Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавра
(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПІДТРИМКИ КОНТРОЛЮ ЗА ТЕХНІЧНИМ ОБСЛУГОВУВАННЯМ АВТОМОБІЛІВ
SOFTWARE FOR CAR MAINTENANCE TRACKING

Виконав: студент(ка) 4 курсу, групи КНТ-218сп
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)
Освітня програма (спеціалізація)
Комп'ютерні науки

Тодоров М.А.
(прізвище та ініціали)
Керівник Скрупський С.Ю.
(прізвище та ініціали)
Рецензент Зеленьова І.Я.
(прізвище та ініціали)

2021

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____С.О. Субботін
"____" _____2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Тодорова Микити Анатолійовича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення підтримки контролю за технічним обслуговуванням автомобілів
Software for Car Maintenance Tracking

керівник проєкту (роботи) Скрупський Степан Юрійович, к.т.н., доцент ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від " 30 " березня 2021 року № 103
2. Строк подання студентом проєкту (роботи) 18.05.2021
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Розробка програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Скрупський С.Ю., доцент		
Нормоконтролер	Андреєв М.О., асистент		

7. Дата видачі завдання “ 12 ” березня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка) _____ Тодоров М.А.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ Скрупський С.Ю.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 83 с., 29 рис., 1 табл., 3 дод., 15 джерел.

ТЕХНІЧНЕ ОБСЛУГОВУВАННЯ, АВТОМОБІЛЬ, МЕНЕДЖЕР З ОБСЛУГОВУВАННЯ, ПОСТАЧАЛЬНИКИ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

Об’єкт роботи – процес технічного обслуговування автомобілів.

Предмет роботи – програмне забезпечення підтримки контролю за технічним обслуговуванням автомобілів.

Мета роботи – розроблення програмного забезпечення підтримки контролю за технічним обслуговуванням автомобілів для забезпечення розподіленого підходу до обслуговування.

Матеріали, методи та технічні засоби: мова програмування Python, вебфреймворк Django, система керування базами даних MySQL.

Результати. Виконано аналіз проблеми і програмних аналогів, на основі цього визначено функціональні вимоги до програми, що дозволило забезпечити моделювання сценаріїв використання програми, виконати проєктування бази даних, визначити необхідні для реалізації цієї функціональності в межах шаблону Model View Controller класи та реалізувати їх і програмне забезпечення загалом.

Висновки. Програма призначена для централізованого перегляду і керування даними технічного обслуговування автомобілів. Програма дозволяє фіксувати виконані роботи за автомобілем, параметри їх виконання, використані деталі для проведення ремонту або обслуговування, переглядати результати обслуговування як за замовленнями загалом, так і за виконаними роботами.

Галузь використання – технічне обслуговування автомобілів з потенційною наявністю системи розподілених відділень.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 83 pages, 29 figures, 1 table, 3 appendixes, 15 sources.

MAINTENANCE, CAR, MAINTENANCE MANAGER, SUPPLIERS, SOFTWARE.

The object of work is the process of car maintenance.

The subject of work is software for car maintenance tracking.

The purpose of the work is to develop software to support the maintenance of cars to ensure a distributed approach to maintenance.

Materials, methods and technical means: Python programming language, Django web framework, MySQL database management system.

Results. The problem and software analogues were analyzed. The functional requirements of the software were determined based on the obtained results. It allowed to provide modeling of software usage scenarios, database design, determine necessary classes for implementation of this functionality within the Model View Controller template and implement these classes and software in general.

Conclusions. The program is designed for centralized viewing and management of car maintenance data. The software allows to record works performed on the car maintenance, parameters of car maintenance, the parts used for repairs or maintenance, to view the results of maintenance both on orders in general and on the work performed.

Field of usage is maintenance of cars with the potential presence of a system of distributed departments.

ЗМІСТ

С.

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

MVC – Model View Controller;

VIN – vehicle identification number;

БД – база даних.

ВСТУП

Щодо розвитку галузі автосервісів можна виділити наступне:

– автосервісна галузь зіткнулася з серйозними проблемами в 2020 році, прогнозована втрата склала 1,8 мільярда доларів від економічних зривів, спричинених пандемією, однак, як очікується, цього року автомобільна галузь відновиться, 18% американських домогосподарств мають скористатися послугами авторемонту, що відіб'ється в річному рівні зростання в 7%, у свою чергу світовий ринок авторемонту та технічного обслуговування може навіть досягти 828,6 млрд доларів у 2023 році з 691,7 млрд доларів у 2020 році та 693,5 млрд доларів у 2019 році;

– автосервісні майстерні все одно повинні бути пильними у розумінні нових інструментів та тенденцій: як і в інших галузях, технології дедалі більше інтегруються в роботу автосервісних майстерень, компанії з технічного обслуговування автомобілів використовують інтернет речей для отримання даних про компоненти в реальному часі через датчики в транспортних засобах, іншою галузевою тенденцією є телематика, яка збирає та передає дані про використання транспортних засобів, вимоги до технічного обслуговування та обслуговування автомобілів, програмне забезпечення для технічного обслуговування автомобілів може допомогти авторемонтним майстерням адаптуватися до обох цих тенденцій, дозволяючи автомобільним технікам проводити планові перевірки та інспекції з цифрової системи, яку можна підключити до датчиків та інших розумних пристроїв;

– хоча інтернет речей та телематика є прикладами тенденцій, які можуть сприяти зростанню галузі авторемонту, є декілька тенденцій, які можуть зашкодити світовому розвитку, такі як збільшення лізингу автомобілів та зростаюча прихильність до того, щоб дилерські центри займалися ремонтом, автосервісні майстерні можуть пом'якшити наслідки цих двох тенденцій шляхом співпраці з дилерськими центрами, які здають в оренду та ремонт автомобілів, а також за допомогою програмного

забезпечення для обслуговування автомобілів для просування та управління відносинами з клієнтами, повідомлення цінності своєї роботи споживачам може допомогти зміцнити позицію автосервісів серед мінливого та конкурентоспроможного ринку [1].

Необхідність розроблення програмного забезпечення також продиктована тією ситуацією, що часто результати технічного обслуговування залишаються недоступними клієнтам і мають відновлюватися, якщо це можливо, за пам’яттю, іноді зберігаються в паперовому вигляді і при цьому недоступні в інших відділеннях сервісів. Тому важливим завданням є створення програмного забезпечення підтримки контролю за технічним обслуговуванням автомобілів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Проблема розроблення програмного забезпечення підтримки контролю за технічним обслуговуванням автомобілів

Зі зростом нових автомобільних технологій, збільшенням глобальних продажів автомобілів та більшим використанням спільних автомобілів завдяки послугам електронного зв’язку, потреба у відповідному програмному забезпеченні підтримки контролю за технічним обслуговуванням автомобілів звичайно що тільки актуалізується ще більше, для даного програмного забезпечення важливими є наступні риси:

– програмне забезпечення для обслуговування автомобілів повинно допомогти ефективно спілкуватися: незважаючи на те, що технічне обслуговування автомобілів – переважно технічна послуга, як послуга, яку надає сервіс, це також продовження бренду чи іміджу автосервісу, використання заплутаного програмного забезпечення для технічного обслуговування автомобілів може не тільки спричинити отримання у незручному вигляді даних про виконані послуги, але також може призвести до дорогих помилок між власниками та автотехніками;

– програмне забезпечення для технічного обслуговування автомобілів має допомогти керувати бізнесом: окрім перевірки та технічного обслуговування автомобілів, програмне забезпечення для технічного обслуговування автомобілів має включати всі можливі операції в єдину потужну систему, це повинно не тільки допомогти зробити транспортні засоби більш ефективними, але це також повинно допомогти команді бути більш ефективними: таке програмне забезпечення для технічного обслуговування автомобілів має допомогти покращити бізнес в цілому, а не лише з одного боку чи послуги;

– програмне забезпечення для обслуговування автомобілів має допомогти збирати та використовувати дані: завдяки орієнтованій на дані технології автосервісні майстерні повинні активізувати свою діяльність та

відійти від традиційних систем на паперовій основі, однак важливо пам’ятати, що дані, які використовуються неналежним чином, не мають значення, і програмне забезпечення для обслуговування автомобіля має забезпечити безперебійний, простий та автоматизований збір, візуалізацію та доступ до даних [1]-[2].

Нездійснення ефективної системи технічного обслуговування автомобілів або авторемонту може призвести до зменшення обсягу бізнесу та збільшення кількості скарг від клієнтів, щоб уникнути цього, існують основні правила щодо технічного обслуговування автомобілів як для автомобільних техніків, так і для керівників автосервісних майстерень:

– виконувати перевірки послідовно: автомобільні техніки, як правило, можуть визначити, чи можуть у машини виникнути проблеми через регулярне технічне обслуговування автомобіля, можна запланувати загальну перевірку технічного обслуговування автомобіля, а також окремі перевірки для кожної функції автомобіля, щоб переконатися, що потенційні проблеми виявляються до того, як вони стануть реальними проблемами;

– програмне забезпечення для технічного обслуговування автомобілів дозволяє керівникам автосервісних майстерень призначати завдання відібраним автотехнікам, заплановані перевірки технічного обслуговування автомобілів мають час початку і завершення, а керівники автосервісу автоматично отримують повідомлення про те, що запланована перевірка технічного обслуговування автомобіля завершена або пропущена, періодичні перевірки технічного обслуговування автомобіля можна запланувати для загальних та конкретних оцінок, щоб гарантувати, що нічого не пропущено, окрім планування технічного обслуговування автомобілів, керівники автосервісу можуть відслідковувати кількість невдалих дій та стан коригувальних дій, керівники автосервісу також можуть встановлювати нагадування, щоб гарантувати, що завдання з технічного обслуговування автомобіля виконуються та виконуються вчасно;

– до сервісного обслуговування автомобіля потрібно ставитись серйозно, оскільки це має вирішальне значення для безпеки власників автомобілів та репутації автосервісу: менеджери повинні переконатися, що перевірки технічного обслуговування автомобіля виконані правильно, це означає відсутність пропущених пунктів, відсутність поспішних оцінок і нечітких спостережень, включаючи фотографії, анотації та примітки, автомобільні техніки отримують всебічне уявлення про стан автомобіля;

– підходи до технічного обслуговування автомобілів можуть відрізнятися залежно від автомобіля чи автосервісу: який би спосіб не був найкращим, програмне забезпечення для технічного обслуговування автомобілів дозволяє налаштувати процес відповідно до специфікацій виробника автомобілів або так, як вважає за потрібне автотехнік, за допомогою програмного забезпечення для обслуговування автомобілів керівники автосервісних майстерень можуть встановити конкретні типи реагування для автомобільних техніків, використовуючи програмне забезпечення для обслуговування автомобілів, керівники автосервісних майстерень отримують необхідну інформацію, і автомобільні техніки витрачають менше часу на те, щоб зрозуміти, як провести перевірку [1], [3].

Виділені особливості вказують на актуальність та важливість проблеми розроблення програмного забезпечення контролю за технічним обслуговуванням автомобілів.

1.2 Огляд програмних аналогів

У даному підрозділі у додаток до виконаного аналізу проблеми буде виконано аналіз існуючих програмних аналогів.

AutoFluent [4] (рис. 1.1) – це потужне програмне забезпечення для всіх потреб автомобільного та шинного магазинів, забезпечує просту у використанні інвентаризацію та управління бухгалтерським обліком, що

спрощує роботу бухгалтерів та торгових представників, надаючи бізнесу професійний імідж [1].

Рисунок 1.1 – Програма AutoFluent [5]

Програма AutoFluent має цілий набір функціональних можливостей, які охоплюють різні підгалузі:

– завантаження даних про транспортний засіб з номером vehicle identification number (VIN);

– перегляд історії обслуговування автомобіля та отримання повідомлень про відкриті дії;

– сканування за допомогою інвентарного штрих-коду та VIN;

– створення та відстежування рекомендованих послуг для кожного автомобіля, надсилаючи нагадування електронною поштою;

– відстеження прибутку у межах робочого замовлення;

– пошук моделі автомобіля та розміру шин;

– замовлення шин;

– відстеження зовнішніх покупок, автоматично створюючи рахунки-фактури;

– імпортування прайс-листів на продукцію постачальника для зручного пошуку, розміщення замовлення;

– перегляд та керування запасами в різних місцях;

– відправлень замовлень на придбання безпосередньо постачальникам;

– поповнення запасів генерацією замовлень на замовлення та автоматичними замовленнями для деяких постачальників;

– продаж товарів;

– автоматизація маркетингу електронної пошти за допомогою індивідуальних моделей та вдосконалених фільтрів;

– пошук клієнтів за іменем, телефоном, адресою, номером транспортного засобу або рахунком-фактурою;

– перегляд історії клієнтів та автомобілів;

– відображення попереджень для клієнтів із закінченим терміном строків або з затримкою та можуть бути призначені клієнтам для відображення при відкритті транзакцій;

– доступ клієнтам до покупок в інтернет-магазині;

– керування графіками роботи співробітників та відстеження графіків, святкових днів та надурочних робіт;

– звіти для моніторингу ефективності та продуктивності роботи механіків та торгових представників;

– звітність та моніторинг консолідованих продажів: можна порівнювати прибуток, прогрес у досягненні цілей продажів, розділених на настроювані категорії;

– керування запасами;

– відображення історії рахунків клієнта [6].

AutoFluent не є вільним програмним забезпеченням і потребує сплати 95 американських доларів на місяць за версію з базовою функціональністю.

Недоліками програми є те, що кількість інструментів є величезною, відповідно в багатьох засобах це не зовсім програмна система контролю за технічним обслуговуванням автомобілів, підтримка роботи не акцентована саме на сервісах – багато засобів стосуються зокрема продажу. Також необхідно зазначити, що програма має досить утилітарний інтерфейс, який не відповідає сучасним вимогам.

AutoRepair Cloud [7] (рис. 1.2) надає можливість виконання таких груп операцій:

– оцінка робочого часу: міститься каталог для кожного транспортного засобу, база даних про роботи для понад 10 000 транспортних засобів, близько 500 робіт на кожен транспортний засіб;

– управління автозапчастинами: для цього організований швидкий пошук понад 8 мільйонів деталей, потрібно ввести назву необхідної деталі під час ремонту та отримати список пропозицій від провідних постачальників деталей, пропозиції містять номери деталей виробника, а запчастини відповідають поточному транспортному засобу;

– діагностика: програма дозволяє працювати з кодами несправностей [7].

Мінімальна версія AutoRepair Cloud передбачає сплату 35 американських доларів на місяць з одного користувача.

Проведений огляд дозволив констатувати, що програмного забезпечення, що повністю повторює функціональність системи, що має розроблятися, у відкритому доступі немає. В основному в даній галузі використовуються системи підтримки продажів або саме технічного обслуговування (з відслідковуванням кодів помилок), які при цьому мають як додаткову функціональність функціональні можливості, що можна віднести до контролю за технічним обслуговуванням, тобто фіксації виконаних робіт.

Ідея створення єдиної картки потребує якісної програмної реалізації на даний момент.

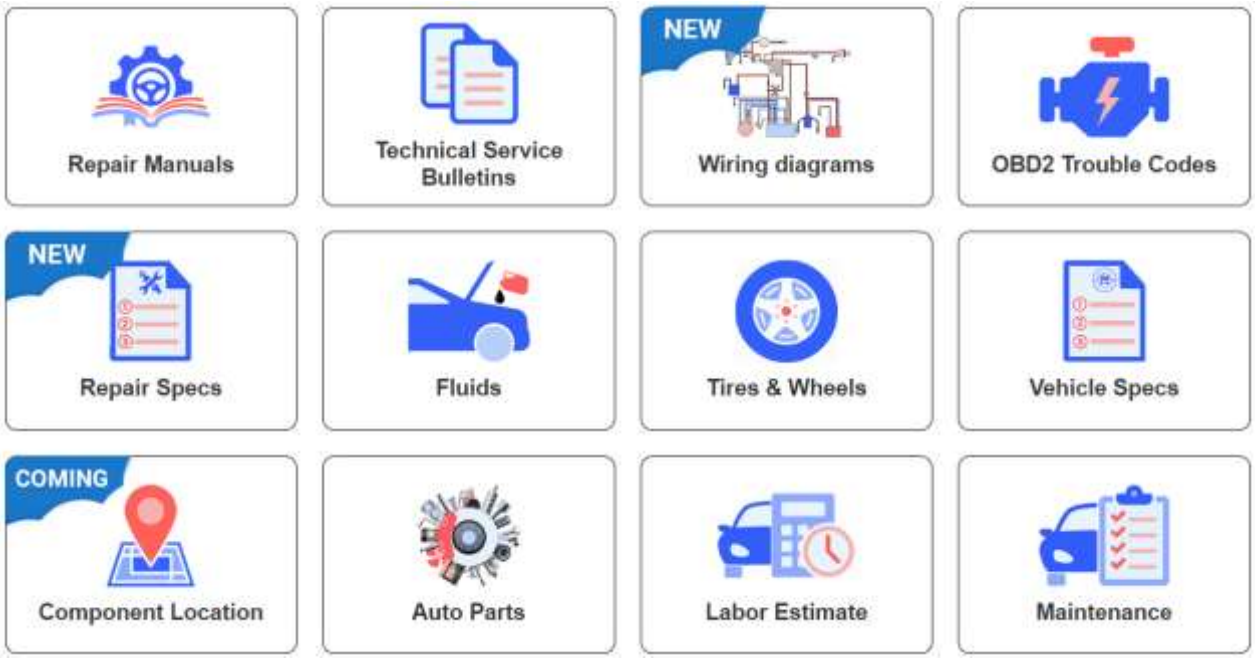

Рисунок 1.2 – Програма AutoRepair Cloud [8]

1.3 Вимоги до програми

Загальними вимогами до програми в результаті аналізу проблеми було визначено наступні:

– замовлення деталей у постачальників, тобто не передбачається направленість системи на продаж деталей, хоча для власних клієнтів відділення і може за потреби замовити якусь деталь, проте це не основна специфіка діяльності, найчастіше замовлення деталей будуть виконуватися в межах забезпечення виконання робіт з технічного обслуговування;

– підтримка внесення робіт з обслуговування різними відділеннями, які можуть знаходитися в різних містах;

– створення єдиної картки обслуговування автомобіля, коли незалежно від того, де саме було виконано обслуговування, власник

автомобіля та працівники інших відділень можуть отримати доступ до цієї інформації;

– підтримка розкладу обслуговувань, тобто система має забезпечувати створення розкладу робіт;

– підтримка роботи з пристроїв різного типу в мережі, адже програма має використовуватися різними відділеннями з обслуговування, відповідно не можна бути впевненим в тому, які саме пристрої будуть використані в конкретному місці, до того ж арсенал технічних засобів технічних працівників може значно відрізнятися від засобів менеджерів.

