Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

Пояснювальна записка

До бакалаврської дипломної роботи на тему
Інформаційна система обліку транспортних засобів

Виконав: студент 4 курсу, групи 1КІ-22мс
спеціальності 123 — «Комп'ютерна інженерія»
освітня програма – «Комп'ютерна інженерія»
Щербатюк О.В.
Керівник к.т.н., доц. каф. ОТ
Кисюк Д.В.
Рецензент к.ф-м.н., доц. каф. МБІС
Шиян А.А.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
"20" 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо—кваліфікаційний рівень бакалавр
Галузь знань — 12 Інформаційні технології
Спеціальність — 123 Комп'ютерна інженерія
Освітня програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ
проф., д.т.н. Азаров О. Д.
"06" 02 2024 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Щербатюку Олександру Вікторовичу

1 Тема роботи: «Інформаційна система обліку транспортних засобів», керівник роботи Кисюк Дмитро Васильович, старший викладач кафедри обчислювальної техніки, затверджено наказом вищого навчального закладу від «11» березня 2024 року.

2 Строк подання студентом роботи 20.06.2024р.

3 Вихідні дані до роботи:

4 Зміст розрахунково-пояснювальної записки: вступ, обґрунтування доцільності розробки інформаційної системи, розробка клієнтської частини системи обліку автомобілів, розробка хостингової частини, список використаних джерел.

5 Перелік графічного матеріалу: головна сторінка платформи JSolutions, графічне представлення головної вкладки, вкладка редагування даних.

6 Консультанти розділів роботи наведені в таблиці 1.

7 Дата видачі завдання 12.03.2024 р.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	ст. викладач каф. ОТ Кисюк Д. В.		

8 Календарний план приведений в таблиці 2.

Таблиця 2— Календарний план

№ з/п	Назва етапів дипломної роботи	Строк виконання	Підпис
1	Обґрунтування доцільності розробки інформаційної системи		
2	Розробка клієнтської частини системи обліку автомобілів		
3	Розробка хостингової частини		
4	Оформлення пояснювальної записки		
5	Перевірка якості виконання бакалаврської роботи та усунення недоліків		

Студент Олександр ЩЕРБАТЮК

Керівник роботи Дмитро КИСЮК

АНОТАЦІЯ

УДК 004

В процесі виконання бакалаврської дипломної роботи проводилось створення інформаційної системи обліку транспортних засобів з використанням сучасних технологій ASP.NET.

Після проведення дослідження була розроблена інформаційна система, яка допоможе ефективно вести облік автомобілів, забезпечуючи оптимальне використання ресурсів та зниження витрат як економічних, так і часових. Були показані її функціональні можливості, методи збору та обробки даних про автомобілі, а також процес забезпечення безпеки та конфіденційності інформації.

Досягнуті техніко-експлуатаційні показники: працююча система обліку автомобілів, що дає змогу створювати, редагувати, видаляти, сортувати та фільтрувати надану інформацію.

ABSTRACT

In the course of the bachelor's thesis, the creation of an information system for keeping records of cars was carried out using modern ASP.NET technologies.

As a result of the research, an information system was developed that will help to effectively keep records of cars, ensuring optimal use of resources and reducing both economic and time costs. Its functionality, methods of collecting and processing data about cars, as well as the process of ensuring the security and confidentiality of information were shown.

Achieved technical and operational indicators: a working vehicle registration system that allows you to create, edit, delete, sort and filter the information provided.

Зміст

ВСТУП

За останні десятиліття інформаційні технології викликали масштабні трансформації у всіх сферах діяльності, включаючи управління транспортними ресурсами. інформаційна система обліку транспортних засобів стає необхідним інструментом для ефективного управління автопарком підприємства або організації. Швидкий розвиток технологій та зростання обсягів транспортних потоків створюють потребу у зручних та надійних інструментах для обліку та аналізу даних про автотранспорт.

Актуальність теми. У зв'язку зі зростанням числа автотранспортних засобів і складності управління ними, виникає потреба у впровадженні сучасних інформаційних систем, які дозволять ефективно вирішувати завдання з обліку, моніторингу та аналізу транспортних ресурсів. Ця проблема набуває особливого значення в умовах зростання конкуренції на ринку та підвищення вимог до оперативності та точності прийняття управлінських рішень.

Мета і завдання дослідження. Метою є розробка інформаційної системи обліку автомобілів, спрямованої на автоматизацію процесів обліку, моніторингу та аналізу даних про транспортні засоби. Розроблена система має за завдання забезпечити зручний доступ до інформації про автопарк, підвищити ефективність управління транспортними ресурсами та зменшити час та зусилля, необхідні для проведення аналізу даних.

Об'єкт дослідження. Процеси обліку, моніторингу та аналізу транспортних засобів у підприємствах та організаціях.

Предмет дослідження. Аналіз процесів, методів та інструментів, які забезпечують автоматизацію та оптимізацію обліку транспортних засобів. Це включає в себе комплексний підхід до збору, зберігання, обробки та аналізу даних про автомобілі та їх власників, а також вивчення технічних, організаційних та управлінських аспектів, пов'язаних зі створенням та експлуатацією такої системи.

Методи дослідження. Охоплюють різноманітні підходи, як теоретичні, так і практичні, що дозволяють комплексно оцінити всі аспекти проекту.

Розроблена інформаційна система матиме практичне значення для підприємств та організацій, що використовують автотранспорт у своїй діяльності. Впровадження системи дозволить зменшити час і зусилля, необхідні для ведення обліку автомобілів, покращити контроль за їхнім використанням, а також забезпечить зручний доступ до актуальної інформації про транспортні ресурси.

Після детального аналізу даної роботи формуються наступні задачі для дослідження:

1) визначити потреби і вимоги потенційних користувачів системи обліку автомобілів;

2) встановити функціональні можливості системи. Наприклад облік автомобілів, моніторинг технічного стану, планування обслуговування та ремонту, управління запасними частинами тощо;

3) проаналізувати існуючі інформаційні системи обліку автомобілів та їхні переваги і недоліки для визначення оптимального підходу до розробки нової системи;

4) проаналізувати та пояснити вибір інструментів для розробки;

5) реалізувати модулі авторизації, адміністрування наповнення даних в реальному часі;

6) провести тестування розробленого програмного коду.

Даний проект має потенціал сприяти підвищенню ефективності управління транспортними ресурсами та зростанню конкурентоспроможності підприємств у сучасних умовах ринкової конкуренції.

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

1.1 Аналіз предметної області

Інформаційна система відіграє важливу роль у сучасному управлінні автопарками. Вона спрямована на забезпечення ефективного контролю за автомобілями, їх використанням та обслуговуванням. Основні функції такої системи включають в себе облік транспортних засобів, ведення їхньої технічної та експлуатаційної інформації, моніторинг роботи автомобілів, а також аналіз витрат на їх утримання та експлуатацію.

Важливим елементом інформаційної системи являється збір та обробка даних про автомобілі. Основними критеріями являється збір інформації про технічний стан автомобілів, їх розташування, пробіг, витрати на паливо та інші параметри. Завдяки цим даним, керівництво може отримувати актуальну інформацію про стан автопарку в режимі реального часу і приймати обгрунтовані подальші рішення.

У сучасному бізнес-світі, де час та ефективність грають важливу роль, інформаційна система обліку транспортних засобів обов’язково буде необхідним інструментом для підприємств будь-якої галузі, що активно використовують автомобілі. Вона дозволяє оптимізувати ресурси, підвищувати продуктивність та ефективність управління транспортними ресурсами, що є критичними факторами успіху в умовах сучасного бізнесу.

У предметній області інформаційної системи обліку автомобілів основними об'єктами є автомобілі та власники:

1) автомобілі включають такі характеристики, як марка, модель, рік випуску, VIN (унікальний ідентифікаційний номер), реєстраційний номер, колір і тип кузова. Ці дані є базовими для кожного автомобіля;

2) власники можуть бути як фізичними, так і юридичними особами. інформація про власників включає їхні імена, контактні дані та адреси. Це дозволяє відстежувати, хто саме володіє автомобілем у певний момент часу.

Автомобілі пов'язані з власниками, історією обслуговування, реєстраційними даними та технічним станом. Кожен автомобіль має бути прив'язаний до одного або кількох власників в різні періоди часу, відображаючи зміни в праві власності. Дані про автомобіль та його власників повинні оновлюватись при кожній зміні автомобіля чи власника відповідно. історія обслуговування фіксує всі проведені ремонтні роботи автомобіля, що дозволяє відстежувати його технічний стан та планувати наступні дати для обслуговування.

Для забезпечення роботи системи необхідна база даних для зберігання всіх необхідних даних. інтерфейс користувача повинен бути зручним для введення, редагування та перегляду даних. інтеграція з іншими системами, такими як державні реєстри та сервіси технічного огляду, дозволить автоматизувати багато процесів, однак інтеграція планується вводитись у подальшому майбутньому при виході наступних оновлень. Безпека даних є критично важливою для захисту від несанкціонованого доступу та забезпечення конфіденційності інформації.

Основні проблеми та ризики включають точність та актуальність даних, конфіденційність інформації та технічні проблеми. Регулярне оновлення даних та верифікація є необхідними для підтримки актуальної інформації. Захист особистих даних власників автомобілів є важливим аспектом безпеки. Технічні збої та потреба в регулярному обслуговуванні системи також можуть впливати на її роботу.

Основні переваги системи включають в себе підвищення ефективності за рахунок автоматизації процесів обліку та управління даними, можливість створення звітів та аналізу даних для прийняття обґрунтованих рішень, а також зручність використання для користувачів системи, що спрощує процедури реєстрації та обліку.

Таким чином, інформаційна система обліку автомобілів дозволяє значно підвищити ефективність управління автопарком, забезпечує надійний облік і контроль за станом транспортних засобів, а також сприяє підвищенню рівня безпеки на дорогах.

1.2 Суть технічної проблеми та шляхи її вирішення

Технічна проблема, яка часто виникає при розробці та впровадженні інформаційних систем обліку транспортних засобів, полягає у забезпеченні точності, надійності та ефективності обробки великого обсягу даних, які містить автопарк. При зборі, аналізі та використанні таких даних можуть виникати різноманітні проблеми, такі як недостатня швидкодія системи при обробці великої кількості даних, неоднорідність формату даних, нестабільність та можливість втрати даних через технічні збої тощо.

Одним з можливих шляхів вирішення цієї проблеми є використання потужних серверів та баз даних, які забезпечують високу швидкодію та надійність при обробці великого обсягу інформації. Також важливо ретельно спроектувати архітектуру системи, враховуючи потреби її користувачів, і використовувати ефективні алгоритми для обробки даних.

Ще одним способом вирішення цієї проблеми є використання технологій обробки даних в реальному часі (real-time data processing), які дозволяють обробляти та аналізувати дані миттєво, без затримок. Це дозволяє оперативно реагувати на зміни в умовах експлуатації автопарку та приймати швидкі та обгрунтовані управлінські рішення.

Додатково, важливим аспектом є забезпечення безпеки даних та захисту від несанкціонованого доступу. Використання сучасних методів шифрування та аутентифікації дозволяє захистити конфіденційні дані користувачів та забезпечити цілісність інформації в системі.

Використання інформаційної системи надає ряд корисних можливостей та функціоналів для керування автопарком. Нижче наведені приклади:

1) реєстрація автомобілів. Можливість додавання нових автомобілів до системи з введенням основних технічних характеристик, таких як марка, модель, рік випуску, номер кузова тощо;

2) облік технічного стану. Ведення історії технічного обслуговування та ремонту автомобілів, включаючи відомості про проведені роботи, дати, пробіг, витрати тощо;

3) моніторинг палива. Відстеження споживання палива для кожного автомобіля з можливістю аналізу витрат та ефективності використання;

4) управління запасними частинами. Система може вести облік запасних частин та матеріалів, їх розподіл по автомобілях та здійснювати автоматичне повідомлення про необхідність поповнення запасів;

5) графіки обслуговування. Забезпечення планування технічного обслуговування та періодичних оглядів автомобілів з відповідними нагадуваннями та повідомленнями;

6) аналітика та звітність. Генерація звітів і аналітичних даних про ефективність використання автопарку, витрати на обслуговування, витрати на паливо, технічний стан автомобілів тощо;

7) інтеграція з GPS і системами відстеження. Можливість інтеграції з GPS-трекерами та системами відстеження, що дозволить відслідковувати місцезнаходження автомобілів у реальному часі.

Недоліки використання такої системи:

1) витрати на впровадження. Розробка та впровадження інформаційної системи можуть вимагати значних витрат на програмне забезпечення, обладнання та навчання персоналу;

2) складність використання. Якщо система не буде досить інтуїтивно зрозумілою або не буде враховувати потреби користувачів, вона може стати складною у використанні, що може призвести до опору з боку персоналу;

3) проблеми з безпекою. Зберігання та обробка конфіденційної інформації про автомобілі (наприклад, особисті дані власників автомобілів,

відомості про ремонт та технічний стан) може стати об'єктом для кібератак або несанкціонованого доступу;

4) несправності системи. Технічні проблеми, такі як відмови обладнання або програмне збої, можуть призвести до перерв у роботі системи та втрати даних;

5) необхідність постійного оновлення. Зміна потреб бізнесу або технологічного середовища може вимагати постійного оновлення або модернізації системи, що може бути часовим - та ресурсозатратним;

6) обмеження віддаленого доступу. Якщо інформаційна система не підтримує віддалений доступ або мобільні додатки, це може обмежити зручність користувачів, особливо у випадку роботи на віддалених майданчиках;

7) несправжня або застаріла інформація. Якщо дані в системі не оновлюються регулярно або неправильно введені, це може призвести до неправильних рішень та втрати точності в обліку;

8) недостатня інтеграція з іншими системами. Якщо інформаційна система не може ефективно співпрацювати з іншими системами, це може створити труднощі у обміні даними та взаємодії між різними підрозділами організації.

