми додати роутинг для додатку, та який CSS препроцесор буде використовуватися чи

просто стилі будуть записуватися у вигляді звичайного CSS. Оскільки роутинг в додатку звичайно ж буде присутній, то його було вибрано. Також у якості CSS препроцесора був овибрано SCSS, оскільки цей препроцесор значно прискорює написання CSS-стилів.

Angular CLI згенерував велику кількість різних файлів, а також завантажив Node Modules, тобто набір бібліотек, які дозволять працювати додатку. Після запуску команди npm start проект буде запущено і це дозволить переконатися в тому, що додаток було успішно згенеровано.

Отже, після цього вже можна побачити, що було отримано в якості проекта. Середовищем для розробки проекту було вибрано «Webstorm». Webstorm володіє багатьма перевагами, а саме:

- виконання підсвітки і автодоповнення коду;
- перевірка коду на помиллки;
- швидка навігація по проекту, та автоматичне збереження внесених змін;
- багато корисних плагінів;
- можливість використання GIT прямо в консолі;
- збереження локальної історії;
- підтримка Angular CLI (один з найбільших плюсів).

За допомогою Webstorm розробляти стає набагато комфортніше, а також він допомагає запобіганню великої кількості помилок, і робить розробку набагато швидшою.

Після створення проекту, було отримано структуру, яку зображено на рисунку 3.3.

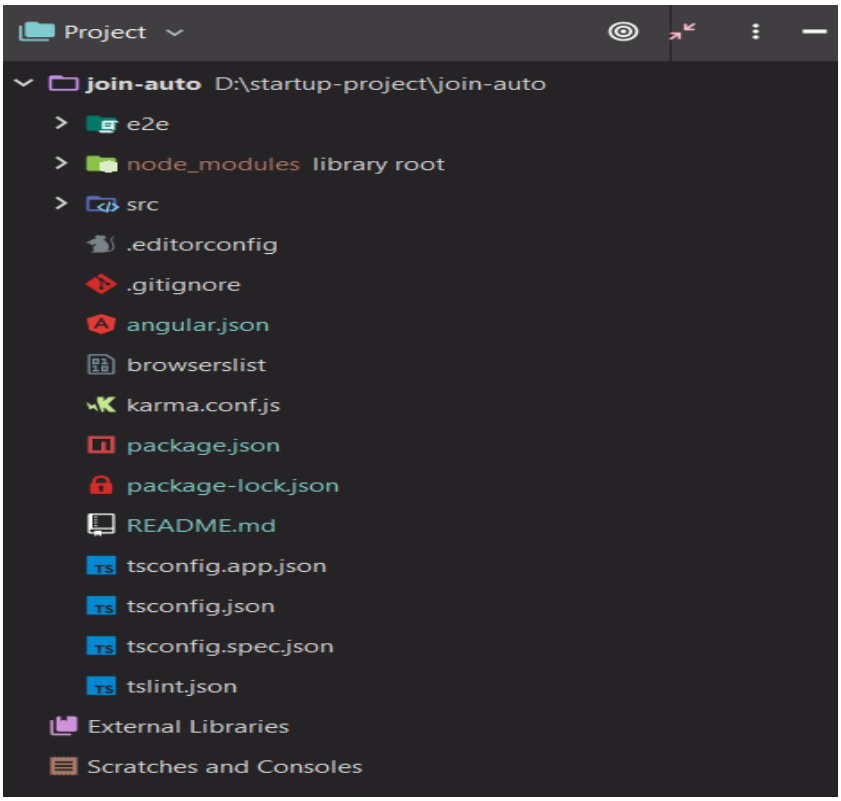

Рисунок 3.3 - Структура проекту

В структурі проекту присутній ряд папок та файлів. Перша папка в списку називається «e2e» і перекладається як «end to end», яка відноситься до тестування, проте тестування не було використано при розробці проекту. Наступна папка це «node_modules» і в ній зберігається вихідний код зі всіх бібліотек залежностей, які потрібні для роботи Angular, тобто вона генерується автоматично. В проекті також присутній ряд файлів. Перший файл – це «.editorconfig», який задає якусь певну стилістику для редактора, наприклад якесь загальне кодування, стиль відступу, розмір табу і так дальше. Наступний файл – це «.gitignore» і він слугує для того, щоб приховувати якісь певні файли чи папки від системи контролю версій «GIT». Для прикладу він приховує папку «node_modules» та папку «idea», які не потрібно буде додавати у GIT. В файлі «angular.json» зберігаються всі налаштування для додатку. Він формується автоматично, але ми також можемо його змінити, якщо в цьому буде потреба. В ньому написано, наприклад, що ми використовуємо SCSS стилі, деякі шляхи:

шлях до index.html файлу, до головного файлу, який запускає додаток, до файлу з поліфілами, а також до файлу з різнмим статичними елементами і статичними стилями. Також в ньому ми можемо підключати скрипти. В файлі «browserslist» описано деякі вимоги для роботи додатку. Файл «package.json» - це файл, для того, щоб описувати додаток та дивитися різні команди. В ньому написана версія додатку, а також присутнє дуже цікаве поле «scripts», в якому є деякі скрипти, які дозволяють керувати додатком, а саме «ng serve» - команда, яка дозволяє запускати додаток, за допомогою команди «npm build» можна білдити (збирати) додаток, а також тестувати за допомогою команди «ng test». В полі «dependencies» перераховані залежності, які потрібні іменно для самого додатку. По замовчуванню Angular установлює ті пакети, які дозволяють комфортно працювати з додатком, а саме пакет з анімаціями, пакет з якимось загальними речами, в якому, наприклад знаходиться пакет HTTP[11], пакет з компілятором, формами, а також пакет для роботи з браузером. Також в файлі присутній список з пакетами, які потрібні на стадії розробки, в тому числі є поле «typescript». Файл «package-lock.json» - це системний файл, який зберігає версії пакетів, для легшого керування ними. В файлі «tsconfig.app.json» ми можемо вказувати різноманітні опції для typescript і цей файл наслідується від файлу «tsconfig.json», в якому присутні якісь базові настройки. Останній файл в списку – це «tslint.json», в якому можна вказувати різні правила для стилістики коду.

Найважливіша частина додатку – це папка «src», в якій зберігається весь вихідний код проекту. В цій папці розміщено ще декілька папок, серед яких «app», «assets» та «environments». В папці «assets» зберігається деяка статика та присутній файл «.gitkeep», для того, щоб система контролю версій «GIT» не видаляла порожні папки. В папці «environments» зберігаються файли, в яких можна тримати деякі конфігураційні значення. Файл «favicon.ico» - це іконка по замовчуванню. Для проекту було вибрано іконку, яку зображено на рисунку 3.4.

Рисунок 3.4 - Іконка проекту

Файл «index.html» - це той самий єдиний файл index.html у Single Page Application, тобто якраз-таки за допомогою Angular ми і створюємо його. Головним файлом у проекті є «main.ts», тобто це файл, який запускає додаток. У файлі «polyfills.ts» зберігаються підказки, як користуватися поліфілами. Для того, щоб можна було додавати глобальні стилі для всього додатку, в папці «src» є файл, з назвою «styles.scss». Тобто тут можна писати глобальні стилі, які не відносяться до стилів локальних компонентів. Останнім файлом у папці є «test.ts», який слугує для unit-тестів.

Найцікавішою зі всіх папок є папка «app», в якій зберігаються дві сутності: «app.module.ts» і «app.component.ts» та різні його варіації. Angular дотримується певного стайл-гайду в назвах файлів, для того, щоб цим було легше керувати. Ідея полягає в тому, що на початку ми пишемо назву елемента чи сутності і потім через крапку ми пишемо, який тип у цеї сутності тобто якщо сутність – компонент, то потрібно писати назву сутності і додавати слово «component», якщо модуль то «module». Будь який компонент складається з чотирьох елементів: html-елемент, scss-елемент, тобто елемент зі стилями, елемент з тестами і, звичайно ж, сам файл в якому знаходиться логіка даного компонента. В модулі структура зовсім інша і відрізняється від структури компонента.

Головним файлом в додатку є файл «main.ts», з якого і все починається. В ньому присутній дуже важливий метод, який називається «platformBrowserDynamic». На початку ми запускаємо цей метод, який повертає іншу функцію, яка називається «bootstrapModule»[13]. В цю функцію ми передаємо головний модуль додатку – AppModule. Тобто ми запускаємо додаток і запускаємо певний модуль. Після цього нам повертається проміс, тобто обіцянка і якщо є якісь помилки, то вони виводяться в консоль. Тобто по суті ми запускаємо модуль. В модулі AppModule присутній масив «declarations», в якому ми реєструємо AppComponent. Також в модулі присутнє поле «bootstrap», яке також являється масивом і тут ми вказуємо, який компонент є головним в додатку, тобто той компонент, який потрібно запускати. Оскільки в файлі «main.ts» ми запускаємо потрібний нам модуль, то цей модуль вже знає, який компонент нам потрібно запускати. Тобто він його реєструє і в ньому присутній селектор «app-root» і потім в файлі «index.html» ми просто записуємо його як html-тег і Angular його запускає.

3.2 Створення головного layout

Для того, щоб додати загальні стилі для додатку, було використано файл «styles.sccs». В ньому було задано деякі нативні css-змінні, які навіть не відносяться до scss, а також набір певних стилів, які потрібні для розробки додатку. Основним шрифтом для додатку було вибрано шрифт «Fira Code», а в якості другорядного шрифта було вибрано шрифт «Scada». Для того, щоб використати ці шрифти їх потрібно було імпортувати з бібліотеки шрифтів «Google fonts». Для того, щоб імпортувати шрифти в Angular, було використано правило «@import» та вказано шлях до шрифта. Після цього за допомогою селектора css «*» шрифт «Fira Code» було додано для всього додатку. В якості

альтернативного шрифта було використано шрифт «monospace». Основні кольори в додатку такі:

- rose: #ff225d;
- brown: #323232;
- grey: #707070;
- dark: #333;
- bright: #f7f7f7;
- dark-green: #043136.

Фрагмент зі загальними стилями, та записом змінних для кольору зображено на рисунку 3.5.

```scss
/* You can add global styles to this file, and also import other style files */
@import url('https://fonts.googleapis.com/css2?family=Fira+Code:wght@300;400;500;600;700&display=swap');
@import url('https://fonts.googleapis.com/css2?family=Indie+Flower&family=Josefin+Sans:wght@700&display=swap');
@import url('https://fonts.googleapis.com/css2?family=Scada:wght@700&display=swap');
@import '~ngx-owl-carousel-o/lib/styles/scss/owl.carousel';
@import '~ngx-owl-carousel-o/lib/styles/scss/owl.theme.default';
@import "~@ng-select/ng-select/themes/default.theme.css";

:root {
  --rose: #ff225d;
  --brown: #323232;
  --grey: #707070;
  --dark: #333;
  --bright: #f7f7f7;
  --dark-green: #043136;
}

* {
  font-family: 'Fira Code', monospace;
}

body {
  background-color: var(--bright) !important;

  p, h1, h2, h3, h4, span {
    margin: 0;
  }
```

Рисунок 3.5 - Фрагмент з загальними стилями для додатку

Також до загальних стилів було вирішено винести стилі для кнопок та інпутів, оскільки такі стилі будуть використовуватися в різних місцях.

Для того, щоб створити перший компонент в додатку – layout-component, було використано Angular CLI та команду «ng generate component layout». Але перед тим як запустити цю команду, було створено папку «layouts», в якій будуть зберігатися всі можливі лейаути для додатку. Отже після створення цієї папки та переходу в неї, за допомогою команди «ng generate component layout» було створено іншу папку, в якій згенерувалися всі сутності, які потрібні для роботи компонента Layout. Єдине що Angular CLI не згенерував – це сутність модуля. Для того, щоб згенерувати LayoutModule, було виконано команду «ng generate module layout». Після виконання цієї команди було згенеровано модуль лейаута, в який можна буде імпортувати інші модулі додатку.

Головний лейаут додатку було розділено на два інших лейаути: лейаут для десктоп-версії та лейаут для мобільної версії. В лейауті для десктопної версії є хедер та футер, а в мобільної версії ще й меню зліва, яке буде відкриватися по натисканню на кнопку відкриття цього меню. Хедер та футер для головного лейаута було вирішено винести в окремі компоненти, які було додано в масив «declarations» в LayoutModule. Якщо не додати ці компоненти в масив «declarations» модуля, то вони не будуть доступні в межах цього модуля і ми не зможемо використати ці компоненти у лейауті. Розділення хедера та футера на компоненти дозволяє спростити структуру лейаута і можна буде писати стилі для компонентів хедера та футера окремо.

В хедері були додані такі кнопки: «Купити авто», «Продати авто» та «Замовити авто». Також справа додано кнопку «Увійти», при натисканні на яку, користувач перейде на сторінку логіна та зможе увійти у свій обліковий запис. Зліва знаходиться логотип проекту, який водночас є кнопкою, при натисканні на яку, користувач перейде на головну сторінку додатку. При натисканні на кнопку «Купити авто», користувач перейде на сторінку зі списком всіх автомобілів, які

доступні для продажу. Якщо користувач натисне на кнопку «Продати авто», то перейде на сторінку, на якій потрібно буде заповнити форму для продажу авто. І якщо користувач натисне на кнопку «Замовити авто», то перейде на сторінку замовлення авто, на якій також потрібно буде заповнити форму і вказати всі дані про авто, яке він бажає замовити. Дизайн хедера зображено на рисунку 3.6.

Рисунок 3.6 - Дизайн хедера

Для кращого візуального сприйняття кнопкам було додано ховер-ефект, тобто при наведенні на кнопку, вона буде несильно збільшуватися в розмірі.

Дизайн хедера для мобільної версії додатку трохи відрізняється, а саме в ньому немає трьох головних кнопок: «Купити авто», «Продати авто» та «Замовити авто». Ці кнопки були перенесені в меню зліва. Також в меню зліва було перенесено кнопку «Увійти». Було створено кнопку меню, яка з'являється тільки тоді, коли вхід в веб-додаток було здійснено з мобільного телефона або планшета. На рисунку 3.7 зображено дизайн меню для мобільних пристроїв.

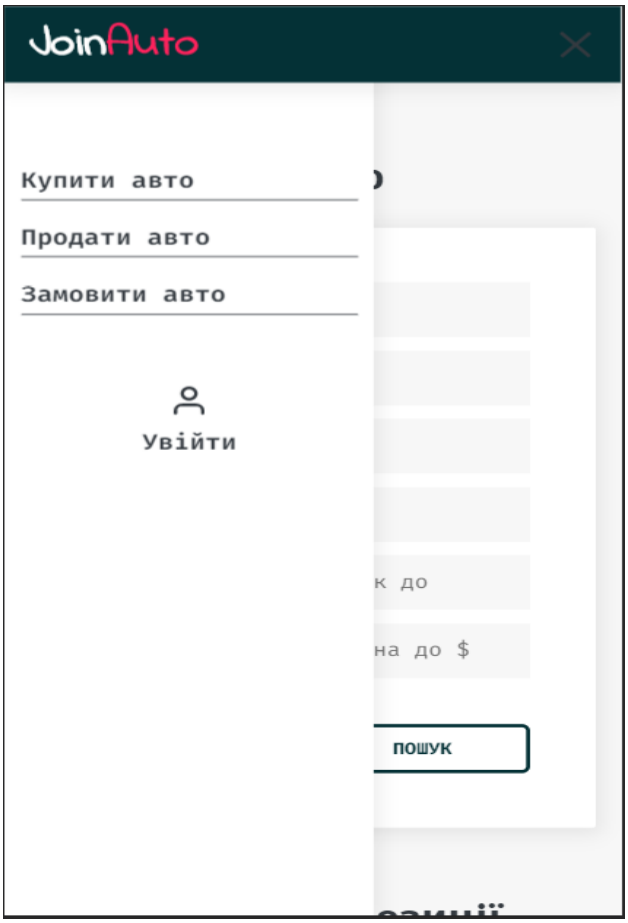

Рисунок 3.7 - Дизайн меню для мобільних пристроїв

Футер в додатку дуже простий. Він однаковий як для мобільної версії, так і для десктопної. Єдина інформація, яка присутня в ньому – це назва додатку. Футер веб-сайту зображено на рисунку 3.8.

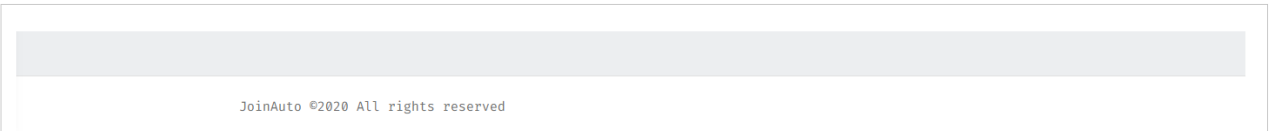

Рисунок 3.8 - Футер веб-сайту

Дизайн, який дозволяє для різних видів пристроїв показувати різний шаблон ще називається «responsive design», тобто адаптивний дизайн. Додаток

було вирішено створити адаптивним і для цього було використано Bootstrap[12]. Bootstrap – це найпопулярніша в світі бібліотека фронт-енд компонентів. Взагальному, Bootstrap – це простий CSS-файл і простий Javascript файл. Ці два файли дають цілу тонну різного функціоналу та стилів. Але це не означає, що можна просто додати ці два файли в проект і все. Для того, щоб використовувати Bootstrap у Angular додатку було запущено команду «npm install bootstrap». Після виконання цієї команди, бібліотека Bootstrap була додана до проекту. Але цього було недостатньо. Для Angular було створено свій власний Bootstrap, який розширює можливості звичайної версії. Звичайна версія Bootstrap дає змогу використовувати готові CSS-класи, які дозволяють додавати властивості до елементів. NgBootstrap, тобто версія для Angular дозволяє використовувати готові віджети, серед них: акордион, кнопки, карусель, рейтинг, таблиці та інші. Для того, щоб мати змогу використовувати NgBootstrap у проекті, було запущено команду «ng add @ng-bootstrap/ng-bootstrap», яка встановила його. Ще раз можна побачити зручність використання Angular CLI, оскільки команда «ng add» належить до CLI. Після того, як було добавлено NgBootstrap до проекту, потрібно додати модуль «NgbModule» у масив «imports» головного модуля AppModule.