Зважаючи на загальні розглянуті особливості такого програмного забезпечення було вирішено використовувати наступні принципи при організації функціональних можливостей:

– технічні працівники мають отримувати можливість відслідковування процедур виконання тих чи інших робіт зі специфікою щодо конкретний моделей та версій моделей автомобілів;

– має надаватися інструментар відслідковування роботи технічних працівників, щоб зрозуміти ефективність їх роботи хоча би за одним показником;

– необхідно забезпечити роботу з розкладом обслуговування для збільшення зручності роботи сервісу;

– внесення деталей виконаних робіт може виконуватися і технічним працівником, і менеджером.

1.4 Висновки за розділом 1

Проаналізовано особливості і вимоги до програмного забезпечення контролю за технічним обслуговуванням автомобілів. Проаналізовані програмні аналоги мають в основному функціональність, пов’язану з роботою над продажами, а вже в додаткову чергу – визначення обслуговування автомобілів. Саме розроблення програмного забезпечення,

яке буде направлено на зручність роботи над технічним обслуговуванням, є особливістю роботи.

Завдання дипломної кваліфікаційної роботи:

– визначення вимог до програмного забезпечення, що розробляється;

– проєктування програмного забезпечення;

– реалізація програмного забезпечення;

– тестування і налагодження програмного забезпечення.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір засобів розробки

У процесі аналізу і вибору мов програмування для реалізації проєкту було враховано вимоги до програми, що передбачають віддалений доступ з пристроїв різного типу, що зручно задовольнити у вигляді вебзастосунку. Для розроблення вебзастосунку можна використати мову програмування Java [9], адже за багатьма статистичними даними саме ця мова є найпопулярнішою мовою програмування на поточний момент, до того ж вона може використовуватися саме для веброзробки. Ще одним варіантом стала мова програмування Python [10].

У процесі порівняння цих мов було виділено наступні важливі особливості:

– мова Python дозволяє більш продуктивно працювати над проєктом;

– якщо важливим є максимально швидке отримання результатів, то саме мова Python дозволяє створювати такі рішення;

– програми мовою Python дещо повільніші за програми мовою Java, але жодна з цих мов не призводить до створення ідеальних у цьому компонентів рішень.

Зважаючи на те, що Python [11] не значно програє Java за швидкістю виконання коду, до того ж не передбачається, що кількість звернень до програми буде надто критичною, адже в основному з нею будуть працювати представники сервісів з технічного обслуговування, а клієнти будуть мати обмежену функціональність, але при цьому Python дозволяє створити максимально швидко готові проєкти, зокрема за рахунок динамічної типізації, то для програмної реалізації було віддано перевагу мові Python.

Результати порівняння мов програмування для програмної реалізації представлено в табл. 2.1.

Таблиця 2.1 – Порівняння мов програмування для програмної реалізації

Параметр	Java	Python
Продуктивність розробників	Середня	Висока
Ліцензія	Відкрита	Відкрита
Тип мови	Компільована	Інтерпретована
Спільнота розробників	Велика	Велика
Швидкість виконання коду	Вища за середню	Середня
Вебфреймворки	Spring	Flask, Django
Підтримка вебзастосунків з великим трафіком	Так	Так

Розробку мовою Python буде виконано за допомогою вебфреймворку Django, оскільки він надає можливості об’єктно-реляційного відображення, що додатково спрощує і пришвидшує процес розробки програми, спрощує процес перенесення бази даних (БД). Вебфреймворк дозволяє створювати вебзастосунки на основі шаблону Model View Controller (MVC) з 3 основних структурних елементів:

– модель – це елемент вебзастосунку, який виступає посередником між інтерфейсом вебсайту та БД, у технічному плані це об’єкт, який реалізує логіку для домену даних програми, бувають випадки, коли програма може брати дані лише в певному наборі даних і безпосередньо надсилати їх до уявлення (компонент інтерфейсу користувача), не потребуючи жодної БД, тоді набір даних розглядається як модель, хоча у випадку застосунку, що розробляється, саме БД є основним елементом організації єдиної картки, що

є вимогою до програми, загалом модель – це елемент, який містить бізнес-логіку в архітектурі Django;

– уявлення: цей елемент містить логіку інтерфейсу в архітектурі Django, уявлення насправді є інтерфейсом користувача вебпрограми та містить такі частини, як HTML, CSS та інші технології інтерфейсу, як правило, цей інтерфейс створюється з компонента Models, тобто вміст надходить з компонента Models;

– контролер є основним компонентом управління – це означає, що контролер обробляє взаємодію користувача та вибирає уявлення відповідно до моделі, основне завдання контролера – вибрати компонент перегляду відповідно до взаємодії користувача, а також застосувати компонент моделі [12].

Схему взаємодії між елементами в структурі MVC за використання шаблону проєктування Django представлено на рис. 2.1.

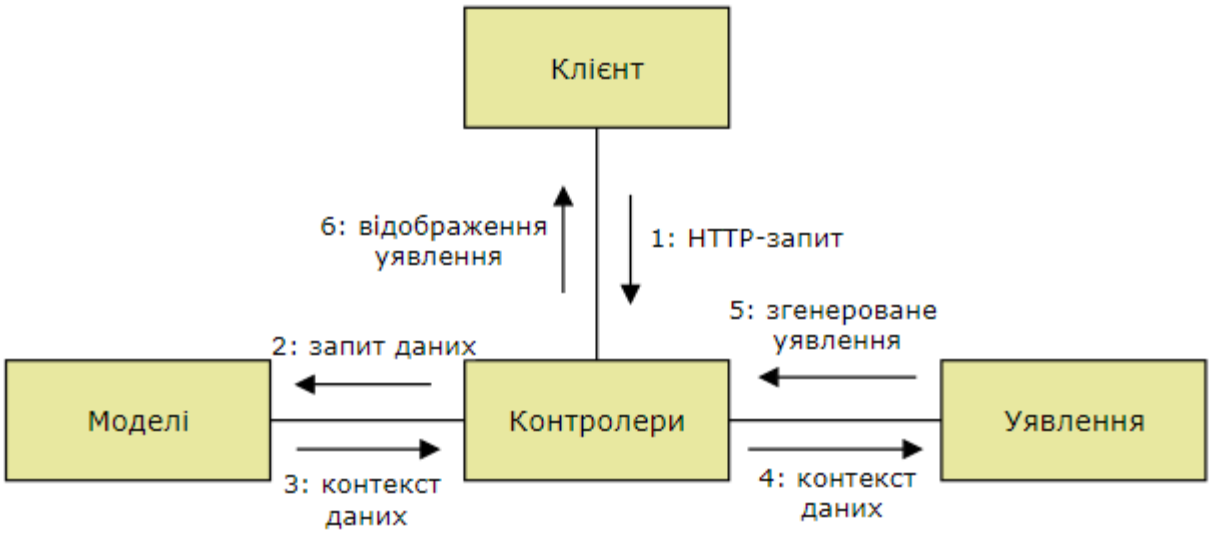

Рисунок 2.1 – Послідовність комунікації всередині MVC

2.2 Функціональні вимоги до програмного забезпечення

У даному підрозділі було визначено функціональні вимоги до програми і виконано моделювання сценаріїв використання системи за

відповідними функціями.

З програмою можуть працювати клієнти (власники автомобілів), менеджери і технічні працівники сервісу технічного обслуговування та адміністратори.

Для клієнта мають бути реалізовані наступні функціональні можливості (рис. 2.2):

– перегляд робіт, виконаних за автомобілем;

– записування на обслуговування;

– перегляд розкладу власних обслуговувань;

– авторизація.

Рисунок 2.2 – Діаграма прецедентів для клієнта

Для технічного працівника мають бути реалізовані наступні функціональні можливості (рис. 2.3):

– перегляд робіт, які мають бути виконані за замовленням;

– перегляд процедури виконання роботи;

– визначення початку виконання роботи з обслуговування;

– визначення завершення виконання роботи з обслуговування;

– перегляд наявності деталей;

– відкриття замовлення на постачання деталей;

– авторизація.

Рисунок 2.3 – Діаграма прецедентів для технічного працівника

Для технічного працівника під час виконання керування роботою з обслуговування стандартний потік включає визначення початку виконання роботи та визначення завершення виконання роботи, а альтернативний потік передбачає перегляд процедури виконання роботи, що знадобиться в тому випадку, якщо працівник не пам’ятає деталей виконання роботи за певною версією моделі автомобіля.

Для адміністратора мають бути реалізовані наступні функціональні можливості (рис. 2.4):

– керування даними працівників;

– керування даними виробників, моделей і версій моделей автомобілей;

– керування (в тому числі редагування) замовленнями;

– керування каталогом деталей;

– керування даними постачальників;

– керування даними відділень;

– керування каталогом робіт з обслуговування;

– авторизація.

Рисунок 2.4 – Діаграма прецедентів для адміністратора

Для менеджера мають бути реалізовані наступні функціональні можливості (рис. 2.5):

– створення замовлення на обслуговування;

– підтвердження виконання замовлення на обслуговування;

– внесення виконаних робіт за замовленням на обслуговування;

– внесення використаних деталей для обслуговування;

– замовлення деталей у постачальника;

– внесення прибуття деталей від постачальника;

– перегляд виконаних робіт за автомобілем;

– пошук замовлень за номером телефону власника;

– пошук придбаних деталей власником;

– визначення робіт, які найчастіше виконувались за автомобілем;

– перегляд замовлень, що знаходяться в роботі;

– пошук замовлення за номером;

– перегляд затримки виконання робіт за робітниками;

– внесення даних клієнтів;

– редагування даних клієнта.

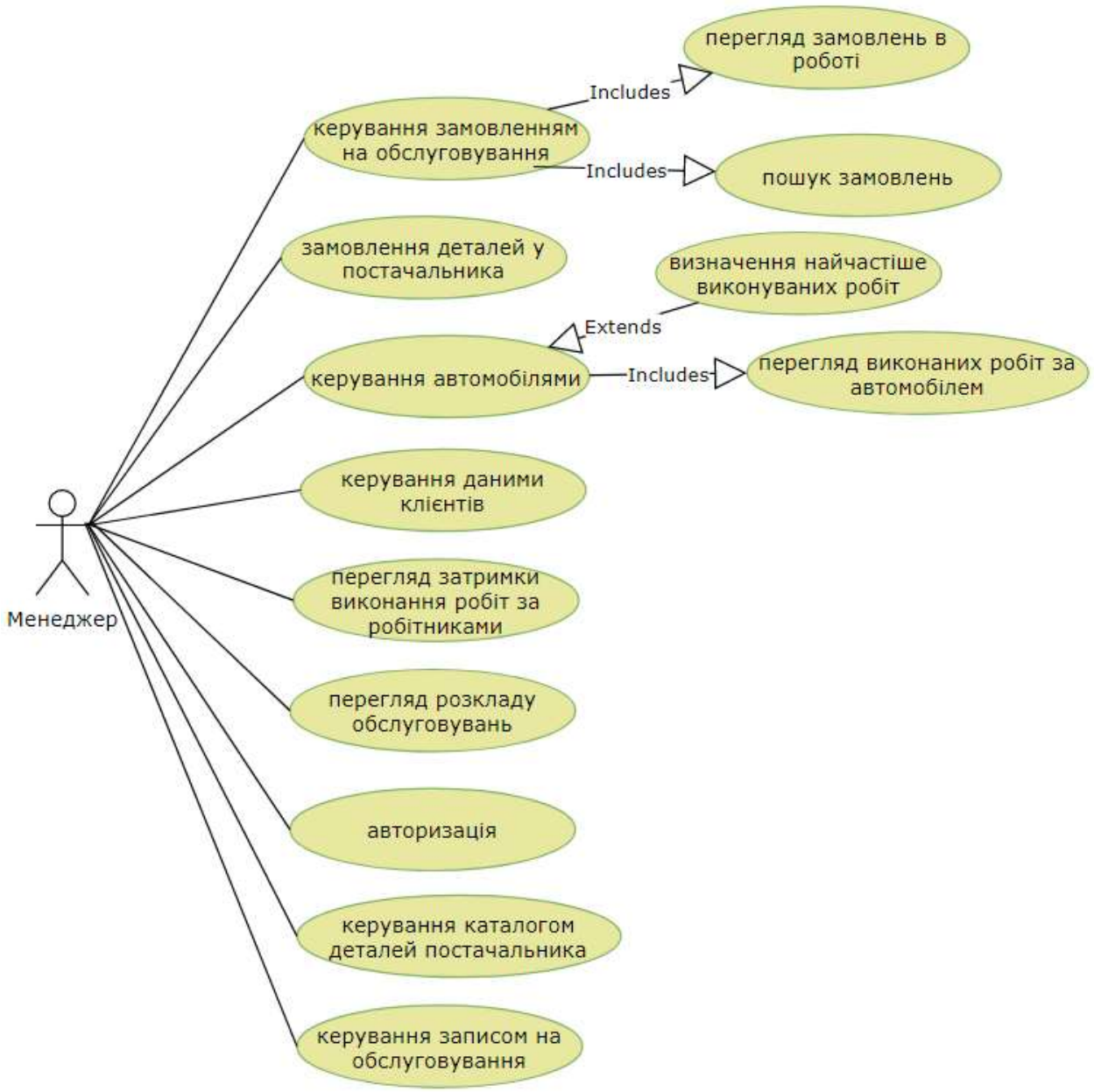

Рисунок 2.5 – Діаграма прецедентів для менеджера

Окрім того менеджер повинен мати можливість виконувати:

– внесення даних автомобіля;

– редагування даних автомобіля;

– створення і редагування каталогу деталей постачальника;

– перегляд розкладу обслуговувань;

– підтвердження запису на обслуговування;

– скасування запису на обслуговування;

– авторизація власного облікового запису.

2.3 Проєктування структури даних

Для підтримки роботи БД було використано систему управління БД MySQL [13]. На основі її використання було запроєктовано структуру БД.

Таблиця Owner призначена для збереження інформації про власників автомобілів, обслуговування яких виконується на сервісі, для цього містить такі поля:

– id – ідентифікаційний номер клієнта;

– lastname – прізвище клієнта;

– firstname – ім’я клієнта;

– thirdname – по батькові клієнта;

– phone – номер телефону клієнта;

– license – водійське посвідчення клієнта;

– userid – обліковий запис користувача.

Таблиця Staff призначена для збереження інформації про працівників сервісу, для цього містить такі поля:

– id – ідентифікаційний номер працівника;

– lastname – прізвище працівника;

– firstname – ім’я працівника;

– thirdname – по батькові працівника;

– phone – номер телефону працівника;

– position – посада працівника;

– started – час початку роботи працівника;

– userid – обліковий запис користувача;

– departmentID – відділення, в якому працює працівник.

Таблиця Department призначена для збереження інформації про відділення сервісу, для цього містить такі поля:

– id – ідентифікаційний номер відділення;

– title – назва відділення;

– cityID – місто, де розташоване відділення;

– address – адреса відділення;

– phone – номер телефону відділення;

– days – кількість робочих днів на тиждень;

– slots – кількість слотів (автомобілів, що одночасно можуть опрацьовуватись).