Для вирішення потенційних проблем інформаційної системи ведення обліку автомобілів можна проводити ретельне навчання персоналу з використання системи, а також враховувати їхні потреби та відгуки для постійного вдосконалення інтерфейсу користувача. Для забезпечення безпеки даних необхідно впровадити ефективні методи захисту, такі як шифрування даних, аутентифікація користувачів та моніторинг доступу. Проблеми з несправністю системи можна вирішити шляхом регулярного технічного обслуговування та підтримки програмного забезпечення.

Окрім цього, постійне оновлення та підтримка системи дозволить уникнути застарілості інформації та забезпечить її відповідність поточним потребам бізнесу. інтеграція з іншими системами та платформами може бути

здійснена через стандартизовані інтерфейси або за допомогою спеціальних інтеграційних рішень.

Загальний підхід до вирішення цих проблем полягає в постійному моніторингу та аналізі роботи системи, враховуючи потреби користувачів і забезпечуючи постійне вдосконалення та підтримку.

1.3 Огляд аналогів

При огляді аналогів інформаційної системи обліку транспортних засобів важливо врахувати різноманітність доступних на ринку рішень. Кожна із них має свої переваги та недоліки, які варто враховувати при виборі оптимального варіанту для конкретної організації або підприємства. Приклад такої системи наведений на рисунку 1.1.

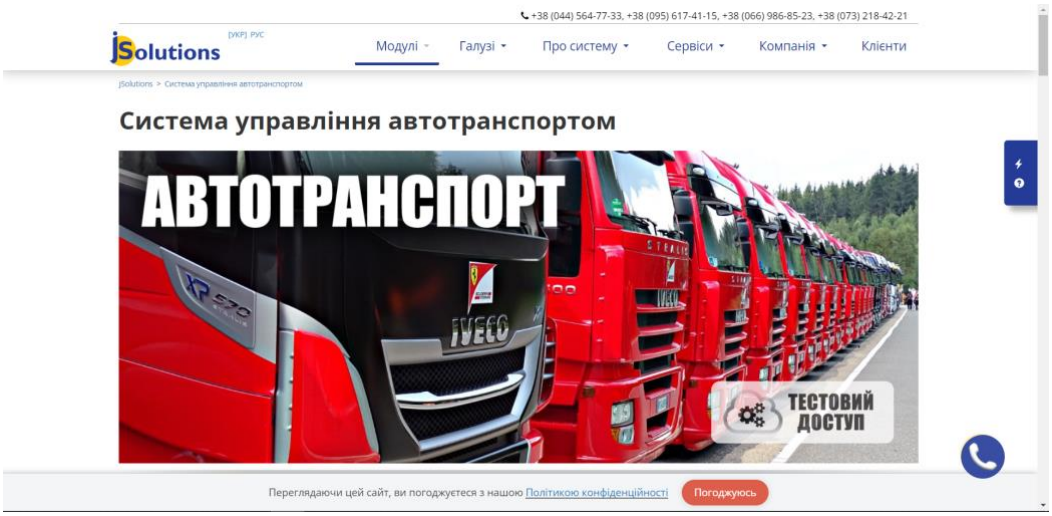

Рисунок 1.1 — Головна сторінка платформи JSolutions

JSolutions – це компанія, яка займається розробкою програмного забезпечення, в тому числі інформаційних систем обліку автомобілів. Аналізуючи їх підхід до створення такої системи, можна виділити наступні переваги та недоліки користуванням їх послугами.

Переваги:

1) JSolutions має досвід у розробці складних інформаційних систем, що забезпечує високу якість їх продуктів. Вони використовують сучасні технології та інструменти, що дозволяє створювати надійні та масштабовані рішення;

2) їхні системи зазвичай мають зручний та інтуїтивно зрозумілий інтерфейс користувача. Це спрощує навчання нових користувачів і зменшує кількість помилок при введенні даних;

3) інформаційні системи від JSolutions часто мають модульну архітектуру, що дозволяє легко налаштовувати та розширювати функціонал відповідно до потреб клієнтів. Це дає можливість адаптувати систему під конкретні вимоги організації;

4) JSolutions приділяє велику увагу безпеці даних, використовуючи сучасні методи шифрування та автентифікації. Це забезпечує високий рівень захисту від несанкціонованого доступу та зберігає конфіденційність інформації;

5) продукти JSolutions зазвичай підтримують інтеграцію з іншими інформаційними системами та сервісами, що дозволяє забезпечити безперервний обмін даними та покращити загальну ефективність роботи.

Недоліки:

1) одним із основних недоліків рішень від JSolutions може бути їх висока вартість. Це включає як початкові витрати на впровадження системи, так і постійні витрати на обслуговування та підтримку;

2) незважаючи на високу якість продуктів, процес впровадження інформаційної системи може бути складним і вимагати значних зусиль від організації. Це включає необхідність навчання персоналу та адаптації існуючих процесів під нову систему;

3) використання системи від JSolutions може призвести до залежності від конкретного постачальника програмного забезпечення. Це може стати проблемою у випадку, якщо компанія вирішить змінити постачальника або якщо JSolutions змінить умови обслуговування;

4) хоч і модульна архітектура і налаштовуваність є перевагами, у деяких випадках може виникнути обмеження у можливостях налаштування під

специфічні потреби організації. Це може вимагати додаткових зусиль для адаптації системи;

5) незважаючи на підтримку інтеграції з іншими системами, в деяких випадках можуть виникати технічні труднощі при сполученні з існуючими системами або сервісами. Це може потребувати додаткових ресурсів для вирішення;

6) якщо організація використовує застаріле обладнання або програмне забезпечення, це може ускладнити впровадження нової системи від JSolutions і потребувати додаткових інвестицій у модернізацію технічної інфраструктури.

Інформаційні системи обліку автомобілів від JSolutions мають багато переваг, включаючи високу якість, зручний інтерфейс, безпеку даних та можливість інтеграції з іншими системами. Однак, ці рішення також можуть мати недоліки, такі як висока вартість, складність впровадження та залежність від постачальника. В ході цього було вирішено створити власну інформаційну модель задля полегшення складності впровадження та автоматизації робочого процесу.

1.4 Економічні переваги використання системи обліку автомобілів

Впровадження інформаційної системи обліку автомобілів надає значні економічні переваги, які можуть вплинути як на загальні витрати організації, так і на ефективність її діяльності. Створення власної інформаційної моделі обліку автомобілів забезпечує різні аспекти економічної вигоди, включаючи зниження операційних витрат, підвищення продуктивності, оптимізацію ресурсів, поліпшення управління активами та багато іншого.

Однією з найбільших економічних переваг є суттєве зниження операційних витрат. Автоматизація процесів обліку, які раніше виконувалися вручну, зменшує необхідність у великій кількості адміністративного персоналу. Це призводить до скорочення витрат на заробітну плату, а також зменшує ризик людських помилок, які можуть бути дорогими для виправлення. Крім того,

автоматизована система зменшує витрати на паперову документацію, що є ще одним джерелом економії.

Припустимо, у середній організації, яка займається обліком автомобілів, працює 5 адміністраторів, кожен із яких отримує заробітну плату 20 000 грн на місяць. Загальні витрати на персонал складуть 100 000 грн на місяць.

Автоматизація процесів дозволяє зменшити кількість адміністраторів до 2 осіб, що знижує витрати на персонал до 40,000 грн на місяць. Таким чином, щомісячна економія становитиме 60 000 грн, а річна – 720 000 грн.

Додатково витрати зменшуються і на паперову документацію. В середньому організація витрачає 5000 грн на місяць на паперову документацію, включаючи друк, папір, зберігання та архівування. Впровадження інформаційної системи дозволяє майже повністю відмовитися від паперової документації, зменшуючи ці витрати до мінімуму. Припустимо, нові витрати складуть 1000 грн на місяць на випадкові друковані матеріали. Щомісячна економія становитиме 4000 грн, а річна – 48 000 грн.

Окрім цього автопарк із 50 автомобілів витрачає в середньому 10 000 грн на місяць на позапланові ремонти через несвоєчасне обслуговування. Впровадження інформаційної системи дозволяє краще планувати обслуговування, що може зменшити ці витрати на 30%. В результаті щомісячна економія становитиме 3000 грн, а річна – 36 000 грн.

У висновку загальна економія для середньостатистичної фірми, що має автопарк буде становити близько 1 000 000 грн.

1.5 Вибір архітектури застосунку, структури його модулів та методу зберігання даних

У процесі проектування інформаційної системи автомобільного обліку вибір архітектури додатків є важливим кроком, оскільки від цього залежить ефективність та масштабованість системи. Вибираючи архітектуру, ви повинні враховувати потреби вашого бізнесу, технічні вимоги, розмір проекту та інші

фактори. Отже, проект буде розроблятися на основі шаблону ASP.NET Модель MVC. Це архітектурний шаблон, який розбиває додаток на 3 основні компоненти: моделі, подання та контролери. Кожен з цих компонентів виконує чітко визначені функції, покращує розподіл обов'язків і підтримує принцип поділу інтерфейсу і логіки програми.

Дана модель представляє архітектуру, а також логіку програми та взаємодіє з даними для виконання різних операцій. Вона відповідає за доступ до даних, обробку даних та виконання логіки програми. У цій моделі використовуються автомобілі, власники та інші об'єкти даних. Можуть бути класи, що представляють ці об'єкти. Окрм цього у моделі можуть бути присутні класи, які надають логіку для різних операцій з використанням цих об'єктів, таких як створення, видалення або оновлення даних.

Архітектура ASP.NET MVC відповідає за створення HTML-сторінки, яка відображає дані для користувача та відображається у браузері. Для цього використовуються дані, отримані з моделі. Представлення не має прямого доступу до даних або логіки програми, а відображає лише інформацію, отриману від контролера.

Контролер отримує запити від користувача, викликає відповідні методи обробки цих запитів, надсилає отримані дані до відповідного представлення та відображає їх. Після включає дії, які обробляють різні типи запитів, такі як вилучення даних, перегляд даних або їх збереження.

ASP.NET MVC використовує маршрутизацію, щоб визначити, який контролер і дію викликати для кожного запиту. Це робиться на основі URL-адреси, що підвищує гнучкість та масштабованість програми. Архітектура ASP.NET MVC забезпечує зручний та ефективний спосіб розробки веб-додатків, що дозволяє легко розділити логіку додатка на компоненти, які полегшуючи розробку додатків.

Після аналізу необхідно подумати про архітектуру додатку. Для інформаційної системи обліку автомобілів може бути вибрана багатошарова архітектура типу «клієнт-сервер». У такій архітектурі клієнтська частина

системи (інтерфейс користувача) взаємодіє з серверною частиною, де розташована база даних. Даний підхід забезпечує розділення функціональності та дозволяє зберігати дані централізовано, що спрощує подальше керування та підтримку працездатності системи.

Структура модулів системи повинна представляти логічну організацію функцій і робочих компонентів системи обліку. Модулі можуть бути організовані за їхнім типом функціональності: модуль ведення обліку автомобілів, модуль планування технічного обслуговування, модуль моніторингу палива тощо. Кожен модуль повинен відповідати власному певному завданню і мати чітко визначені інтерфейси для взаємодії з іншими модулями.

Для зберігання даних, можна використовувати бази даних, такі як PostgreSQL або Microsoft SQL Server. Реляційні бази даних найкраще підходять для зберігання даних, які використовуються в інформаційних системах обліку. Вони забезпечують надійність, стабільність та можливість використання різноманітних інструментів для роботи з даними. У моєму ж випадку використовуватиметься PostgreSQL та Docker, які будуть виступати у ролі localhost.

Загалом, вибір архітектури, структури модулів та методу зберігання даних повинен відповідати потребам та вимогам проєкту, забезпечуючи ефективну та надійну роботу інформаційної системи обліку автомобілів.

Тепер необхідно переглянути загальну структуру для кожного модуля. При розробці програмного забезпечення варто зважати на те, що написаний код може підтримуватися, допрацьовуватися та тестуватися іншими розробниками. Щоб врахувати цей аспект часто використовуються певні шаблони.

1.6 Структурне моделювання інформаційної системи

Структурне моделювання інформаційної системи обліку автомобілів являється процесом проектування системи, який включає в себе визначення структури даних, логіки програм та інтерфейсів, що дозволяють ефективно

зчитувати, зберігати, обробляти та використовувати інформацію про автомобілі. Нижче наведено детальний опис кожного пункту підходу:

1. Аналіз вимог. Даний етап передбачає вивчення потреб користувачів та вимог до системи. Це може включати визначення функцій системи, типів даних, які потрібно зберігати, і операцій, які система повинна здійснювати.

2. Проектування бази даних. На цьому етапі визначається структура бази даних, яка буде використовуватися для зберігання інформації про автомобілі. Це включає в себе створення таблиць, визначення взаємозв'язків між ними та вибір відповідних типів даних для кожного поля.