Отже, бібліотеку Bootstrap було добавлено і це дало змогу робити адаптивний дизайн для сайту[20]. Bootstrap використовує свої класи, для яких вже було прописано окремі стилі, тобто просто додавши якийсь певний клас до HTML-тега, для нього будуть застосовані стилі, які були записані в класі. Для того, щоб зробити хедер додатку адаптивним, було використано ряд таких класів. Кнопки CTA, які знаходяться в хедері («Купити авто», «Продати авто», «Замовити авто») було обгорнуто блоком div та до нього було додано такі класи: «action-wrapper», «d-none» та «d-lg-flex». Клас «action-wrapper» - це звичайний клас, в якому було прописано стилі для цього блоку, а от класи «d-none» та «d-lg-flex» - це Bootstrap-класи, які дозволяють або показувати елемент обгортки і відповідно контент в ньому, або навпаки приховувати його. В Bootstrap є своя сітка, яка називається «Bootstrap grid». В ній є так звані брейкпоінти, для яких

задано мінімальну ширину зони перегляду, і вони дозволяють масштабувати елементи по мірі зміни ширини зони перегляду. Є такі брейкпоінти: xs, sm, md, lg, xl, для яких вказано мінімальну ширину зони перегляду в пікселях. Клас «d-lg-flex» ніби говорить, що потрібно додавати стиль «display: flex» для блока, коли ширина зони перегляду буде більша або рівна 992 пікселі. А клас «d-none» додає стиль «display: none» для елемента, якщо ширина знаходиться в діапазоні від 0 до 992 пікселів. Тобто це означає, що блок з кнопками потрібно додавати в розмітку тільки тоді, коли ширина зони перегляду буде більша ніж 992 пікселі або рівна 992 пікселям. Рішення про створення адаптивного дизайну для хедера було прийнято через те, що при малій ширині дисплея, наприклад такій ширині як у мобільних телефонів, кнопки будуть відображуватися некоректно і користувачу буде неприємно бачити це, а також користуватися веб-додатком також буде незручно. Для цього і було додано меню зліва, а також кнопку, яка відкриває це меню. Кнопку було створено у вигляді трьох ліній, які при натисканні змінюються на хрестик, тобто в цієї кнопки присутня анімація. Для того, щоб відображати цю кнопку тільки тоді, коли користувач зайшов в веб-додаток з мобільного пристрою, обгортці цієї кнопки, тобто блоку, в якій вона знаходиться, було додано Bootstrap-клас «d-block» та «d-lg-none». Це означає, що блок з кнопкою меню потрібно відображати тільки тоді, коли ширина зони перегляду буде меншою ніж 992 пікселі, тобто від 0 до 992 пікселів.

Отже, кнопку для відкривання меню було додано, а отже, щоб це меню відкрити, потрібно додати ще якусь логіку. На кнопку меню було навішено подію, яка називається «click». Це звичайна Javascript-подія і коли вона настане, то виконається callback-функція в якій була записана вся логіка, яка стосується відкривання меню. Функцію callback було названо «openMenu», що дозволяє легко зрозуміти, що вона виконує якусь дію для відкривання меню. Назви функцій та змінних в Javascript прийнято писати у вигляді camelCase, тобто якщо назва складається з двох слів, то перше слово потрібно писати з малої букви а наступне з великої. Оскільки компонент «header.component.ts» та

«header.component.html» зв’язані між собою, то всю логіку для хедера було написано в «header.component.ts». Для того, щоб відстежувати стан меню, було створено змінну «isMenuActive», якій на початку було присвоєно булеан значення «false», тобто це меню по замовчуванню закрито. Після натискання на кнопку меню, викликається метод «openMenu()», який змінює значення змінної «isMenuActive» на протилежне. Тобто таким способом можна було створено логіку для закривання та відкривання меню. До змінної «isMenuActive» прив’язано додавання класу «active» для меню. Тобто було використано механізм додавання динамічних класів, який присутній в Angular. Для того, щоб додати динамічний клас до елемента, було використано директиву «[ngClass]». Оскільки ця директива була забайджена, тобто записана в квадратних дужках, то в якості значення атрибута було передано об'єкт. Для цього працює такий принцип: в якості ключа об’єкта пишеться назву класу, який ми хочемо застосувати динамічно, а в якості значення ключа потрібно записати умову по якій потрібно добавити даний клас. Тобто в нас є змінна[9] «isMenuActive», і якщо її значення – «true», то потрібно додавати клас «active», а якщо значення – «false», то не потрібно. На рисунку 3.9 зображено принцип додавання динамічного класу «active».

```
<div class="side-menu-btn d-block d-lg-none">
  <button class="menu-btn" [ngClass]="{'active': isMenuActive}" (click)="openMenu()">
    <span></span><span></span><span></span>
  </button>
</div>
```

Рисунок 3.9 - Додавання динамічного класу

Для того, щоб надати невеликої жвавості хедеру, для нього було створено механізм показування та зникання. Тобто, коли користувач починає гортати веб-сторінку вниз, то хедер зникає з невеликою затримкою. А коли користувач

навпаки починає гортати сторінку вгору, то хедер з’являється. Для цього було придумано деяку логіку. Було створено дві змінні: перша – це змінна, яка відповідає за те, чи було здійснено гортання вниз, а друга змінна – це змінна, яка зберігає кількість пікселів, на яку було проскролено і на початку її значення - 0. Логіка тут така: коли користувач починає скролити вниз, то в змінну «scrolledPixels» записується кількість пікселів, на яку було проскролено і якщо ця кількість більша від 0, то змінна «isScrolledBottom» отримує значення «true». Для хедера також було застосовано динамічний клас, який було названо «scrolled». І коли значення змінної («isScrolledBottom»), до якої прив'язаний цей клас стане «true», хедер зникне з шаблону і його буде не видно. Кількість пікселів, на яку було проскролено, збережеться в змінну «scrolledPixels» і при наступному скролі, значення цієї змінної буде порівнюватися зі значенням «window.scrollY». Якщо значення «scrolledPixels» менше, ніж значення «window.scrollY» то це буде означати те, що було проскролено вгору, а значить потрібно відображати хедер, встановивши значення «false» для змінної «isScrolledBottom»[14]. Логіка для показування/приховування хедера зображена на рисунку 3.10.

```
addScroll() {
  if ((window.scrollY || window.pageYOffset || document.documentElement.scrollTo) > this.scrolledPixels) {
    this.isScrolledBottom = true;
    this.scrolledPixels = pageYOffset;
  } else {
    this.isScrolledBottom = false;
    this.scrolledPixels = pageYOffset;
  }
}
```

Рисунок 3.10 - Логіка для показування/приховування хедера

3.3 Створення сторінки реєстрації та логіну

Для вебдодатку було додано можливість реєстрації та логіну користувача. Для цього було створено дві сторінки: сторінку реєстрації та сторінку логіну.

Для того щоб потрапити на сторінку логіну, користувачу потрібно натиснути кнопку «Логін», яка знаходиться зправа в хедері. Кнопка «Логін» являє собою посилання, до якого було додано атрибут «routerLink». Атрибут «routerLink» - це спеціальний атрибут Angular, який дозволяє переміщатися по додатку без перезагрузки сторінки. Тобто замість того, щоб написати в тег «a» атрибут «href», в нього записується атрибут «routerLink» і звичайно ж адресу, по якій потрібно перейти. Тобто коли користувач переходить на сторінку логіна, Angular говорить JavaScript[15], щоб він відмалював готовий компонент і при цьому не відправляються ніякі запити на сервер, не отримуються ніякі данні від нього, а все знаходиться всередині. Пакет Angular Routing дозволяє з цим взаємодіяти. Тобто він дозволяє динамічно без перезагрузки переміщуватися по додатку. Перше, що необхідно було зробити, щоб відобразити сторінку логіна, це звичайно її створити і зареєструвати цю сторінку в списку роутів додатку. Для того, щоб створити сторінку логіна було використано Angular CLI та запущено команду «ng generate component login» і після її виконання, було створено компонент логіна. Також було створено ще модуль логіна для того, щоб не зареєстровувати компонент в головному модулі AppModule, а зареєструвати його в своему власному модулі, що в майбутньому дозволить зробити ліниву загрузку цього модуля. Лінива загрузка модуля LoginModule дозволить загружати цей модуль тільки тоді, коли ми перейдем на цю сторінку. В додатку є модуль, в якому записуються всі роути. Він називається «AppRoutingModule» і був створений ще на етапі генерації проекту. В цьому модулі є масив зі всіх роутів, які присутні в додатку і в ньому потрібно додати шлях, який буде визначатися по строці «login» і потім замість того, щоб прописувати потрібний компонент, потрібно звернутися до поля, яке називається «LoadChildren» і сюди

як строкове значення передати шлях до того модуля, який потрібно підгрузити, а саме шлях до модуля LoginModule. Але цього ще не достатньо. Потрібно перейти в самий LoginModule додати в список роутів роут з префіксом, який буде дорівнювати пустій строці, адже по замовчуванню у всього цього модуля уже є префікс «login». Після того як буде зроблено перехід по посиланню «login», Angular створить файл, який називається «login-page-module», тобто окремий «чанк». Створення таких чанків дає сильний приріст в швидкості загрузки додатку та оптимізованості, оскільки не потрібно буде грузити всі модулі одразу в один файл, як це робиться в SinglePageApplication.

Для сторінки логіна було придумано сучасний, та досить легкий в сприйнятті дизайн. Дизайн сторінки логіна зображено на рисунку 3.11.

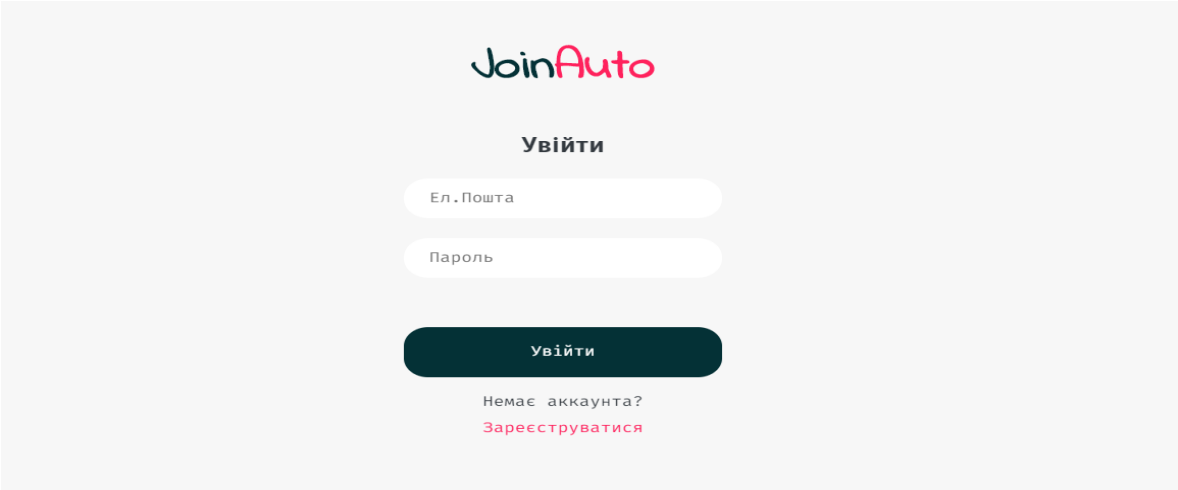

Рисунок 3.11 - Дизайн сторінки логіна

Всі елементи сторінки було розміщено посередині. В верхній частині було додано назву проекта і під нею було створено форму для логіна користувача та кнопку «Увійти». Під кнопкою додано підказку, з текстом «Зареєструватися». Тобто, якщо користувач ще не зареєстрований, то він зможе перейти за посиланням та зареєструватися. Основні поля вводу – це електронна пошта, та пароль. Форму для логіну користувача було створено за допомогою Angular

Reactive Forms а отже перше, що необхідно було зробити – це вказати Angular, що будуть використовуватися реактивні форми. Тобто іншими словами потрібно в масиві «imports» модуля LoginModule імпортувати модуль, який називається ReactiveFormsModule, який в свою чергу імпортується з бібліотеки «@angular/forms». І після таких дій можна працювати з реактивними формами. Реактивні форми означають те, що вся ініціалізація цих форм проходить не в шаблоні, а робиться програмно і це додає дуже багато зручностей для роботи з формами.

Для того щоб створити саму форму, в компоненті LoginComponent, було створено змінну, яка називається «form» і додано їй тип «FormGroup», який так само імпортується з бібліотеки «@angular/forms». Після того, як було створено змінну, її потрібно ще проініціалізувати і для цього було використано метод життєвого циклу «OnInit», в якому відбувається ініціалізація компонента. В цьому методі потрібно зробити ініціалізацію форми, написавши «this.form = new FormGroup()». FormGroup – це клас, який дозволяє створювати формгрупу, або просто форму. В конструктор класу FormGroup було передано об’єкт, який називається «controls». Цей об'єкт зберігає наступний формат: як ключ потрібно передавати назву поля, наприклад «email», а як значення, потрібно проініціалізувати це поле через ключове слово «new» і клас, який називається «FormControl». Цей клас приймає в себе декілька аргументів. На початку потрібно передати початкове значення даного контролу, наприклад пуста строка, і другим параметром потрібно передати набір валідаторів. В формі було додано два формконтроли: «email» та «password». Для формконтрола «email» в якості валідаторів було передано валідатор «email», який буде перевіряти чи правильний формат у електронної пошти, яку ввів користувач, а також формконтрол «required», який буде перевіряти наявність значення в полі форми. Для формконтрола «password» в якості валідаторів було передано валідатор «required», який буде також перевіряти наявність значення в полі форми. Після того, як було створено форму, її потрібно зв’язати з формою, яка присутня в

шаблоні HTML. Для цього було створено тег «form» і для того, щоб передати в нього змінну «form», яка знаходиться в typeScript файлі, було додано директиву, яка називається «[formGroup]» і зробити байндинг, передавши як значення атрибута об'єкт «form». Тобто за допомогою цієї директиви було зв'язано html-форму, а також форму, яку було створено. Оскільки у формі присутня кнопка «Увійти», то їй потрібно вказати тип «submit», тому що при кліку на цю кнопку буде виконуватися підтвердження форми. Після цього у самої форми буде доступна подія, яка називається «ngSubmit». Для цього цю подію було обгорнуто в круглі дужки, щоб можна було її слухати, а також було передано назву методу, який потрібно викликати. Такий метод було названо «submit», який означає те, що потрібно підтвердити форму. Ще одну річ, яку потрібно було зробити – це зв'язати формконтроли з typescript файлу з формконтролами в шаблоні. Для цього потрібно кожному тегу input в формі добавити атрибут, який називається «formControlName», тобто назву формконтрола. Як значення цього атрибута потрібно передати назву цього ж формконтрола з typescript файлу. Тобто для інпута вводу електронної пошти атрибут «formControlName» буде приймати значення «email», а для інпута вводу пароля цей атрибут буде приймати значення «password». Після того, як додати формконтроли для полів вводу даних, та зв'язати форму з формою з typescript файлу, Angular зможе контролювати стани цих інпутів, тобто перевіряти їхні значення, розпізнавати, чи на інпут було натиснено. І відповідно після цього Angular зможе сам валідувати інпути, і не потрібно буде вручну створювати валідації для пароля та електронної пошти. Отже для того, щоб валідувати формконтроли Angular використовує свої власні валідатори і для того, щоб додати їх для формконтролів потрібно в конструктор класу передати Validators – великий клас з бібліотеки @angular/forms, який зберігає в собі список валідаторів, вбудованих в Angular. Існують такі валідатори: email, minLength, min, required, compose, composeAsync, max, maxlength, nullValidator, pattern та requiredTrue. Серед цих всіх валідаторів є валідатор pattern, який допомагає проводити валідацію по шаблону, який

вказаний для нього. В якості шаблона можна передати якийсь регулярний вираз, тобто regExp і здійснювати валідацію по цьому шаблону. Але можна піти набагато простішим шляхом і використати валідатор email, в якому вже було заготовлено шаблон для валідації і не потрібно витрачати час на пошук потрібного регулярного виразу. Validators – це клас у якого є статичні методи. І в ньому дуже важливо не викликати функцію «email()» як валідатор, тому що вона поверне якийсь результат, який не відповідає типу, тобто в якості валідатора потрібно вказувати просто посилання на дану функцію. Але одного валідатора email в даному випадку буде недостатньо, оскільки потрібно, щоб користувач точно ввів електронну пошту. І для цього потрібно другий параметр, який приймає клас FormControl перетворити в масив, для того, щоб додавати ще інші валідатори. Наступним валідатором після email потрібно передати «required», тобто це ніби говорить, що електронна пошта обов'язково повинна бути введена. Такі ж самі маніпуляції було проведено і для формконтрола «password». І для нього було додано тільки Validators.required, щоб користувач обов’язково ввів пароль. Механізм створення формконтролів в додатку та додавання до них валідаторів зображено на рисунку 3.12.

```
ngOnInit() {
  this.form = new FormGroup( controls: {
    email: new FormControl( formState: '',  validatorOrOpts: [Validators.required, Validators.email]),
    password: new FormControl( formState: '',  validatorOrOpts: [Validators.required])
  });
}
```

Рисунок 3.12 - Створення формконтролів та валідаторів для форми

Отже створення валідаторів дозволяє визначити та показати користувачу, що на якомусь контролі є помилка, зв’язана з тим, що валідатор, який було вказано не пройшов. І для цього в додатку було створено механізм відображення

помилок, зв’язаних з формконтролом, для того, щоб користувач міг зрозуміти, що він ввів неправильно чи що пішло не так. Для цього під полями вводу (інпутами) було додано повідомлення про помилки. В шаблоні було створено блок з класом «error-block», в якому будуть відображуватися повідомлення з помилками. Оскільки цей блок не повинен відображатися, якщо помилок немає, то для нього було створено умову, по якій він буде відображуватися. За допомогою директиви «*ngIf» було додано логіку для відображення цього блока. Інпутів у формі є два, а отже під кожним з них було створено такі блоки з помилками. Для того, щоб зробити перевірку на те, чи потрібно показувати блок з помилками, в об’єкта «form» прямо всередині шаблону було викликано метод, який називається «get», в який було передано назву формконтрола в строковому форматі. Для блока з електронною поштою було передано «email», а для блока з паролем – «password». Далі ми отримуємо доступ до контролів, в яких присутні різні стани. Наприклад якщо контрол «invalid», тобто невалідний, то тоді потрібно показувати повідомлення про помилки. Також потрібно здійснювати перевірку на те, чи було натиснено на інпут, оскільки на початку формконтрол невалідний, а отже помилки будуть відображатися завжди. Для цього було додано ще одну умову «touched». В результаті ці дві умови повертаються булеве значення і валідність формконтрола буде залежати від цього значення. Для формконтрола email було додано два повідомлення про помилки: «Введіть email» - якщо електронну пошту було взагалі не введено, та «Введіть коректний email» - коли електронну пошту було введено в неправильному форматі. Для формконтрола password було додано тільки одне повідомлення про помилку – «Введіть пароль». Таку помилку буде показано, якщо користувач не введе пароль. Повідомлення з помилками для форми логіна зображено на рисунку 3.13.