Таблиця City призначена для збереження інформації про міста, для цього містить такі поля:

– id – ідентифікаційний номер міста;

– title – назва міста.

Таблиця CarBrand призначена для збереження інформації про виробників автомобілів, для цього містить такі поля:

– id – ідентифікаційний номер виробника автомобілів;

– title – назва виробника;

– phone – номер телефону для вирішення запитань;

– address – адреса;

– site – вебсайт.

Таблиця CarModel призначена для збереження інформації про моделі автомобілів, для цього містить такі поля:

– id – ідентифікаційний номер моделі автомобіля;

– title – назва моделі;

– carbrandID – виробник автомобіля.

Таблиця CarModelVersion призначена для збереження інформації про версії моделей автомобілів, для цього містить такі поля:

– id – ідентифікаційний номер версії моделі автомобіля;

– carmodelID – модель автомобіля;

– year1 – рік початку випуску;

– year2 – рік завершення випуску;

– documentation – документація щодо даного випуску автомобіля;

– features – особливості версії.

Таблиця Car призначена для збереження інформації про автомобілі, для цього містить такі поля:

– id – ідентифікаційний номер автомобіля;

– vin – VIN-номер автомобіля;

– made – дата виробництва автомобіля;

– specification – специфікація автомобіля (може включати дані двигуна, додаткові особливості);

– ownerID – поточний власник автомобіля;

– carmodelversionID – версія моделі автомобіля.

Таблиця ServiceGroup призначена для збереження інформації про категорії обслуговування, для цього містить такі поля:

– id – ідентифікаційний номер типу обслуговування;

– title – назва типу обслуговування.

Таблиця Service призначена для збереження інформації про доступні роботи з обслуговування, для цього містить такі поля:

– id – ідентифікаційний номер роботи з обслуговування;

– title – назва роботи;

– servicegroupID – тип обслуговування;

– procedure – опис процедури виконання робіт.

Таблиця ServicePrice призначена для збереження інформації про вартість конкретних процедур з обслуговування автомобілів (прив’язані до специфіки моделей автомобілів за версіями), для цього містить такі поля:

– id – ідентифікаційний номер запису;

– serviceID – обслуговування;

– carmodelversionID – версія моделі автомобіля;

– price – вартість обслуговування;

– duration – тривалість виконання роботи (в хвилинах);

– specification – особливості виконання роботи;

– procedure – процедура реалізації роботи;

– departmentID – відділення, де встановлено такі ціни.

Таблиця CarService призначена для збереження інформації про конкретні (окремі) виконані роботи з обслуговування конкретного автомобіля, для цього містить такі поля:

– id – ідентифікаційний номер обслуговування автомобіля;

– serviceorderID – замовлення на обслуговування, в межах якого було виконано роботу;

– staffID – працівник, який виконував роботу;

– serviceID – виконана робота;

– price – вартість;

– specification – особливості виконання роботи;

– date1 – дата початку виконання роботи;

– date2 – дата завершення виконання роботи.

Таблиця ServiceOrder призначена для збереження інформації про замовлення з обслуговування автомобіля (об’єднує різні роботи), для цього містить такі поля:

– id – ідентифікаційний номер;

– staffID – працівник, який створював замовлення;

– carID – автомобіль, обслуговування якого було реалізовано;

– ownerID – клієнт (власник автомобіля, який замовляв роботи): фіксація необхідна окрім автомобіля, адже з часом обслуговування у автомобіля може змінюватися власник, в записі про автомобіль фіксується

саме поточний власник, за даним полем за необхідності можна буде відслідкувати власників автомобіля;

– date1 – початок обслуговування;

– date2 – завершення обслуговування;

– price – сумарна вартість замовлення;

– created – дата створення замовлення.

Таблиця Order призначена для збереження інформації про замовлення деталей, для цього містить такі поля:

– id – ідентифікаційний номер деталі в замовленні;

– cardID – автомобіль, для якого замовлено деталі;

– supplierID – постачальник, у якого оформлено замовлення;

– serviceorderID – обслуговування, в межах якого було виконано замовлення;

– made – дата створення замовлення;

– performed – дата виконання замовлення;

– price – сумарна вартість замовлення.

Таблиця Supplier призначена для збереження інформації про постачальників, для цього містить такі поля:

– id – ідентифікаційний номер постачальника;

– title – назва постачальника;

– phone – номер телефону постачальника;

– address – адреса постачальника;

– site – вебсайт.

Таблиця ComponentOrder призначена для збереження інформації про замовлення деталей, для цього містить такі поля:

– id – ідентифікаційний номер деталі;

– componentID – деталь, яку було замовлено;

– orderID – замовлення;

– price – вартість деталі.

Фактично реалізує зв’язок багато до багатьох між таблицями Component і Order.

Таблиця ComponentGroup призначена для збереження інформації про категорії деталей, для цього містить такі поля:

– id – ідентифікаційний номер категорії деталей;

– title – назва категорії деталей.

Таблиця Component призначена для збереження інформації про деталі, для цього містить такі поля:

– id – ідентифікаційний номер деталі;

– title – назва деталі;

– componentgroupID – категорія деталі;

– carmodelversionID – версія моделі автомобіля;

– supplierID – поточний постачальник деталі, з яким працює сервіс;

– price – поточна вартість деталі;

– quantity –наявна в даний момент кількість деталей;

– departmentID – відділення, якому відповідають ці дані.

Таблиця Timetable призначена для збереження інформації про розклад робіт з обслуговування автомобілів, для цього містить такі поля:

– id – ідентифікаційний номер обслуговування;

– departmentID – відділення;

– carID – автомобіль, обслуговування якого буде здійснюватися;

– slot – слот на обслуговування (для можливості одночасного обслуговування декількох автомобілів);

– description – опис особливостей;

– date1 – початок обслуговування;

– date2 – завершення обслуговування;

– approved – підтвердження запиту менеджером.

Схема БД представлена на рис. 2.6.

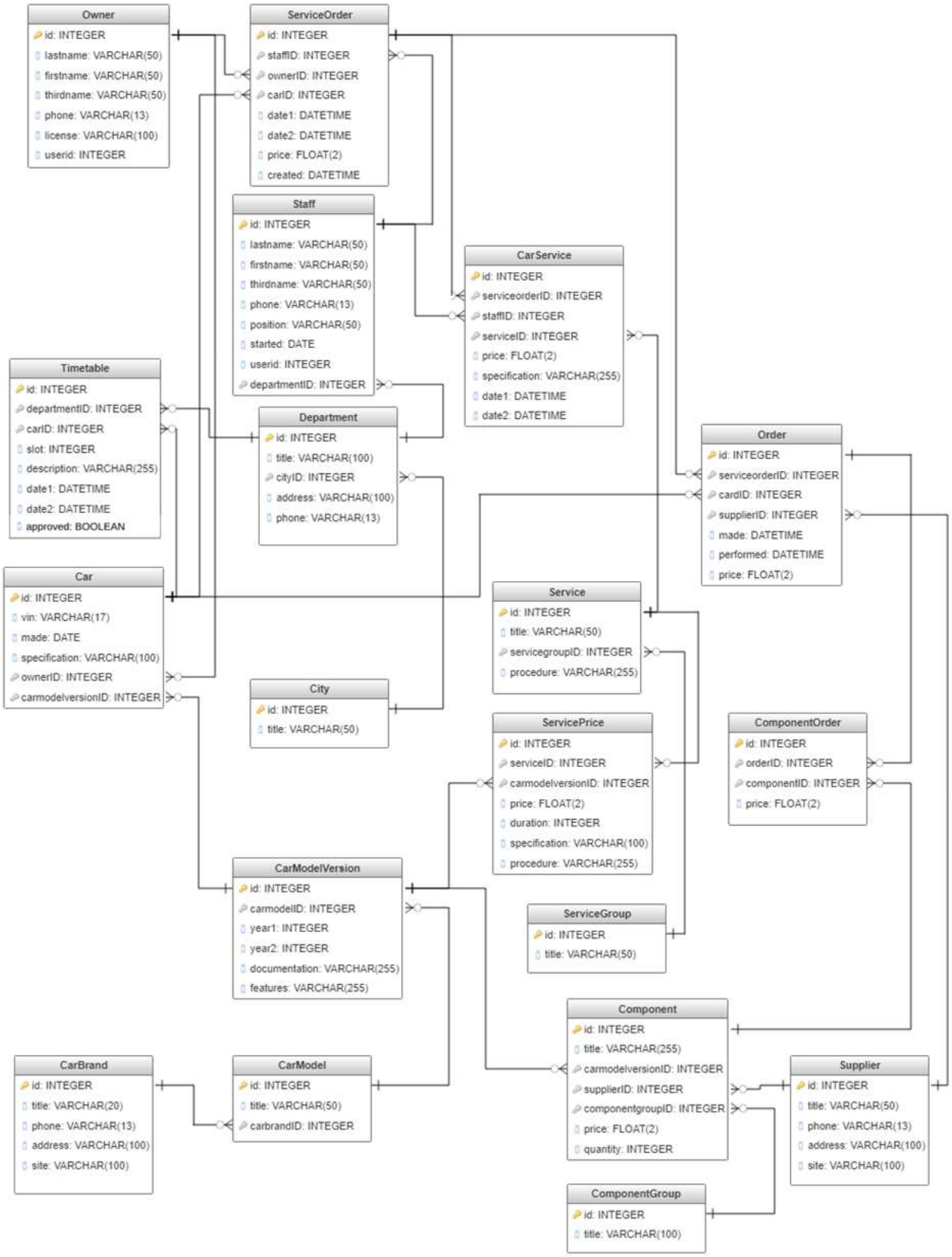

Рисунок 2.6 – Структурна схема БД

І таблиця ServiceOrder, і таблиця Order містять carID тому, що не кожне замовлення Order обов’язково містить ServiceOrder, тобто сервіс може

за запитом клієнта оформити постачання певної деталі, що не потребує ніяких дій, тоді в замовленні потрібно внести інформацію, до якого саме автомобіля це замовлення стосується.

2.4 Висновки за розділом 2

Виконано вибір мови програмування Python та вебфреймворку Django для розробки вебзастосунку, проаналізовано архітектурний шаблон, який лежить в основі Django, визначено функціональні вимоги до програми, виконано моделювання сценаріїв роботи клієнтів, технічних працівників, менеджерів та адміністраторів з програмою, виконано проєктування БД, в результаті чого створено схему БД.

3 РОЗРОБКА ПРОГРАМИ

3.1 Структура розроблених класів

У відповідності з логікою організації проєктів Django та шаблону MVC було розроблено множину класів, що охоплюють форми, моделі та уявлення проєкту. У свою чергу шаблони, менеджер адрес розробляються у вигляді скриптів, а не класів.

Усі розроблені класи було об’єднано у пакети. Всього до складу проєкту Django входить 3 основні пакети:

– пакет models, що включає класи моделей вебзастосунку;

– пакет forms, що включає класи форм вебзастосунку, які побудовані на основі пакету models;

– пакет views, що включає уявлення проєкту.

До складу пакету views входять наступні підпакети:

– підпакет serviceorders включає набір класів, що пов’язані з уявленнями замовлень і надають можливість створення, редагування, перегляду наборів замовлень та деталей замовлень;

– підпакет catalogue включає набір класів, що пов’язані з уявленнями створення, редагування, перегляду наборів виробників, моделей автомобілів, версій цих моделей, деталей, постачальників, відділень, міст, каталогу робіт і їх категорій і працівників;

– підпакет cars включає набір класів, що пов’язані з автомобілями власників;

– підпакет timetable включає класи уявлень перегляду, формування розкладу і редагування записів;

– підпакет orders включає класи уявлень створення, редагування, перегляду деталей та наборів замовлень деталей.

На рис. 3.1 представлена діаграма пакетів розробленого програмного проєкту.

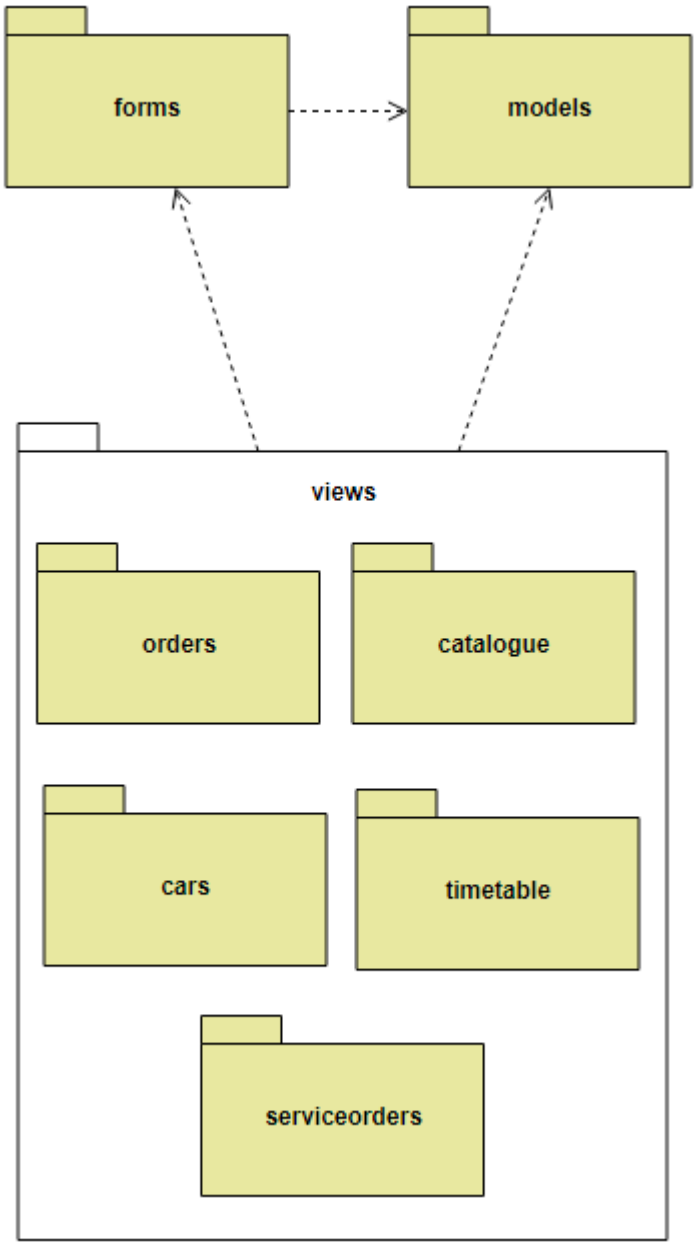

Рисунок 3.1 – Діаграма пакетів

3.2 Опис реалізації програми

Усі розроблені класи уявлень передбачають початкову авторизацію, адже тільки домашня сторінка може відображатися неавторизованому користувачу, а також порожні версії сторінок розкладу, замовлень та картки. Тому усі класи уявлень були створені шляхом наслідування класу LoginRequiredMixin.

Детальніше розглянемо спосіб створення класів уявлень на основі підпакету класів serviceorders пакету views, адже замовлення обслуговування є однією з основних сутностей предметної області.

Клас AdminServiceOrdersView створює уявлення, яке виводить на сторінці відповідного шаблону adminsorders.html перелік усіх замовлень з можливостями пошуку замовлень. Клас створений наслідуванням класу ListView, який створює сторінку перегляду набору об’єктів. Для визначення параметрів його роботи визначено наступні поля класу:

– модель, з об’єктами якої працює клас, – ServiceOrder (поле model);

– шаблон сторінки, в яку буде направлено сформований перелік об’єктів, – adminsorders.html (поле template_name);

– кількість об’єктів замовлення, що буде відображатися на одній сторінці, – paginate_by (встановлено значення 10, тобто по 10 замовлень на сторінці).

Пошук замовлень виконується за допомогою встановлення чотирьох параметрів, які передаються через GET-запит і включають:

– номер замовлення self.request.GET.get('number');

– відділення, в якому виконується замовлення self.request.GET.get('department');

– інтервал початку виконання замовлення від self.request.GET.get('date1') до self.request.GET.get('date2').

Метод get_queryset(self) класу виконує підготовку набору об’єктів для відображення на сторінці, для цього спочатку реалізується формування переліку замовлень на обслуговування з усіх, що є в БД (у випадку, якщо відділення не встановлено в пошуку), або з замовлень, що внесені працівником сервісу з заданого відділення:

```
if self.department:
    orders = ServiceOrder.objects.filter(staff__department__pk = self.department)
```

```
else:
    orders = ServiceOrder.objects.all()
```

Для визначення замовлень, початок виконання яких потрапляє в заданий інтервал від d1 до d2, реалізується фільтрація наступним чином:

```
if self.date1:
    orders = orders.filter(date1__gte=d1)
if self.date2:
    orders = orders.filter(date1__lte=d2)
```

Клас ManagerServiceOrdersView виконує створення уявлення перегляду менеджером замовлень. Структура побудови даного класу аналогічна попередньому. Менеджер може переглядати тільки ті замовлення, які були створені працівниками відділення, в якому працює поточний авторизований користувач. Визначення такого відділення відбувається шляхом витягання даних про поточного користувача:

```
iduser = self.request.user.id
staff = Staff.objects.get(userid = iduser)
dep = staff.department
orders = ServiceOrder.objects.filter(department = dep)
```

В основі відображення замовлень для менеджера лежить також пошук. Якщо значення параметрів не встановлено, то відображаться всі замовлення. У менеджера на відміну від можливості вибору конкретного відділення відділення визначається автоматично, але при цьому надається в якості параметрів встановлювати номер телефону власника автомобіля.