3. Розробка програмної логіки. На даному етапі розробляються програмні компоненти, які забезпечують функціональність системи. Це може включати створення класів або модулів для обробки даних, логіки методів та взаємодії з користувачем.

4. Створення інтерфейсу користувача. Розробка інтерфейсу, який дозволяє користувачам взаємодіяти з системою. Це може включати створення веб або десктопних додатків, форм, таблиць та інших елементів, які дозволяють введення та відображення даних.

5. Тестування і відлагодження. Після розробки всіх компонентів системи проводиться тестування для перевірки їхньої працездатності та виявлення помилок. Після цього виконується відлагодження, аби виправити будь-які виявлені проблеми.

6. Впровадження та підтримка. Після успішного завершення тестування система готова до впровадження. Це означає встановлення системи на серверах або комп'ютерах користувачів та навчання персоналу щодо її використання. Після впровадження необхідно забезпечити підтримку системи, включаючи виправлення помилок, розширення функціональності та забезпечення безперебійної роботи.

2 РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ СИСТЕМИ ОБЛІКУ АВТОМОБІЛІВ

2.1 Створення та налаштування проекту для розробки інформаційної системи обліку автомобілів

Для того щоб створити ASP .NET проект для початку потрібно створити проект у Visual Studio використовуючи шаблон ASP .NET Web Application (рисунок 2.1). Серед усіх шаблонів потрібно обрати саме ASP .NET Web Application (Model View Controller).

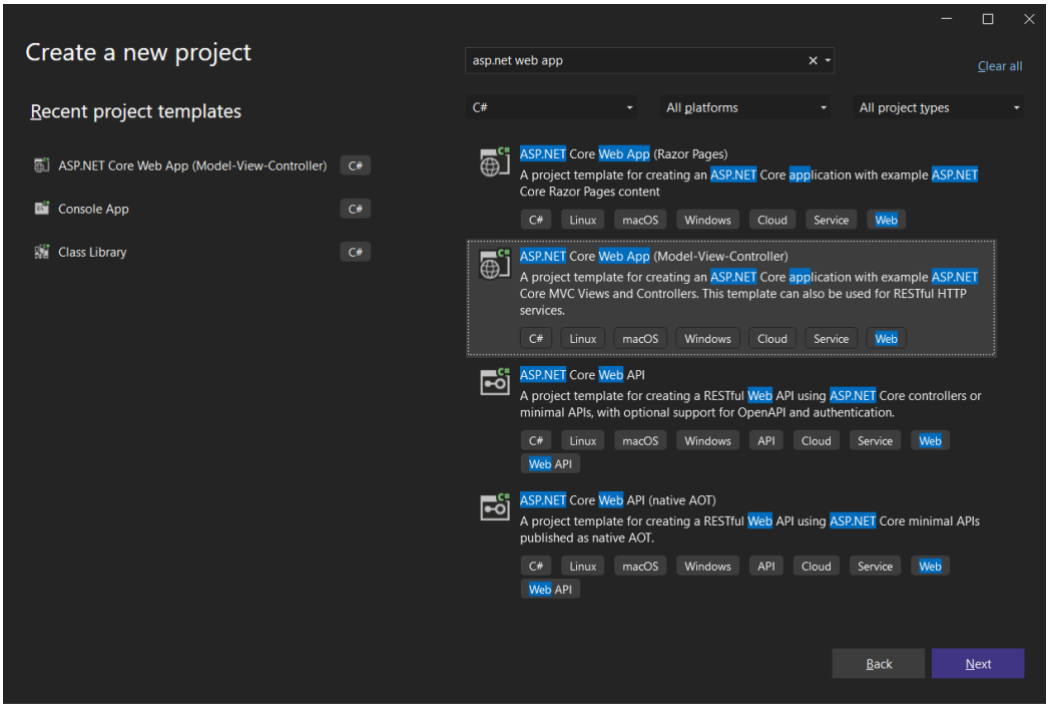

Рисунок 2.1 — Створення проекту по необхідному шаблону

Саме даний шаблон дозволяє розділити вашу програму на три основні частини: модель (Model), представлення (View) та контролер (Controller). Даний шаблон має більшу гнучкість у виборі технологій для різних компонентів веб-додатку. З його допомогою можна використовувати будь-які ORM для моделей, будь-які шаблони для представлень і контролери можуть бути реалізовані на C#.

Наступним кроком стане налаштування підключення бібліотек за допомогою NuGet Package Manager (рисунок 2.2).

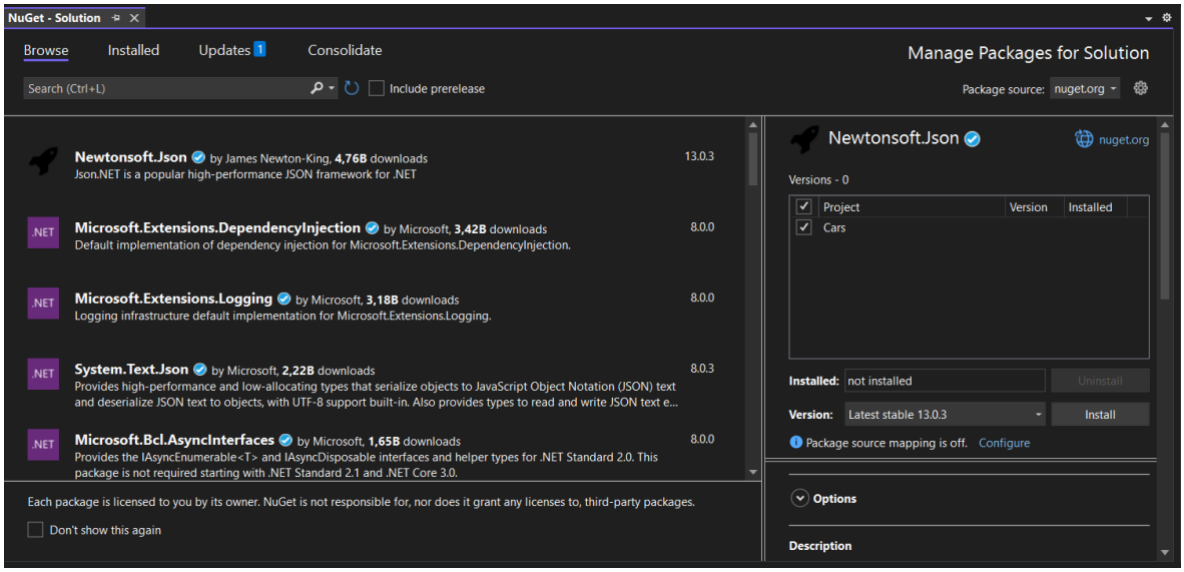

Рисунок 2.2 — Налаштування підключення бібліотек

Даний інструмент призначений для керування пакетами у розробці програмного забезпечення на платформі .NET. Він дозволяє легко встановлювати, оновлювати та видаляти пакети залежностей для кожного проекту. Нижче наведені ключові особливості та переваги NuGet Package Manager:

1) простота використання. NuGet інтегровано з інструментами розробки, такими як Visual Studio, що робить його дуже зручним для встановлення пакетів та керування ними безпосередньо з IDE;

2) різноманітність пакетів. NuGet має велику екосистему пакетів, яка охоплює широкий спектр функціональності, від бібліотек та фреймворків до утиліт та різноманітних розширень;

3) версіонування пакетів. NuGet дозволяє встановлювати конкретні версії пакетів, а також керувати їх версіями під час оновлення;

4) залежності та конфлікти. NuGet вирішує проблеми залежностей та конфліктів між різними версіями пакетів, забезпечуючи, що всі залежності будуть встановлені правильно та сумісно;

5) пакети для власного використання. Крім встановлення зовнішніх пакетів, ви також можете створювати свої власні NuGet-пакети для використання в інших проектах або для спільного використання з іншими розробниками;

6) підтримка командного рядка. NuGet також доступний через командний рядок, що дозволяє виконувати операції керування пакетами безпосередньо з командного рядка вашої операційної системи.

Після налаштування бібліотек можна зайнятись безпосередня створенням структури проекту (створити необхідні класи і т.д.). У даному випадку в папку Models було добавлено 2 класи Cars та Owners (рисунок 2.3).

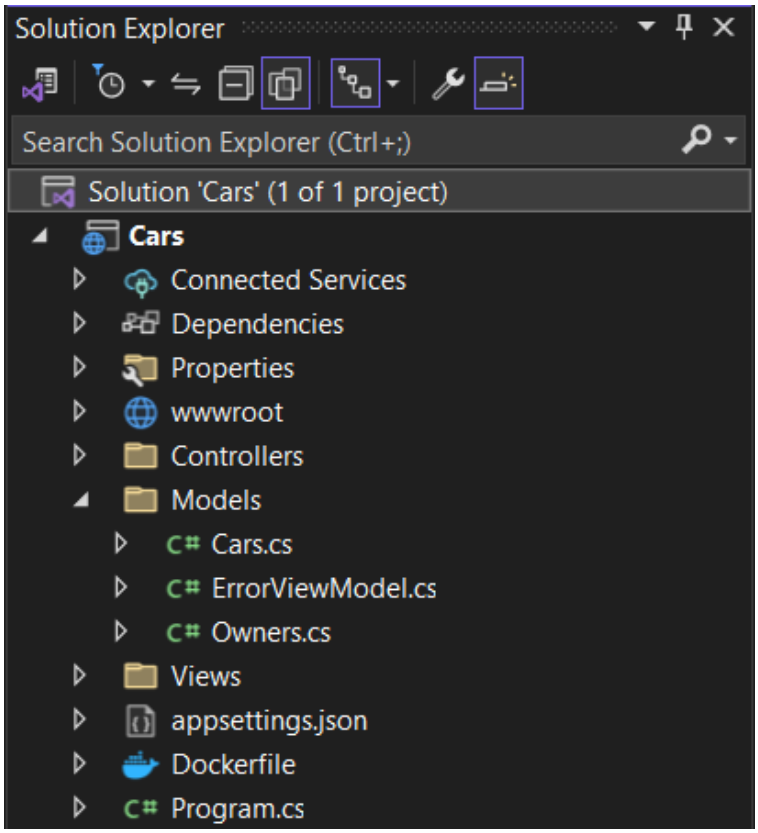

Рисунок 2.3 — Створення класів Cars та Owners

Дані класи міститимуть параметри для автомобілів та власників, зможуть відображати таблиці баз даних та інші параметри. Основні параметри, які можуть бути у цьому класі наведені нижче:

1) Id - унікальний ідентифікатор автомобіля;

2) Brand - марка автомобіля (наприклад, Toyota, BMW, Honda);

3) Model - модель автомобіля (наприклад, Camry, X5, Civic);

4) Year - рік випуску автомобіля;

5) Color - колір автомобіля;

6) RegistrationNumber - реєстраційний номер автомобіля;

7) OwnerId - ідентифікатор власника автомобіля (якщо автомобіль має власника);

8) Mileage - пробіг автомобіля;

9) FuelType - тип палива, яке використовує автомобіль (бензин, дизель, гібрид тощо);

10) EngineCapacity - об'єм двигуна;

11) TransmissionType - тип трансмісії (автоматична, механічна тощо);

12) Price - ціна автомобіля;

13) PurchaseDate - дата придбання автомобіля;

14) InsuranceDetails - інформація про страхування автомобіля (номер полісу, дата закінчення тощо);

15) LastServiceDate - дата останнього технічного обслуговування автомобіля.

Дані параметри дозволять адміністратору чи механіку дізнатись якомого більше інформації про автомобіль та у подальшому упростити процес обслуговування.

Наступним кроком потрібно зайнятися створенням так званих контролерів, які будуть містити методи, що відповідатимуть за обробку, створення, видалення даних та інше. Для початку потрібно створити клас CarController та OwnerController у папці Controllers (рисунок 2.4).

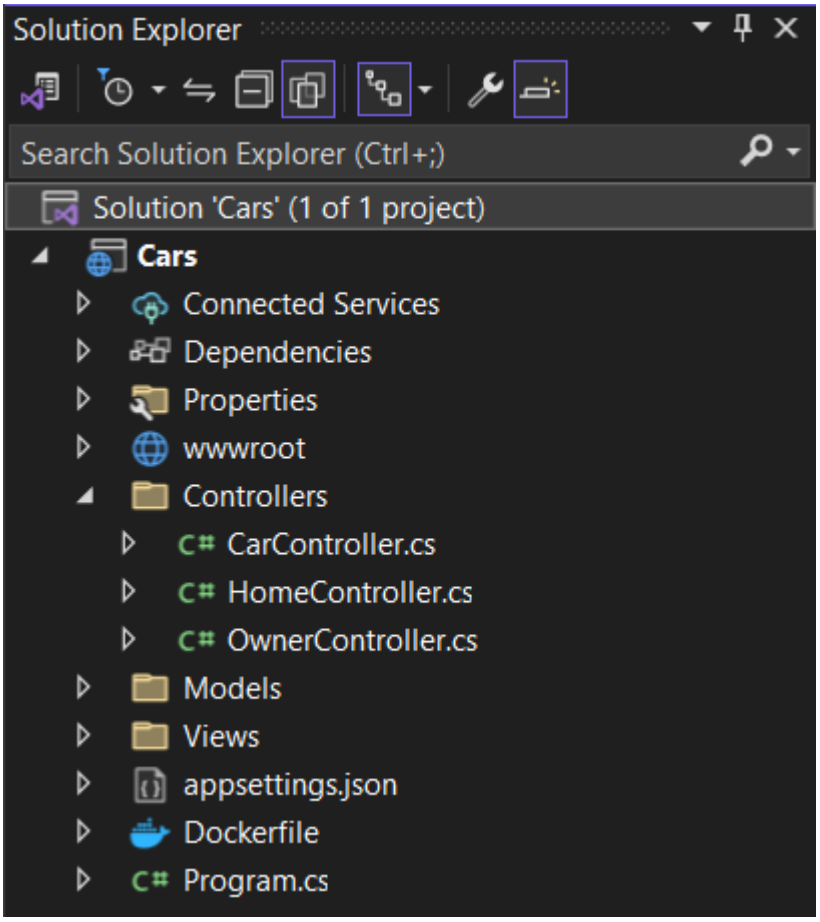

Рисунок 2.4 — Створення класів контролерів

Контролери CarController і OwnerController відповідатимуть за обробку запитів, пов'язаних з автомобілями та власниками відповідно. Нижче наведена загальна відповідальність кожного з цих контролерів.