Рисунок 3.13 - Помилки вводу даних

Логіку відображення помилок в шаблоні зображено на рисунку 3.14

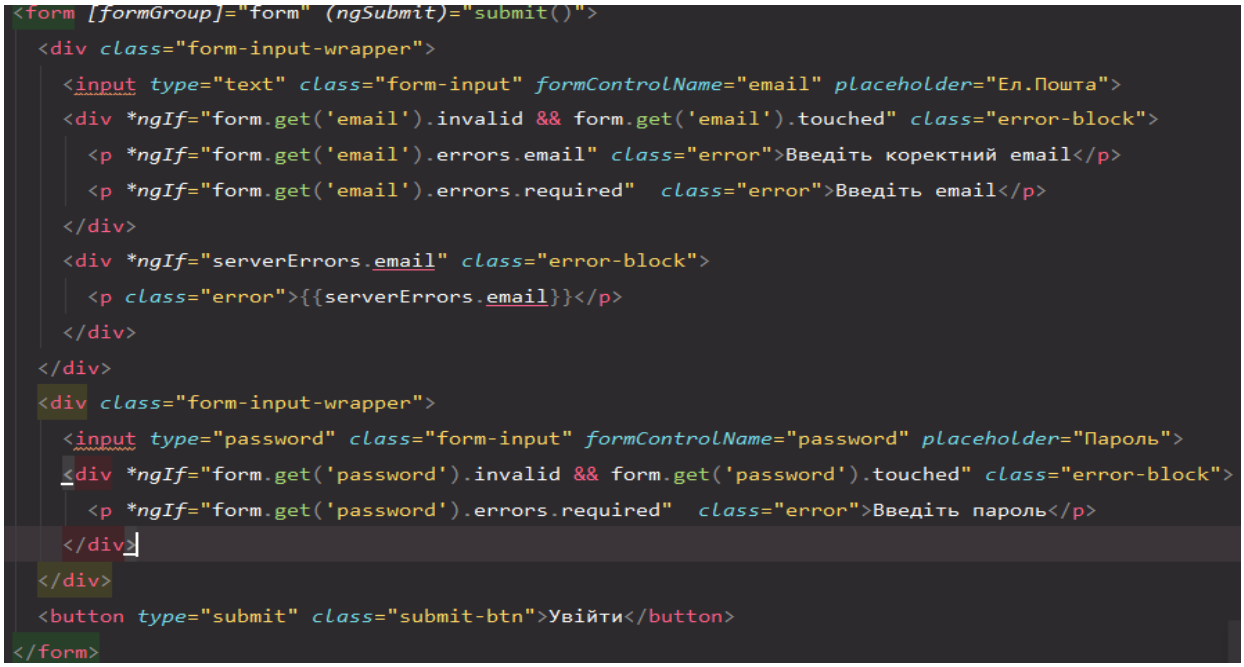

Рисунок 3.14 - Логіка відображення помилок в шаблоні

Отже після того, як користувач введе неправильні дані, він побачить помилки, та зможе їх виправити завдяки підказкам. Коли користувач виправляє помилки то він зможе натиснути кнопку «Увійти», та дані, які він ввів будуть відправлені на сервер для перевірки. Всю логіку для роботи з сервером в Angular прийнято виносити в окремий сервіс. Тому для аутентифікації користувача було створено сервіс з назвою AuthService. В ньому було також створено всі необхідні методи для роботи і в компоненті LoginComponent будуть просто викликатися такі методи. В сервісі AuthService було додано декоратор «@Injectable», для того, щоб інжектувати в нього дані. Також цей сервіс було зареєстровано в кореневому модулі, записавши «providedIn: “root”». В цьому сервсі було створено метод конструктора, куди було інжектовано HttpClient, тобто це означає що в цьому сервісі ми будемо працювати з об’єктом HttpClient. Після того, як було інжектовано HttpClient, потрібно створити всі методи, необхідні, для того, щоб працювати з аутентифікацією користувача. Для того, щоб працювати з логіном користувача, було створено метод login(), який приймає об’єкт body, що зберігає всі введені дані користувача (email, password). У сервісі також було створено

змінну, яка відповідає за адресу API («https://join-auto.herokuapp.com/v1»). В методі login() дані будуть надсилатися за допомогою rest API метода «post»[5], який доступний в пакеті HttpClient. Метод post повертає об'єкт проміса і після цього можна викликати ще один метод – «subscribe». Проте цей метод не потрібно викликати в межах сервісу, тому що логіка тут присутня совсім інша. Підписуватися потрібно в самому компоненті LoginComponent, так як логіка для логіна реалізована іменно в ньому. В цьому і проявляється комфорт використання сервісів, тому що в самому сервісі ми ініціалізуємо стрім (stream) і не обов'язково, щоб він виконувався в сервісі, тобто немає значення де цей стрім виконається. Важливо те, що ми його тут створюємо, а де він буде працювати, це вже наше рішення. Для цього з метода login() просто повертається цей стрім.

Щоб мати змогу працювати з сервісом AuthService в компоненті LoginComponent, потрібно зробити інжектування цього сервіса в метод конструктора класу LoginComponent. Для цього було створено приватну змінну «authService» та вказано їй тип «AuthService» (private authService: AuthService). Такі дії дали змогу працювати з сервісом та використовувати методи, які були реалізовані в ньому. Отже, в класі LoginComponent було створено метод submit, який буде відправляти всі дані, які ввів користувач на сервер. Ці дані будуть відправлятися в тому випадку, якщо форма валідна, тобто якщо у всіх формконтролів цієї форми valid === true. Для того, щоб надіслати дані на сервер, було використано метод login(), який доступний з сервісу AuthService. Оскільки цей метод повертає проміс, то ми викликаємо його та передаємо в об'єкт body всі дані які ввів користувач у форматі, який попросив бекенд (email: <data>, password: <data>). Для повернутого проміса застосовуємо метод «subscribe», за допомогою якого всі дані будуть надіслані на сервер. Цей метод працює по принципу методів than та catch, які доступні у промісах. Але тут все дещо спрощено. Замість того, щоб викликати метод than, коли проміс отримав стан «resolved» та метод catch, коли проміс отримав стан «rejected», метод subscribe

запускає callback-функції, які було записано. Першим параметром є callback-функція, яка виконається при успішному логіні користувача, а другим параметром є callback-функція, яка виконається тоді, коли будуть якісь помилки, тобто тоді, коли проміс отримає стан «rejected». У випадку успішного логіна, користувач перенаправиться на головну сторінку додатку. Якщо сервер поверне якісь помилки, то користувач, звичайно ж не перейде на головну сторінку, і йому висвітляться помилки на рахунок того, де було вказано неправильні дані. Для того, щоб зберігати помилки, які повернув сервер, було створено змінну. Ця змінна – це об'єкт, в якому ключ – це поле в якому була допущена помилка, а значення – це текст з помилкою. Об'єкт з помилками також було прив'язано до шаблону та здійснено перевірку: якщо серверні помилки є, то відображати їх, а якщо ні то навпаки. Після того, як користувач почне виправляти ці помилки та вводити правильні дані, то серверні помилки зникнуть. На рисунку 3.15 зображено логіку для логіну користувача.

Рисунок 3.15 - Логіка для логіну користувача

Якщо користувач ще не зареєструвався, то відповідно він не зможе залогінитися. Для цього було створено сторінку реєстрації користувачів. Для того, щоб перейти на сторінку реєстрації, користувачу потрібно натиснути на кнопку «Зареєструватися», яка знаходиться під кнопкою «Увійти». Кнопку «Зареєструватися» було виділено рожевим кольором, для того щоб її було краще помітно. Отже перший крок до того, щоб створити сторінку реєстрації – це генерація компонента SignUpComponent. Так само, як і в випадку з логін-сторінкою, сторінці реєстрації було також створено модуль та зареєстровано компонент цієї сторінки в ньому. Цей модуль буде загружуватися ліниво, тобто для нього було добавлено функціонал лінивої загрузки модулів. В масиві роутів

головного компонента AppModule було створено новий об’єкт. Значення «path» було вказано як «sign-up», а значення «loadChildren» - шлях до цього модуля. Такі маніпуляції дозволять завантажувати цей модуль тоді, коли користувач здійснить перехід по посиланню «/sign-up».

На сторінці реєстрації користувача буде форма, в якій потрібно буде ввести такі дані: ім'я, електронну пошту, пароль, місце проживання, а також номер телефону. Для того, щоб створити форму, в компоненті було створено змінну «form» з типом «FormGroup». За допомогою метода життєвого циклу «ngOnInit» було проініціалізовано цю змінну та додано їй формконтроли, назва яких співпадає з назвою поля, яке потрібно заповнити. Для формконтролів також було додано валідатори для перевірки коректного введення даних, а саме для контрола «email» було додано валідатор, який перевіряє правильність написання електронної пошти, та валідатор «required», який перевіряє чи поле не пусте. Для контрола «password» було додано валідатор «required», а також валідатор «minLength», якому було задано значення «6», що означає те, що довжина пароля повинна бути не менше шести символів. Контролу «name» було додано тільки валідатор «required». Для того, щоб перевіряти правильність введення номеру телефону користувача, контролу «phoneNumber» було додано валідатор «pattern» з шаблоном «^\\+?3?8(0\\d{9})$». Це означає, що перевірка на валідність буде здійснюватися по цьому шаблону. Останній контрол у формі – «location», якому було додано валідатор «minLength» зі значенням «5». Для роботи з реєстрацією користувача в сервісі AuthService було створено метод з назвою «register»[19]. Подібно до метода «login», метод «register» також приймає об’єкт «body» і повертає проміс. Для того, щоб використовувати AuthService в компоненті SignUp, потрібно зробити інжектування цього сервісу в конструктор класу. Після цього можна використовувати методи сервіса AuthService. Але перед тим як надсилати дані на сервер, їх потрібно отримувати від користувача. Для цього форму з typescript файлу було з’єднано з формою в шаблоні, тобто було створено атрибут «[formGroup]» і як значення цього атрибута було додано змінну «form».

Всім контролам у формі було додано атрибут «formControlName», що також дозволило прив’язати поля вводу даних до формконтролів, які записані у файлі. Для підтвердження форми було додано кнопку «Зареєструватися», до якої було прив’язано подію «click». Після того, як ця подія відбудеться, виконається метод «submit», у якому присутня деяка логіка. В цьому методі буде здійснена перевірка на те, чи форма валідна, тобто, якщо користувач введе невалідні дані, то не зможе зареєструватися. Якщо форма валідна, то всі дані, які ввів користувач збережуться в об’єкті «form» і потім надішлються на сервер. В цьому методі також реалізований метод «subscribe», який в своїх параметрах приймає дві callback-функції. Якщо користувач введе валідні дані, то здійсниться запит на сервер і потім виконається перша callback-функція. Після успішної реєстрації, сервер поверне токен користувача, який зареєструвався. Цей токен потрібно записати у localStorage, тобто у локальне сховище браузера. Наявність цього токена у сховищі буде свідчити про те, що користувач зареєструвався, чи залогінився. Після реєстрації користувача буде перенаправлено на домашню сторінку веб-додатку. Якщо сервер поверне якісь помилки, то вони збережуться у змінній «serverErrors». Як і в випадку з логіном, ця змінна теж є об'єктом, у який будуть записані всі помилки, які повернув сервер. Ці помилки буде виведено в шаблоні і користувач зможе виправити їх та спробувати зареєструватися знову. На рисунку 3.16 зображено логіку для реєстрації користувача.

Рисунок 3.16 - Логіка для реєстрації користувача

Після того, як користувач здійснить логін або реєстрацію, він вже не зможе зробити це ще раз, доки не розлогіниться. Кнопка «Увійти» зміниться на кнопку «Вийти». Для того, щоб користувач не міг зайти на ці сторінки повторно, було створено систему захисту роутів[18]. Для захисту роутів застосовано концепцію, яка присутня у Angular та називається «Guards». Для реалізації цього концепту було створено новий файл, який названо «auth.guard.ts». Для того, щоб цей файл дійсно став «гуардом» було здійснено імплементацію від інтерфейса, який називається «CanActivate». На початку потрібно реалізувати цей метод і він приймає в себе декілька параметрів. Перший параметр – «route» з типом «ActivatedRouteSnapshot». Другий параметер – «state», в якого тип – «RouterStateSnapshot». Для того, щоб знати чи користувач увійшов в систему, в

сервісі AuthService було створено ще один метод «isAuthenticated», який повертає «true», якщо користувач увійшов і «false», якщо користувач не увійшов. Щоб мати доступ до цього метода в «AuthGuard» потрібно інжектувати сервіс AuthService. Якщо користувач увійшов в систему і хоче перейти на сторінку логіна чи реєстрації, йому буде не дозволено це зробити та цього користувача буде переправлено на домашню сторінку. Для того, щоб здійснити редірект користувача, у конструктор AuthGuard було інжектовано сервіс роута. AuthGuardService зображено на рисунку 3.17.

```
import { Injectable } from '@angular/core';
import {Router, CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot} from '@angular/router';
import { AuthService } from './auth.service';
@Injectable(
  {providedIn: 'root'}
)

export class AuthGuardService implements CanActivate {
  constructor(public auth: AuthService, public router: Router) {}
  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): boolean {
    if (!this.auth.isAuthenticated()) {
      this.router.navigate( commands: ['/login']);
      return false;
    } else {
      this.router.navigate( commands: ['/']);
      return true;
    }
  }
}
```

Рисунок 3.17 – GuardService

Для того, щоб захистити шлях «/login» та шлях «/signup», у об'єкти роутера було додано поле, яке називається «canActivate», куди було передано AuthGuard. Принцип захисту сторінок логіна та реєстрації зображено на рисунку 3.18.

Рисунок 3.18 - Захист сторінок логіна та реєстрації

3.4 Створення основного функціоналу

При вході на веб-сайт перше, що користувач буде бачити – це головну сторінку. Для того, щоб згенерувати головну сторінку, було використано Angular CLI. Всі сторінки в проекті було розбито на окремі модулі, тобто для кожної сторінки було створено свій модуль. Для головної сторінки було також створено такий модуль та в масив «declarations» додано компонент HomePageComponent. Єдине, що цей модуль не буде грузитися ліниво, тому його було імпортовано в масив «imports» головного модуля додатку AppModule. В модулі AppRoutingModule було вказано шлях для домашньої сторінки а саме шлях «/». Тобто це означає, що домашня сторінка буде грузитися першою при загрузці самого веб-додатку. В домашній сторінці створено фільтр, за допомогою якого користувач зможе підшукати автомобіль за певними характеристиками та перейти на сторінку зі всіма автомобілями, які відповідають фільтру. Також другою секцією на домашній сторінці є секція з недавно доданими автомобілями. На рисунку 3.19 зображено домашню сторінку додатку.