Клас ManagerProcessServiceOrdersView виконує створення уявлення перегляду поточних замовлень обслуговування у відділенні для менеджера.

Замовлення не містить напряму значення стану, тому стан розрізняється за встановленими датами. Якщо початкову дату встановлено, а кінцеву ще ні (тобто вона визначена у розкладі обслуговування, але не зазначена в замовленні), то замовлення вважається таким, що виконується. Якщо кінцева дата також встановлена, то замовлення вважається завершеним.

Для пошуку замовлень, що виконуються, відбір реалізується шляхом визначення замовлень, дата і час початку яких менше за поточний час, тобто вони вже виконуються, але дата і час завершення виконання замовлення не встановлено:

```
orders = orders.filter(date1__lte=datetime.datetime.now())
orders = orders.filter(date2__isnull=True)
```

Як видно з виконаного опису, функціональні можливості і відповідні уявлення поділені на варіанти для адміністратора, технічного працівника, менеджера та власника. Ця інформація не зберігається у БД. Усі користувачі розділені на 2 групи, але в БД не зберігається, яка з 4 ролей використовується для цього користувача. Для реалізації аутентифікації користувачів використана базова система для Django. Тобто усі користувачі системи мають бути авторизовані за допомогою засобів Django. Django зберігає у своїй БД відповідну інформацію про користувача. У кожного такого користувача є характеристика Groups у стандартній БД. Усього є 4 групи користувачів: технічні працівники, менеджери, адміністратори, власники автомобілів. В залежності від цієї ролі реалізується запуск відповідного сценарію роботи з програмою, відповідно відображення шаблонів і уявлень. Усі шаблони поділені на 4 папки для відповідної ролі користувача і створені на основі вільного шаблону [14].

Клас OwnerServiceOrdersView виконує створення уявлення перегляду замовлень власника автомобіля. Для нього виділяються замовлення, у яких власником вказано авторизованого користувача.

Клас TechServiceOrdersView виконує створення уявлення перегляду технічним працівником замовлень. Для технічного працівника це робиться наступним чином: вони належать тому ж відділенню, де працює авторизований користувач, їх виконання розпочалося у минулому і не завершилося у поточний момент. Тобто саме такий обмежений перелік замовлень відображається технічному працівнику, щоб йому не потрібно було виконувати пошук замовлень, працювати з великим їх обсягом. До того ж під час сортування вони розташовуються за зменшенням ідентифікаційного номеру, це дозволяє розташовувати старі замовлення, які не було завершено, нижче у переліку, тобто працівник може за потреби до них звернутися, але першочергово він отримує поточні замовлення.

Клас TechServiceOrdersSearchView дозволяє виконати пошук замовлень за VIN-номером автомобіля. Цей засіб за потреби дозволяє знайти саме потрібне замовлення, якщо з переліку в попередньому варіанті знайти його не вдалося або було недостатньо зручно. Сортування також реалізується за зменшенням ідентифікаційного номеру, тобто новіші замовлення розташовуються раніше.

Клас CreateServiceOrderView дозволяє створити замовлення і першочергово направлений на роботу з ним менеджера. Окрім менеджера створити замовлення може ще адміністратор, адже йому доступні всі засоби. Але оскільки це основний засіб менеджера, то уявлення розташовано саме в цій папці. Цей клас реалізований шляхом наслідування класу CreateView, адже направлений на уявлення для створення нового об’єкту, яке у даному випадку э замовленням. Даний клас реалізовано на основі відповідного класу форми створення замовлення ServiceOrderCreateForm, що підтримує поля визначення автомобіля, власника, дати початку і завершення обслуговування.

Метод form_valid(self, form) реалізує валідацію форми з програмним внесенням дати і часу створення замовлення, працівника, який його створив:

```
mdata = form.save(commit=False)
```

```
mdata.created = datetime.datetime.now()
us = self.request.user
st = get_object_or_404(Staff, user=us)
mdata.staff = st
mdata.save()
```

Одночасно під час створення замовлення виконується внесення даних у розклад обслуговування за необхідності. Необхідність виникає у тому випадку, якщо попередній запис автомобіля на обслуговування не було виконано. Це потрібно для того, щоб у розкладі було видно, що в даний період відповідний слот обслуговування зайнятий.

Для цього спочатку виконується відбір всіх записів про обслуговування заданого автомобіля у відділенні, де працює авторизований користувач, у період часу, що співпадає або є ширшим за період, встановлений під час створення замовлення.

Якщо запис не було знайдено, то він вноситься до БД. Дата і час завершення обслуговування з форми створення замовлення використовуються саме для запису в розклад, дата завершення в самому замовленні зазначається тільки тоді, коли менеджер визначає завершення роботи над замовленням.

Для даного класу також реалізовано метод get_form_kwargs(self, *args, **kwargs). Окрема реалізація знадобилась для розширення контексту, щоб внести до форми створення замовлення інформації про всіх замовників і автомобілів, дані про які наявні в БД. Окрім того передаються дані про всіх власників з БД та для кожного власника встановлюється набір автомобілів, за якими цей власник оформлював замовлення хоча би раз в системі.

Розширення контексту реалізується шляхом додавання ключів cars, owners та carowners до словника контексту.

Менеджер також визначає завершення виконання замовлення. Цей процес реалізовано шляхом створення функції DoneServiceOrder(request, pk),

оскільки у даному випадку не потрібно відображати окрему сторінку для роботи. У цій функції реалізується оновлення замовлення шляхом внесення поточної дати і часу до дати і часу завершення роботи над замовленням, окрім того реалізується підрахунок вартості замовлення. Підрахунок вартості реалізується через обчислення суми робіт, виконаних за замовленням, та суми вартостей всіх деталей, внесених у замовлення.

Клас ServiceOrderDetailView виконує створення уявлення для перегляду деталізованих даних замовлення шляхом наслідування класу DetailView [15]. Окрім внесення стандартним чином через наслідування значень параметрів замовлення реалізується також розширення контексту через додавання робіт з обслуговування замовлення та деталей у замовленні з даним номером:

```
context['services'] = CarService.objects.filter(serviceorder=self.kwargs.get('pk'))
context['components'] = Order.objects.filter(serviceorder =self.kwargs.get('pk'))
```

Клас UpdateServiceOrderView реалізований через наслідування класу UpdateView і забезпечує створення уявлення редагування замовлення на обслуговування. Даний клас використовується тільки адміністратором. Менеджер забезпечує створення замовлення і визначення часу завершення роботи над ним, а адміністратор – повне редагування даних.

Під час витягання даних описані класи взаємодіють з класами моделей, з ними також взаємодіють класи форм.

Для кожного класу було визначено набір полів з визначенням відповідних типів цих полів, засоби перетворення класу до рядкової форми та автоматизоване визначення абсолютного шляху до відповідної сторінки деталізації об’єкту класу моделі.

3.3 Висновки за розділом 3

Описано розроблені класи, що дозволили впровадити архітектурний шаблон на основі використання вебфреймворку Django, в структуру застосунку. Структуру організації розроблених класів представлено на основі діаграми пакетів. Розроблено програмне забезпечення у вигляді вебзастосунку.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програма призначена для централізованого перегляду і керування даними технічного обслуговування автомобілів. Програма дозволяє фіксувати виконані роботи за автомобілем, параметри їх виконання, використані деталі для проведення ремонту або обслуговування, переглядати результати обслуговування як за замовленнями загалом, так і за виконаними роботами.

4.2 Умови виконання програми

Апаратні вимоги до організації сервера включають:

– процесор: 4 ядра, тактова частота – 2,9 ГГц;

– оперативна пам’ять – 10 Гб;

– платформа – 32-розрядна або 64-розрядна;

– жорсткий диск – 1 Гб вільного простору.

4.3 Робота з програмою

При виконанні запиту до вебсервера, на якому розташовано вебзастосунок, користувачу за замовчуванням відображається вебзастосунок в режимі роботи власника автомобіля. Початково відображається домашня сторінка власника автомобіля (рис. 4.1).

Розділи меню власника автомобіля включають:

– розділ «Домашня», що дозволяє відображати базову сторінку, з якої розпочинається робота з вебсайтом;

– розділ «Картка», що дозволяє переглядати інформацію щодо робіт з обслуговування власного автомобіля;

– розділ «Розклад», що дозволяє переглядати власні записи на технічне обслуговування та створити новий запис.

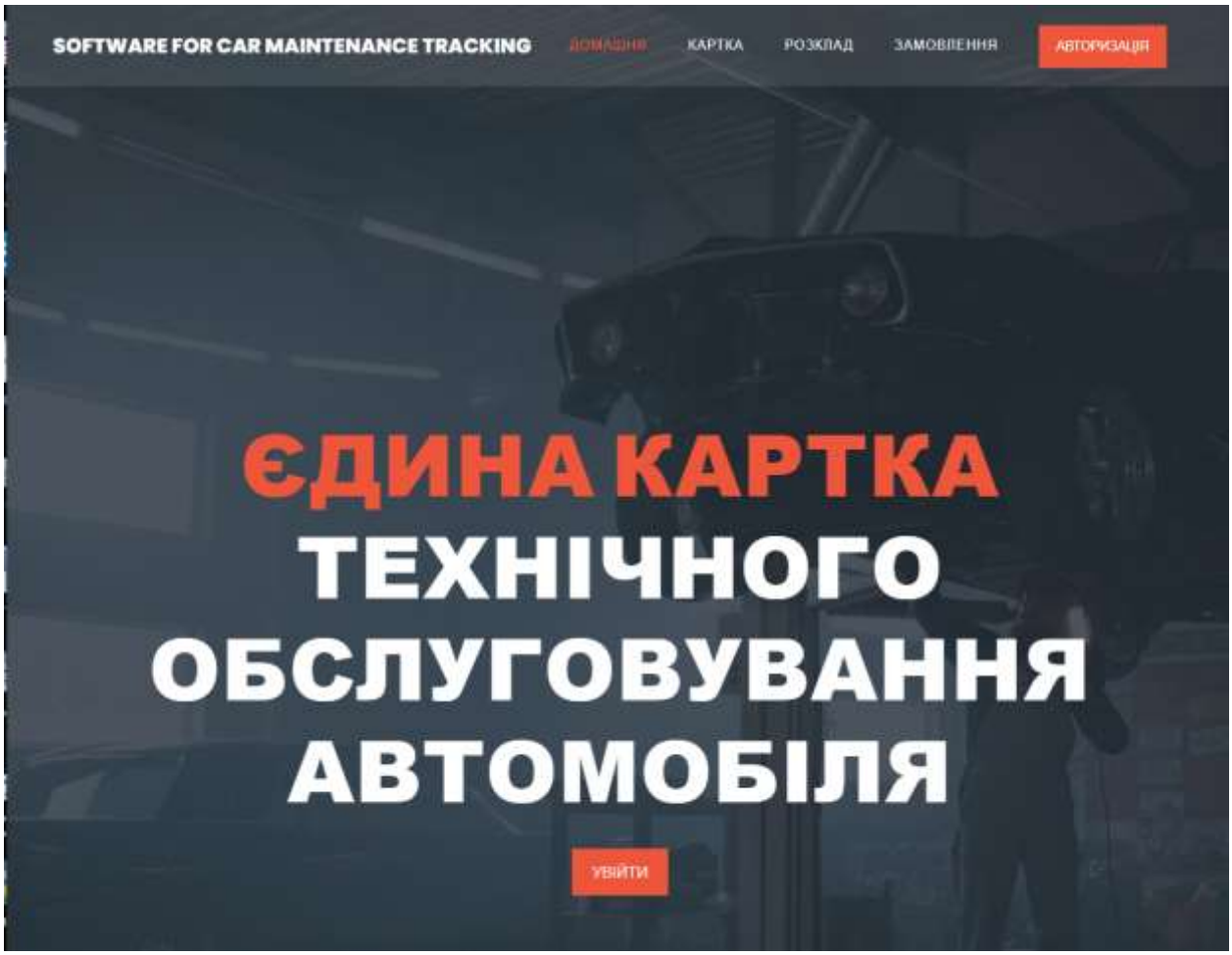

Рисунок 4.1 – Домашня сторінка для клієнта

У випадку, якщо користувач ще не авторизувався, йому буде відображено всі розділи власника, але при цьому при виконанні переходу на ці сторінки він отримає сторінку, де буде виведено повідомлення про неможливість отримання даних для нього і необхідність авторизуватися. На необхідність авторизуватися вказує і домашня сторінка, а в головному меню відображається кнопка «Авторизація».

Після виконання авторизації власник автомобіля зможе використовувати згадані вище розділи.

На сторінці «Картка» (рис. 4.2) реалізується принцип єдиної картки автомобіля, коли на ній можна побачити всі дані про ремонтні роботи автомобіля, проведені в усіх сервісах, що використовують вебзастосунок.

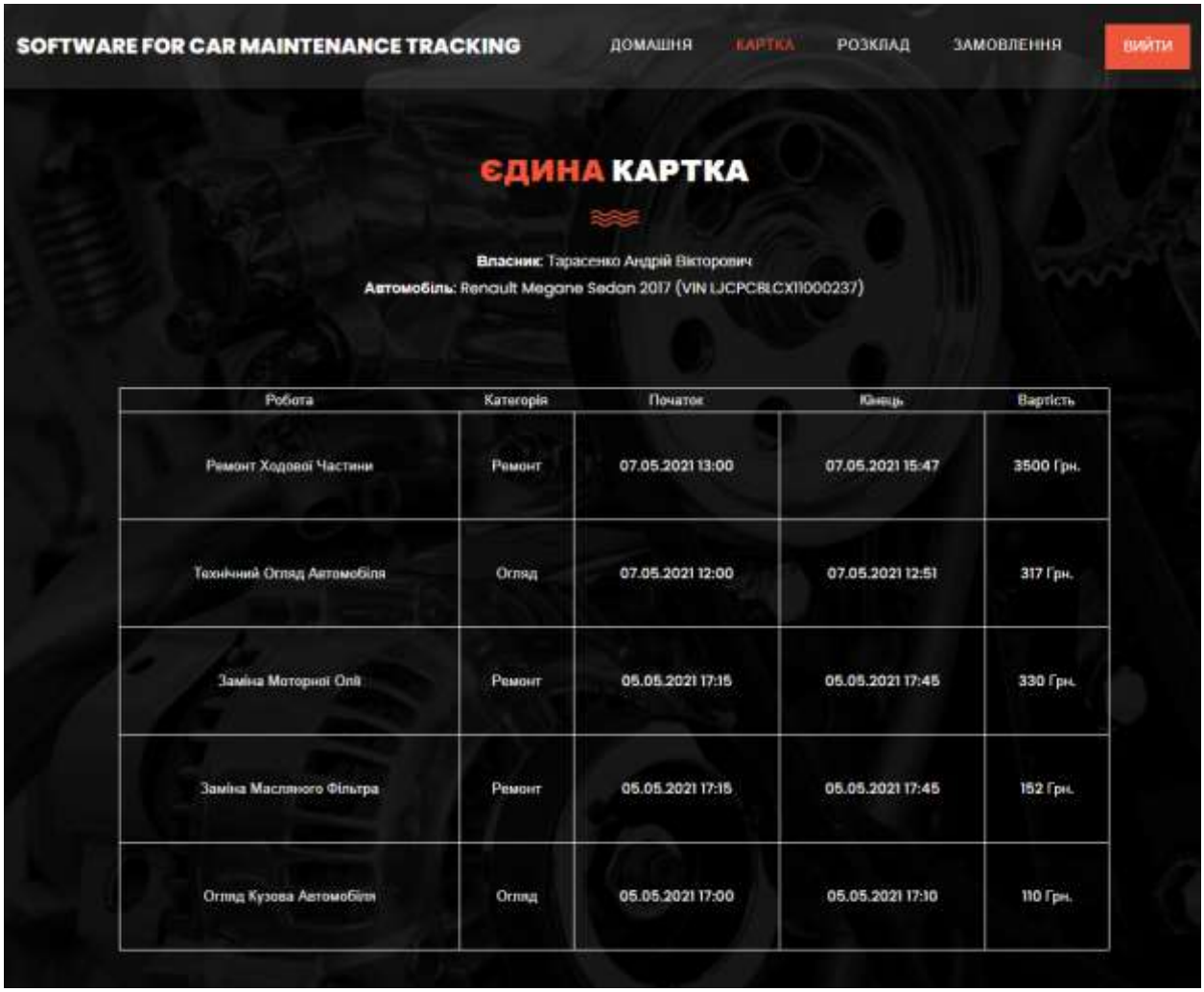

Рисунок 4.2 – Перегляд власником автомобіля робіт, виконаних за ним

На даній сторінці відображаються саме роботи, а не окремі замовлення. За кожною роботою відображається назва роботи, категорія роботи, дата і час початку роботи, дата і час завершення роботи, вартість роботи. Усі роботи розташовані, починаючи від поточної дати.