За що відповідають методи контролера CarController:

1. Index - відображає список всіх автомобілів.
2. Details - відображає детальну інформацію про певний автомобіль за його Id.
3. Create - відображає форму для створення нового автомобіля.
4. Edit - відображає форму для редагування існуючого автомобіля за його Id.
5. Delete - відображає підтвердження видалення автомобіля та видаляє його з бази даних.

6. Search - обробляє запити на пошук автомобіля за різними критеріями.

7. Filter - обробляє запити на фільтрацію списку автомобілів за певними умовами (наприклад, за маркою, за роком випуску тощо).

За що відповідають методи контролера OwnerController:

1. Index - відображає список всіх власників автомобілів.

2. Details - відображає детальну інформацію про певного власника автомобіля за його Id.

3. Create - відображає форму для створення нового власника автомобіля.

4. Edit - відображає форму для редагування існуючого власника автомобіля за його Id.

5. Delete - відображає підтвердження видалення власника автомобіля та видаляє його з бази даних.

6. Search - обробляє запити на пошук власника за різними критеріями (ім'я, прізвище, номер телефону).

7. Filter - обробляє запити на фільтрацію списку власників за певними умовами (наприклад, за місцем проживання, за кількістю автомобілів).

Дані контролери будуть взаємодіяти з відповідними моделями даних (Car та Owner), а також з відповідними представленнями для відображення даних користувачам та обробки їхних запитів.

2.2 Створення та налаштування представлень для відповідних класів

У ASP.NET MVC папка Views відповідає за зберігання представлень, які використовуються для відображення даних користувачам. Представлення - це шаблони HTML, які можуть містити динамічний вміст, що передається з контролерів. Основні функції та структура папки Views включають:

1) представлення для кожного контролера зберігаються в окремих підпапках, що мають назви відповідних контролерів. Наприклад, представлення

для CarController зберігаються в папці Views/Car, а представлення для OwnerController - в папці Views/Owner;

2) кожна дія контролера має своє представлення, яке називається відповідно до дії. Наприклад, для дії Index контролера CarController буде представлення Index.cshtml в папці Views/Car;

3) папка Views/Shared зберігає спільні представлення, які можуть використовуватися кількома контролерами. Наприклад, спільні макети (layouts), часткові представлення (partial views) та інші компоненти інтерфейсу;

4) у папці Views/Shared також зазвичай знаходяться макети, такі як _Layout.cshtml, які визначають загальну структуру сторінки (шапка, навігація тощо). Макети дозволяють використовувати спільну структуру на багатьох сторінках;

5) представлення можуть містити Razor синтаксис, який дозволяє використовувати C# код у межах HTML. Це дає змогу динамічно змінювати вміст сторінки залежно від даних, що передаються з контролера.

На рисунку 2.5 показана загальна структура вмісту папки Views.

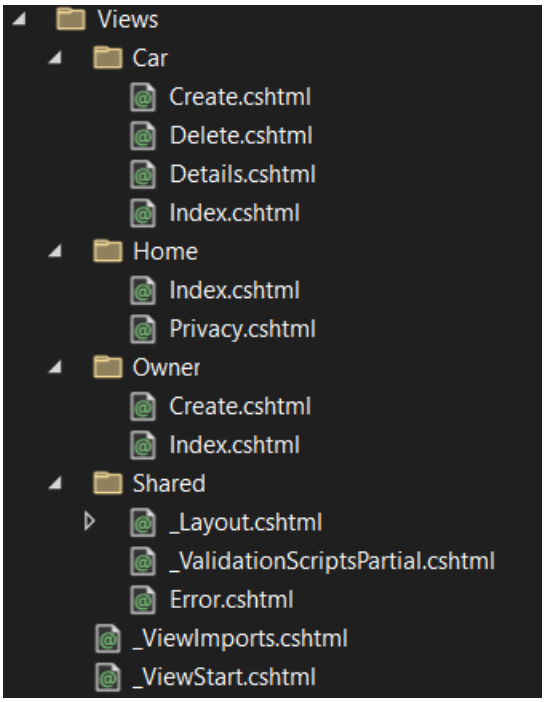

Рисунок 2.5 — Структура вмісту папки Views

Коли користувач робить запит до контролера, контролер обробляє цей запит, взаємодіє з моделями для отримання даних і повертає відповідне представлення. Наприклад, при запиті до дії Index контролера CarController, повернеться представлення Views/Car/Index.cshtml, яке відобразить список автомобілів. Основні компоненти представлень:

1) HTML - основна структура сторінки;

2) Razor - синтаксис для впровадження C# коду в HTML, наприклад, для відображення даних, умовних операторів, циклів тощо;

3) CSS - стилі для покращення вигляду сторінок;

4) JavaScript - скрипти для динамічної взаємодії з користувачем та обробки подій на стороні клієнта.

Макети визначають загальну структуру сторінки, і кожне представлення може наслідувати цю структуру. Наприклад, _Layout.cshtml може містити заголовок сторінки, навігаційне меню. Окремі представлення будуть вставляти свій вміст у визначену секцію макету (зазвичай @RenderBody()). Це забезпечує єдиний стиль і структуру сторінок та спрощує їх підтримку та оновлення.

2.3 Налаштування маршрутів

Маршрутизація в ASP.NET MVC - це механізм, який визначає, як URL-адреси на веб-сайті відповідають певним контролерам і діям. Маршрутизація визначає правила, за якими вхідні запити обробляються і спрямовуються до відповідних компонентів вашого додатка. Основні концепції маршрутизації:

1) маршрут - це правило, яке визначає, які URL-адреси відповідають яким контролерам і діям;

2) контролер - це клас, який обробляє запит і повертає відповідь (зазвичай у вигляді представлення);

3) Дія (Action) - метод в контролері, який обробляє конкретний запит;

4) Параметри - це дані, які передаються у вигляді частин URL або як запитні параметри.

Маршрутизація в ASP.NET MVC зазвичай налаштовується в файлі Startup.cs (для ASP.NET Core) або в файлі RouteConfig.cs (для старіших версій ASP.NET MVC). Нижче наведений приклад налаштування маршрутизації.

Лістинг 2.1 - Частина коду файлу Startup.cs

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Home/Error");
            app.UseHsts();
        }
        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseRouting();
        app.UseAuthorization();
        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllerRoute(
                name: "default",
                pattern: "{controller=Car}/{action=Details}/{id?}");
        });
    }
```

У цьому прикладі конфігурації визначено наступний маршрут:

1) pattern - "{controller=Car}/{action=Details}/{id?}";

2) controller - ім'я контролера "Car";

3) action - ім'я дії "Details";

4) id - необов'язковий параметр, який можна передати до дії.

Тобто при виклику даного маршруту буде використовуватися посилання на контролер Car та викликати ім’я дії Details, що буде показувати детальну інформацію про авто. Окрім цього існують додаткові параметри маршрутизації:

1) route Constraints - обмеження, які дозволяють визначити, які URL відповідають маршруту (наприклад, обмеження типу для параметрів);

2) attribute Routing - використання атрибутів прямо в контролерах і діях для визначення маршрутів.

Маршрутизація є ключовим компонентом ASP.NET MVC, оскільки вона забезпечує зв'язок між URL-адресами та логікою додатка, дозволяючи користувачам взаємодіяти з вашим веб-додатком через прості та інтуїтивно зрозумілі URL-адреси.

2.4 Налаштування міграцій для PostgreSQL в ASP.NET Core

Міграція в контексті розробки баз даних означає процес внесення змін до структури бази даних в керований спосіб. У ASP.NET Core, зокрема з використанням Entity Framework Core, міграції дозволяють розробникам створювати, змінювати та видаляти схеми баз даних, а також підтримувати їх версіонування. Це дозволяє зберігати зміни у версії схеми бази даних та розгортати ці зміни на різних середовищах (наприклад, у розвитку, тестуванні та виробництві).

В першу чергу слід підключити Entity Framework Core для роботи з PostgreSQL (рисунок 2.6).

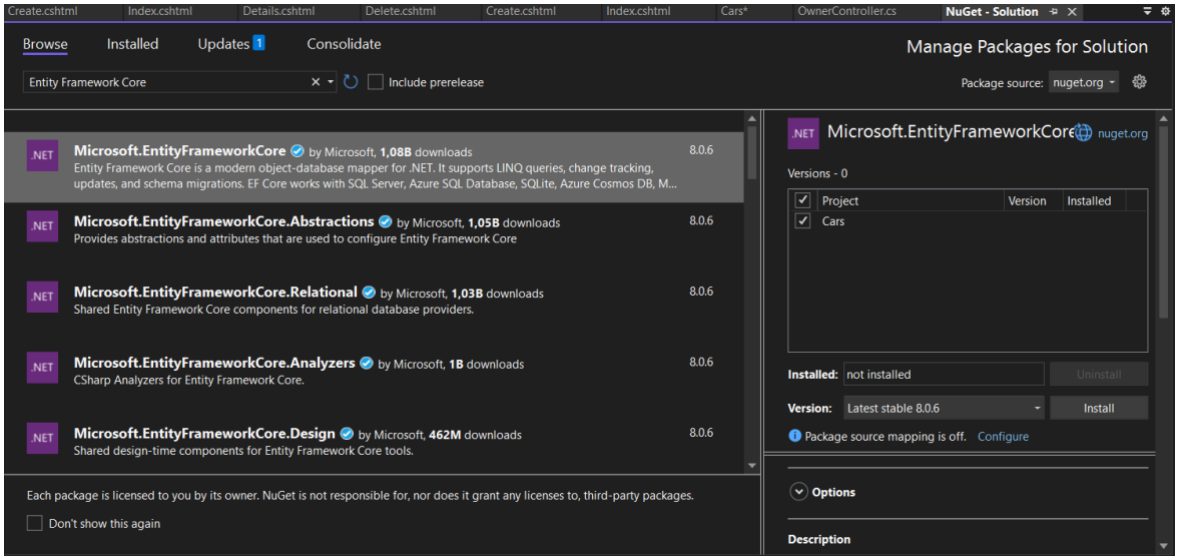

Рисунок 2.6 — Підключення Entity Framework Core

Далі необхідно створити клас контексту даних ApplicationDbContext. Даний клас буде відображати моделі даних на таблиці бази даних. Реалізація частини коду наведена нижче:

Лістинг 2.2 - Реалізація класу контексту даних ApplicationDbContext

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
    public DbSet<Car> Cars { get; set; }
}
```

Після слід знайти файл appsettings.json. Даний файл відповідає за налаштування програми. У цьому файлі потрібно додати рядок підключення до PostgreSQL.

Лістинг 2.3 - Реалізація підключення до бази даних

```
{
  "ConnectionStrings": {
        "DefaultConnection": "Host=localhost;Database=CarDatabase;
Username=admin;Password=”password"
  }
}
```

Окрім цього слід оновити файл startup.cs додавши метод Configure. Даний метод використовується для реєстрації служб в контейнері впровадження залежностей (DI - Dependency Injection). Цей метод викликається під час запуску програми і дозволяє додавати необхідні служби, конфігурації та налаштування, які будуть використовуватися в додатку.

За допомогою Package Manager Console підключаємо застосування міграцій (рисунок 2.7). Дана команда створить файли міграції, які відображають поточну модель даних.

```
Add-Migration InitialCreate
```

Рисунок 2.7 — Створення файлів міграцій

За допомогою команди “Update-Database” застосуємо створені міграції до бази даних, створюючи необхідні таблиці та інші об'єкти.

2.5 Реалізація служб аутентифікації та авторизації

Авторизація в ASP.NET Core включає в себе процес аутентифікації користувачів і визначення їх прав доступу до різних ресурсів. Процес реалізації даних служб описаний нижче.

Перш за все, потрібно налаштувати служби аутентифікації та авторизації в методі ConfigureServices в Startup.cs.