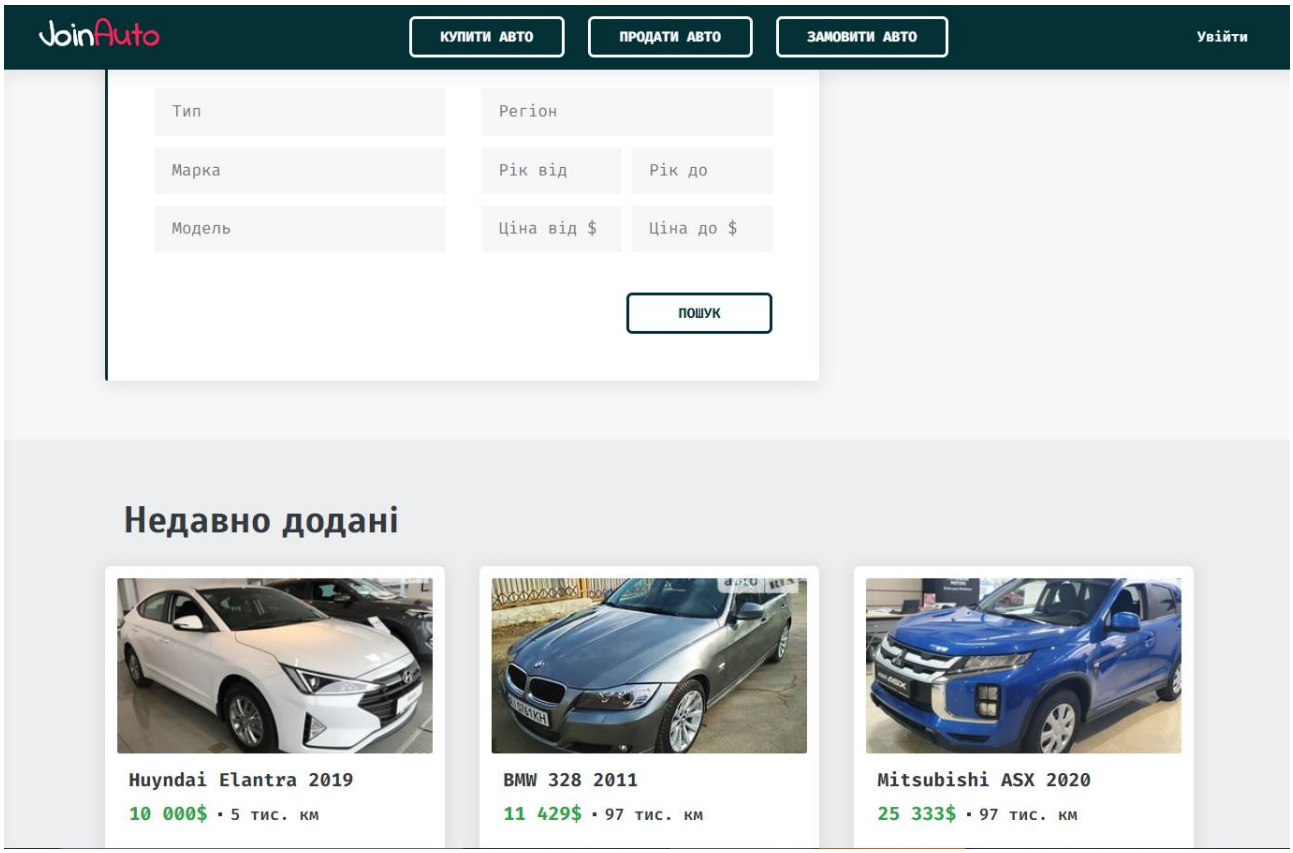

Рисунок 3.19 - Домашня сторінка додатку

Для того, щоб користувач зміг знайти підходящий автомобіль, у фільтрі було додано такі поля вводу: тип автомобіля, марка автомобіля, модель, регіон, рік випуску від, рік випуску до, ціна від та ціна до. Ці інпути – не прості, а інпути автоматичного доповнення, тобто коли користувач почне вводити якісь дані, йому зразу ж відобразяться запропоновані варіанти. Для того, щоб використовувати такі інпути, було додано бібліотеку «ng-autocomplete», яка володіє функціоналом автодоповнення. Щоб додати цю бібліотеку, було викликано команду «npm install ng-autocomplete» і після цього цю бібліотеку було встановлено. Для фільтра створено окрему форму, при сабміті якої користувач переходить на сторінку зі всіма автомобілями, які задовільняють критерій фільтрування. В кожного поля форми присутній формконтрол, який зв'язує інпут в шаблоні з інпутом в typescript файлі. Також в інпуті присутні такі атрибути, які було забайнджено: «[data]», «[searchKeyword]», «[itemTemplate]»,

та «[notFoundTemplate]». Вони використовуються для того, щоб динамічно відображати дані по якомусь ключовому слові в об'єкті, до якого вони прив'язані. Якщо результату не буде знайдено, то користувач отримає повідомлення про те, що за його запитом нічого не було знайдено. На рисунку 3.20 зображено приклад автодоповнення.

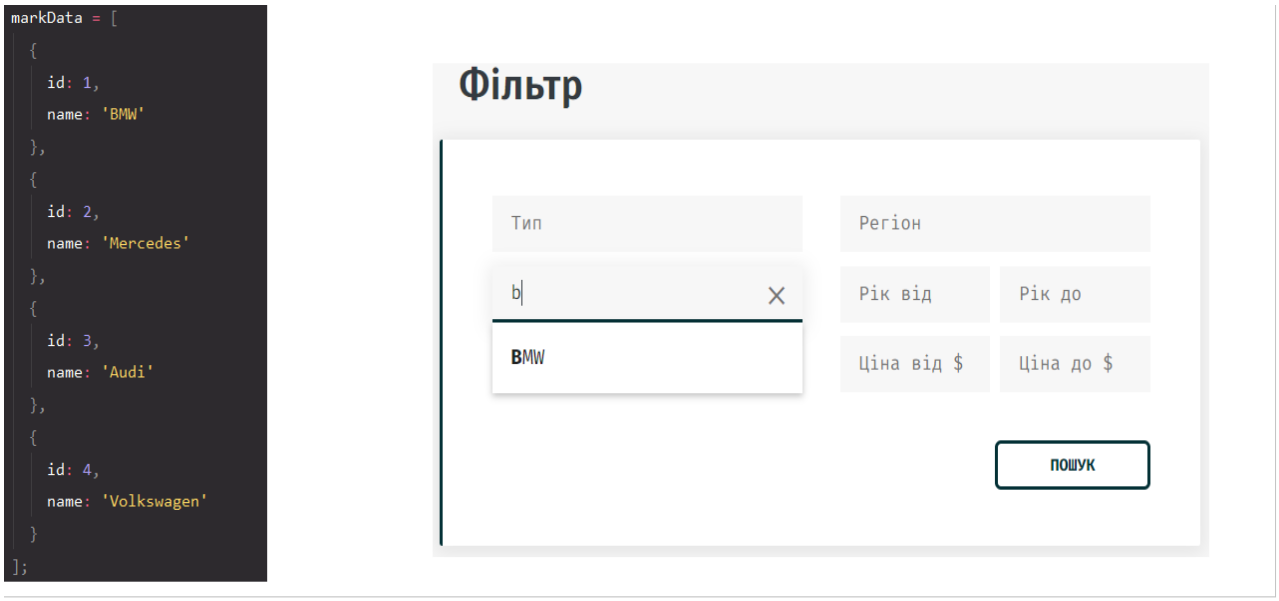

Рисунок 3.20 – Автодоповнення

У другій секції головної сторінки відображаються автомобілі, які були недавно додані на сайт. Для того, щоб відобразити дані про автомобіль було створено окремий компонент, який можна перевикористовувати у багатьох інших компонентах. Це додало дуже великої зручності у розробці додатку. Також для такого компонента було створено окремий модуль, що дозволило імпортувати даний модуль в модуль головної сторінки. Для того, щоб отримати список автомобілів, було створено сервіс Vehicles, в якому створено метод «getVehicles()». В компоненті VehicleList було інжектовано цей сервіс в метод конструктора. Отримання списку автомобілів здійснюється при ініціалізації компонента, тобто в методі життєвого циклу «ngOnInit» виконується запит до

сервера, який повертає масив з об'єктів. Для того, щоб зберегти цей масив об'єктів у компоненті було створенно змінну «productsList», яка на початку є пустим масивом. Після того, як було отримано масив з автомобілями з сервера, змінну «productsList» було перезаписано та задано їй нове значення, тобто масив з цими автомобілями. Для того, щоб перебрати масив , який повернув сервер, було використано директиву «*ngFor». Ця директива – це цикл, який виконує динамічний перебір масиву та генерує нові елементи. В даному випадку таким масивом є «productsList». Якщо користувач бажає знайти підходящий автомобіль по критерію, то він заповнює форму, де вказує характеристики автомобіля, який бажає знайти і після цього по натисканню на кнопку «Пошук» виконується метод «submit», який перенаправляє користувача на сторінку з автомобілями, в якій будуть виведені тільки ті автомобілі, які підходять під критерій пошуку. Приклад сторінки зі списком автомобілів зображено на рисунку 3.21

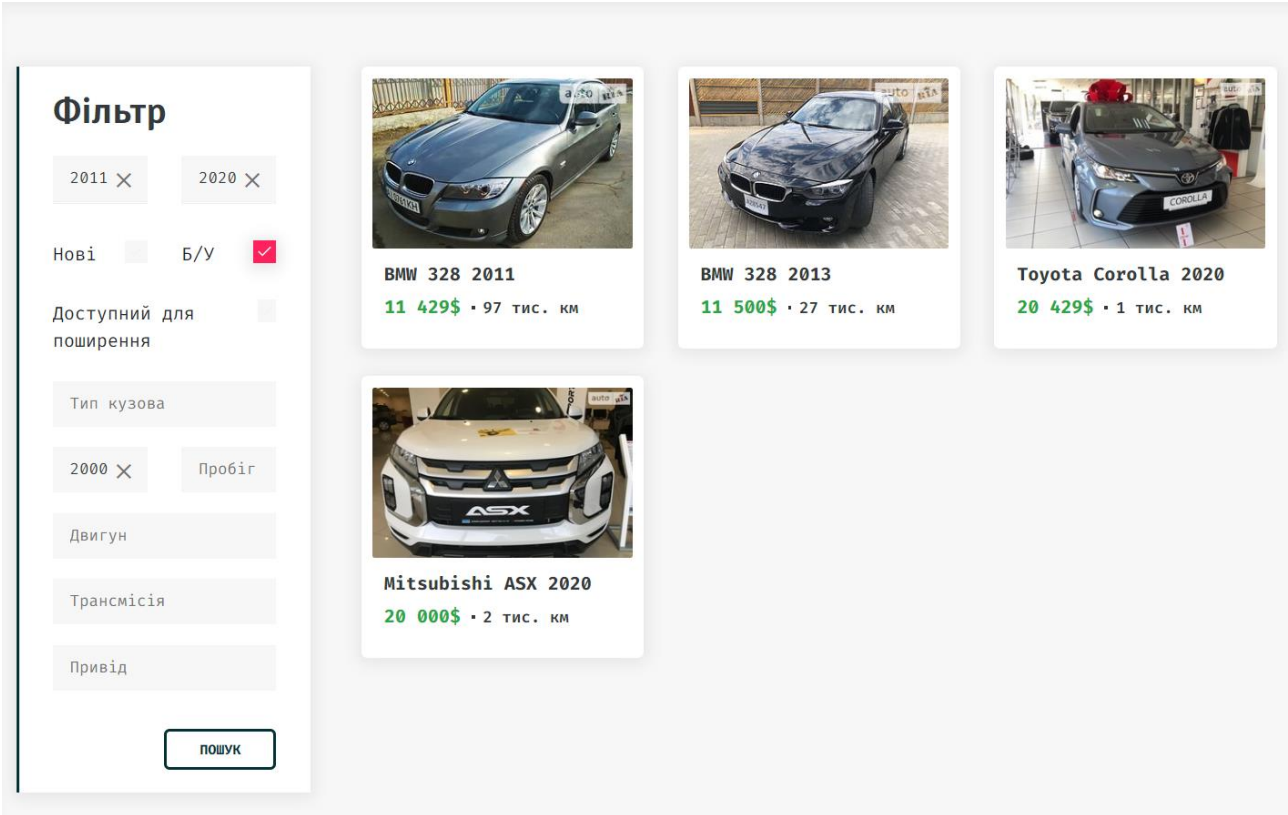

Рисунок 3.21- Сторінка зі списком автомобілів

Для сторінки зі списком автомобілів також було створено окремий модуль та додано можливість лінивої загрузки цього модуля. При переході на роут «vehicle-list», Angular створює окремий chunk, тобто окремий файл, в якому будуть зберігатися всі інші файли, які потрібні для роботи цього модуля. Для того, щоб відобразити список автомобілів, при ініціалізації компонента, в його методі життєвого циклу виконується запит до сервера, який повертає список цих автомобілів. Цей список зберігається в змінній «productsList» і за допомогою циклу «*ngFor» перебирається. Для того, щоб відобразити автомобілі, всі дані про них передаються компоненту ProductCard, який в свою чергу приймає ці дані та виводить їх. В лівій стороні сторінки було додано фільтр, за допомогою якого користувач зможе знайти бажаний автомобіль по критеріях, які будуть вказані. Після натиснення на кнопку «Пошук» виконається метод «find()», який прийме всі параметри, які вказав користувач, та зробить запит до сервера. Сервер в свою чергу поверне новий масив з автомобілями і після цього в змінну «productsList» буде записано новий масив. Цикл ngFor зрозуміє, що значення змінної змінилося та зробить повторний прохід по елементах масиву та користувачу виведуться нові автомобілі, які відповідають заданим критеріям.

Картки з автомобілями зроблено клікабельними, тобто коли користувач натисне на таку картку, то відбудеться перенаправлення на адресу «/product-page». У кожного автомобіля в списку є свій власний id, за допомогою якого можна отримати всі дані про нього. При загрузці сторінки з автомобілем, виконається метод «ngOnInit», в якому буде зроблено запит до сервера. Для того, щоб сервер знав дані про який автомобіль потрібно повернути, в запиті було додано id цього атомобіля. Щоб дізнатися id автомобіля, потрібно звернутися до поля id об’єкта з даними про автомобіль. Сторінку автомобіля зображено на рисунку 3.22.

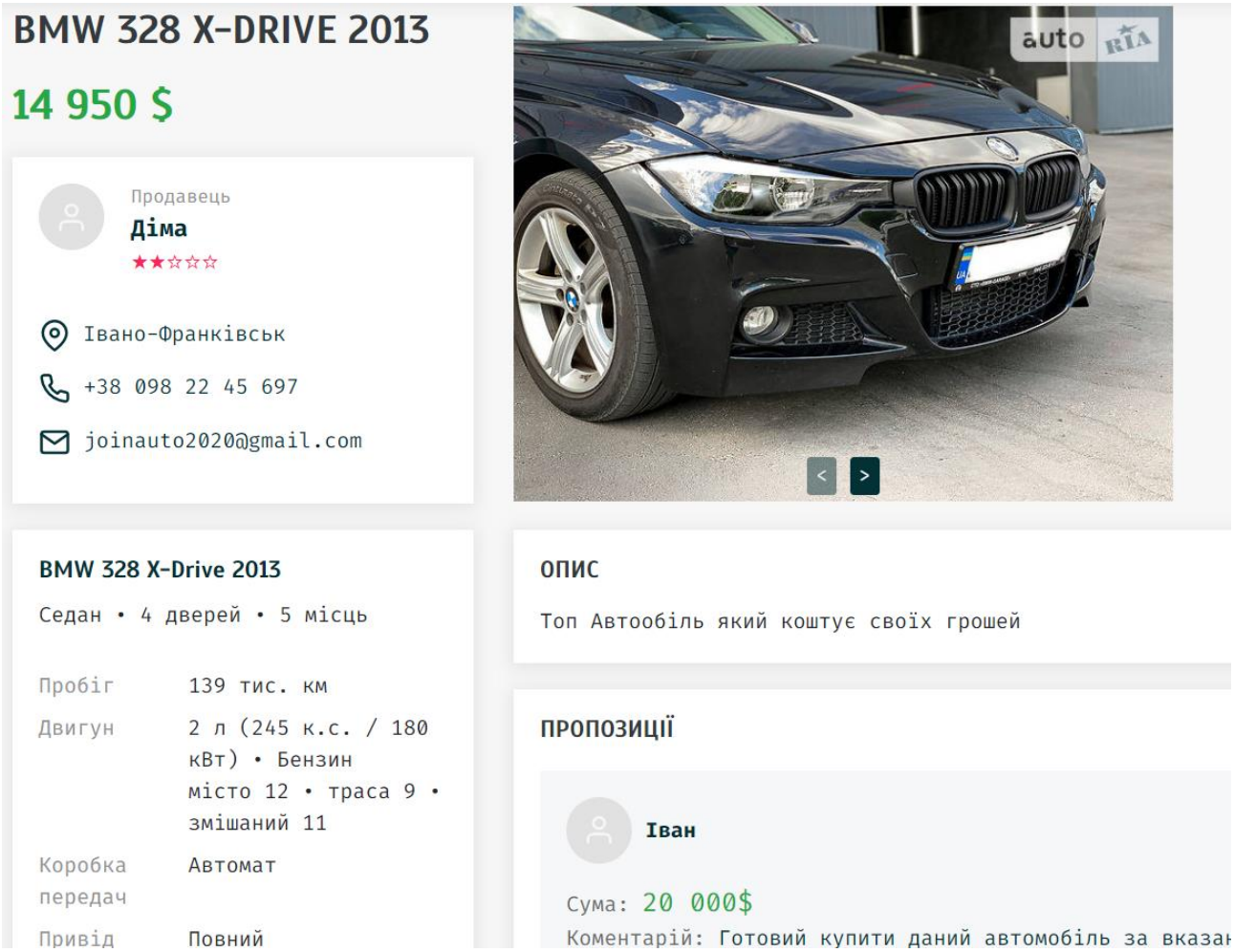

Рисунок 3.22 - Сторінка автомобіля

На сторінці з автомобілем відображуються основні дані про нього. Також є інформація про власника автомобіля, та список пропозицій, які було висунуто іншими користувачами. Оскільки фотографій у автомобіля може бути декілька, то було використано плагін «owl-carousel», за допомогою якого можна додавати карусель в html-розмітку, щоб користувач міг гортати фото автомобіля. Для того, щоб установити цей плагін було виконано команду «npm install owl-carousel». Після того, як плагін було установлено, його модуль було імпортовано в масив «imports» модуля «ProductPageModule» і потім додано компонент «owl-carousel-o» в шаблон. В цей компонент також потрібно було передати ще деякі опції, а саме об'єкт «customOptions», в якого тип «OwlOptions». Опції для каруселі зображені на рисунку 3.23.

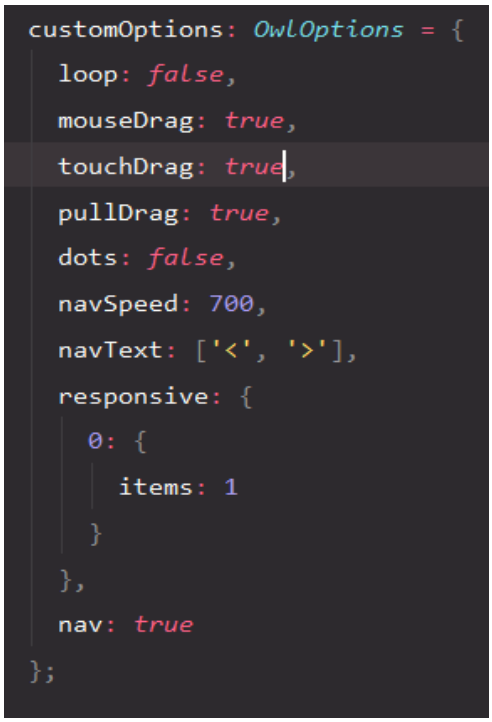

Рисунок 3.23 - Опції для каруселі

Одна з найважливіших опцій в каруселі – це «touchDrag», яка відповідає за те, щоб можна було гортати фотографії за допомогою пальця на сенсорних екранах. Оскільки додаток адаптований під мобільні пристрої, то така опція повинна бути присутня. Також в каруселі є опція «responsive», яка відповідає за кількість картинок які потрібно відображати на одному слайді. Цій опції було виставлено значення 1, тобто потрібно відображати тільки одну картинку в слайді.