У верхній частині сторінки відображається інформація про автомобіль, його номер та власника. Якщо власник, який авторизувався в системі, має дані про обслуговування декількох автомобілів, то зверху також відображається випадаючий перелік з усіма доступними для даного власника

автомобілями, з якого потрібно обрати необхідний, а за замовчуванням відображаються дані про автомобіль, робота за яким виконувалась останньою.

На сторінці «Розклад» надається інформація про записи на технічне обслуговування, розташовані починаючи від поточної дати. За допомогою кнопки «Записатися» можна перейти до сторінки оформлення запиту на обслуговування (рис. 4.3).

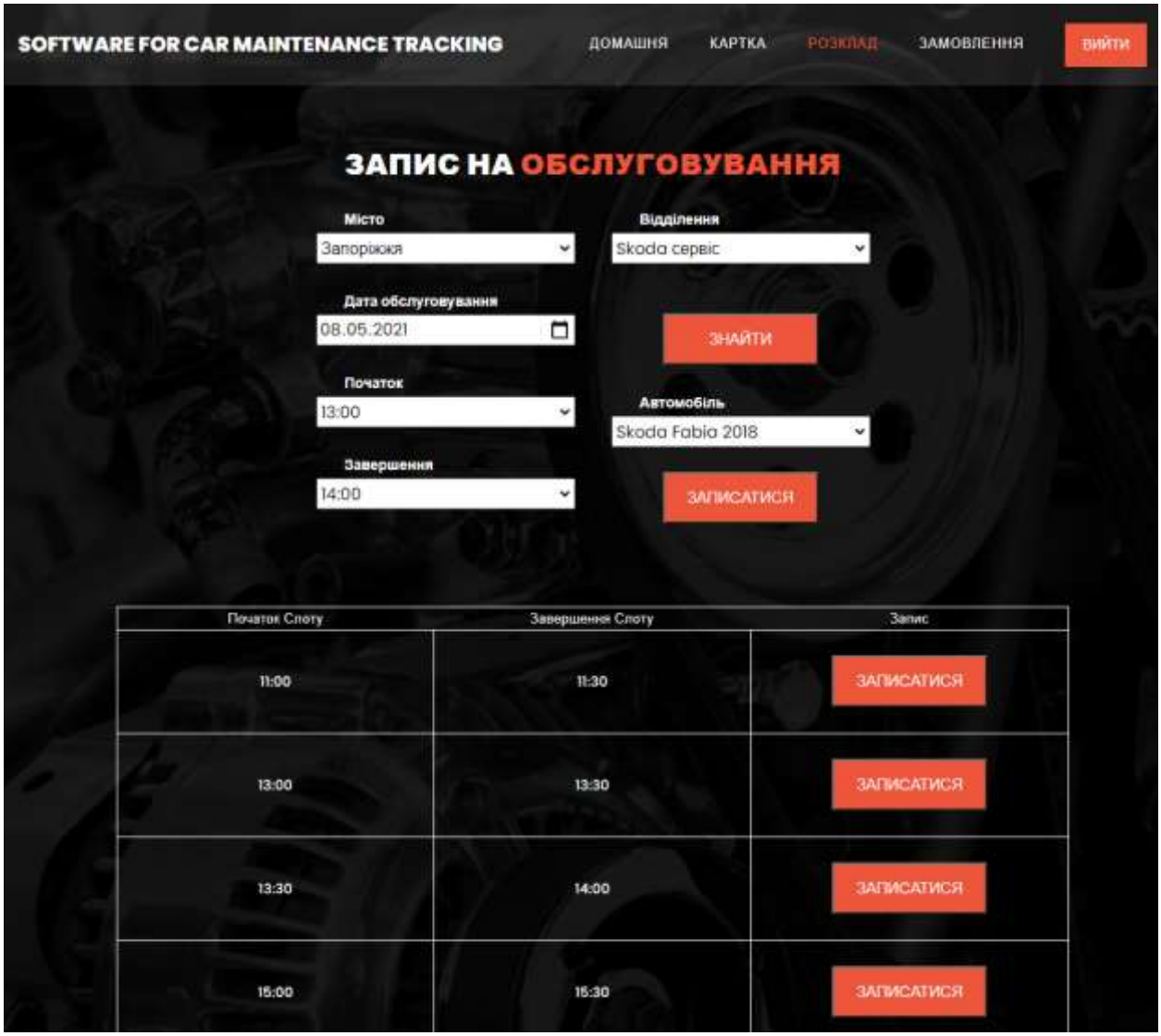

Рисунок 4.3 – Записування на обслуговування

На даній сторінці потрібно обрати місто сервісу з випадаючого переліку, потім сам сервіс у цьому місті, а далі обрати з календаря дату

обслуговування. Після цього потрібно натиснути на кнопку «Знайти», після чого актуалізуються дані нижче.

У таблиці з доступними слотами відображається на заданий день у заданому відділенні час початку, час завершення слоту. Для кожного слоту можна натиснути на кнопку «Записатися» для оформлення запиту на обслуговування. Після оформлення запиту менеджер має підтвердити цей запит, щоб він зайняв місце в розкладі і не відображався усім іншим користувачам.

Перед оформленням запиту потрібно обрати з випадаючого переліку також автомобіль власника. У переліку відображаються тільки автомобілі, за якими було раніше у цього власника сформовано замовлення. Самостійно користувач не може внести свій новий автомобіль, для цього потрібна робота менеджера. Це необхідно для забезпечення безпеки даних зокрема.

Якщо потрібно використати варіант обслуговування на більшу тривалість за слоти, що відображаються, то потрібно задати в переліку «Початок» один з варіантів, для нього обрати час завершення з переліку «Завершення». За допомогою кнопки «Записатися» можна оформити відповідний запит.

Таким чином, власник автомобіля може використовувати програму для моніторингу робіт за власним автомобілем та для роботи з розкладом технічного обслуговування.

Розділи меню технічного працівника включають:

– розділ «Замовлення», що дозволяє вносити інформацію про виконання робіт за замовленнями;

– розділ «Деталі», що дозволяє виконати пошук наявних у каталозі деталей;

– розділ «Картка», що дозволяє отримати доступ до даних єдиної картки обслуговування автомобіля і переглянути історію робіт, які було раніше виконано.

Технічний працівник зі сторінки «Замовлення» може переглядати замовлення в роботі (рис. 4.4). У нього є також можливість виконати пошук замовлення за VIN-номером автомобіля. Це необхідно для таких випадків, коли працівник потребує інформації про минулі ремонти автомобіля.

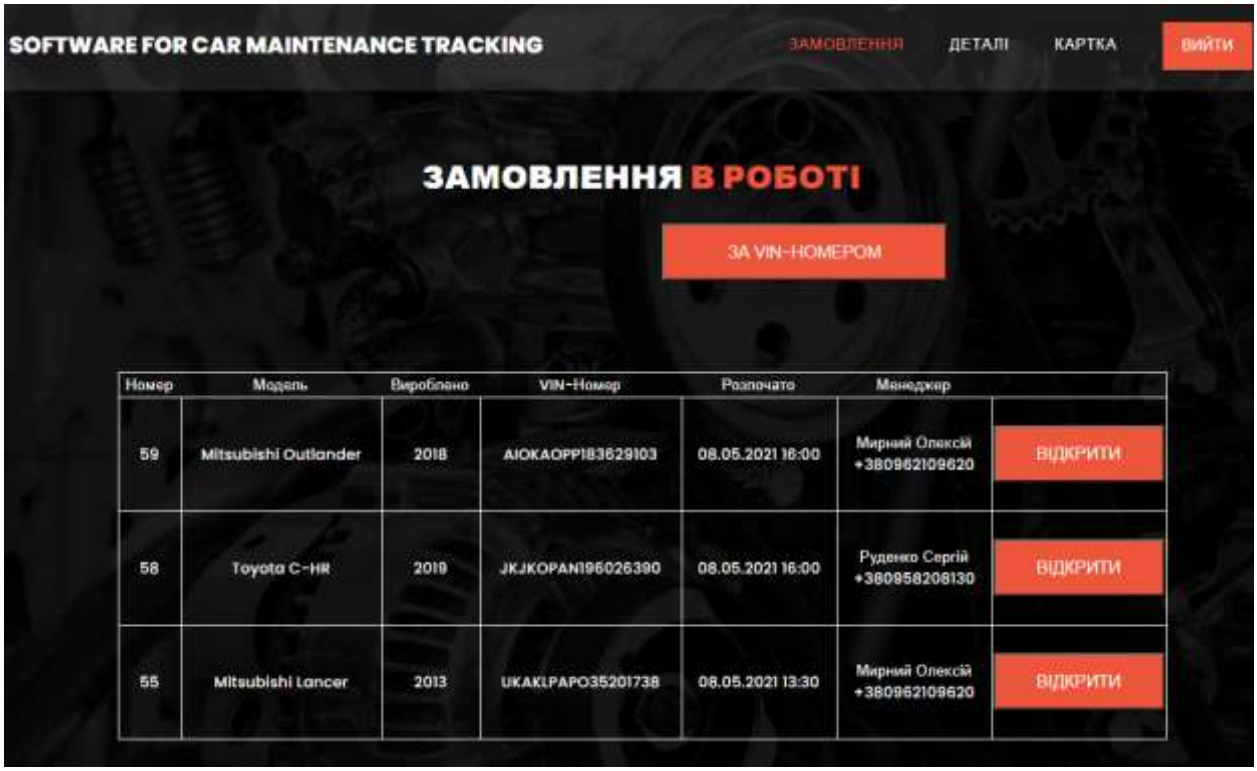

Рисунок 4.4 – Перегляд замовлень технічним працівником

Зі стандартного інтерфейсу цієї сторінки працівник може перейти до складу замовлення, звідки йому буде доступно перелік робіт і деталей (рис. 4.5).

Звідси працівник може переглянути набір робіт, переглянути деталі виконання відповідної роботи (зокрема процедуру виконання роботи) за допомогою кнопки «Переглянути», може зафіксувати початок виконання роботи за допомогою кнопки «Розпочати», зафіксувати завершення виконання роботи за допомогою кнопки «Завершити», натиснувши на кнопку «Деталі» перейти до сторінки внесення деталей, використаних під час робіт, а за допомогою кнопки «Замовити» оформити замовлення на постачання необхідних деталей.

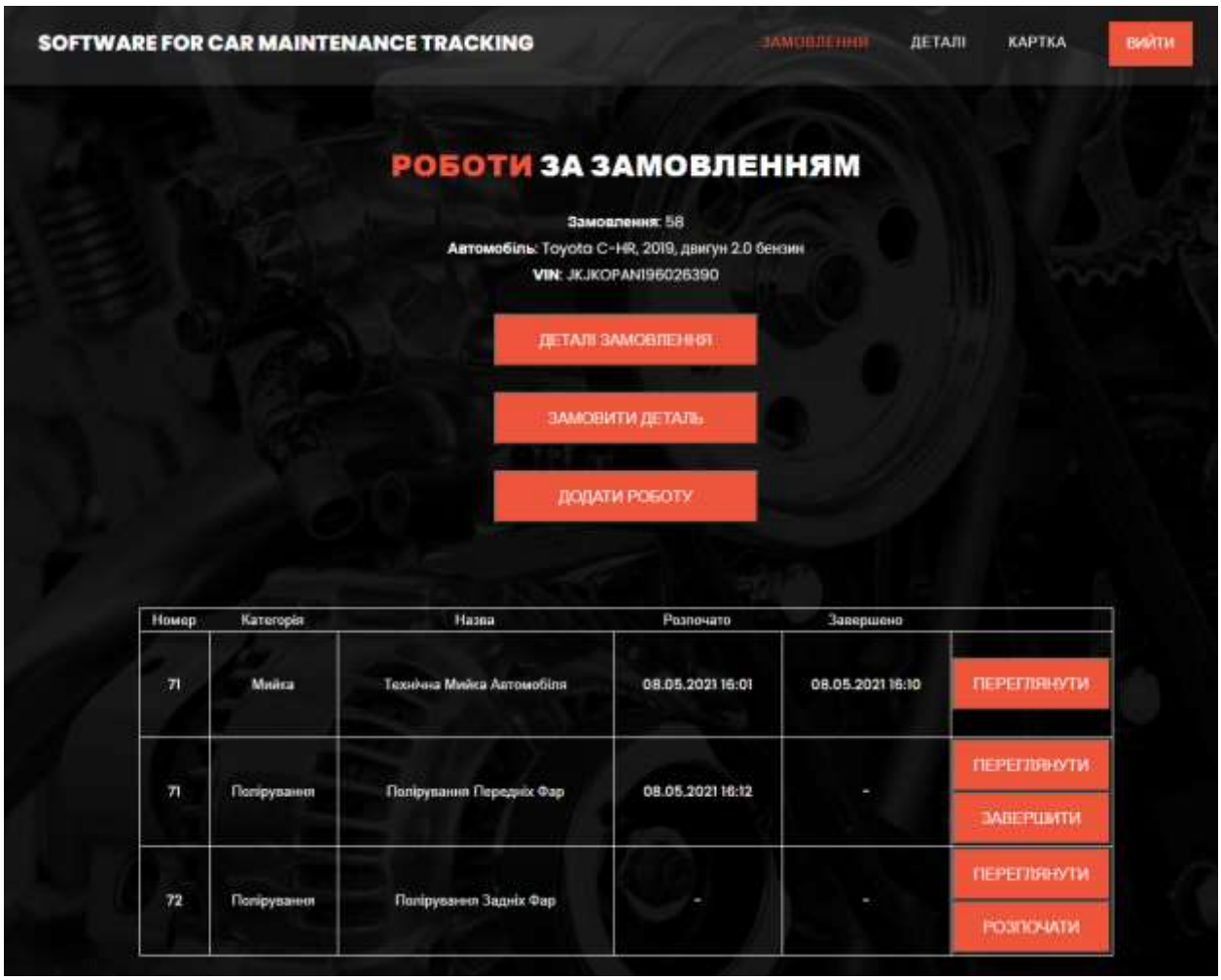

Рисунок 4.5 – Керування складом замовлення технічним працівником

Зі сторінки перегляду деталей роботи можна також переглянути особливості виконання, внесені самим працівником, а також перейти до редагування (за потреби), наприклад, якщо було внесено некоректно дані. Але дата і час фіксуються тільки натисканням кнопок, вручну встановити їх не можна.

Технічний працівник не може завершити замовлення цілком, адже для цього менеджер має перевірити сплату, але технічний працівник може зафіксувати факт завершення виконання окремих робіт у складі замовлення.

Розділи меню адміністратора включають:

– розділ «Авто», який дозволяє створювати каталог автомобілів, за допомогою якого потім можна вносити інформацію по конкретним автомобілям, обслуговування яких виконується;

– розділ «Деталі», який дозволяє створювати каталог деталей, з якими працюють сервіси, а також керувати постачальниками, від яких ці деталі отримуються;

– розділ «Сервіси», який дозволяє керувати відділеннями для технічного обслуговування автомобілів та працівниками у цих відділеннях;

– розділ «Роботи», який дозволяє керувати каталогом робіт, які виконуються під час технічного обслуговування;

– розділ «Замовлення», який дозволяє за необхідності (для коригування будь-якої внесеної інформації, до якої після того не має доступ менеджер) виконувати повне редагування замовлень, які було раніше створено менеджерами.

Розділи меню менеджера включають:

– розділ «Замовлення», де можна керувати процесом виконання замовлень, що є однією із головних задач менеджера;

– розділ «Деталі», де можна замовляти деталі та вносити інформацію щодо наявності деталей:

– розділ «Авто», де можна керувати даними автомобілів;

– розділ «Клієнти», де можна керувати даними клієнтів:

– розділ «Розклад», де можна переглядати розклад роботи сервісу та керувати запитами на обслуговування від клієнтів, тобто формувати розклад обслуговування.

Менеджер може працювати з замовленнями в двох режимах. На сторінці, що відкривається при виборі розділу меню «Замовлення» (рис. 4.6), за замовчуванням відображаються всі замовлення, що знаходяться в роботі відділення. Якщо натиснути на кнопку «Усі замовлення», то відкриється сторінка, на якій відображаються всі взагалі замовлення відділення, у яких можна виконувати пошук, встановлюючи номер телефону власника автомобіля, номер замовлення, часовий проміжок замовлень.