Лістинг 2.4 - Реалізація методу ConfigureServices

```
public void ConfigureServices(IServiceCollection services)
   {
        // Додавання підтримки контролерів з представленнями (MVC)
        services.AddControllersWithViews();
        // Налаштування контексту бази даних
       services.AddDbContext<ApplicationDbContext>(options =>
options.UseNpgsql(Configuration.GetConnectionString("DefaultConnection"))
);
      // Налаштування аутентифікації за допомогою кукі
services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationSche
me)
         .AddCookie(options =>
         {
              options.LoginPath = "/Account/Login";
// Шлях до сторінки входу
            options.AccessDeniedPath = "/Account/AccessDenied";
// Шлях до сторінки відмови в доступі
         });
      // Налаштування політик авторизації
      services.AddAuthorization(options =>
      {
        // Додавання політики, що вимагає роль "Admin"
options.AddPolicy("RequireAdminRole", policy => policy.
RequireRole("Admin"));
      });
   }
```

Метод ConfigureServices у класі Startup налаштовує необхідні сервіси для додатку ASP.NET Core. Цей метод приймає параметр IServiceCollection, який використовується для реєстрації сервісів у контейнері впровадження залежностей.

Спочатку метод додає підтримку контролерів з представленнями, що дозволяє використовувати MVC-патерн. Це робиться за допомогою виклику services.AddControllersWithViews(). Даний метод додає всі необхідні служби для роботи з контролерами та представленнями, включаючи підтримку моделей, контролерів і видів.

Далі налаштовується контекст бази даних. У даному проекту використовується база даних PostgreSQL. Контекст бази даних налаштовується за допомогою методу AddDbContext<ApplicationDbContext>, де ApplicationDbContext представляє клас контексту бази даних. Метод options.UseNpgsql використовується для налаштування контексту бази даних з використанням з'єднувального рядка, отриманого з конфігурації додатку (Configuration.GetConnectionString("DefaultConnection")).

Наступним кроком є налаштування аутентифікації за допомогою кукі. Це робиться за допомогою методу services.AddAuthentication, де вказується схема аутентифікації, яка використовується за замовчуванням (CookieAuthenticationDefaults.AuthenticationScheme). Потім додається налаштування кукі за допомогою методу AddCookie, де можна вказати параметри аутентифікації, такі як шлях до сторінки входу (options.LoginPath = "/Account/Login") і шлях до сторінки відмови в доступі (options.AccessDeniedPath = "/Account/AccessDenied").

Останнім кроком є налаштування політик авторизації. Це робиться за допомогою методу services.AddAuthorization, який приймає делегат для налаштування політик. У цьому прикладі додається політика з назвою RequireAdminRole, яка вимагає, щоб користувач мав роль "Admin". Ця політика додається за допомогою методу options.AddPolicy, де вказується ім'я політики та конфігурація політики (policy.RequireRole("Admin")).

Наступним кроком буде прописання команди “services.AddControllers WithViews();”, яка додає служби, необхідні для роботи контролерів MVC і представлень, та налаштування контексту бази даних.

Після налаштування контексту налаштовуються аутентифікації за допомогою кукі.

Лістинг 2.5 - Налаштування аутентифікації

```
services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(options =>
    {
        options.LoginPath = "/Account/Login"; // Шлях до сторінки входу
        options.AccessDeniedPath = "/Account/AccessDenied"; // Шлях до сторінки відмови в доступі
    });
```

Даний фрагмент коду налаштовує аутентифікацію в ASP.NET Core з використанням кукі. Спочатку метод “services.AddAuthentication” викликається з параметром “CookieAuthenticationDefaults.AuthenticationScheme”, що задає схему аутентифікації за замовчуванням, яку буде використовувати додаток. Це означає, що додаток буде використовувати кукі для зберігання інформації про аутентифікованих користувачів.

Далі, метод AddCookie використовується для налаштування параметрів кукі-аутентифікації. Внутрішній блок коду, що передається в метод AddCookie, задає два основних параметри. Параметр “options.LoginPath” встановлює шлях до сторінки входу, яка буде відображатися, коли неаутентифікований користувач намагатиметься отримати доступ до захищеного ресурсу. У цьому випадку, шлях до сторінки входу встановлюється на "/Account/Login".

Другий параметр “options.AccessDeniedPath” визначає шлях до сторінки відмови в доступі. Ця сторінка буде відображатися, коли аутентифікований

користувач намагатиметься отримати доступ до ресурсу, до якого у нього немає дозволу. У цьому випадку, шлях до сторінки відмови в доступі встановлюється на "/Account/AccessDenied". Ці налаштування забезпечують керування навігацією користувача у випадках, коли потрібно здійснити вхід або коли доступ до ресурсу заборонений.

Наступним кроком створюється контролер AccountController для керування аутентифікацією користувачів. Фрагмент даного коду та зміст папки Controllers після оновлення класів наведено на рисунку 2.8.

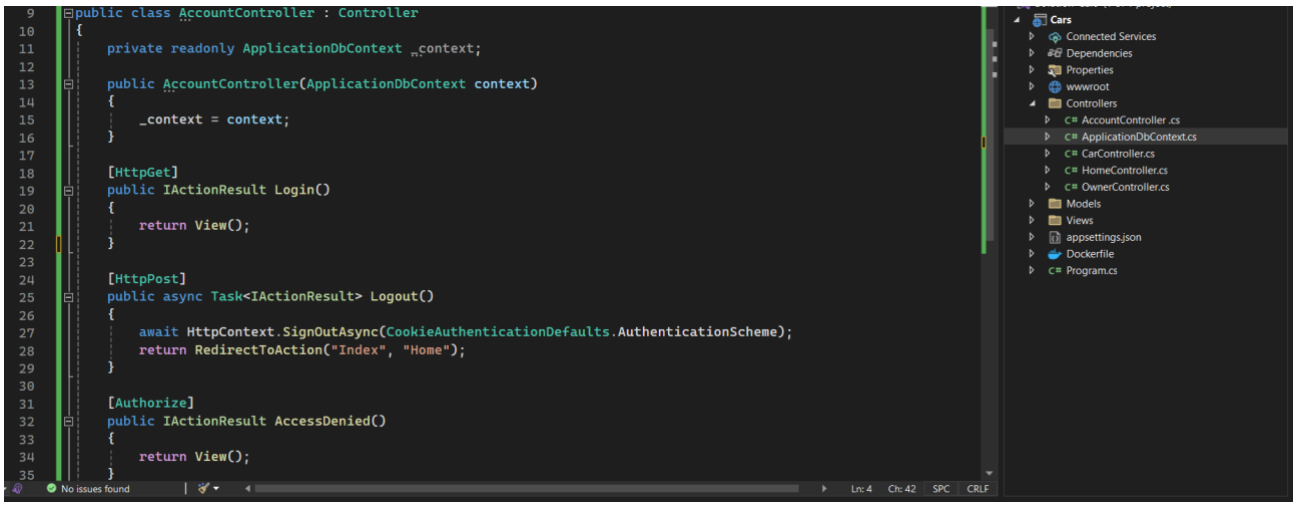

Рисунок 2.8 — Вміст папки Controllers та реалізація фрагменту коду

На цьому фрагменті коду ми маємо контролер AccountController, який відповідає за автентифікацію та авторизацію користувачів у системі. Реалізація даного коду виконується так:

1) контролер має конструктор, який приймає об'єкт контексту бази даних ApplicationDbContext. Це дозволяє контролеру взаємодіяти з базою даних для перевірки автентифікаційних даних користувача;

2) метод Login (GET) відображає сторінку входу для користувача. Він викликається при HTTP GET-запиті на шлях /Account/Login;

3) метод Login (POST) обробляє відправлену форму входу. Він перевіряє введені користувачем дані, перевіряє їх правильність в базі даних та

аутентифікує користувача, якщо дані вірні. Після успішної аутентифікації він перенаправляє користувача на домашню сторінку;

4) метод Logout відображається при виклику HTTP POST-запиту на шлях /Account/Logout. Він видаляє куки аутентифікації, щоб вийти з системи, та перенаправляє користувача на домашню сторінку;

5) метод AccessDenied відображає сторінку доступу заборонено, якщо користувач намагається отримати доступ до ресурсу, на який він не має дозволу.

2.6 Створення представлень для служб аутентифікації та авторизації

Для створення представлень для служб аутентифікації та авторизації в ASP.NET Core потрібно спочатку створити HTML-файли, які будуть відображати форми входу та інші сторінки.

Усі ці представлення повинні зберігатися у папці Views/Account проекту ASP.NET Core (рисунок 2.9).

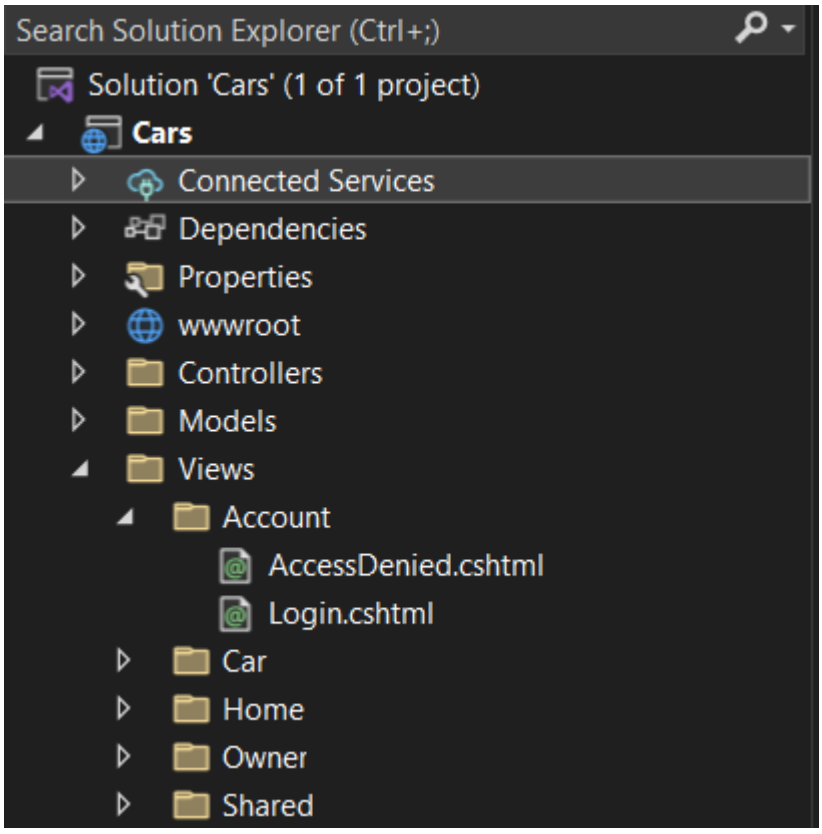

Рисунок 2.9 — Створення .cshtml файлів для реалізації представлень

Представлення для сторінки входу має форму, яка використовує метод HTTP POST для відправлення інформації для аутентифікації. Після введення користувачем інформації і натискання кнопки "Login", дані будуть надіслані на сервер для перевірки. У випадку невірних даних або невдачної аутентифікації може відображатися повідомлення про помилку. Графічне представлення форми авторизації зображена на рисунку 2.10.

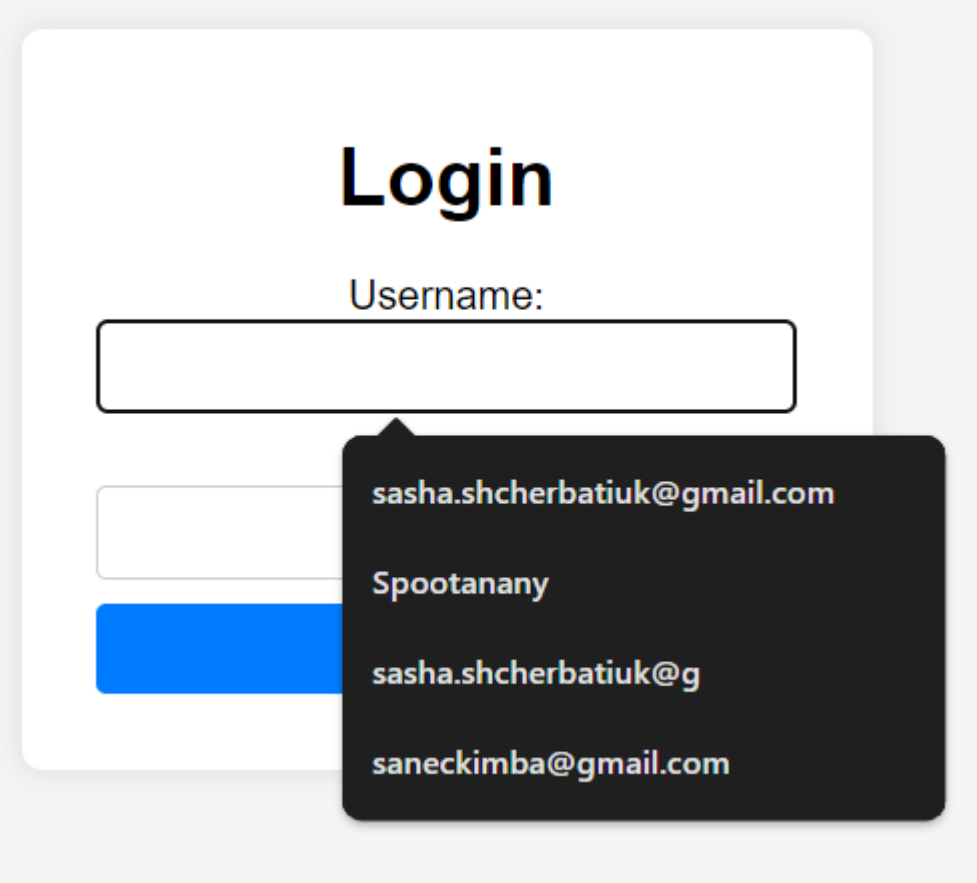

Рисунок 2.10 — Графічне представлення форми авторизації

В першу чергу для написання системи на даний момент використовувалися не реальні, а тестові дані, наближені до реальних. Приклад тестового зразка інформаційної системи показаний на рисунку 2.11. На даному рисунку представлена головна вкладка, яка дозволяє взаємодіяти із даними, отримувати детальну інформацію, редагувати дані та створювати нові. При переході на вкладку детальна інформація можна буде більш детально дізнатися інформацію про автомобіль. На рисунку 2.12 зображено вкладку редагування даних.