Ім’я власника автомобіля було зроблено у вигляді посилання на його особисту сторінку. Тобто в тег «a» було додано атрибут «routerLink», і встановлено йому значення «/user-profile» та додано id користувача для того, щоб сервер міг повертати дані про конкретного користувача. Для того, щоб отримати дані про користувача та вивести їх в шаблон, в методі «ngOnInit» здійснюється запит до сервера та в шлях додається id користувача, дані про якого потрібно вивести. Після того, як сервер поверне дані, вони записуються в змінну «data». Ця змінна – це об'єкт, в якому зберігаються такі дані про користувача: ім'я, електронна пошта, номер телефону та місце проживання. Для того щоб вивести

ці дані в шаблон було застосовано інтерполяцію Angular. Інтерполяція допомагає прив'язати вміст html-елемента до змінної. Вивід даних про користувача зображено на рисунку 3.24

```
  <svg-icon src="../../../assets/images/user.svg"></svg-icon>
  <span class="font-big">
    {{data.name}}
  </span>
</p>
<p class="user-options">
  <svg-icon src="../../../assets/images/mail.svg"></svg-icon>
  <a href="mailto: {{'joinauto2020@gmail.com'}}" title="{{'joinauto2020@gmail.com'}}">
      {{data.email}}
  </a>
</p>
<p class="user-options">
  <svg-icon src="../../../assets/images/phone.svg"></svg-icon>
  <a href="tel: {{'+38 098 22 45 697'}}" title="{{'+38 098 22 45 697'}}">
      {{data.phoneNumber}}
  </a>
</p>
<p class="user-options">
  <svg-icon src="../../../assets/images/map-pin.svg"></svg-icon>
    {{data.location}}
</p>
```

Рисунок 3.24 - Вивід даних про користувача

Також на сторінці користувача є список його товарів. Для того, щоб вивести список товарів, також було використано директиву «*ngFor» яка дозволяє перебрати масив з об'єктів, в яких зберігаються дані про автомобілі, які належать користувачу. В масиві є автомобілі як самого користувача, так і автомобілі, які користувач зашарив.

Для того, щоб користувач зміг продати свій автомобіль на сайті, було створено сторінку продажу автомобіля. На сторінці створено декілька кроків, які потрібно пройти, щоб зробити пост автомобіля для продажу. Перший крок – можливість додати фото автомобіля, а то і не одне. Щоб користувач зміг загрузити фотографії свого авто, в розмітку було додано інпут з типом «file».

Такий інпут дозволяє вибрати фото та загрузити його. Після того, як фото загрузиться, воно поміститься в масив «event.target.files». Для того, щоб витягти це фото з масиву та зберегти його у свій масив, було створено метод «addPhoto», який приймає об'єкт event, в якому якраз-таки знаходиться це фото. Для збереження всіх фото, які загружає користувач, створено змінну «photosList», яка на початку є пустим масивом. На інпут навішено подію «change», яка буде викликати метод addPhoto. В ньому реалізовано метод масивів «push», якому передано перше значення масива event.target.files. Тобто метод addPhoto буде додавати в масив всі файли, які завантажив користувач. Для того, щоб відобразити ці фото (файли) в шаблоні, за допомогою директиви *ngFor здійснюється перебір масиву та генерація html-елементів. Оскільки до інпутів в html можна прив'язати лейбл (label), то за допомогою Bootstrap класа «d-none» інпут для завантаження фото було приховано з шаблону, та до нього прив'язано label, якому додано стилі. При кліку на такий лейбл виконується емуляція кліку на самий інпут, тому приховання інпута з шаблону нічого не змінило. Користувачу також було додано можливість видалення фото, які він завантажив. Для цього до блока з файлом у верхньому правому куті створено кнопку «x», яку виділено червоним кольором, та при кліку по ній буде виконуватися метод «removePhoto», який містить логіку видалення елементів з масиву. А саме цей метод приймає індекс елемента, який потрібно видалити. Для того, щоб дізнатися цей індекс, цикл *ngFor зберігає індекси елементів, які перебирає, тобто в ньому було створено змінну index та передано її в метод «removePhoto». Директива *ngFor динамічно перебирає елементи масиву, тобто якщо з масива буде видалено елемент або навпаки додано, то цикл здійснить повторний прохід по такому масиву та перемалює html-шаблон, але не повністю весь, а тільки те, що змінилося. На рисунку 3.25 зображено методи додавання та видалення фото.

```
addPhoto(event) {
  this.photosList.push(event.target.files[0]);
}

removePhoto(index) {
  this.photosList.splice(index, deleteCount: 1);
}
```

Рисунок 3.25 - Додавання та видалення фото

Наступним кроком заповнення форми для продажу авто є додавання основної інформації про автомобіль. Щоб користувачу було простіше додавати інформацію, було використано компоненти «ng-autocomplete», які динамічно доповнюють те, що вводить користувач. Ці компоненти – це звичайні інпути, яким додано деяку логіку. Для того, щоб використати інпути автодоповнення, в шаблон додано тег «<ng-autocomplete>» та передано йому деякі дані у вигляді атрибутів, які було забайнджено, тобто здійснено прив'язку значень атрибутів до змінних в typescript файлі. Дані, які користувач повинен ввести – це тип транспорту, марка авто, модель авто, рік випуску, тип кузова, пробіг, двигун, трансмісія, привід, колір. Для того, щоб користувач знав які дані де вводити, для кожного інпута додано «placeholder», тобто підказку.

Останні два кроки – це додавання ціни автомобіля та опис автомобіля. Ціну користувач повинен ввести в доларах і для цього в інпут додано placeholder «Введіть ціну в $». Поле для додавання опису створено у вигляді html-тега textarea, тобто в ньому користувач зможе написати особливості автомобіля.

Під формою створено кнопку «Розмістити оголошення» з типом «submit», тобто при кліку на цю кнопку буде відбуватися підтвердження форми. Для того, щоб форма знала, що при кліку на цю кнопку потрібно виконувати якийсь метод, їй додано подію «ngSubmit». При настанні цієї події, тобто при кліку на кнопку, викличеться метод «submit», який прийме дані, які ввів користувач, та передасть ці дані на сервер за домогою rest API метода POST, який приймає об'єкт (body),

в який якраз-таки буде додано ці дані у форматі, який просить бекенд. Після того, як користувач заповнить форму, автомобіль буде опубліковано на сторінці з машинами, які доступні для продажу. Сторінку додавання оголошеня для продажу автомобіля зображено на рисунку 3.26.

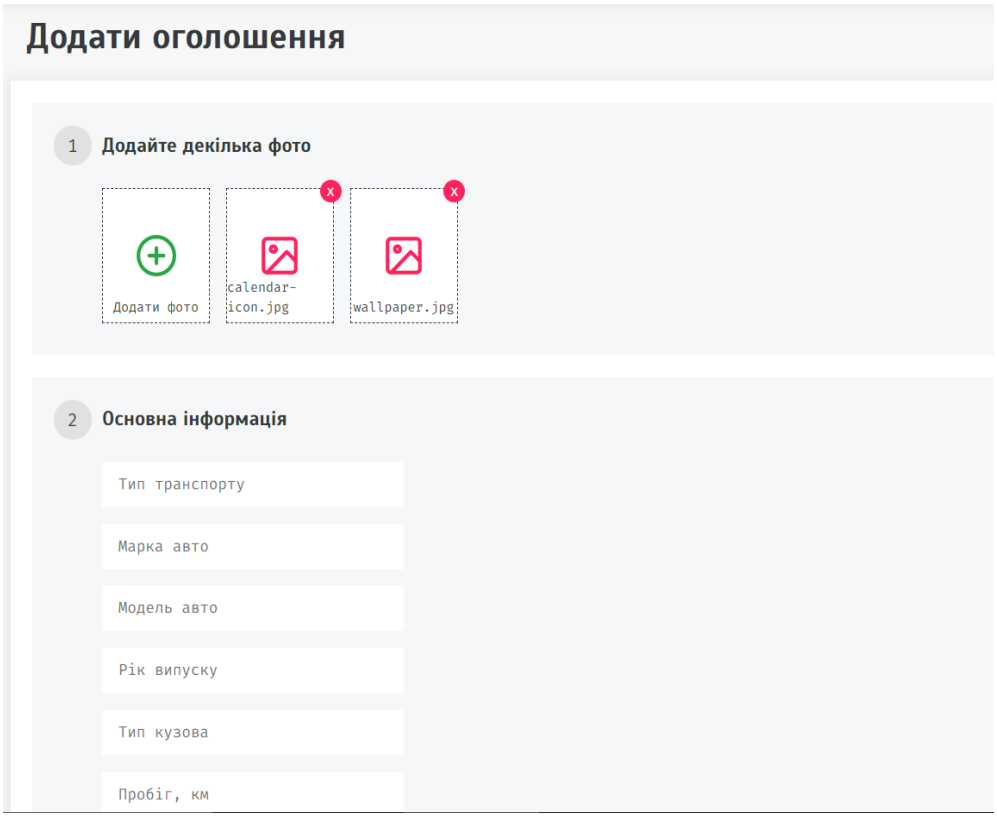

Рисунок 3.26 - Сторінка додавання оголошення для продажу авто

Для того, щоб користувач зміг замовити авто, було створено сторінку PostOrder, тобто таку сторінку на якій можна заповнити форму та зробити пост автомобіля для замовлення. Для компонента PostOrder створено окремий модуль, в якому задекларовано цей компонент. В головному модулі AppRoutingModule створено об'єкт, в якого «path», тобто шлях до модуля – «post-order». Тобто коли користувач натисне на кнопку «Замовити авто» у хедері веб-додатку, то він буде перенаправлений на сторінку замовлення авто, яка доступна за посиланням «post-order». Для модуля PostOrderModule також добалено

можливість лінивої загрузки для покращення оптимізації додатку. На сторінці замовлення авто присутні такі ж поля вводу, як і на сторінці продажу авто. Єдине, що було видалено - це можливість завантаження фото, оскільки у цьому немає потреби. Також змінено формат вводу року випуску автомобіля, а саме створено два інпути, які знаходяться один навпроти одного, та для них додано підказку, тобто placeholder «Рік від» та «Рік до». Завдяки цьому, користувач зможе зрозуміти, які дані потрібно вводити у ці поля. Також змінено формат вводу ціни. Ціну було розділено на «Ціна від» та «Ціна до», що дозволяє користувачу ввести приблизну ціну, яку він готовий заплатити за автомобіль. Внизу сторінки додано кнопку «Розмістити замовлення», якій звичайно ж додано тип «submit», оскільки при кліку на цю кнопку буде здійснюватися підтвердження форми, та викликатися однойменний метод «submit», який буде надсилати дані на сервер. Для того, щоб надіслати дані, які ввів користувач, використано метод «post», який присутній у пакеті HttpClient, та йому передано дані у об’єкт body. Сторінку замовлення авто зображено на рисунку 3.27.

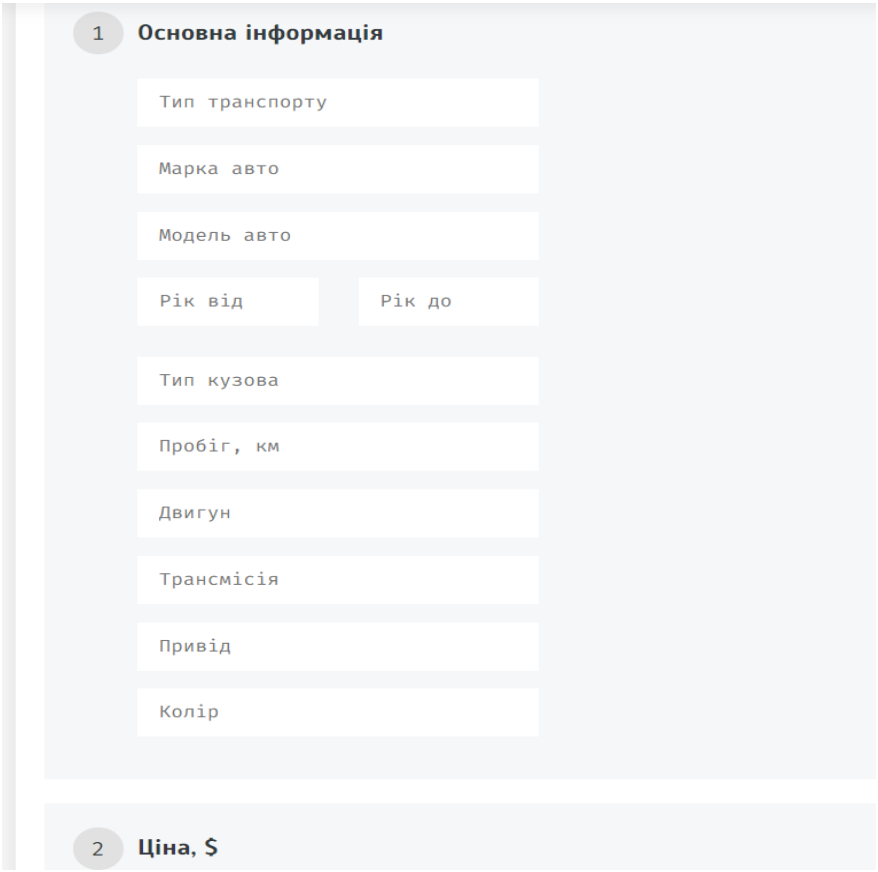

Рисунок 3.27 - Сторінка замовлення авто

В методі submit використано інший метод сервісу Router, який дозволяє здійснювати навігацію по сайті. А саме йому передано масив, в якому вказано шлях до строінки на яку потрібно перейти. Після того, як користувач зробить пост оголошення про замовлення, його потрібно перенаправити на сторінку замовлень, а отже в масиві було вказано шлях «order-list». Тобто після натиснення на кнопку «Розмістити замовлення» користувач потрапить на сторінку зі всіма замовленнями. При загрузці сторінки виконається метод «ngOnInit», в якому записано логіку завантаження даних для цієї сторінки, а саме було використано метод «get» з пакету HttpClient Angular, в який передано URL API, з якого потрібно завантажити дані. Після того, як дані завантажуться, виконається функція повторного виклику, в яку в якості параметра буде передано дані, що повернув сервер. Ці дані – це масив з об'єктів. Для того, щоб зберегти дані, які повернув сервер, було створено змінну, яку на початку проінізіалізовано

як порожній масив. Потім у колбек-функції здійснено переприсвоєння значення цієї змінної та присвоєно нове значення – масив, який повернув сервер. У шаблоні здійснено візуалізацію цих даних за допомогою циклу *ngFor, який згенерував картки з замовленнями.

Картки замовлень також було зроблено клікабельними, тобто у розмітці вони створені у вигляді тега «a» з атрибутом «routerLink», якому встановлено значення «order-page» та додано id автомобіля, який отримано з об’єкта з даними про автомобіль. Цей id потрібний для того, щоб вказати серверу, які дані йому потрібно повертати. Для замовлення, було створено окрему сторінку на якій водображуються всі дані про замовлення та пропозиції, які висувають користувачі щодо замовлення цього автомобіля.

4 БІЗНЕС ПЛАН

4.1 Резюме

Автобізнес з кожним роком все набуває популярності. Автомобіль – це та річ, без якої важко уявити теперішній світ. Люди звикли переміщатися за допомогою автомобілів та отримувати від цього задоволення, тому навряд чи настане час, коли людство відмовиться від цього виду транспорту.

Проект буде займатися наданням послуг з купівлі та продажу автомобілів. Він буде представлений у вигляді веб-сайту, на якому користувачі зможуть розміщувати свої автомобілі для продажу, або навпаки купувати автомобілі, які вже розміщені на веб-сайті. Також користувачі зможуть замовляти автомобілі по своєму бажанню та інші користувачі зможуть здійснювати замовлення. Цільовою аудиторією будуть люди які бажають продати чи купити автомобіль.

Оскільки проект будуть розробляти декілька людей, відповідно їм потрібно буде платити зарплату. На виплату зарплати потрібно близько двадцяти тисяч доларів. Також для того, щоб популяризувати проект, потрібно купити рекламу, яка буде коштувати в районі двадцяти тисяч доларів. Для того, щоб працівники змогли ефективно працювати, потрібно орендувати комфортний офіс, на який потрібно витратити п'ятнадцять тисяч доларів. Працівників потрібно забезпечити обладнанням для того, щоб вони могли працювати. На обляднання потрібно п'ятнадцять тисяч доларів.

Основним джерелом фінансових інвестицій будуть інвестори, з якими буде проведено переговори і запропоновано привабливий відсоток від розміру доходів. Оскільки проект володіє високою перспективою, то інвесторів, які згодяться на фінансування проекту буде декілька і відповідно стартовий капітал для розвитку проекту буде великий.

За перший рік роботи очікується близько сімдесят тисяч доларів чистого доходу. Загальний розмір витрат, потрібний для реалізації проекту – вісімдесят тисяч доларів. Формою організації майбутнього бізнесу буде діяльність фізичної особи підприємця. Для створення та підтримки проекту буде залучено шість осіб. Оскільки за перший рік очікується сімдесят тисяч доларів чистого прибутку, то на наступний рік сума повинна зрости у два – три рази. Тобто за другий рік проект отримає сто п'ятдесят – двісті тисяч доларів прибутку.

4.2 Маркетинг

Основним видом послуг, який буде запропоновано клієнту – це покупка та продаж автомобіля. Тобто клієнт зможе зайти на сайт та продати чи купити автомобіль. У випадку якщо він не знайде потрібний автомобіль, для нього буде присутня опція замовлення автомобіля. Головною сферою використання продукту буде сфера малого бізнесу.