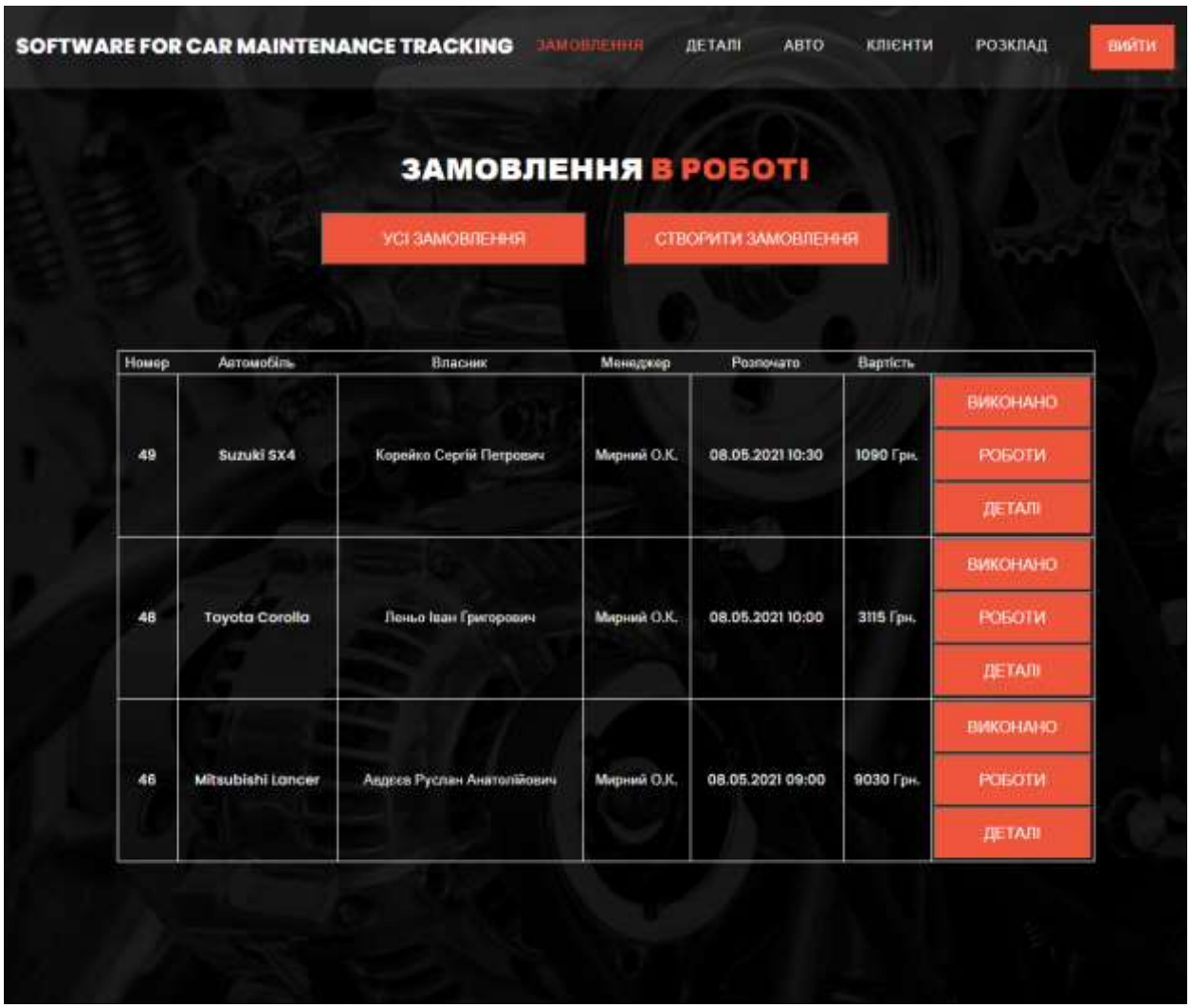

Рисунок 4.6 – Керування замовленнями в роботі

З цієї ж сторінки можна перейти до створення замовлення за допомогою кнопки «Створити замовлення».

Для кожного замовлення в роботі з таблиці відображається його номер, автомобіль, обслуговування якого виконується, власник автомобіля, менеджер, який створив замовлення, дату і час початку виконання замовлення, розраховану за внесеними роботами вартість замовлення та набір кнопок для виконання дій над замовленням.

За допомогою кнопки «Виконано» менеджер може завершити роботу над замовленням, після цього воно зникне з даної сторінки.

За допомогою кнопки «Роботи» менеджер може перейти до переліку робіт замовлення, звідки він може додавати нові роботи зокрема.

За допомогою кнопки «Деталі» менеджер може перейти до переліку деталей, що були використані при виконанні замовлення, звідси можна їх вносити.

Розділ «Розклад» дозволяє переглянути розклад обслуговування у відділенні для авторизованого менеджера на вказану шляхом вибору з календаря дату (рис. 4.7).

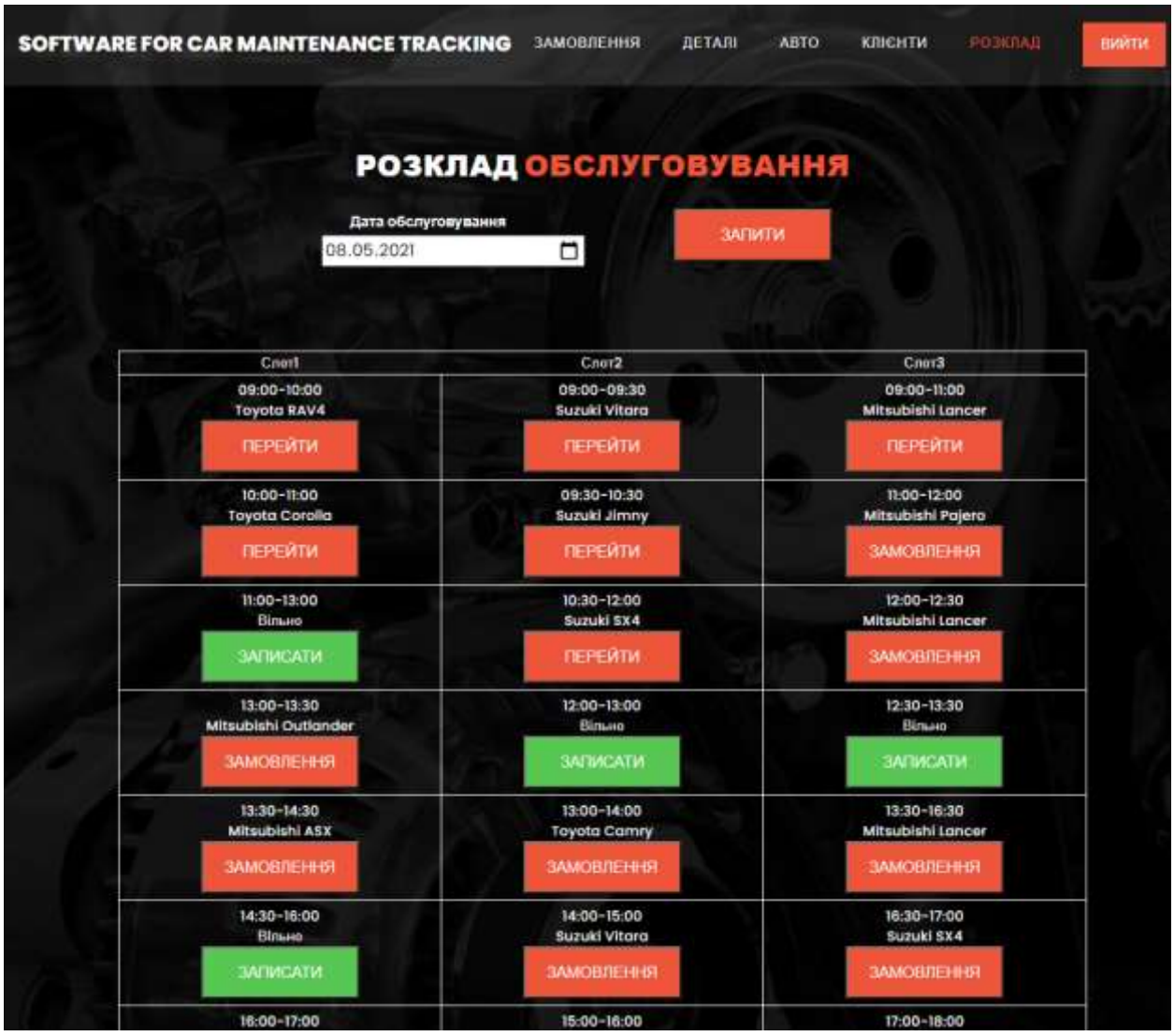

Рисунок 4.7 – Розклад обслуговування у відділенні

За вказаною датою відображається таблиця з доступними слотами. За кожним слотом відображаються часові слоти. Кожен часовий слот – це або зарезервований час обслуговування якогось автомобіля, модель якого вказано у клітинці, або тривалість вільного часового інтервалу, в який можна

виконати запис. Вільні позиції відображаються з зеленою кнопкою, що дозволяє перейти до оформлення запису. На відміну від оформлення запиту користувачем у даному випадку запис буде одразу вважатися підтвердженим. Якщо резервування вже відбулось, то відображається червона кнопка. Червона кнопка може мати назву «Перейти» – у випадку, коли обслуговування почалось і замовлення на нього вже було створено, або «Замовлення», якщо замовлення ще не було створено, тоді дана кнопка дозволяє перейти до створення замовлення.

ВИСНОВКИ

За результатами виконання роботи було досягнуто мету – розроблено програмне забезпечення підтримки контролю за технічним обслуговуванням автомобілів для забезпечення розподіленого підходу до обслуговування.

Виконаний аналіз програмних аналогів та особливостей предметної області дозволив сформулювати функціональні вимоги до програмного забезпечення.

Виконано вибір мови програмування Python та вебфреймворку Django для розробки вебзастосунку, проаналізовано архітектурний шаблон Model View Controller, який лежить в основі вебфреймворку Django, виконано моделювання сценаріїв роботи клієнтів, технічних працівників, менеджерів та адміністраторів з програмою, виконано проєктування БД, в результаті чого створено схему БД.

Описано розроблені класи, що дозволили впровадити архітектурний шаблон на основі використання вебфреймворку Django, в структуру застосунку. Структуру організації розроблених класів представлено на основі діаграми пакетів. Розроблено програмне забезпечення у вигляді вебзастосунку.

Програма призначена для централізованого перегляду і керування даними технічного обслуговування автомобілів. Програма дозволяє фіксувати виконані роботи за автомобілем, параметри їх виконання, використані деталі для проведення ремонту або обслуговування, переглядати результати обслуговування як за замовленнями загалом, так і за виконаними роботами. У результаті власники автомобілів, технічні працівники різних відділень та сервісів можуть отримувати єдиний доступ до даних про реалізоване технічне обслуговування.

Усі завдання дипломної кваліфікаційної роботи повністю виконано.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Top 5 Car Maintenance Software [Electronic resource]. – Access mode : https://safetyculture.com/app/car-maintenance/

2. Gelder, K. V. Fundamentals of Automotive Technology [Text] : Principles and Practice (Cdx Learning Systems) / K. V. Gelder. – Berlington : Jones & Bartlett Learning, 2017. – 1866 p.

3. Halderman, J. Automotive Technology [Text] : Principles, Diagnosis, and Service / J. Halderman. – London : Pearson, 2015. – 1664 p.

4. AutoFluent - Software for Tire Dealers and Auto Service Centers [Electronic resource]. – Access mode : https://autorepairsoftware.com/

5. Features – AutoFluent – Software for Tire Dealers and Auto Service Centers [Electronic resource]. – Access mode : https://autorepairsoftware.com/features

6. AutoFluent [Electronic resource] : workshop management software review – Accurate Reviews. – Access mode : https://www.accuratereviews.com/auto-repair-software/autofluent-review/

7. AutoRepairCloud [Electronic resource]. – Access mode : http://www.autorepaircloud.com/?utm_source=capterra&utm_medium=cpc&utm_campaign=ARC

8. AutoRepairCloud [Electronic resource] : Releases. – Access mode : http://www.autorepaircloud.com/releases.html

9. Брюс, Э. Философия Java [Текст] / Э. Брюс. – СПб. : Питер, 2019. – 1168 с.

10. Васильев, А. Программирование на Python в примерах и задачах [Текст] / А. Васильев. – М. : Бомбора, 2020. – 616 с.

11. Дауни, А. Б. Основы Python. Научитесь думать как программист [Текст] / А. Б. Дауни. – М. : Манн, Иванов и Фербер, 2021. – 304 с.

12. Django Architecture – 3 Major Components of MVC Pattern [Electronic resource]. – Access mode : https://data-flair.training/blogs/django-

architecture/

13. Кузнецов, М. В. MySQL 5 [Текст] / М. В. Кузнецов, И. В. Симдянов. – СПб. : БХВ-Петербург, 2007. – 592 с.

14. Free Template 548 Training Studio [Electronic resource]. – Access mode : https://templatemo.com/tm-548-training-studio

15. Generic display views | Django Documentation | Django Electronic resource]. – Access mode : https://docs.djangoproject.com/en/3.2/ref/class-based-views/generic-display/

ДОДАТОК А

Технічне завдання

Вступ

Необхідність розроблення програмного забезпечення продиктована тим, що часто результати технічного обслуговування залишаються недоступними клієнтам і мають відновлюватися за пам’яттю, іноді зберігаються в паперовому вигляді і недоступні в інших відділеннях сервісів. Тому важливим завданням є створення програмного забезпечення підтримки контролю за технічним обслуговуванням автомобілів.

А.1 Підстави для розробки

Підставою для виконуваної розробки в межах дипломної кваліфікаційної роботи бакалавра є завдання, що було визначене за темою «Програмне забезпечення підтримки контролю за технічним обслуговуванням автомобілів» і затверджене у відповідності з наказом № 103 від 30 березня 2021 р. за Національним університетом «Запорізька політехніка».

А.2 Призначення розробки

Програма, що розробляється, призначена для централізованого перегляду і керування даними технічного обслуговування автомобілів. Програма має дозволяти фіксувати виконані роботи за автомобілем, використані деталі для проведення ремонту або обслуговування.

А.3 Основні вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Програма повинна забезпечувати виконання наступних функціональних характеристик:

– перегляд робіт, виконаних за автомобілем;

– записування на обслуговування;

– перегляд розкладу власних обслуговувань;

– авторизація;

– перегляд робіт, які мають бути виконані за замовленням;

– перегляд процедури виконання роботи;

– визначення початку виконання роботи з обслуговування;

– визначення завершення виконання роботи з обслуговування;

– перегляд наявності деталей;

– керування даними працівників;

– керування даними виробників, моделей і версій моделей автомобілей;

– керування (в тому числі редагування) замовленнями;

– керування каталогом деталей;

– керування даними постачальників;

– керування даними відділень;

– керування каталогом робіт з обслуговування;

– створення замовлення на обслуговування;

– підтвердження виконання замовлення на обслуговування;

– внесення виконаних робіт за замовленням на обслуговування;

– внесення використаних деталей для обслуговування;

– замовлення деталей у постачальника;

– внесення прибуття деталей від постачальника;

– пошук замовлень за номером телефону власника;

– пошук придбаних деталей власником;

– визначення робіт, які найчастіше виконувались за автомобілем;

– перегляд замовлень, що знаходяться в роботі;

– пошук замовлення за номером;

– перегляд затримки виконання робіт за робітниками;

– внесення і редагування даних клієнтів.

А.3.2 Вимоги до надійності

Для забезпечення надійності має виконуватися валідація даних, що було введено користувачем в поля форм.

А.3.3 Умови експлуатації

Експлуатація програмного забезпечення має виконуватися на основі клієнт-серверної моделі. Для цього потрібно виділити окремий комп’ютер або скористатися виділеним сервером, який буде забезпечувати підтримку реалізації вебсервера. Вебсервер має бути доступним для запитів користувачів постійно.

А.3.4 Вимоги до складу та параметрів технічних засобів

Склад та параметри технічних засобів вебсервера було визначено для реалізації можливості доступу 300 користувачів одночасно. Відповідно вимоги можуть бути знижені, якщо потрібне забезпечення меншої кількості запитів, а для більшої кількості одночасних запитів вимоги навпаки мають бути збільшені. Мінімальні системні вимоги до вебсервера включають:

– процесор: 4 ядра, тактова частота – 2,9 ГГц;

– оперативна пам’ять – 10 Гб;

– платформа – 32-розрядна або 64-розрядна;

– жорсткий диск – 1 Гб вільного простору.

А.3.5 Вимоги до транспортування та збереження

Безпосередньо програма має транспортуватися та зберігатися через репозиторій вебсервера. Збереження на окремих фізичних носіях не передбачається.

А.3.6 Вимоги до програмної документації

Основою розробки даного програмного проєкту є технічне завдання на нього.

Опис прийнятих рішень виконується в пояснювальній записці. У розділі «Експлуатація та тестування програми» має бути представлено опис процесу роботи з програмою. Інших програмних документів не передбачається.