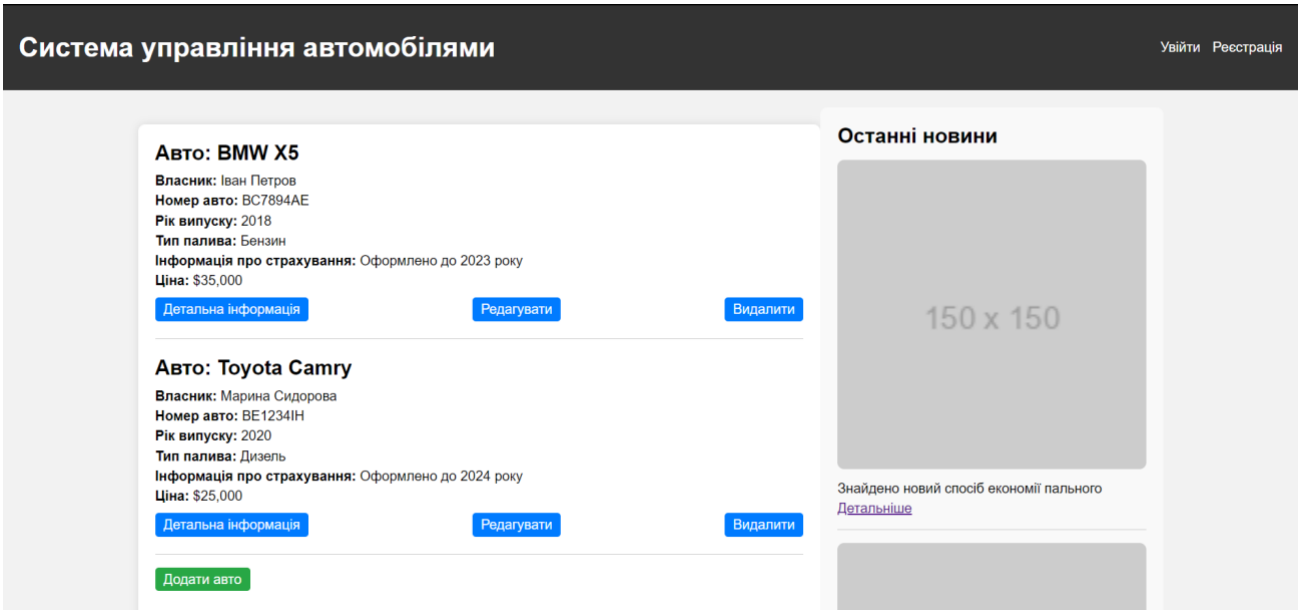

Рисунок 2.11 — Графічне представлення головної вкладки

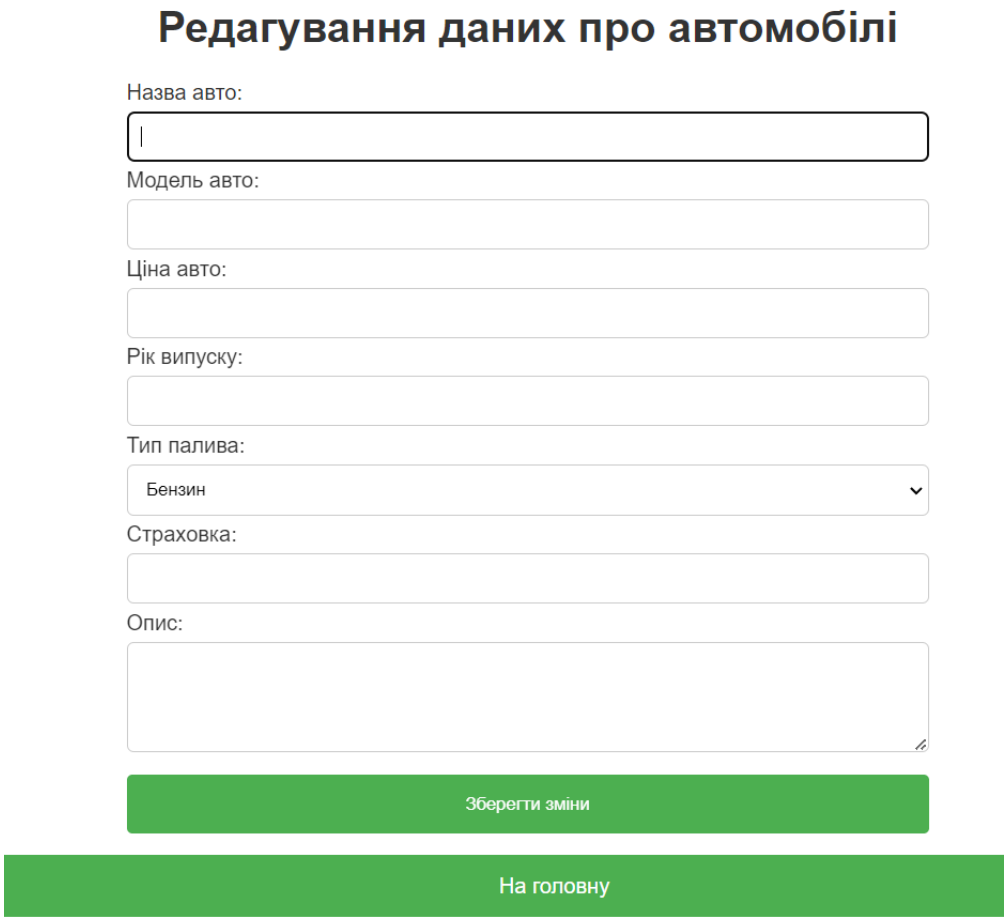

Рисунок 2.12 — Вкладка редагування даних

3 РОЗРОБКА ХОСТИНГОВОЇ ЧАСТИНИ

3.1 Налаштування та завантаження Docker

Для встановлення Docker на комп'ютер, спершу необхідно перевірити, чи операційна система підтримує Docker. Після перевірки слід зробити наступні дії:

1) завантаження Docker - необхідно перейти на офіційний сайт Docker та відкрити сторінку завантаження. Там необхідно обрати потрібну версію Docker, яка відповідає операційній системі;

2) встановлення Docker - запуск завантаженого файлу і виконання інструкції по встановленню;

3) запуск Docker - після успішного встановлення необхідно запустити Docker зі стартового меню або додаткових іконок у випадаючому меню;

4) перевірка встановлення - потрібно відкрити командний рядок та ввести команду docker --version, щоб переконатися, що Docker успішно встановлено і правильно працює. Якщо ви відображається версія Docker, то встановлення пройшло успішно (рисунок 3.1).

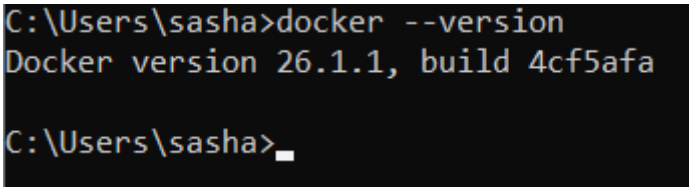

Рисунок 3.1 — Перевірка версії Docker

Після встановлення можна розпочати використовувати Docker для розгортання та керування контейнерами з програмним забезпеченням.

Контейнери - це віртуальні середовища, які дозволяють упаковувати та виконувати програмне забезпечення та його залежності разом з усім необхідним середовищем. Кожен контейнер ізольований від інших контейнерів та має своє власне файло-систему, бібліотеки, змонтовані точки та мережеві інтерфейси.

Основними компонентами контейнерів є образи та контейнери. Образ - це статичний шаблон, що включає в себе програмне забезпечення, його

конфігурацію та залежності. Контейнер - це інстанція образу, яка може бути запущена, виконана та зупинена. Кожен контейнер базується на образі та має свій власний стан та файлову-систему, які можуть бути змінені під час виконання.

Контейнери використовують легковагові віртуалізаційні технології, такі як контроль ядра та простір користувача, щоб забезпечити ізоляцію та безпеку. Вони дозволяють запускати та виконувати програмне забезпечення в різних середовищах без втрати продуктивності або стабільності.

Контейнери також допомагають полегшити процес розгортання та керування програмним забезпеченням. Вони дозволяють швидко створювати та запускати ізольовані середовища, що містять необхідне програмне забезпечення та його залежності. Крім того, контейнери можуть бути легко масштабовані та керовані за допомогою інструментів автоматизації та оркестрації, таких як Docker Compose та Kubernetes.

3.2 Налаштування та завантаження PostgreSQL на Docker

PostgreSQL - це потужна та добре відома система управління базами даних (СУБД), яка дозволяє зберігати та керувати великими обсягами даних. Вона є однією з найпопулярніших відкритих реляційних баз даних, і багато людей використовують її як основу для своїх проектів.

Для встановлення PostgreSQL через Docker потрібно встановити та запустити контейнер з образом PostgreSQL, який буде доступний для роботи з вашими даними. Перш ніж створювати контейнер потрібно завантажити останню версію образу PostgreSQL з Docker Hub на комп'ютер (рисунок 3.2).

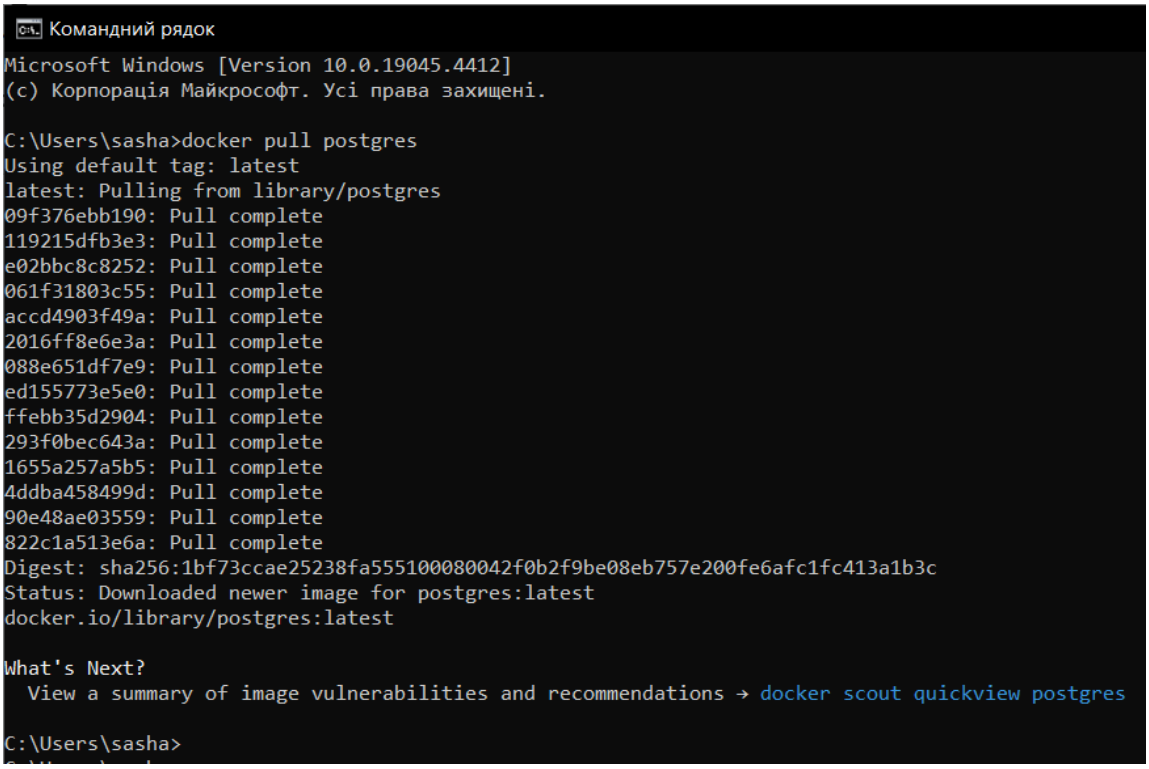

Рисунок 3.2 — Завантаження останньої версії PostgreSQL

Встановлення контейнера можна зробити за допомогою команди наведеної на рисунку 3.3. Дана команда створить контейнер з назвою my_postgres та заданим паролем для користувача postgres як mysecretpassword.