Унікальною торговою пропозицією буде надання можливості користувачу абсолютно безплатно робити пости для продажу автомобіля та здійснювати покупку автомобіля або його замовлення. В можливості замовлення авто визначається унікальність проекту. Можливість безплатного розміщення автомобілів на сайті та їх купівля є основним видом послуг проекту.

Основним недоліком проекту буде здійснення шахрайства з боку користувачів. Для того, щоб уникнути такого виду діяльності, у кожного користувача буде власний рейтинг, за допомогою якого інші юзери сайту зможуть дізнатися чи потрібно довіряти такому користувачу чи ні.

Користувачами чи клієнтами проекту можуть бути люди зі всієї країни. Приблизний середній вік користувачів – 30 років. Це чоловіки з середнім рівнем доходів, які зацікавлені в покупці чи продажі автомобіля.

Основні конкуренти на ринку – це такі бренди автомобільної інтернет торгівлі, як AutoRia, RST, InfoCar, Auto Ua, Avto Bazar, Автопортал, UAvto. Всі вони реалізовують аналогічні послуги. Основним видом послуг цих брендів є пропозиції з продажу та покупки авто, проте вони не дають клієнтам можливості замовлення автомобіля. Основним джерелом доходів таких конкурентів є реклама.

Для того, щоб отримати бажаний об'єм доходів, потрібно популяризувати проект, тобто залучити велику аудиторію. За допомогою реклами дуже легко донести інформацію потенційним користувачам чи клієнтам за досить короткий проміжок часу та залучити їхню увагу. Реклама допоможе сформувати позицію на ринку та закріпитися в списку лідерів і відповідно збільшити аудиторію та дохід. Основною перевагою реклами буде економія часу, тобто не потрібно буде пропонувати послуги власноруч, а реклама зробить все сама. Для того, щоб реклама була максимально ефективною, в ній буде описано кожну особливість проекту та підтверджено все доказами. На рекламу буде виділено максимальну кількість коштів, тобто чим більше буде коштувати реклама, тим вона буде ефективнішою. Приблизна кількість грошей, які потрібно витратити на рекламу– 20 тисяч доларів. Для розміщення реклами вибрано соціальну мережу – Facebook. Це один з найпростіших та найефективніших варіантів для розміщення реклами, оскільки в цій соцмережі знаходиться переважна кількість потенційних клієнтів. За допомогою фейсбук можна розміщувати рекламу відносно розташування, віку, статі а також інтересів. Також для реклами буде використано ремаркетинг, тобто після того, як користувачі відвідають веб-сайт, їм в пошукових системах буде запропоновано відвідати цей сайт ще раз. Реклама в соціальній мережі Facebook допоможе значно підвищити кількість відвідувачів і потім такі відвідувачі перетворяться в клієнтів.

4.3 Обґрунтування необхідності фінансових вкладень

Оскільки проект – це вебсайт, тому розміщення тут неважливе. Тобто для того, щоб залучити велику аудиторію, розміщення для такого виду бізнесу відіграє неважливу роль. Важливу роль в такому роді бізнесу відіграють працівники. Від того, яка мотивація буде у працівників залежить якість проекту та можливість його підтримки в майбутньому. Для того, щоб працівники могли працювати на повну силу та з великим бажанням, потрібні комфортні умови. Для цього потрібно зробити оренду просторого офісу, який буде знаходитися близько до центру міста (розташування в такому випадку враховується) або парку. В кожного працівника повинні бути недешеві робочі інструменти, тобто комп'ютери з високими характеристиками продуктивності. Для покращення швидкості та комфорту розробки, в кожного працівника повинно бути мінімум два монітори на робочому місці. Таке обладнання буде закуплено в магазинах або замовлено через інтернет. У випадку хорошої якості таких товарів, з постачальниками буде здійснено кооперацію та розміщення реклами їх товарів на веб-сайті. Режим роботи компанії буде стандартний, тобто робочий день буде починатися в дев'ятій годині ранку та закінчуватися в шостій годині вечора. В тиждень буде два вихідні (субота та неділя). Державні свята також будуть вихідними. У випадку, якщо працівники будуть швидко та якісно виконувати роботу, їм буде нараховуватися премія та додаткові вихідні, що збільшить лояльність працівників до компанії та відповідно збільшить репутацію самої компанії. Для того, щоб купити обладнання для роботи працівників потрібно орієнтовно десять тисяч доларів. На оренду офісу потрібно витрачати тисячу доларів в місяць.

Для розробки проекту потрібні працівники, тобто програмісти. В IT сфері рівень працівників поділяється на: низький, середній і відповідно високий. Для того, щоб максимально якісно та швидко розробити проект в компанію потрібно найняти програмістів з середнім та високим рівнем, тому, що програмісти з

низьким рівнем, як правило роблять багато помилок і тому проект буде важко підтримувати в майбутньому, а якщо не гірше то взагалі його переробити і втратити дорогоцінний час та гроші. Краще витратити більшу суму грошей на етапі створення проекту, ніж залишити його без перспективи розвитку в майбутньому. Зараз ІТ дуже популярне, тому людей, які шукають роботу є досить багато. Для того, щоб знайти працівників, можна буде використати такі сайти пошуку роботи, як «Rabota.ua», «Work.ua» чи розмістити вакансію на ІТ порталі «DOU». Зарплату працівники будуть отримувати в кінці кожного місяця. Від того, який досвід у працівника, та його вміння, буде залежати зарплата. Можливітсь підвищення заробітньої плати також буде присутня. У випадку якщо працівник буде відчувати, що виконує роботу, за яку потрібно платити більше, він зможе попросити перегляд зарплати.

4.4 Нормативно-правові нюанси

Оскільки проект знаходиться на етапі ідеї, а не є готовим продуктом, тому для нього було вибрано ФОП як організаційно-правову форму. ФОП – це фізична особа підприємець, тобто особа, яка зареєстрвоана як підприємець та не має статусу юридичної особи. Така особа має право на здіснення підприємницької діяльності після того як їй виповниться 18 років. Також така особа може здійснювати найм працівників, що дуже важливо для проекту. В майбутньому можливо буде розглядатися варіант зі зміни організаційно-правової форми.

На початковому етапі вся діяльність щодо розробки проекту буде зосереджена в руках керівника. Тобто керівник буде один. Згодом для керування проектом буде найнято проектного менеджера, тобто людину, яка буде розподіляти завдання для розробки проекту між іншими людьми. Керівник після цього буде займатися вирішенням бізнес питань. Для розробки фронтенд

частини проекту буде найнято двох програмістів і для розробки бекенд частини також буде найнято двох людей. Проектний менеджер буде створювати завдання для них та записувати у таблицю. Після того, як розробники оглянуть завдання, їм потрібно буде поставити час, за який вони зможуть виконати такі завдання. В кінці дня ПМ зможе побачити та оцінити прогрес. На рисунку 4.1 зображено ієрархію проекту.

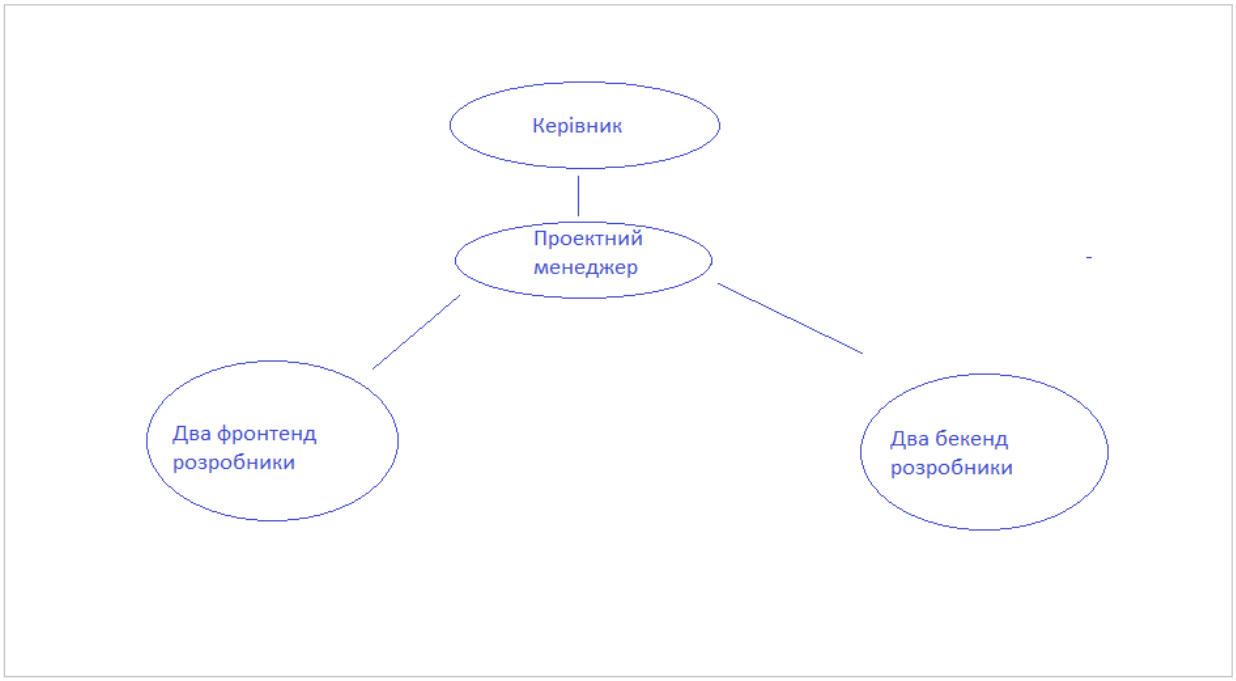

Рисунок 4.1- Ієрархія проекту

На рисунку 4.1 видно, що головну роль в проекті відіграє керівник, тобто та особа, яка займається його управлінням. Йому підкоряється проектний менеджер – особа, яка керує проектом та розподіляє завдання між програмістами. Програмістів розділено на дві групи: фронтед розробники та бекенд розробники. Ці дві групи програмістів підкоряються проектному менеджеру і звітують за зроблену роботу перед ним. Основна робота проектного менеджера – забезпечити правильний хід розробки проекту, тобто правильно розподілити час та завдання і врахувати всі нюанси.

4.5 Складання фінансового бюджету

Складання фінансового бюджету для проекту – один з найважливіших етапів його створення. Для того, щоб скласти правильний фінансовий бюджет, потрібно знайти схожий проект та проаналізувати і використати його модель фінансування. Основним джерелом фінансування будуть інвестори. Інвестор – це особа, яка вкладає кошти в проект з ціллю збереження цих коштів та отримання прибутку. Тобто інвестори примножують гроші за рахунок тих грошей, які вже є. Інвестори займаються пошуком перспективних проектів для того, щоб в них вкласти гроші та заробити ще більше. Така людина розуміє, що будь-які вкладення грошей можуть принести не тільки прибутки, а й втрати. Взагалі, будь-яка інвестиція це ризик. Але чим більший ризик, тим більший прибуток. Для того, щоб розробити проект, потрібно здійснити фінансові витрати, які зображено на рисунку 4.2.

№	Назва грошового вкладення	Вартість в $
1	Оренда офісу	15 000
2	Виплата зарплат працівникам	20 000
3	Покупка обладняння (комп'ютерів)	15 000
4	Покупка реклами	20 000
	Разом	**70 000**

Рисунок 4.2 - Фінансові витрати для проекту

Фінансові витрати для проекту складаються з таких пунктів:

1. Оренда офісу. Для того, щоб розробляти проект потрібний офіс, оренда якого обійдеться в 15 тисяч доларів за період розробки проекту.
2. Виплата зарплат працівникам. Працівникам потрібно буде виплатити близько двадцяти тисяч доларів.
3. Покупка обладнання. В офісі потрібні комп'ютери для того, щоб працівники могли на них розробляти проект.
4. Покупка реклами. Для того, щоб набрати аудиторію, тобто клієнтів, потрібна реклама, яка обійдеться в двадцять тисяч доларів.

В результаті отримаємо 70 тисяч доларів витрат. Найбільше коштів потрібно буде витратити на рекламу. На рекламі економити не можна, тому, що якісна реклама допоможе залучити велику кількість клієнтів, а це в свою чергу дозволить підвищити дохід.

Оскільки основним джерелом доходів буде також реклама, то збільшення аудиторії та відвідувачів призведе до збільшення прибутків. Приблизний дохід та аудиторію за перші три роки зображено на рисунку 4.3.

Рік	Аудиторія , кількість осіб	Прибуток в $
1	50 000	70 000
2	120 000	150 000
3	250 000	300 000

Рисунок 4.3 - Дохід та аудиторія за перші три роки

Перший рік буде найважчим для популяризації проекту, тому і дохід буде порівняно невеликий. Очікувана кількість користувачів які будуть відвідувати сайт – п'ятдесят тисяч. За таку кількість користувачів з реклами можна заробити сімдесят тисяч доларів. На другий рік аудиторія зросте вдвічі та дохід також зросте в два рази. Третій рік буде найприбутковішим, оскільки аудиторія буде дуже велика і відповідно кількість переглядів реклами також буде більшою порівняно з попередніми роками. Очікуваний дохід за третій рік – триста тисяч доларів.

ВИСНОВКИ

Автомобільний бізнес у наш час набуває все більшої популярності. Зв'язано це з тим, що автомобілі стали невід'ємною частиною життя людей. Зараз, напевно, немає людини, яка б не хотіла мати своє власне авто. Для когось автомобіль – це просто транспортний засіб, а для когось розкіш.

Автомобілі – це та річ, яку завжди будуть купувати та продавати. В даний час перепродаж авто є дуже популярним бізнесом. Такий вид діяльності дозволяє непогано заробляти на життя. Кількість автомобілів, які виставляють на продаж просто вражає. Тема купівлі та продажу авто є дуже актуальною, а ще більш актуальною є тема купівлі автомобілів закордоном та ввезення їх на територію України.

За допомогою статистики було з'ясовано, що переважна кількість людей купує автомобілі з європейського ринку. Також багато людей займаються бізнесом, пов'язаним з купівлею таких автомобілів та продажем. Це дозволило дійти до висновку, що автомобільному бізнесу потрібен продукт, за допомогою якого одні люди зможуть замовляти автомобілі з Європи, а інші зможуть виконувати замовлення та отримувати за це нагороду.

Проаналізувавши ринок, було досліджено, що у сайтів продажу авто немає функції замовлення автомобіля. Отже було вирішено створити веб-сайт, в якому буде така можливість, що дозволить людям, які бажають купити авто з Європи замовити його, а людям які займаються купівлею та продажем авто з-за кордону виконувати такі замовлення.

Для того, щоб реалізувати такий проект було використано JavaScript фреймворк Angular. Цей інструмент дозволив швидко та легко розробити додаток, який можна з легкістю масштабувати у майбутньому. Для веб-сайту було створено сучасний дизайн, та чудовий досвід користування, для того, щоб користувачі отримували позитивні емоції від взаємодії з продуктом.

Отже, можна дійти до висновку, що розроблений проект створить революцію у автобізнесі. Відтепер людям, які бажають придбати автомобіль з Європи не потрібно буде турбуватися про те, у кого і де замовити такий автомобіль, а людям, що займаються пригоном авто буде легко знаходити клієнтів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

REFERENCES

1. Документація JavaScript. URL: https://javascript.info/ (дата звернення: 15.01.2020)
2. Документація TypeScript. URL: https://www.typescriptlang.org/docs/home.html (дата звернення: 20.01.2020)
3. Документація Angular. URL: https://angular.io/ (дата звернення: 20.01.2020)
4. Застосування Angular CLI. URL: https://cli.angular.io/ (дата звернення: 20.01.2020)
5. REST API простою мовою. URL: https://codeguida.com/post/601 (дата звернення: 10.02.2020)
6. Вступ до HTML. URL: https://www.w3schools.com/html/html_intro.asp (дата звернення: 12.02.2020)
7. Основи CSS. URL: https://developer.mozilla.org/uk/docs/Learn/Getting_started_with_the_web/CSS_basics (дата звернення: 14.02.2020)
8. Робота з JSON. URL: https://developer.mozilla.org/ru/docs/Learn/JavaScript/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D1%8B/JSON (дата звернення: 20.02.2020)
9. ECMAScript 6. URL: https://www.w3schools.com/js/js_es6.asp (дата звернення: 15.03.2020)
10. Туторіал по UML. URL: https://www.tutorialspoint.com/uml/index.htm (дата звернення: 15.04.2020)

11. Огляд протоколу HTTP. URL:
https://developer.mozilla.org/ru/docs/Web/HTTP/Overview
(дата звернення: 20.04.2020)
12. Документація Bootstrap. URL: https://getbootstrap.com/
(дата звернення: 21.04.2020)
13. Документація Angular powered Bootstrap. URL: https://ng-bootstrap.github.io/#/home
(дата звернення: 21.04.2020)
14. Система питань і відповідей Stack Overflow. URL:
https://stackoverflow.com/
(дата звернення: 22.04.2020)
15. Марейн Хавербек. Виразний JavaScript: Сучасне введення в програмування. 2011. 436 с.
16. Бібо Б., Марас Й., Резіг Д. Секрети JavaScript ніндзя. 2013. 416 с.
17. Рейсиг Дж. JavaScript. Професійні прийоми програмування. 2009. 720 с.
18. M. Kozlenko, V. Tkachuk, and M. Dutchak, "Software implementation of microcomputer based intrusion detection and prevention system with binary neural network," in Proc. 2nd International Scientific-Practical Conference "Problems of Cyber Security of Information and Telecommunication Systems" (PCSITS), O. Oksiiuk et al, Eds. Taras Shevchenko National University of Kyiv, Kyiv, Ukraine, Apr. 11-12, 2019, pp. 371-373.
19. I. Lazarovich and Y. Nikolaychuk, "Method of randomization and its application for adaptive data compression," Second IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2003. Proceedings, Lviv, 2003, pp. 362-364, doi: 10.1109/IDAACS.2003.1249587.
20. П. Федорук і М. Дутчак, “Побудова бази знань адаптивних систем дистанційного навчання на основі фреймової та продукційної моделей

представлення знань,” Управляючі системи і машини (УСіМ), №5, с.35-42, 2012.

