МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інформаційна система керування заявками станції технічного обслуговування»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Шелєхов І.В.**

Студента групи ІН – 73 — **Д'ячук І.О.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2021 р.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-73 спеціальності "Комп'ютерні науки" денної форми навчання Д'ячука Івана Олеговича.

Тема: " Інформаційна система керування заявками станції технічного обслуговування. "

Затверджена наказом по СумДУ

№_____ від _____2021 р.

Зміст пояснювальної записки: 1) аналіз проблеми та постановка задачі; 2) вибір метода розв'язання задачі; 3) розробка інформаційного і програмного забезпечення системи

Дата видачі завдання "_____"_____2021 р.

Керівник випускної роботи _____ Шелєхов І.В.

Завдання прийняв до виконання _____Д'ячук І.О.

РЕФЕРАТ

Записка: 34 стор., 17 рис., 1 табл., 1 додаток, 8 джерел.

Об’єкт дослідження — система керування заявками

Мета роботи — інформаційне та програмне забезпечення системи керування заявками станції технічного обслуговування

Методи дослідження — технології створення інформаційних систем

Результати — розроблено інформаційну систему керування заявками на станції технічного обслуговування, яка дозволяє вести облік автомобілів, що відвідували СТО.

DJANGO, PYTHON, ІНФОРМАЦІЙНА СИСТЕМА, АВТОМОБІЛЬНА ІНДУСТРІЯ

ЗМІСТ

ВСТУП

В епоху діджиталізації, суспільство поступово відмовляється від паперових носіїв інформації та переходить до електронних. Чи можна зараз у бухгалтерії побачити так багато розрахунків на папері, як раніше? Чи в університеті ви йдете до бібліотеки, щоб написати курсову або дипломну роботу? Чи ходите ви на лекції, щоб отримувати знання? Відтепер все це замінюють комп’ютери та мережа інтернет.

Вони поступово стають заміною у багатьох сферах життя. Як приклад, вони допомагають вести різну звітність у комунальних установах, на підприємствах або сервісних центрах.

В даній роботі буде спроектовано та розроблено інформаційну систему керування заявками станції технічного обслуговування. Так як автомобілів все більше, попит на СТО росте, відповідно є потреба у веденні обліку автомобілів, які були доставлені для ремонту, технічного огляду тощо.

Якщо в цьому випадку використовувати паперові носії, це може сповільнити та ускладнити роботу при веденні обліку автомобілів через пошук або допущення помилок. Окрім цього, є ризик на втрату частини документів при непередбачуваних випадках.

Саме тому буде розроблено систему, яка значно спростить ведення обліку автомобілів на станції технічого обслуговування.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1. Аналіз аналогів інформаційної системи

За своїми функціями система керуванням заявками належить до програмного забезпечення операційних служб СТО та центрів сервісного ремонту автомобілів тому порівнювати будемо з відомими аналогами.

Першим аналогом такої системи є «ПТ-Авто 8.3» [2]. Дана система призначена для комплексного обліку та управління в автомобільному бізнесі автоматизації.

Система є багатогранною, тобто її можуть використовувати як станції технічного обслуговування, так і автомобільні дилери.

Рис.1.1. – зовнішній вигляд інтерфейсу ПТ-Авто 8.3

Недоліками цієї системи є:

- Розроблена на 1С, що відразу каже про складність масштабування системи та її налагодження при появі помилок;
- Існує лише версія для персональних комп’ютерів, тобто переглянути звітність в онлайн-режимі з будь-якого іншого пристрою не буде можливим;
- Застарілий дизайн, який навантажує очі співробітників

Наступним аналогом, який ми розглянемо, є «Ремонлайн» [3]. Дане рішення має версії як для малого, середнього, так і для великого бізнесу.

Перевагами, у порівнянні з попередньою системою, є:

- сучасний дизайн, який відповідає новим трендам;
- зручний інтуїтивний інтерфейс, який дозволяє співробітнику будь-якого рівня розібратися в роботі з нею;
- можливість вести облік з будь-якого пристрою завдяки знаходженню системи у мережі інтернет;