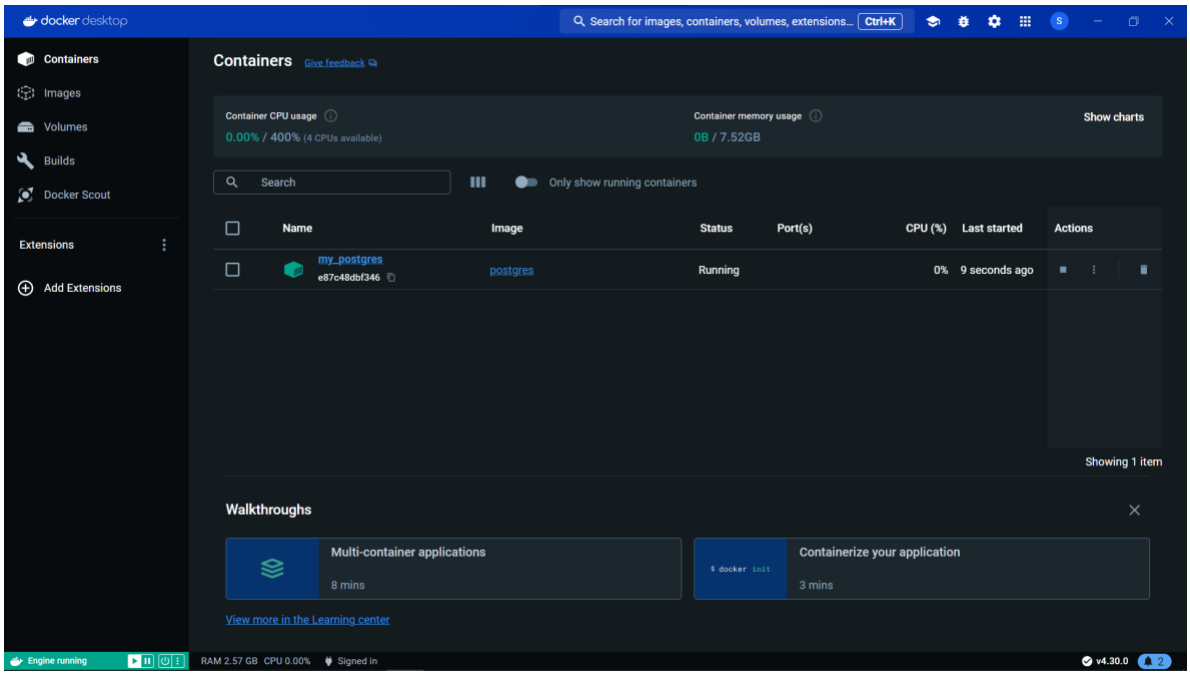

Рисунок 3.3 — Результат створення контейнера

Після створення контейнера можена перевірити його стан за допомогою команди “docker ps”. Ця команда виведе список активних контейнерів (рисунок 3.4).

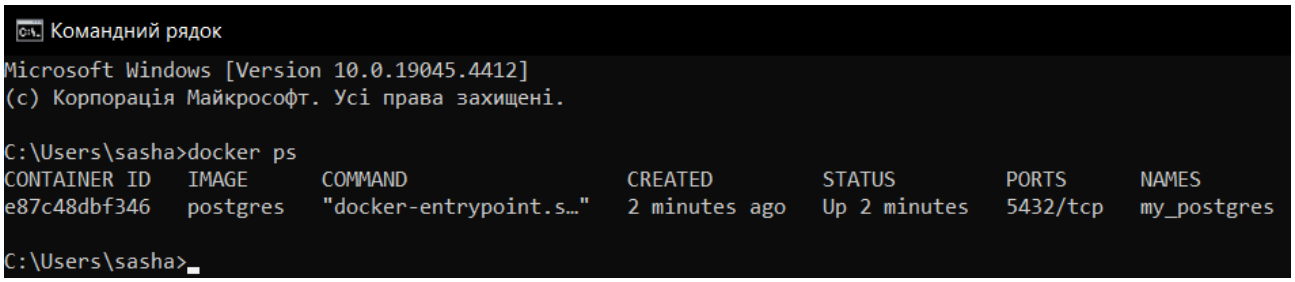

Рисунок 3.4 — Відображення списку контейнерів

Щоб підключитися до вашого сервера PostgreSQL, потрібно скористатися командою psql (рисунок 3.5).

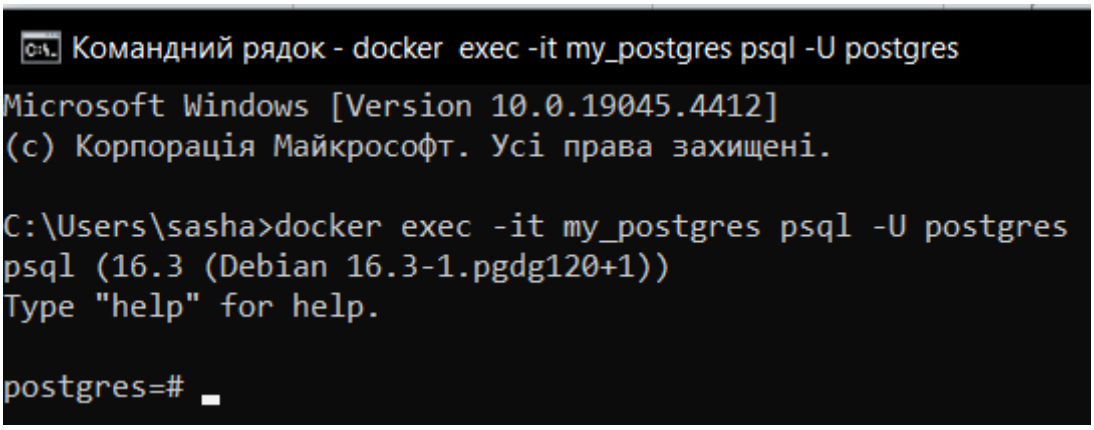

Рисунок 3.5 — Підключення до сервера

Дана команда використовується для запуску інтерактивного інтерфейсу Нижче детально описаний цей процес:

1) docker exec - це команда Docker, яка дозволяє виконувати команди всередині запущеного контейнера;

2) -it - дані прапорці вказують Docker на запуск команди в інтерактивному режимі та забезпечують з'єднання терміналу з контейнером;

3) my_postgres - це ім'я контейнера PostgreSQL, до якого потрібно підключитися;

4) psql -U postgres - це команда, яка викликає інтерфейс командного рядка PostgreSQL, відомий як psql. Параметр "-U postgres" вказує, що ми хочемо увійти в систему як користувач "postgres". Це стандартний користувач за замовчуванням у PostgreSQL.

Після введення цієї команди і натискання клавіші Enter можна увійти в інтерфейс psql всередині контейнера Docker. Тут можна виконувати різноманітні команди SQL, такі як створення таблиць, вставка даних, вибірка даних та багато іншого.

Режим psql дозволяє вам взаємодіяти з базою даних в командному рядку. Можна виконувати SQL-запити, переглядати та змінювати структуру бази даних, керувати ролями та дозволами користувачів, і багато іншого.

В результаті розробки , вийшло дві сутності (таблиці): car, user.

На рисунках 3.6 та 3.7 показано поля сутностей car та user.

Column Name	#	Data type	Identity	Collation	Not Null
name	1	varchar		default	[v]
model	2	varchar		default	[v]
price	3	_numeric			[v]
year_production	4	int4			[v]
fuel type	5	varchar		default	[v]
insurance	6	bool			[ ]
description	7	text		default	[ ]
id	8	serial4			[v]

Рисунок 3.6 — Поля сутності car

Column Name	#	Data type	Identity	Collation	Not Null
id	1	serial4			[v]
name	2	varchar		default	[v]
phone_number	3	varchar		default	[ ]
mail	4	varchar		default	[ ]
car_id	5	serial4			[v]

Рисунок 3.7 — Поля сутності user

На рисунку 3.8 показано ER-діаграму бази даних.

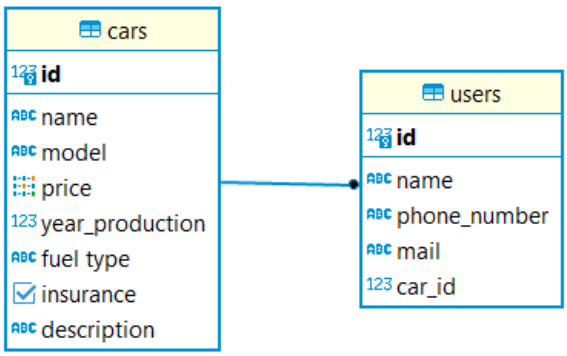

Рисунок 3.8 — ER-діаграма БД

Одна з переваг використання Docker для PostgreSQL - це можливість швидко розгорнути базу даних у будь-якому середовищі розробки або продукції, не турбуючись про конфлікти версій або інші залежності.

Однією з ключових переваг PostgreSQL є його відкритий та безкоштовний характер, що робить його доступним для використання у різних проектах без значних витрат на ліцензії. Крім того, PostgreSQL є потужним інструментом з багатьма функціями, включаючи підтримку транзакцій, конкурентність, реплікацію та велику кількість розширень.

Підключення до PostgreSQL у ASP.NET додатках може бути здійснено за допомогою різних методів, таких як використання драйверів баз даних, наприклад, Npgsql, та формування рядка підключення.

3.3 Процес взаємодії між ASP.NET та PostgreSQL

Коли ASP.NET додаток потребує доступу до бази даних PostgreSQL, він використовує драйвер бази даних, наприклад, Npgsql, для встановлення з'єднання. Цей драйвер функціонує як посередник між додатком та базою даних, дозволяючи їм обмінюватися даними.

Перш ніж почати роботу з базою даних, додаток повинен налаштувати параметри підключення до PostgreSQL. Ці параметри включають в себе

інформацію про хост бази даних, порт, ім'я бази даних, а також ім'я користувача та пароль для доступу.

Після успішного встановлення з'єднання додаток може виконувати різноманітні операції з даними, такі як вибірка, вставка, оновлення та видалення записів. Наприклад, при вході користувача на веб-сайт, ASP.NET додаток може виконати запит до бази даних, щоб перевірити, чи існує користувач з введеними обліковими даними.

Отримані дані можуть бути подальше оброблені додатком. Наприклад, результати запитів можуть бути використані для генерації вмісту сторінок веб-сайту або для аналітичних цілей, таких як створення звітів або графіків.

Таким чином, взаємодія між ASP.NET та PostgreSQL забезпечує ефективну обробку та зберігання даних у веб-додатках, що дозволяє розробникам створювати потужні та функціональні програми.

Загалом, робота з PostgreSQL у контексті розробки веб-додатків на платформі ASP.NET може бути дуже задоволенням, оскільки вона надає розробникам потужний та надійний інструмент для зберігання та керування даними, що дозволяє створювати високоякісні та ефективні програми.

ВИСНОВКИ

У рамках даної дипломної роботи була проведена розробка системи обліку автомобілів. Задля досягнення поставленої мети було використано сучасні технології веб-розробки, а саме ASP.NET MVC, що дозволяє писати мовою C#, Docker та PostgreSQL, які дозволяють запустити локальний хост на власному комп'ютері.

Аналізуючи предметну область, було виявлено необхідність створення власної системи обліку автомобілів, внаслідок малої конкуренції на ринку, а також необхідності в створенні унікального веб-додатку для бізнесу, який являється більш гнучкий та оптимізований під надані потреби.

Під час розробки клієнтської частини було використано технології ASP.NET, які забезпечують ефективну інтеграцію з серверною частиною додатку. ASP.NET надає широкі можливості для створення веб-інтерфейсів з високою швидкодією та масштабованістю. Використання ASP.NET дозволяє створювати сучасні, інтуїтивно зрозумілі веб-додатки з використанням мов програмування, таких як C# або VB.NET, і багатством готових компонентів та бібліотек для розширення функціоналу. Такий підхід сприяє швидкому розвитку та підтримці клієнтської частини системи обліку автомобілів, забезпечуючи високу якість та зручність для кінцевих користувачів.

Хостингова частина була запущена за допомогою PostgreSQL та Docker, що забезпечило ефективне управління базою даних та ізолювало середовище роботи. Використання PostgreSQL як системи управління базами даних забезпечує надійність, стабільність та швидкодію обробки даних. Docker надає можливість легко розгортати та керувати контейнерами, що містять хостингову частину системи, забезпечуючи гнучкість та простоту управління інфраструктурою. Використання цих технологій у поєднанні дозволило створити масштабовану, надійну та зручну для управління хостингову частину інформаційної системи ведення обліку автомобілів.

Сайт інтегрує форми реєстрації та форми роботи з даними, що значно полегшує користувачам процес роботи з системою. Використання таких функціональних елементів дозволяє забезпечити користувачів необхідною гнучкістю та можливістю ефективно взаємодіяти з системою, надаючи їм зручність у введенні та обробці даних.

Форма реєстрації дозволяє користувачам створювати облікові записи в системі, що забезпечує персоналізований доступ та підтримку для подальшої роботи. Створення даних через відповідні форми дозволяє користувачам додавати нові записи або оновлювати існуючі, забезпечуючи актуальність та повноту інформації в системі.

Використання цих функціональних можливостей дозволяє створити зручне та інтуїтивно зрозуміле середовище для користувачів, що сприяє підвищенню продуктивності та ефективності їх роботи з системою обліку автомобілів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ASP.NET Core. URL: https://dotnet.microsoft.com/en-us/apps/aspnet (дата звернення: 08.05.2024).