ДОДАТОК А

Код проекту

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import {DefaultLayoutModule} from './layouts/default-layout/default-
layout.module';
import {HomePageModule} from './routes/home-page/home-page.module';
import {BrowserAnimationsModule} from '@angular/platform-browser/animations';
import {HTTP_INTERCEPTORS, HttpClientModule} from '@angular/common/http';
import {TokenInterceptorService} from './services/token-interceptor.service';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    DefaultLayoutModule,
    HomePageModule,
    BrowserAnimationsModule,
    HttpClientModule
  ],
  providers: [
```

```
        { provide: HTTP_INTERCEPTORS, useClass: TokenInterceptorService, multi:
    true },

      ],

      bootstrap: [AppComponent]

    })

    export class AppModule { }
```

```
import {NgModule} from '@angular/core';

import {Routes, RouterModule} from '@angular/router';

import {DefaultLayoutComponent} from './layouts/default-layout/default-
layout.component';

import {HomePageComponent} from './routes/home-page/home-page.component';

import {AuthGuardService} from './services/auth.guard';




const routes: Routes = [

  {

    path: '', component: DefaultLayoutComponent, children: [

      {path: '', component: HomePageComponent},

      {path: 'vehicle-list', loadChildren: () => import('./routes/vehicle-
list/vehicle-list.module').then(m => m.VehicleListModule)},

      {path: 'product-page', loadChildren: () => import('./routes/product-
page/product-page.module').then(m => m.ProductPageModule)},

      {path: 'post-sell', loadChildren: () => import('./routes/post-sell/post-
sell.module').then(m => m.PostSellModule)},

      {path: 'post-order', loadChildren: () => import('./routes/post-order/post-
order.module').then(m => m.PostOrderModule)},

      {

        path: 'order-page',

        loadChildren: () => import('./routes/order-product-page/order-product-
page.module').then(m => m.OrderProductPageModule)

      },

      {

        path: 'order-list',

        loadChildren: () => import('./routes/order-list/order-
list.module').then(m => m.OrderListModule)
```

```
      },
      {
        path: 'user-profile', loadChildren: () => import('./routes/user-
profile/user-profile.module').then(m => m.UserProfileModule)
      }
    ]
  },
  {
    path: 'login',
    loadChildren: () => import('./routes/login-page/login-page.module').then(m
=> m.LoginPageModule), canActivate: [!AuthGuardService]
  },
  {
    path: 'sign-up', loadChildren: () => import('./routes/sign-up/sign-
up.module').then(m => m.SignUpModule), canActivate: [!AuthGuardService]
  }
];


@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {
}
import { Injectable } from '@angular/core';
import {Router, CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot} from
'@angular/router';
import { AuthService } from './auth.service';
@Injectable(
  {providedIn: 'root'}
)

export class AuthGuardService implements CanActivate {
  constructor(public auth: AuthService, public router: Router) {}
  canActivate(
    route: ActivatedRouteSnapshot,
```

```
    state: RouterStateSnapshot
  ): boolean {
    if (!this.auth.isAuthenticated()) {
      this.router.navigate(['/login']);
      return false;
    } else {
      this.router.navigate(['/']);
      return true;
    }
  }
}
import {Injectable} from '@angular/core';
import {HttpClient} from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  public baseUrl = 'https://join-auto.herokuapp.com/v1';

  constructor(private http: HttpClient) {
  }

  isAuthenticated() {
    return !!localStorage.getItem('user_token');
  }

  register(body) {
    return this.http.post(`${this.baseUrl}/sign-up`, body);
  }

  login(body) {
    return this.http.post(`${this.baseUrl}/login`, body);
  }
}
```

```
import { Injectable } from '@angular/core';

import {HttpEvent, HttpHandler, HttpInterceptor, HttpRequest} from
'@angular/common/http';

import {Observable} from 'rxjs';


@Injectable()

export class TokenInterceptorService implements HttpInterceptor {


  intercept(req: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {

    const userToken = localStorage.getItem('user_token');

    const modifiedReq = req.clone({

      headers: req.headers.set('Authorization', `${userToken}`),

    });

    return next.handle(modifiedReq);

  }

}

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import {RouterModule} from '@angular/router';

import {ReactiveFormsModule} from '@angular/forms';

import {SignUpComponent} from './sign-up.component';



@NgModule({

  declarations: [

    SignUpComponent

  ],

  imports: [

    CommonModule,

    RouterModule.forChild([

      {path: '', component: SignUpComponent}

    ]),

    ReactiveFormsModule

  ],
```

```
  exports: [SignUpComponent]
})
export class SignUpModule { }
import {Component, OnInit} from '@angular/core';
import {FormControl, FormGroup, Validators} from '@angular/forms';
import {AuthService} from '../../services/auth.service';
import {Router} from '@angular/router';

@Component({
  selector: 'app-sign-up',
  templateUrl: './sign-up.component.html',
  styleUrls: ['./sign-up.component.scss']
})
export class SignUpComponent implements OnInit {
  public form: FormGroup;
  public isLoaded = true;
  public serverErrors = {};

  constructor(private authService: AuthService, private router: Router) {
  }

  submit() {
    if (this.form.valid) {
      this.isLoaded = false;
      console.log('sending');
      this.authService.register({
        email: this.form.get('email').value,
        user_name: this.form.get('name').value,
        password: this.form.get('password').value,
        phone_number: this.form.get('phoneNumber').value,
        location: this.form.get('location').value
      }).subscribe(res => {
        localStorage.setItem('user_token', 'auth_token');
        this.router.navigate(['/']);
      }, (errors) => {
```

```
        for (const error of errors.error) {
          this.serverErrors[error.field] = error.error;
        }
      });
    } else {
      this.form.markAllAsTouched();
    }
  }

  ngOnInit() {
    this.form = new FormGroup({
      email: new FormControl('', [Validators.required, Validators.email]),
      password: new FormControl('', [Validators.required,
Validators.minLength(6)]),
      name: new FormControl('', [Validators.required]),
      phoneNumber: new FormControl('', [Validators.required,
Validators.pattern('^\\+?3?8?(0\\d{9})$')]),
      location: new FormControl('', [Validators.required,
Validators.minLength(5)])
    });
  }
}
<section class="login-section">
  <div class="login-form-wrapper">
    <h1 class="logo-title">Join<span>Auto</span></h1>
    <h1 class="title">Зареєструватися</h1>
    <form [formGroup]="form" (ngSubmit)="submit()">
      <div class="form-input-wrapper">
        <input type="text" class="form-input" formControlName="name"
placeholder="Ім'я">
        <div *ngIf="form.get('name').invalid && form.get('name').touched"
class="error-block">
          <p *ngIf="form.get('name').errors.required"  class="error">Введіть
ім'я</p>
        </div>
        <div *ngIf="serverErrors.user_name" class="error-block">
          <p class="error">{{serverErrors.user_name}}</p>
```

```
        </div>
      </div>
      <div class="form-input-wrapper">
        <input type="text" class="form-input" formControlName="email"
placeholder="Ел.Пошта" (input)="serverErrors.email = null">
        <div *ngIf="form.get('email').invalid && form.get('email').touched"
class="error-block">
          <p *ngIf="form.get('email').errors.email" class="error">Введіть
правильний email</p>
          <p *ngIf="form.get('email').errors.required"  class="error">Введіть
email</p>
        </div>
        <div *ngIf="serverErrors.email" class="error-block">
          <p class="error">{{serverErrors.email}}</p>
        </div>
      </div>
      <div class="form-input-wrapper">
        <input type="password" class="form-input" formControlName="password"
placeholder="Пароль">
        <div *ngIf="form.get('password').invalid &&
form.get('password').touched" class="error-block">
          <p *ngIf="form.get('password').errors.minlength" class="error">Пароль
повинен містити не менше 6 символів</p>
          <p *ngIf="form.get('password').errors.required"  class="error">Введіть
пароль</p>
        </div>
        <div *ngIf="serverErrors.password" class="error-block">
          <p class="error">{{serverErrors.password}}</p>
        </div>
      </div>
      <div class="form-input-wrapper">
        <input type="text" class="form-input" formControlName="location"
placeholder="Місце проживання">
        <div *ngIf="form.get('location').invalid &&
form.get('location').touched" class="error-block">
          <p *ngIf="form.get('location').errors.required"
class="error">Заповніть поле</p>
          <p *ngIf="form.get('location').errors.minlength"
class="error">Заповніть поле</p>
```

```
        </div>
        <div *ngIf="serverErrors.location" class="error-block">
          <p class="error">{{serverErrors.location}}</p>
        </div>
      </div>
      <div class="form-input-wrapper">
        <input type="number" class="form-input" formControlName="phoneNumber"
placeholder="Номер телефону">
        <div *ngIf="form.get('phoneNumber').invalid &&
form.get('phoneNumber').touched" class="error-block">
          <p *ngIf="form.get('phoneNumber').errors.required"
class="error">Введіть номер телефону</p>
          <p *ngIf="form.get('phoneNumber').errors.pattern"
class="error">Введіть правильний номер телефону</p>
        </div>
        <div *ngIf="serverErrors.phone_number" class="error-block">
          <p class="error">{{serverErrors.phone_number}}</p>
        </div>
      </div>
      <button type="submit" class="submit-btn">Зареєструватися</button>
    </form>
    <p class="bottom-hint">Є аккаунт? <a routerLink="/login">Увійти</a></p>
  </div>
</section>
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import {RouterModule} from '@angular/router';
import {LoginPageComponent} from './login-page.component';
import {ReactiveFormsModule} from '@angular/forms';


@NgModule({
  declarations: [
    LoginPageComponent
  ],
  imports: [
```

```
    CommonModule,
    RouterModule.forChild([
      {path: '', component: LoginPageComponent}
    ]),
    ReactiveFormsModule
  ],
  exports: [LoginPageComponent]
})
export class LoginPageModule { }
import {Component, OnInit} from '@angular/core';
import {FormControl, FormGroup, Validators} from '@angular/forms';
import {AuthService} from '../../services/auth.service';
import {Router} from '@angular/router';

@Component({
  selector: 'app-login-page',
  templateUrl: './login-page.component.html',
  styleUrls: ['./login-page.component.scss']
})
export class LoginPageComponent implements OnInit {
  public form: FormGroup;
  public serverErrors = {};

  constructor(private authService: AuthService, private router: Router) {
  }

  ngOnInit() {
    this.form = new FormGroup({
      email: new FormControl('', [Validators.required, Validators.email]),
      password: new FormControl('', [Validators.required])
    });
  }

  submit() {
    if (this.form.valid) {
```

```
      this.authService.login(
        {
          email: this.form.get('email').value,
          password: this.form.get('password').value
        }
      )
        .subscribe( res => {
          this.router.navigate(['/home-page']);
        }, error => {
          for (const errorField of error.error) {
            this.serverErrors[errorField.field] = errorField.error;
          }
        })
    } else {
      this.form.markAllAsTouched();
    }
  }
}
<section class="login-section">
  <div class="login-form-wrapper">
    <h1 class="logo-title">Join<span>Auto</span></h1>
    <h1 class="title">Увійти</h1>
    <form [formGroup]="form" (ngSubmit)="submit()">
      <div class="form-input-wrapper">
        <input type="text" class="form-input" formControlName="email"
placeholder="Ел.Пошта" (input)="serverErrors = {}">
        <div *ngIf="form.get('email').invalid && form.get('email').touched"
class="error-block">
          <p *ngIf="form.get('email').errors.email" class="error">Введіть
коректний email</p>
          <p *ngIf="form.get('email').errors.required"  class="error">Введіть
email</p>
        </div>
        <div *ngIf="serverErrors.email" class="error-block">
          <p class="error">{{serverErrors.email}}</p>
        </div>
```

```
      </div>
      <div class="form-input-wrapper">
        <input type="password" class="form-input" formControlName="password"
placeholder="Пароль">
        <div *ngIf="form.get('password').invalid &&
form.get('password').touched" class="error-block">
          <p *ngIf="form.get('password').errors.required"  class="error">Введіть
пароль</p>
        </div>
      </div>
      <button type="submit" class="submit-btn">Увійти</button>
    </form>
    <p class="bottom-hint">Немає аккаунта? <a routerLink="/sign-
up">Зареєструватися</a></p>
  </div>
</section>
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import {HomePageComponent} from './home-page.component';
import {FormsModule, ReactiveFormsModule} from '@angular/forms';
import {AutocompleteLibModule} from 'angular-ng-autocomplete';
import {SharedModule} from '../../shared/shared.module';
import {NgbCollapseModule} from '@ng-bootstrap/ng-bootstrap';


@NgModule({
  declarations: [
    HomePageComponent
  ],
  imports: [
    CommonModule,
    FormsModule,
    AutocompleteLibModule,
    ReactiveFormsModule,
    SharedModule,
    NgbCollapseModule
  ],
```

```
  exports: [HomePageComponent]
})
export class HomePageModule { }
import {Component, OnInit} from '@angular/core';
import {FormControl, FormGroup} from '@angular/forms';

@Component({
  selector: 'app-home-page',
  templateUrl: './home-page.component.html',
  styleUrls: ['./home-page.component.scss']
})
export class HomePageComponent implements OnInit {
  public keywords = 'name';
  private form: FormGroup;
  public isCollapsed = false;
  public productsList = [];
  typeData = [
    {
      id: 1,
      name: 'Будь-який'
    },
    {
      id: 2,
      name: 'Легкові'
    },
    {
      id: 3,
      name: 'Мото'
    },
    {
      id: 4,
      name: 'Грузовики'
    }
  ];
```

```
markData = [
  {
    id: 1,
    name: 'BMW'
  },
  {
    id: 2,
    name: 'Mercedes'
  },
  {
    id: 3,
    name: 'Audi'
  },
  {
    id: 4,
    name: 'Volkswagen'
  }
];

modelData = [
  {
    id: 1,
    name: ''
  },
  {
    id: 2,
    name: ''
  },
  {
    id: 3,
    name: ''
  },
  {
    id: 4,
    name: ''
```

```
    }
  ];

  yearFrom = [
    {name: '2020'}
  ]
  constructor() { }

  ngOnInit() {
    this.form = new FormGroup({
      type: new FormControl(),
      mark: new FormControl(),
      model: new FormControl(),
      region: new FormControl(),
      yearFrom: new FormControl(),
      yearTo: new FormControl(),
      priceFrom: new FormControl(),
      priceTo: new FormControl()
    })
  }

  submit() {
    console.log(this.form)
  }

}
          >
                  </ng-autocomplete>

                  <ng-template #priceToTemplate let-item>
                    <a [innerHTML]="item.name"></a>
                  </ng-template>
                </div>
              </div>
              <div class="button-wrapper d-flex justify-content-end">
```

```
                <button class="button btn-alternative mt-4 w-50">Пошук</button>
              </div>
            </div>
          </div>
        </form>
      </div>
    </div>
  </div>
</div>
<section class="recently-added">
  <div class="container">
    <div class="row">
      <div class="col-12">
        <h1 class="title">Недавно додані</h1>
      </div>
      <div class="col-12">
        <div class="row">
          <div *ngFor="let product of [1,2,4,5,4,4,1,2]" class="products-
container col-sm-6 col-md-4 col-xl-3 mb-4">
            <app-product-card></app-product-card>
          </div>
          <div class="col-12" id="collapseExample" [ngbCollapse]="!isCollapsed">
            <div class="row">
              <div *ngFor="let product of [1,2,4,5,4,4,1,2]" class="products-
container col-sm-6 col-md-4 col-xl-3 mb-4">
                <app-product-card></app-product-card>
              </div>
            </div>
          </div>
        </div>
      </div>
      <div class="col-12 d-flex justify-content-center">
        <button *ngIf="!isCollapsed" class="button btn-alternative mt-4"
                (click)="isCollapsed = !isCollapsed"
                [attr.aria-expanded]="!isCollapsed" aria-
controls="collapseExample">
```

```
          Показати більше...
        </button>
      </div>
    </div>
  </div>
</section>
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import {RouterModule} from '@angular/router';
import {ReactiveFormsModule} from '@angular/forms';
import {AutocompleteLibModule} from 'angular-ng-autocomplete';
import {SharedModule} from '../../shared/shared.module';
import {NgbPaginationModule} from '@ng-bootstrap/ng-bootstrap';
import {OrderListComponent} from './order-list.component';

@NgModule({
  declarations: [
    OrderListComponent
  ],
  imports: [
    CommonModule,
    RouterModule.forChild([
      {path: '', component: OrderListComponent}
    ]),
    ReactiveFormsModule,
    AutocompleteLibModule,
    SharedModule,
    NgbPaginationModule
  ],
  exports: [OrderListComponent]
})
export class OrderListModule { }
<section class="section">
<div class="container">
  <div class="row">
```

```
    <div class="col-12">
      <h2 class="title">Список замовлень</h2>
        <div class="row">
          <div *ngFor="let product of [1]" class="col-12 col-lg-3 mb-4">
            <app-product-card></app-product-card>
          </div>
        </div>

        <div class="row">
          <div class="col-12 d-flex justify-content-center">
            <ngb-pagination [collectionSize]="120" [(page)]="page" [maxSize]="2"
                            [boundaryLinks]="true"></ngb-pagination>
          </div>
        </div>
      </div>
  </div>
</div>
</section>
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import {OrderProductPageComponent} from './order-product-page.component';
import {RouterModule} from '@angular/router';
import {AngularSvgIconModule} from 'angular-svg-icon';
import {NgbRatingModule} from '@ng-bootstrap/ng-bootstrap';
import {CarouselModule} from 'ngx-owl-carousel-o';

@NgModule({
  declarations: [
    OrderProductPageComponent
  ],
  imports: [
    CommonModule,
    RouterModule.forChild([
      {path: '', component: OrderProductPageComponent}
    ]),
```