А.4 Порядок контролю та приймання

Календарний план завдання на дипломну кваліфікаційну роботу визначає набір етапів, за якими має забезпечуватися виконання, а також результати, які мають бути отримані за виконання відповідного етапу. Саме на основі даних результатів має забезпечуватися приймання роботи.

ДОДАТОК Б

Текст програми

Б.1 Текст файла views/orders.py

```
from django.contrib.auth.mixins import LoginRequiredMixin
from django.views.generic import ListView
from django.views.generic import UpdateView
from django.views.generic.detail import DetailView
from django.views.generic.edit import CreateView
from django.utils.decorators import method_decorator
from django.contrib.auth.decorators import login_required
from django.shortcuts import get_object_or_404
from .models import ServiceOrder
from .models import Owner
from .models import Staff
from .models import Car
from .models import CarService
from .models import Order
from .models import Timetable
from .forms import *
import datetime


class AdminServiceOrdersView (LoginRequiredMixin,ListView):
    model = ServiceOrder
    template_name = 'admin/sorders.html'
    paginate_by = 10

    def get_queryset(self):
        orders = []
        self.number = self.request.GET.get('number')
        self.department                                          =
self.request.GET.get('department')
        self.d1 = self.request.GET.get('date1')
        self.d2 = self.request.GET.get('date2')
        if self.department:
            orders                                               =
ServiceOrder.objects.filter(staff__department__pk                =
self.department)
        else:
            orders = ServiceOrder.objects.all()
        if self.number:
            orders = orders.filter(pk = self.number)
        if self.date1:
            orders = orders.filter(date1__gte=d1)
        if self.date2:
            orders = orders.filter(date1__lte=d2)
```

```
            orders = orders.order_by("pk")
            return orders

        def get_context_data(self, **kwargs):
            dcontext      =      super(AdminServiceOrdersView,
self).get_context_data(**kwargs)
            dcontext['number'] = self.number
            dcontext['department'] = self.department
            dcontext['date1'] = self.d1
            dcontext['date2'] = self.d2
        return dcontext


    class                              ManagerServiceOrdersView
(LoginRequiredMixin,ListView):
        model = ServiceOrder
        template_name = 'manager/sorders.html'
        paginate_by = 10

        def get_queryset(self):
            orders = []
            self.number = self.request.GET.get('number')
            self.phone = self.request.GET.get('phone')
            self.d1 = self.request.GET.get('date1')
            self.d2 = self.request.GET.get('date2')
            iduser = self.request.user.id
            staff = Staff.objects.get(userid = iduser)
            dep = staff.department
            orders  =  ServiceOrder.objects.filter(department  =
dep)
            if self.phone:
                orders                                          =
ServiceOrder.objects.filter(staff__phone = self.phone)
            if self.number:
                orders = orders.filter(pk = self.number)
            if self.date1:
                orders = orders.filter(date1__gte=d1)
            if self.date2:
                orders = orders.filter(date1__lte=d2)
            orders = orders.order_by("pk")
            return orders

        def get_context_data(self, **kwargs):
            dcontext      =      super(AdminServiceOrdersView,
self).get_context_data(**kwargs)
            dcontext['number'] = self.number
```

```
            dcontext['phone'] = self.phone
            dcontext['date1'] = self.d1
            dcontext['date2'] = self.d2
        return dcontext


    class                       ManagerProcessServiceOrdersView
(LoginRequiredMixin,ListView):
        model = ServiceOrder
        template_name = 'manager/procsorders.html'
        paginate_by = 10

        def get_queryset(self):
            iduser = self.request.user.id
            staff = Staff.objects.get(userid = iduser)
            dep = staff.department
            orders  =  ServiceOrder.objects.filter(department  =
dep)
            orders                                              =
orders.filter(date1__lte=datetime.datetime.now())
            orders = orders.filter(date2__isnull=True)
            orders = orders.order_by("-date1")
            return orders


    class OwnerServiceOrdersView (LoginRequiredMixin,ListView):
        model = ServiceOrder
        template_name = 'own/sorders.html'
        paginate_by = 10

        def get_queryset(self):
            orders = []
            iduser = self.request.user.id
            ow = Owner.objects.get(userid = iduser)
            orders = ServiceOrder.objects.filter(owner = ow)
            orders = orders.order_by("pk")
            return orders


    class TechServiceOrdersView (LoginRequiredMixin,ListView):
        model = ServiceOrder
        template_name = 'tech/sorders.html'
        paginate_by = 10

        def get_queryset(self):
            orders = []
```

```
            iduser = self.request.user.id
            staff = Staff.objects.get(userid = iduser)
            dep = staff.department
            orders                                                =
ServiceOrder.objects.filter(staff__department = dep)
            orders                                                =
orders.filter(date1__lte=datetime.datetime.now())
            orders = orders.filter(date2__isnull=True)
            orders = orders.order_by("-pk")
            return orders


    class                          TechServiceOrdersSearchView
(LoginRequiredMixin,ListView):
        model = ServiceOrder
        template_name = 'tech/vinsorders.html'
        paginate_by = 10

        def get_queryset(self):
            orders = []
            iduser = self.request.user.id
            staff = Staff.objects.get(userid = iduser)
            dep = staff.department
            self.carvin = self.request.GET.get('carvin')
            orders  =  ServiceOrder.objects.filter(department  =
dep)
            orders = orders.filter(car__vin=self.carvin)
            orders = orders.order_by("-pk")
            return orders

        def get_context_data(self, **kwargs):
            dcontext      =      super(AdminServiceOrdersView,
self).get_context_data(**kwargs)
            dcontext['carvin'] = self.carvin
        return dcontext


    class CreateServiceOrderView(CreateView):
        template_name = 'manager/createsorder.html'
        form_class = ServiceOrderCreateForm

        def form_valid(self, form):
            mdata = form.save(commit=False)
            mdata.created = datetime.datetime.now()
            us = self.request.user
            st = get_object_or_404(Staff, user=us)
```

```
            mdata.staff = st
            timetables    =    Timetable.objects.filter(car    =
mdata.car)
            timetables    =    timetables.filter(department    =
mdata.staff.department)
            timetables                                          =
timetables.filter(date1__lte=mdata.date1)
            timetables                                          =
timetables.filter(date2__gte=mdata.date2)
            if not timetables:
                timetables                                      =
Timetable.objects.filter(date1__lte=mdata.date1)
                timetables                                      =
timetables.filter(date2__gte=mdata.date2)
                timetables  =  timetables.filter(department  =
mdata.staff.department)
                maxslot = 0
                for t in timetables:
                    if t.slot > maxslot:
                        maxslot = t.slot
                timetbl = Timetable(
                    department = mdata.staff.department,
                    car = mdata.car,
                    slot = maxslot + 1,
                    date1 = mdata.date1,
                    date2 = mdata.date2,
                    approved = True)
                timetbl.save()
            mdata.price = 0
            mdata.date2 = None
            mdata.save()

        def get_form_kwargs(self, *args, **kwargs):
            kwargs       =       super(CreateServiceOrderView,
self).get_form_kwargs(*args, **kwargs)
            kwargs['cars'] = Car.objects.all()
            kwargs['owners'] = Owner.objects.all()
            carowners = Owner.objects.all()
            rescarowners = []
            for ow in carowners:
                servorders  =  ServiceOrder.objects.filter(owner
= ow)
                cars=[]
                for so in servorders:
                    if so.car not in cars:
                        cars.append(so.car)
```

```
                obj={}
                obj['owner'] = ow
                obj['cars'] = cars
                rescarowners.append(obj)
            kwargs['carowners'] = rescarowners
            return kwargs

        @method_decorator(login_required)
        def dispatch(self, *args, **kwargs):
            return                super(CreateServiceOrderView,
self).dispatch(*args, **kwargs)


    def DoneServiceOrder(request, pk):
        carservices = CarService.objects.filter(serviceorder =
pk)
        orders = Order.objects.filter(serviceorder = pk)
        tot = 0
        for serv in carservices:
            tot = tot + serv.price
        for order in orders:
            tot = tot + order.price

ServiceOrder.objects.filter(pk=pk).update(date2=datetime.datetim
e.now(), price = tot)


    class            ServiceOrderDetailView(LoginRequiredMixin,
DetailView):
        model = ServiceOrder
        template_name = 'manager/serviceorder.html'

        def get_context_data(self, **kwargs):
            context['services']                                =
CarService.objects.filter(serviceorder=self.kwargs.get('pk'))
            context['components']                              =
Order.objects.filter(serviceorder =self.kwargs.get('pk'))
            return context


    class            UpdateServiceOrderView(LoginRequiredMixin,
UpdateView):
        template_name = 'admin/updatesorder.html'
        form_class = ServiceOrderUpdateForm
```

Б.2 Текст файла models.py

```
from django.db import models
from django.urls import reverse


class Owner (models.Model):
    lastname  = models.CharField(max_length=50)
    firstname  = models.CharField(max_length=50)
    thirdname  = models.CharField(max_length=50)
    phone  = models.CharField(max_length=13, unique=True)
    license          =     models.CharField(max_length=100,
unique=True)
    userid   =  models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE)

    def __str__(self):
        return self.lastname + " " + self.phone

    def get_absolute_url(self):
        return               reverse('owner.views.details',
args=[str(self.id)])


class Staff  models.Model):
    lastname  = models.CharField(max_length=50)
    firstname  = models.CharField(max_length=50)
    thirdname  = models.CharField(max_length=50)
    phone  = models.CharField(max_length=13, unique=True)
    position  = models.CharField(max_length=50)
    started = models.DateField()
    department       =       ForeignKey       (Department,
on_delete=models.CASCADE)

    def __str__(self):
        return self.position + " " + self.lastname + " " +
self.phone

    class Meta:
        ordering = ['-when']

    def get_absolute_url(self):
        return               reverse('staff.views.details',
args=[str(self.id)])
```

```
    class Department (models.Model):
        title = models.CharField(max_length=100)
        city = ForeignKey (City, on_delete=models.CASCADE)
        address = models.CharField(max_length=100)
        phone = models.CharField(max_length=13, unique=True)
        days = models.IntegerField()
        slots = models.IntegerField()

        def __str__(self):
            return self.title

        class Meta:
            ordering = ['-title']

        def get_absolute_url(self):
            return          reverse('department.views.details',
args=[str(self.id)])


    class City  (models.Model):
        title = models.CharField(max_length=50, unique=True)


    class CarBrand (models.Model):
        title = models.CharField(max_length=20, unique=True)
        phone = models.CharField(max_length=13, unique=True)
        address = models.CharField(max_length=100)
        site = models.CharField(max_length=100)

        def __str__(self):
            return self.title

        class Meta:
            ordering = ['title']

        def get_absolute_url(self):
            return            reverse('carbrand.views.details',
args=[str(self.id)])


    class CarModel (models.Model):
        title = models.CharField(max_length=50)
        carbrand        =        ForeignKey        (CarBrand,
on_delete=models.CASCADE)

        def __str__(self):
```

```
            return self.title

        class Meta:
            ordering = ['title']


    class CarModelVersion (models.Model):
        carmodel = ForeignKey (CarModel,
on_delete=models.CASCADE)
        year1 = models.IntegerField()
        year2 = models.IntegerField()
        documentation = models.CharField(max_length=255)
        features = models.CharField(max_length=255)

        def __str__(self):
            return self.title

        class Meta:
            ordering = ['carmodel']

        def get_absolute_url(self):
            return reverse('carmodelversion.views.details',
args=[str(self.id)])


    class Car (models.Model):
        vin = models.CharField(max_length=17, unique=True)
        made = models.DateField()
        specification = models.CharField(max_length=100)
        owner = ForeignKey (Owner, on_delete=models.CASCADE)
        carmodelversion = ForeignKey (CarModelVersion,
on_delete=models.CASCADE)

        def __str__(self):
            return self.vin

        class Meta:
            ordering = ['vin']

        def get_absolute_url(self):
            return reverse('car.views.details',
args=[str(self.id)])


    class ServiceGroup (models.Model):
        title = models.CharField(max_length=50)
```

```
        def __str__(self):
            return self.title

        class Meta:
            ordering = ['title']


    class Service (models.Model):
        title = models.CharField(max_length=50)
        servicegroup      =      ForeignKey      (ServiceGroup,
on_delete=models.CASCADE)
        procedure = models.CharField(max_length=255)

        def __str__(self):
            return self.title

        class Meta:
            ordering = ['title']

        def get_absolute_url(self):
            return             reverse('service.views.details',
args=[str(self.id)])


    class ServicePrice (models.Model):
        service         =         ForeignKey         (Service,
on_delete=models.CASCADE)
        carmodelversion    =    ForeignKey    (CarModelVersion,
on_delete=models.CASCADE)
        price = models.FloatField()
        duration = models.IntegerField()
        specification = models.CharField(max_length=100)
        procedure = models.CharField(max_length=255)
        department       =       ForeignKey       (Department,
on_delete=models.CASCADE)

        def get_absolute_url(self):
            return        reverse('serviceprice.views.details',
args=[str(self.id)])


    class CarService (models.Model):
        serviceorder      =      ForeignKey      (ServiceOrder,
on_delete=models.CASCADE)
        staff = ForeignKey (Staff, on_delete=models.CASCADE)
```

```
        service = ForeignKey (Service,
on_delete=models.CASCADE)
        price = models.FloatField()
        specification = models.CharField(max_length=255)
        date1 = models.DateTimeField()
        date2 = models.DateTimeField()

        def get_absolute_url(self):
            return reverse('carservice.views.details',
args=[str(self.id)])


    class ServiceOrder (models.Model):
        staff = ForeignKey (Staff, on_delete=models.CASCADE)
        car = ForeignKey (Car, on_delete=models.CASCADE)
        owner = ForeignKey (Owner, on_delete=models.CASCADE)
        date1 = models.DateTimeField()
        date2 = models.DateTimeField()
        price = models.FloatField(decimal_places=2)
        created = models.DateTimeField()

        def get_absolute_url(self):
            return reverse('serviceorder.views.details',
args=[str(self.id)])


    class Order (models.Model):
        car = ForeignKey (Car, on_delete=models.CASCADE)
        supplier = ForeignKey (Supplier,
on_delete=models.CASCADE)
        serviceorder = ForeignKey (ServiceOrder,
on_delete=models.CASCADE)
        made = models.DateTimeField()
        performed = models.DateTimeField()
        price = models.FloatField(decimal_places=2)

        class Meta:
            ordering = ['performed']

        def get_absolute_url(self):
            return reverse('order.views.details',
args=[str(self.id)])


    class Supplier (models.Model):
        title = models.CharField(max_length=50, unique=True)
```

```
        phone = models.CharField(max_length=13)
        address = models.CharField(max_length=100)
        site = models.CharField(max_length=100)

        def __str__(self):
            return self.title

        class Meta:
            ordering = ['title']

        def get_absolute_url(self):
            return            reverse('supplier.views.details',
args=[str(self.id)])


    class ComponentOrder (models.Model):
        component        =        ForeignKey        (Component,
on_delete=models.CASCADE)
        order = ForeignKey (Order, on_delete=models.CASCADE)
        price = models.FloatField(decimal_places=2)

        def get_absolute_url(self):
            return      reverse('componentorder.views.details',
args=[str(self.id)])


    class ComponentGroup (models.Model):
        title = models.CharField(max_length=100)

        def __str__(self):
            return self.title

        class Meta:
            ordering = ['title']


    class Component (models.Model):
        title = models.CharField(max_length=255)
        componentgroup     =     ForeignKey     (ComponentGroup,
on_delete=models.CASCADE)
        carmodelversion    =    ForeignKey    (CarModelVersion,
on_delete=models.CASCADE)
        supplier        =        ForeignKey        (Supplier,
on_delete=models.CASCADE)
        price = models.FloatField(decimal_places=2)
        quantity = models.IntegerField()
```

```
          department       =       ForeignKey       (Department,
on_delete=models.CASCADE)

          def __str__(self):
              return self.title

          class Meta:
              ordering = ['title']

          def get_absolute_url(self):
              return           reverse('component.views.details',
args=[str(self.id)])


      class Timetable (models.Model):
          department       =       ForeignKey       (Department,
on_delete=models.CASCADE)
          car = ForeignKey (Car, on_delete=models.CASCADE)
          slot = models.IntegerField()
          description = models.CharField(max_length=255)
          date1 = models.DateTimeField()
          date2 = models.DateTimeField()
          approved = models.BooleanField()

          class Meta:
              ordering = ['date1']

          def get_absolute_url(self):
              return           reverse('timetable.views.details',
args=[str(self.id)])
```