2. ASP Documents. URL: https://asp.icc-cpi.int/ASPDocIndex (дата звернення: 10.05.2024).

3. MVC Design Pattern. URL: https://www.geeksforgeeks.org/mvc-design-pattern/ (дата звернення: 10.05.2024).

4. Model–view–controller. URL: https://en.wikipedia.org/wiki/Model%E2%80%9 3view%E2%80%93controller (дата звернення: 24.04.2024).

5. Ознайомлення з патерном MVC. URL: https://javarush.com/ua/groups /posts/uk.2536.chastina-7-oznayomlennja-z-paternom-mvc-model-view-controller (дата звернення: 27.04.2024).

6. Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application. URL: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application (дата звернення: 02.05.2024).

7. Unit of Work Application. URL: https://martinfowler.com/eaaCatalog /unitOfWork.html (дата звернення: 06.05.2024).

8. Repository Pattern. URL: https://deviq.com/design-patterns/repository-pattern (дата звернення: 08.05.2024).

9. Repository Design Pattern. URL: https://www.geeksforgeeks.org /repository-design-pattern/ (дата звернення: 02.05.2024).

10. Docker. URL: https://qagroup.com.ua/publications/shcho-take-docker-i-navishcho-vin/ (дата звернення: 13.05.2024).

11. DOCKER: НОВИЙ ПіДХіД ДО РОЗРОБКИ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. URL: https://freehost.com.ua/ukr /faq/wiki/docker-novij-podhod-k-razrabotke-i-vnedreniju-programmnogo-obespechenija/ (дата звернення: 09.05.2024).

12. PostgreSQL: The World's Most Advanced Open Source Relational Database. URL: https://www.postgresql.org/ (дата звернення: 10.05.2024).

ДОДАТКИ

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
_____ проф., д.т.н. О.Д. Азаров
«__» _____ 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ
Інформаційна система обліку транспортних засобів
08-54. БДР.026.00.000 ТЗ

Керівник роботи: к.т.н., доц. каф. ОТ
_____Кисюк Д.В.
Виконав: студентка гр. 1КІ-22мс
_____Щербатюк О.В.

ВНТУ 2024 рік

1 Підстава для виконання дипломної роботи (БДР)

1.1 Тема бакалаврської дипломної роботи «Інформаційна система обліку транспортних засобів».

1.2 Наказ про затвердження теми бакалаврської дипломної роботи.

2 Мета БДР і призначення розробки

2.1 Мета роботи – розробка інформаційної системи обліку автомобілів, спрямованої на автоматизацію процесів обліку, моніторингу та аналізу даних про транспортні засоби. Розроблена система має за завдання забезпечити зручний доступ до інформації про автопарк, підвищити ефективність управління транспортними ресурсами та зменшити час та зусилля, необхідні для проведення аналізу даних.

2.2 Призначення розробки –

3 Вихідні дані для виконання БДР

3.1 Аналіз предметної області та технологій розробки.

3.2 Аналіз та обгрунтування вибору технологій.

4 Технічні вимоги до виконання БДР

5 Етапи роботи та очікуванні результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи БДР

№ з/п	Назва етапів дипломної роботи	Строк виконання	Підпис
1	Обґрунтування доцільності розробки інформаційної системи		
2	Розробка клієнтської частини системи обліку автомобілів		
3	Розробка хостингової частини		
4	Оформлення пояснювальної записки		
5	Перевірка якості виконання бакалаврської роботи та усунення недоліків		

6 Матеріали, що подаються до захисту БДР

До захисту подаються: пояснювальна записка БДР, графічні та ілюстративні матеріали, протокол попереднього захисту роботи на кафедрі, відгук наукового керівника, рецензія опонента, анотації українською та іноземною мовами, нормоконтроль про відповідність оформлення діючим вимогам.

7 Порядок контролю виконання та захисту БДР

Виконання етапів графічної та розрахункової документації БДР контролюється науковим керівником згідно зі встановленими термінами. Захист відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлювання та порядок виконання БДР

8.1 При оформлювання БДР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— Методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія»). Кафедра обчислювальної техніки ВНТУ 2022;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання БДР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:2

ДОДАТОК В

Лістинг файлу AccountController.cs

```
using Cars.Controllers;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;
using System.Threading.Tasks;


public class AccountController : Controller
{
    private readonly ApplicationDbContext _context;

    public AccountController(ApplicationDbContext context)
    {
        _context = context;
    }


    [HttpGet]
    public IActionResult Login()
    {
        return View();
    }


    [HttpPost]
    public async Task<IActionResult> Logout()
    {
        await
HttpContext.SignOutAsync(CookieAuthenticationDefaults.
    AuthenticationScheme);
        return RedirectToAction("Index", "Home");
    }
```

```
    [Authorize]
    public IActionResult AccessDenied()
    {
        return View();
    }
}
```

ДОДАТОК Г

Лістинг файлу CarController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace CarsInfo.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class CarController : ControllerBase
    {
        private readonly ApplicationDbContext _context;

        public CarController(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: api/Car
        [HttpGet]
        public async Task<ActionResult<IEnumerable<Car>>> GetCars()
        {
            return await _context.Cars.ToListAsync();
        }

        // GET: api/Car/5
        [HttpGet("{id}")]
        public async Task<ActionResult<Car>> GetCar(int id)
        {
            var car = await _context.Cars.FindAsync(id);

            if (car == null)
            {
                return NotFound();
```

```
                }

                return car;
            }
            // POST: api/Car
            [HttpPost]
            public async Task<ActionResult<Car>> PostCar(Car car)
            {
                _context.Cars.Add(car);
                await _context.SaveChangesAsync();

                return CreatedAtAction(nameof(GetCar), new { id =
car.Id }, car);
            }
            // DELETE: api/Car/5
            [HttpDelete("{id}")]
            public async Task<IActionResult> DeleteCar(int id)
            {
                var car = await _context.Cars.FindAsync(id);
                if (car == null)
                {
                    return NotFound();
                }
                _context.Cars.Remove(car);
                await _context.SaveChangesAsync();

                return NoContent();
            }
        }
    }
```

ДОДАТОК Д

Лістинг файлу OwnerController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace CarsInfo.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class OwnerController : ControllerBase
    {
        private readonly ApplicationDbContext _context;

        public OwnerController(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: api/Owner
        [HttpGet]
        public     async     Task<ActionResult<IEnumerable<Owner>>>
GetOwners()
        {
            return await _context.Owners.ToListAsync();
        }

        // GET: api/Owner/5
        [HttpGet("{id}")]
        public async Task<ActionResult<Owner>> GetOwner(int id)
        {
            var owner = await _context.Owners.FindAsync(id);

            if (owner == null)
            {
```

```
                return NotFound();
            }

            return owner;
        }

        // POST: api/Owner
        [HttpPost]
        public async Task<ActionResult<Owner>> PostOwner(Owner owner)
        {
            _context.Owners.Add(owner);
            await _context.SaveChangesAsync();

            return CreatedAtAction(nameof(GetOwner), new { id = owner.Id }, owner);
        }

        // DELETE: api/Owner/5
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteOwner(int id)
        {
            var owner = await _context.Owners.FindAsync(id);
            if (owner == null)
            {
                return NotFound();
            }

            _context.Owners.Remove(owner);
            await _context.SaveChangesAsync();

            return NoContent();
        }
    }
}
```

ДОДАТОК Е

Сторінки додатку

Рисунок Е.1 — Головна сторінка платформи JSolutions

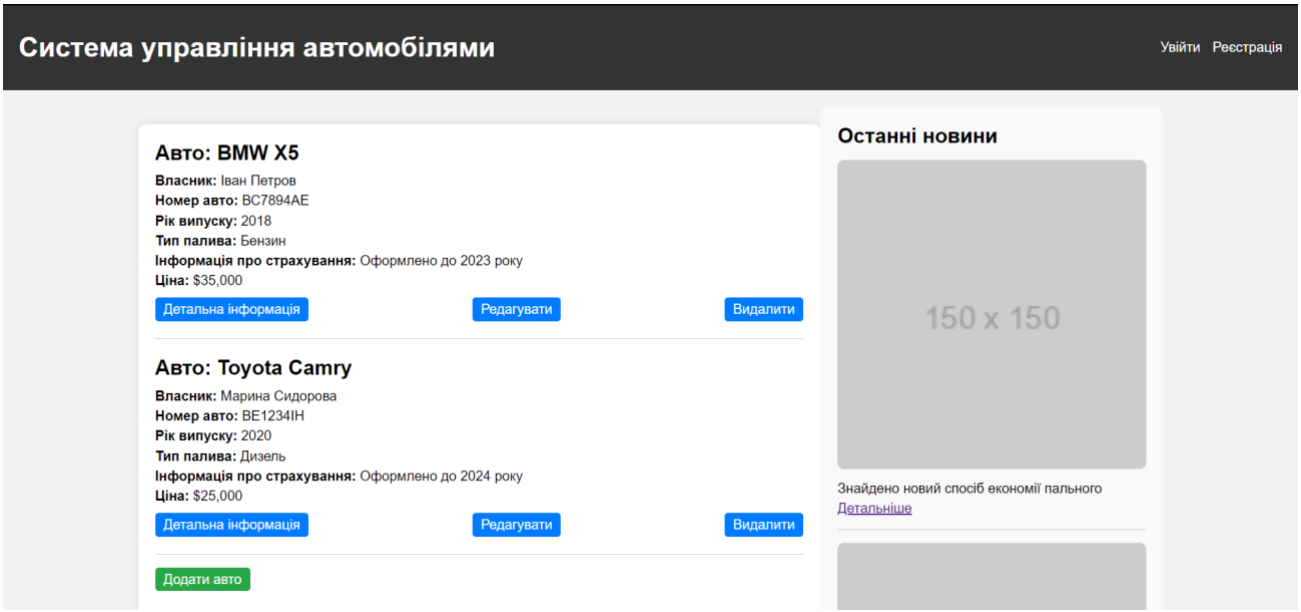

Рисунок Е.2 — Графічне представлення головної вкладки

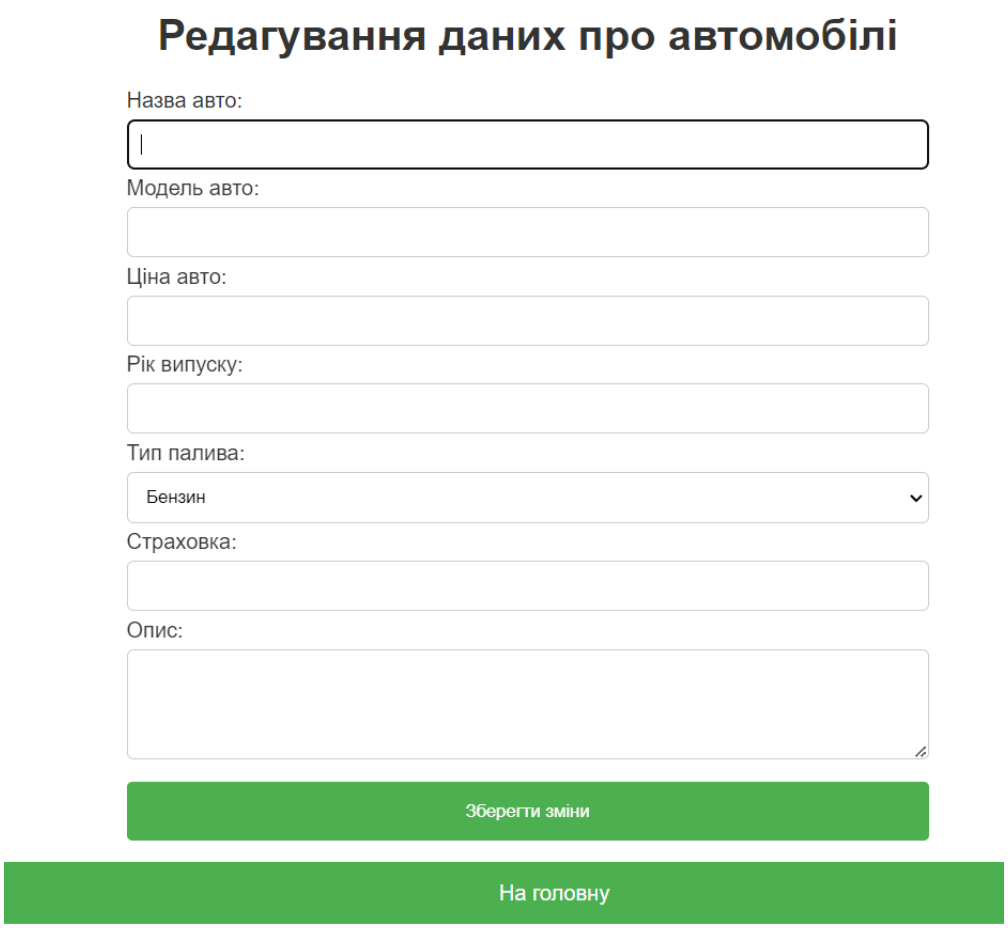

Рисунок Е.3 — Вкладка редагування даних

ДОДАТОК Ж

ПРОТОКОЛ

ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Інформаційна система обліку транспортних засобів

Тип роботи: бакалаврська дипломна робота

(БДР, МКР)

Підрозділ кафедра обчислювальної техніки

(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність % Схожість %

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С. М.

(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Щербатюк О.В.

(підпис) (прізвище, ініціали)

Керівник роботи _____ Кисюк Д. В.

(підпис) (прізвище, ініціали)