```
    AngularSvgIconModule,
    NgbRatingModule,
    CarouselModule
  ],
  exports: [OrderProductPageComponent]
})
export class OrderProductPageModule { }
    <div class="shadow-box background">
                        <div class="offer-person-wrapper">
                          <div class="img-wrapper">
                            <svg-icon src="../../../assets/images/user.svg"></svg-
icon>
                          </div>
                          <p class="name">Іван</p>
                        </div>
                        <p class="gray-text">Сума: <span class="green-text">20
000$</span></p>
                        <p class="gray-text">Коментарій: <span class="dark-
text">Готовий купити даний автомобіль за вказану суму</span></p>
                      </div>
                    </li>
                  </ul>
                </div>
              </div>
            </div>
          </div>
      </div>
    </div>
  </div>
</section>
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { PostOrderComponent } from './post-order.component';
import {RouterModule} from '@angular/router';
import {AngularSvgIconModule} from 'angular-svg-icon';
import {AutocompleteLibModule} from 'angular-ng-autocomplete';
```

```
import {FormsModule, ReactiveFormsModule} from '@angular/forms';

@NgModule({
  declarations: [
    PostOrderComponent
  ],
  imports: [
    CommonModule,
    RouterModule.forChild([
      {path: '', component: PostOrderComponent}
    ]),
    AngularSvgIconModule,
    AutocompleteLibModule,
    FormsModule,
    ReactiveFormsModule
  ],
  exports: [PostOrderComponent]
})
export class PostOrderModule { }
import { Component, OnInit } from '@angular/core';
import {FormControl, FormGroup} from '@angular/forms';

@Component({
  selector: 'app-post-order',
  templateUrl: './post-order.component.html',
  styleUrls: ['./post-order.component.scss']
})
export class PostOrderComponent implements OnInit {
  public keywords = 'name';
  form: FormGroup;
  data = [
    {
      id: 1,
      name: 'Usa'
    },
```

```
    {
      id: 2,
      name: 'England'
    }
  ];

  constructor() { }

  ngOnInit() {
    this.form = new FormGroup({
      vehicleType: new FormControl(),
      name: new FormControl(),
      model: new FormControl(),
      yearFrom: new FormControl(),
      yearTo: new FormControl(),
      bodyType: new FormControl(),
      mileage: new FormControl(),
      engine: new FormControl(),
      transmittion: new FormControl(),
      wheelDrive: new FormControl(),
      color: new FormControl(),
      priceFrom: new FormControl(),
      priceTo: new FormControl()
    });
  }

  submit() {

  }
}
pe="number" class="input-default" placeholder="Пробіг, км"
formControlName="mileage">
                </div>

                <div class="select-wrapper">
```

```
    <ng-autocomplete
      formControlName="engine"
      placeHolder="Двигун"
      [data]="[1,2,3]"
      [searchKeyword]="keywords"
      [itemTemplate]="engineTemplate"
      [notFoundTemplate]="notFoundEngineTemplate">
    </ng-autocomplete>

    <ng-template #engineTemplate let-item>
      <a [innerHTML]="item.name"></a>
    </ng-template>

    <ng-template #notFoundEngineTemplate>
      <div [innerHTML]="'Не знайдено'"></div>
    </ng-template>
  </div>

  <div class="select-wrapper">
    <ng-autocomplete
      formControlName="transmittion"
      placeHolder="Трансмісія"
      [data]="[1,2,3]"
      [searchKeyword]="keywords"
      [itemTemplate]="transmittionTemplate"
      [notFoundTemplate]="notFoundTransmittionTemplate">
    </ng-autocomplete>

    <ng-template #transmittionTemplate let-item>
      <a [innerHTML]="item.name"></a>
    </ng-template>

    <ng-template #notFoundTransmittionTemplate>
      <div [innerHTML]="'Не знайдено'"></div>
    </ng-template>
```

```
</div>

<div class="select-wrapper">
  <ng-autocomplete
    formControlName="wheelDrive"
    placeHolder="Привід"
    [data]="[1,2,3]"
    [searchKeyword]="keywords"
    [itemTemplate]="wheelDriveTemplate"
    [notFoundTemplate]="notFoundWheelDriveTemplate">
  </ng-autocomplete>

  <ng-template #wheelDriveTemplate let-item>
    <a [innerHTML]="item.name"></a>
  </ng-template>

  <ng-template #notFoundWheelDriveTemplate>
    <div [innerHTML]="'Не знайдено'"></div>
  </ng-template>
</div>

<div class="select-wrapper">
  <ng-autocomplete
    formControlName="color"
    placeHolder="Колір"
    [data]="[1,2,3]"
    [searchKeyword]="keywords"
    [itemTemplate]="colorTemplate"
    [notFoundTemplate]="notFoundColorTemplate">
  </ng-autocomplete>

  <ng-template #colorTemplate let-item>
    <a [innerHTML]="item.name"></a>
  </ng-template>
```

```
                <ng-template #notFoundColorTemplate>
                  <div [innerHTML]="'Не знайдено'"></div>
                </ng-template>
              </div>
            </div>
          </div>

          <div class="section-box">
            <div class="header">
              <div class="section-number"><span>2</span></div>
              <h2 class="section-title">
                Ціна, $
              </h2>
            </div>
            <div class="select-info-wrapper">
              <div class="select-wrapper d-flex justify-content-between">
                <input type="number" class="input-default double-input"
placeholder="Ціна від" formControlName="priceFrom">
                <input type="number" class="input-default double-input"
placeholder="Ціна до" formControlName="priceTo">
              </div>
            </div>
          </div>

          <div class="section-box">
            <div class="header">
              <div class="section-number"><span>3</span></div>
              <h2 class="section-title">
                Додайте коментар
              </h2>
            </div>
            <div class="select-info-wrapper">
              <div class="select-wrapper textarea-wrapper">
                <textarea class="input-default" cols="30" rows="40"
placeholder="Опис"></textarea>
              </div>
```

```
              </div>
            </div>
            <div class="d-flex justify-content-center">
              <button type="submit" class="button btn-alternative hover-
effect">Розмістити замовлення</button>
            </div>
          </form>
        </div>
      </div>
    </div>
  </div>
</section>
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import {PostSellComponent} from './post-sell.component';
import {RouterModule} from '@angular/router';
import {AngularSvgIconModule} from 'angular-svg-icon';
import {AutocompleteLibModule} from 'angular-ng-autocomplete';
import {FormsModule, ReactiveFormsModule} from '@angular/forms';


@NgModule({
  declarations: [
    PostSellComponent
  ],
  imports: [
    CommonModule,
    RouterModule.forChild([
      {path: '', component: PostSellComponent}
    ]),
    AngularSvgIconModule,
    AutocompleteLibModule,
    FormsModule,
    ReactiveFormsModule
  ],
  exports: [PostSellComponent]
```

```
})
export class PostSellModule { }
import {Component, OnInit} from '@angular/core';
import {FormControl, FormGroup} from '@angular/forms';

@Component({
  selector: 'app-post-sell',
  templateUrl: './post-sell.component.html',
  styleUrls: ['./post-sell.component.scss']
})
export class PostSellComponent implements OnInit {
  public photosList = [];
  public keywords = 'name';
  form: FormGroup;
  data = [
    {
      id: 1,
      name: 'Usa'
    },
    {
      id: 2,
      name: 'England'
    }
  ];

  constructor() {
  }

  addPhoto(event) {
    this.photosList.push(event.target.files[0]);
  }

  removePhoto(index) {
    this.photosList.splice(index, 1);
  }
```

```
  ngOnInit() {
    this.form = new FormGroup({
      vehicleType: new FormControl(),
      name: new FormControl(),
      model: new FormControl(),
      year: new FormControl(),
      bodyType: new FormControl(),
      mileage: new FormControl(),
      engine: new FormControl(),
      transmittion: new FormControl(),
      wheelDrive: new FormControl(),
      color: new FormControl(),
      price: new FormControl()
    });
  }

  submit() {
    console.log(this.form)
  }
}
           <ng-template #notFoundBodyTypeTemplate>
                    <div [innerHTML]="'Не знайдено'"></div>
                  </ng-template>
                </div>

                <div class="select-wrapper">
                    <input type="number" class="input-default"
placeholder="Пробіг, км" formControlName="mileage">
                </div>

                <div class="select-wrapper">
                  <ng-autocomplete
                    formControlName="engine"
                    placeHolder="Двигун"
```

```
    [data]="[1,2,3]"
    [searchKeyword]="keywords"
    [itemTemplate]="engineTemplate"
    [notFoundTemplate]="notFoundEngineTemplate">
  </ng-autocomplete>

  <ng-template #engineTemplate let-item>
    <a [innerHTML]="item.name"></a>
  </ng-template>

  <ng-template #notFoundEngineTemplate>
    <div [innerHTML]="'Не знайдено'"></div>
  </ng-template>
</div>

<div class="select-wrapper">
  <ng-autocomplete
    formControlName="transmittion"
    placeHolder="Трансмісія"
    [data]="[1,2,3]"
    [searchKeyword]="keywords"
    [itemTemplate]="transmittionTemplate"
    [notFoundTemplate]="notFoundTransmittionTemplate">
  </ng-autocomplete>

  <ng-template #transmittionTemplate let-item>
    <a [innerHTML]="item.name"></a>
  </ng-template>

  <ng-template #notFoundTransmittionTemplate>
    <div [innerHTML]="'Не знайдено'"></div>
  </ng-template>
</div>

<div class="select-wrapper">
```

```
    <ng-autocomplete
      formControlName="wheelDrive"
      placeHolder="Привід"
      [data]="[1,2,3]"
      [searchKeyword]="keywords"
      [itemTemplate]="wheelDriveTemplate"
      [notFoundTemplate]="notFoundWheelDriveTemplate">
    </ng-autocomplete>

    <ng-template #wheelDriveTemplate let-item>
      <a [innerHTML]="item.name"></a>
    </ng-template>

    <ng-template #notFoundWheelDriveTemplate>
      <div [innerHTML]="'Не знайдено'"></div>
    </ng-template>
  </div>

  <div class="select-wrapper">
    <ng-autocomplete
      formControlName="color"
      placeHolder="Колір"
      [data]="[1,2,3]"
      [searchKeyword]="keywords"
      [itemTemplate]="colorTemplate"
      [notFoundTemplate]="notFoundColorTemplate">
    </ng-autocomplete>

    <ng-template #colorTemplate let-item>
      <a [innerHTML]="item.name"></a>
    </ng-template>

    <ng-template #notFoundColorTemplate>
      <div [innerHTML]="'Не знайдено'"></div>
    </ng-template>
```

```
            </div>
          </div>
        </div>

        <div class="section-box">
          <div class="header">
            <div class="section-number"><span>3</span></div>
            <h2 class="section-title">
              Ціна, $
            </h2>
          </div>
          <div class="select-info-wrapper">
            <div class="select-wrapper">
              <input type="number" class="input-default"
placeholder="Введіть ціну в $" formControlName="price">
            </div>
          </div>
        </div>

        <div class="section-box">
          <div class="header">
            <div class="section-number"><span>4</span></div>
            <h2 class="section-title">
              Додайте опис
            </h2>
          </div>
          <div class="select-info-wrapper">
            <div class="select-wrapper textarea-wrapper">
              <textarea class="input-default" cols="30" rows="40"
placeholder="Опис"></textarea>
            </div>
          </div>
        </div>
        <div class="d-flex justify-content-center">
        <button type="submit" class="button btn-alternative hover-
effect">Розмістити оголошення</button>
```

```
            </div>
          </form>
        </div>
      </div>
    </div>
  </div>
</section>
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import {RouterModule} from '@angular/router';
import {ProductPageComponent} from './product-page.component';
import {AngularSvgIconModule} from 'angular-svg-icon';
import {NgbRatingModule} from '@ng-bootstrap/ng-bootstrap';
import {CarouselModule} from 'ngx-owl-carousel-o';
import {ViewPhotosModalModule} from '../../modals/view-photos-modal/view-photos-
modal.module';



@NgModule({
  declarations: [
    ProductPageComponent
  ],
  imports: [
    CommonModule,
    RouterModule.forChild([
      {path: '', component: ProductPageComponent}
    ]),
    AngularSvgIconModule,
    NgbRatingModule,
    CarouselModule,
    ViewPhotosModalModule
  ],
  exports: [ProductPageComponent]
})
export class ProductPageModule { }
```

```
import {Component, OnInit} from '@angular/core';
import {OwlOptions} from 'ngx-owl-carousel-o';
import {NgbModal} from '@ng-bootstrap/ng-bootstrap';
import {ViewPhotosModalComponent} from '../../modals/view-photos-modal/view-photos-modal.component';


@Component({
  selector: 'app-product-page',
  templateUrl: './product-page.component.html',
  styleUrls: ['./product-page.component.scss']
})
export class ProductPageComponent implements OnInit {
  customOptions: OwlOptions = {
    loop: false,
    mouseDrag: true,
    touchDrag: true,
    pullDrag: true,
    dots: false,
    navSpeed: 700,
    navText: ['<', '>'],
    responsive: {
      0: {
        items: 1
      }
    },
    nav: true
  };

  constructor(
    private modal: NgbModal
  ) {
  }

  ngOnInit() {
  }
```

```
}
            </p>
               </div>
             </div>


             <div class="shadow-box">
               <div class="description-block">
                 <h2 class="title">Пропозиції</h2>
                 <ul class="sell-offers-list">
                   <li *ngFor="let offer of [1,2,3,4]" class="offer">
                     <div class="shadow-box background">
                       <div class="offer-person-wrapper">
                         <div class="img-wrapper">
                           <svg-icon src="../../../assets/images/user.svg"></svg-
icon>
                         </div>
                         <p class="name">Іван</p>
                       </div>
                       <p class="gray-text">Сума: <span class="green-text">20
000$</span></p>
                       <p class="gray-text">Коментарій: <span class="dark-
text">Готовий купити даний автомобіль за вказану суму</span></p>
                     </div>
                   </li>
                 </ul>
               </div>
             </div>
           </div>
         </div>
       </div>
     </div>
   </div>
</section>
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import {RouterModule} from '@angular/router';
```

```
import {UserProfileComponent} from './user-profile.component';
import {AngularSvgIconModule} from 'angular-svg-icon';
import {SharedModule} from '../../shared/shared.module';


@NgModule({
  declarations: [
   UserProfileComponent
  ],
  imports: [
    CommonModule,
    RouterModule.forChild([
      {path: '', component: UserProfileComponent}
    ]),
    AngularSvgIconModule,
    SharedModule
  ],
  exports: [UserProfileComponent]
})
export class UserProfileModule { }
<section class="section">
  <div class="container">
    <div class="row">
      <div class="col-12">
        <div class="image-container">
          <div class="image-wrapper">
            <p class="photo">
              <svg-icon src="../../../assets/images/user.svg"></svg-icon>
            </p>
          </div>
        </div>
      </div>
      <div class="col-12">
        <div class="user-info-wrapper">
          <p class="user-options">
```

```
            <svg-icon src="../../../assets/images/user.svg"></svg-icon>
            <span class="font-big">
              {{data.name}}
            </span>
          </p>
          <p class="user-options">
            <svg-icon src="../../../assets/images/mail.svg"></svg-icon>
            <a href="mailto: {{'joinauto2020@gmail.com'}}"
title="{{'joinauto2020@gmail.com'}}">
                {{data.email}}
            </a>
          </p>
          <p class="user-options">
            <svg-icon src="../../../assets/images/phone.svg"></svg-icon>
            <a href="tel: {{'+38 098 22 45 697'}}" title="{{'+38 098 22 45
697'}}">
                {{data.phoneNumber}}
            </a>
          </p>
          <p class="user-options">
            <svg-icon src="../../../assets/images/map-pin.svg"></svg-icon>
              {{data.location}}
          </p>
        </div>
      </div>
    </div>
  </div>
</section>

<section class="color-section">
  <div class="container">
    <div class="row">
      <div class="col-12">
        <h2 class="header">
          Список товарів
```

```
        </h2>
        <div class="products-wrapper">
          <div class="row">
            <div *ngFor="let product of [1,2]" class="col-lg-3 mb-4 products-
container">
              <app-product-card></app-product-card>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</section>
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import {VehicleListComponent} from './vehicle-list.component';
import {RouterModule} from '@angular/router';
import {ReactiveFormsModule} from '@angular/forms';
import {AutocompleteLibModule} from 'angular-ng-autocomplete';
import {SharedModule} from '../../shared/shared.module';
import {NgbPaginationModule} from '@ng-bootstrap/ng-bootstrap';

@NgModule({
  declarations: [
   VehicleListComponent
  ],
  imports: [
    CommonModule,
    RouterModule.forChild([
      {path: '', component: VehicleListComponent}
    ]),
    ReactiveFormsModule,
    AutocompleteLibModule,
    SharedModule,
    NgbPaginationModule
```

```
  ],
  exports: [VehicleListComponent]
})
export class VehicleListModule { }
import { Component, OnInit } from '@angular/core';
import {FormControl, FormGroup} from '@angular/forms';

@Component({
  selector: 'app-vehicle-list',
  templateUrl: './vehicle-list.component.html',
  styleUrls: ['./vehicle-list.component.scss']
})
export class VehicleListComponent implements OnInit {
  form: FormGroup;
  page = 1;
  bodyTypeData: [{
    name: 'Седан',
    id: 1
  }]
  public keywords = 'name';
  constructor() { }

  ngOnInit() {
    this.form = new FormGroup({
      priceFrom: new FormControl(),
      priceTo: new FormControl(),
      isNew: new FormControl(),
      vehicleType: new FormControl(),
      isAvailableForPromote: new FormControl(),
      bodyType: new FormControl(),
      mileageFrom: new FormControl(),
      mileageTo: new FormControl(),
      engine: new FormControl(),
      transmittion: new FormControl(),
      wheelDrive: new FormControl()
```

```
    })
  }

  submit() {
    console.log(this.form)
  }
}
/form>
        </div>
      </div>
      <div class="col">
        <div class="container">
          <div class="row">
            <div *ngFor="let product of [1,2,3,4]" class="col-12 col-lg-4 mb-4">
              <app-product-card></app-product-card>
            </div>
          </div>

          <div class="row d-none">
            <div class="col-12 d-flex justify-content-center">
              <ngb-pagination [collectionSize]="120" [(page)]="page"
[maxSize]="2"
                              [boundaryLinks]="true"></ngb-pagination>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</section>
```