ДОДАТОК В

Слайди презентації

Рисунок В.1 – Слайд № 1

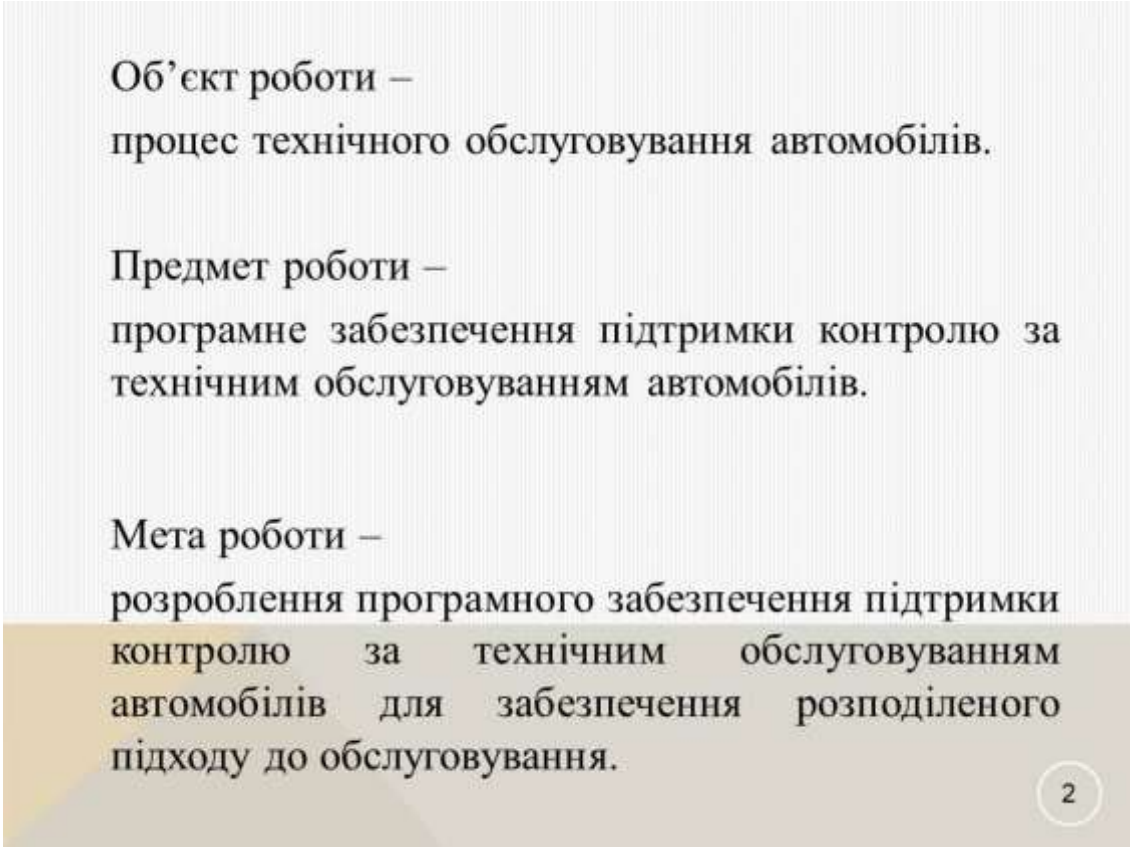

Рисунок В.2 – Слайд № 2

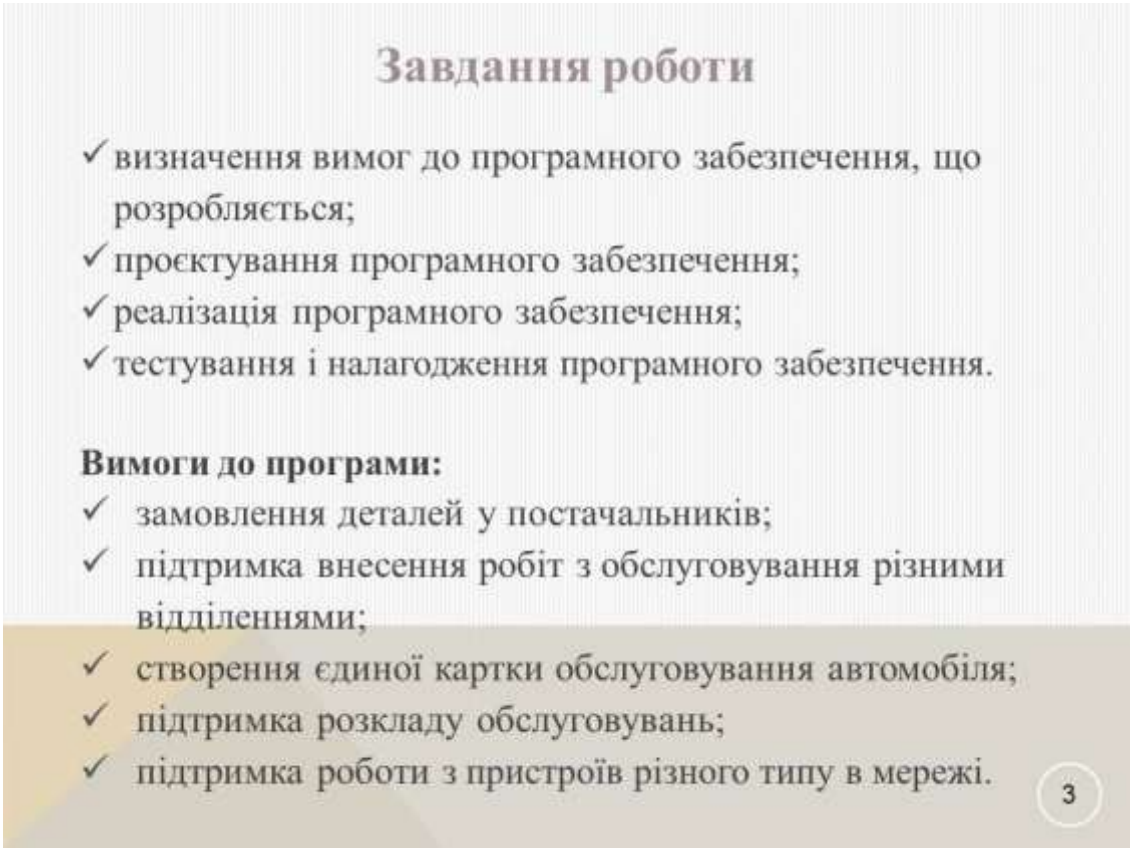

Рисунок В.3 – Слайд № 3

Рисунок В.4 – Слайд № 4

Рисунок В.5 – Слайд № 5

Рисунок В.6 – Слайд № 6

Рисунок В.7 – Слайд № 7

Рисунок В.8 – Слайд № 8

Рисунок В.9 – Слайд № 9

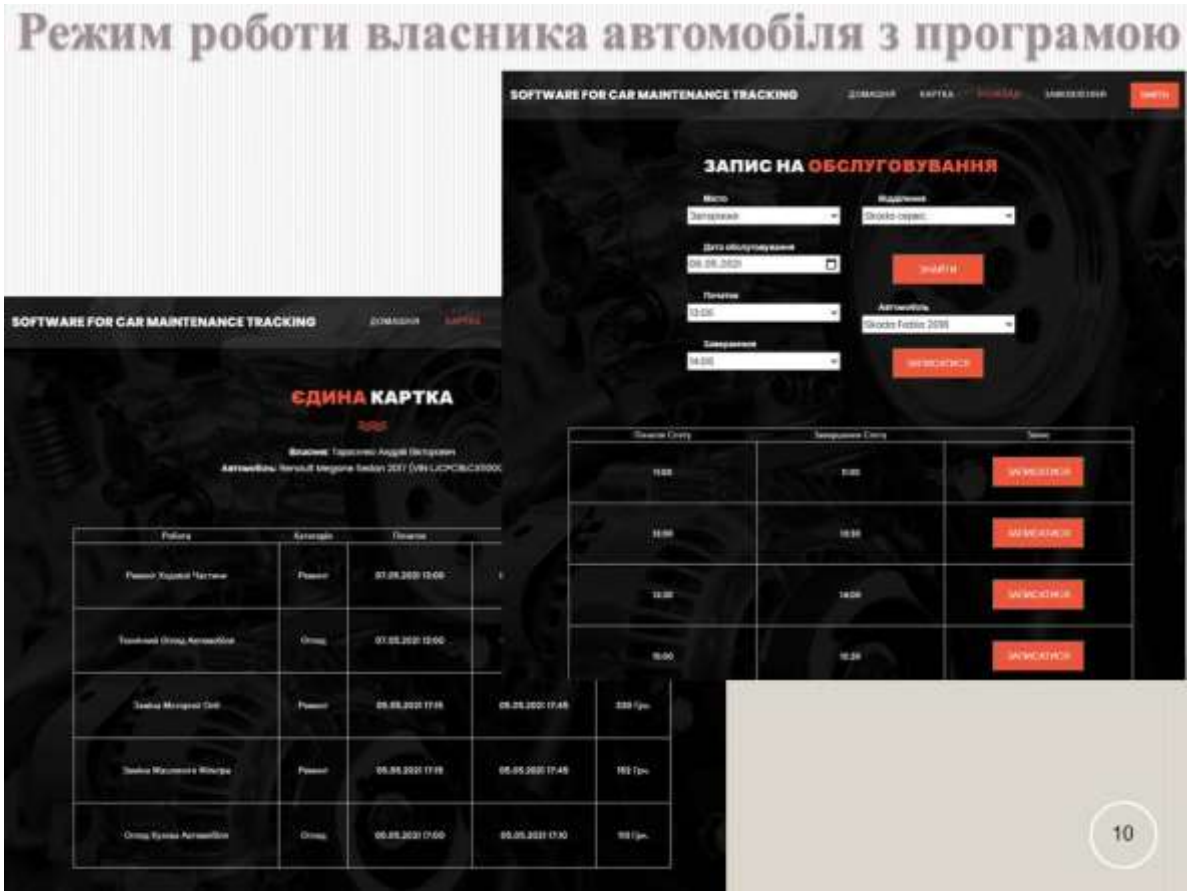

Рисунок В.10 – Слайд № 10

Рисунок В.11 – Слайд № 11

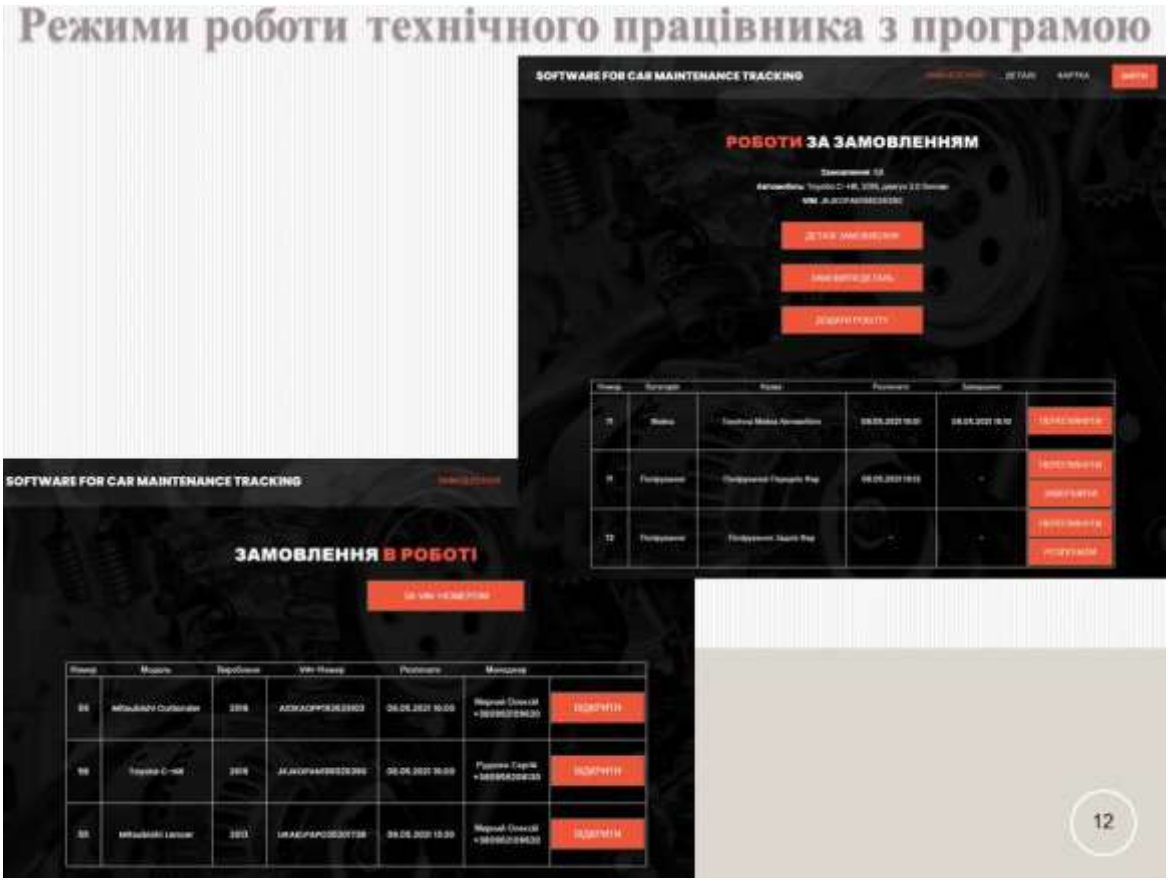

Рисунок В.12 – Слайд № 12

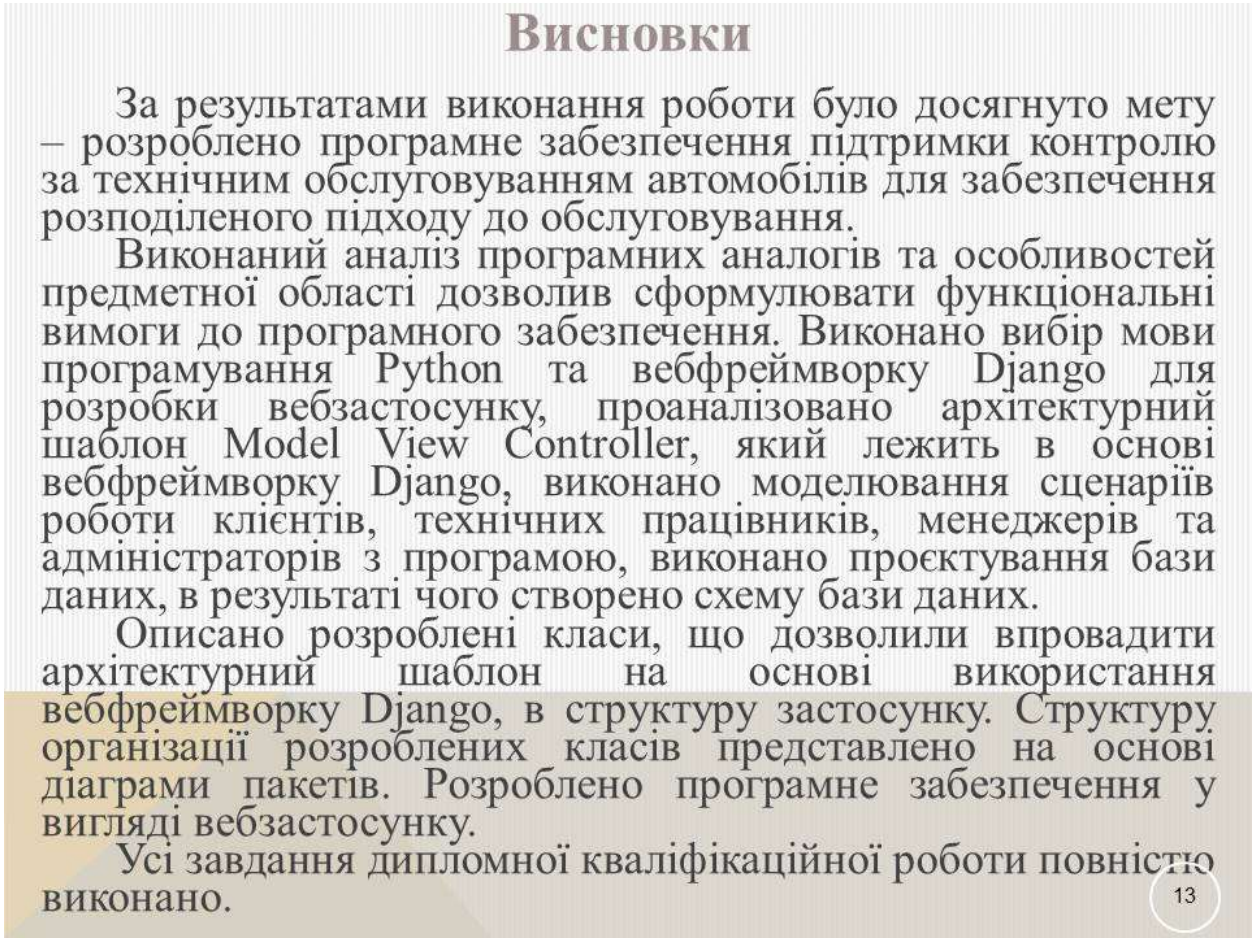

Рисунок В.13 – Слайд № 13