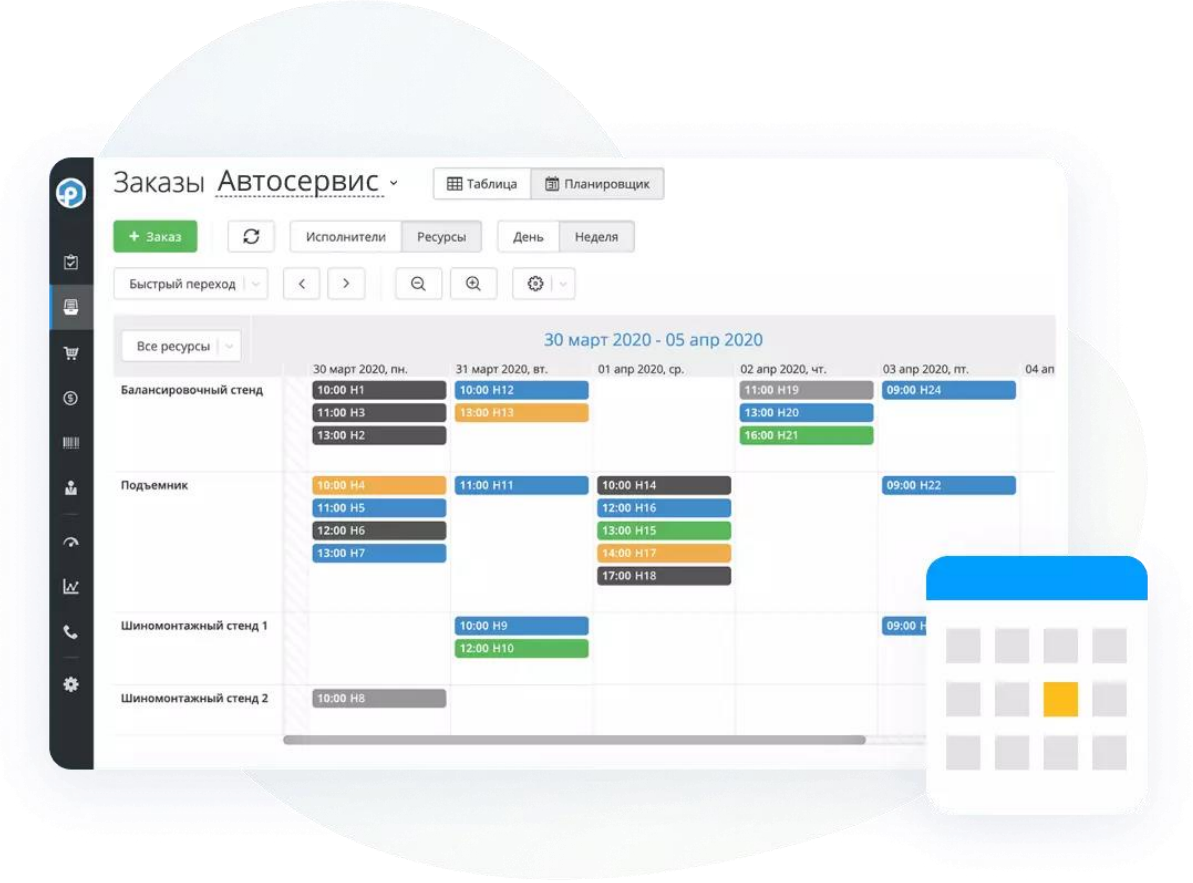

Рис.1.2 – зовнішній вигляд системи для СТО «Ремонлайн»

Проте, у системи можна виділити такі недоліки:

- високі ціни на тарифи відповідно обмежень (рис.1.3)
- так як система знаходиться на сервері розробника, є відсутність гарантії появи особистих даних клієнтів та компанії на «чорному ринку», що може суттєво вплинути на репутацію СТО;

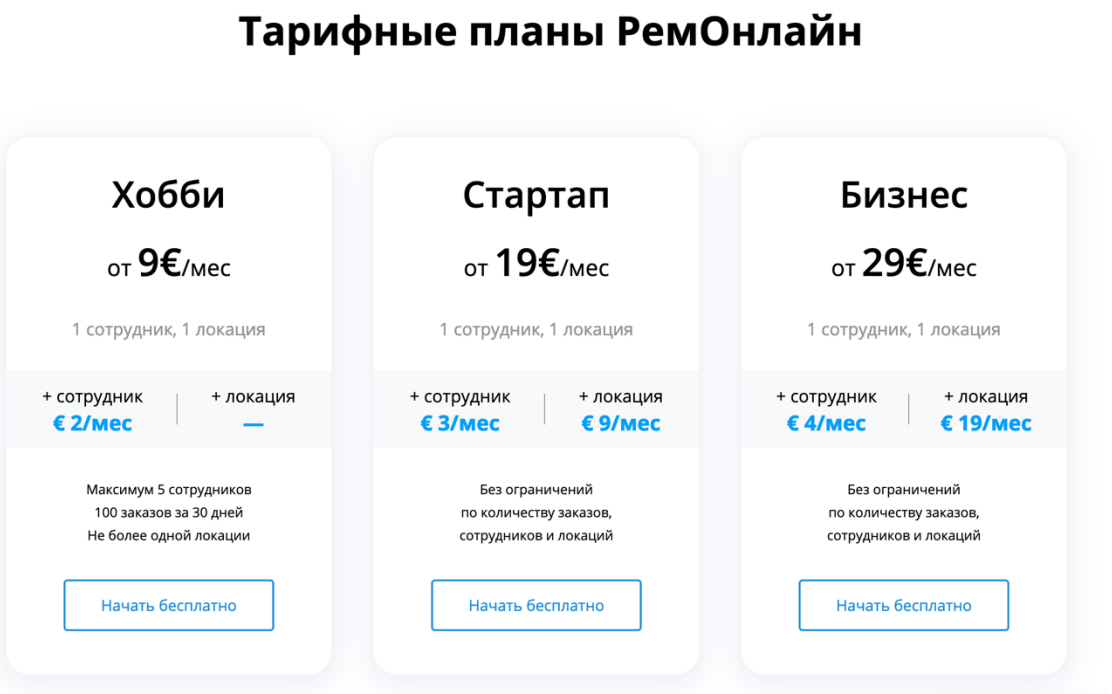

Рис.1.3. – тарифи системи для СТО «Ремонлайн».

Останнім аналогом, який ми розглянемо, є «Універсальна система обліку» [4], яка розроблена спеціалістами з Казахстану та має широкий функціонал.

Перевагою цієї системи є те, що вона має мінімалістичний та інтуїтивно зрозумілий дизайн.

Недоліками цієї системи є:

- офіційний сайт, що не відповідає трендам дизайну і потенційний користувач може легко заплутатися у його структурі, що вплине на придбання;
- відсутність можливості слідкувати за системою з іншого пристрою, попередньо не налаштувавши його для цього;
- система існує лише для операційної системи Windows;

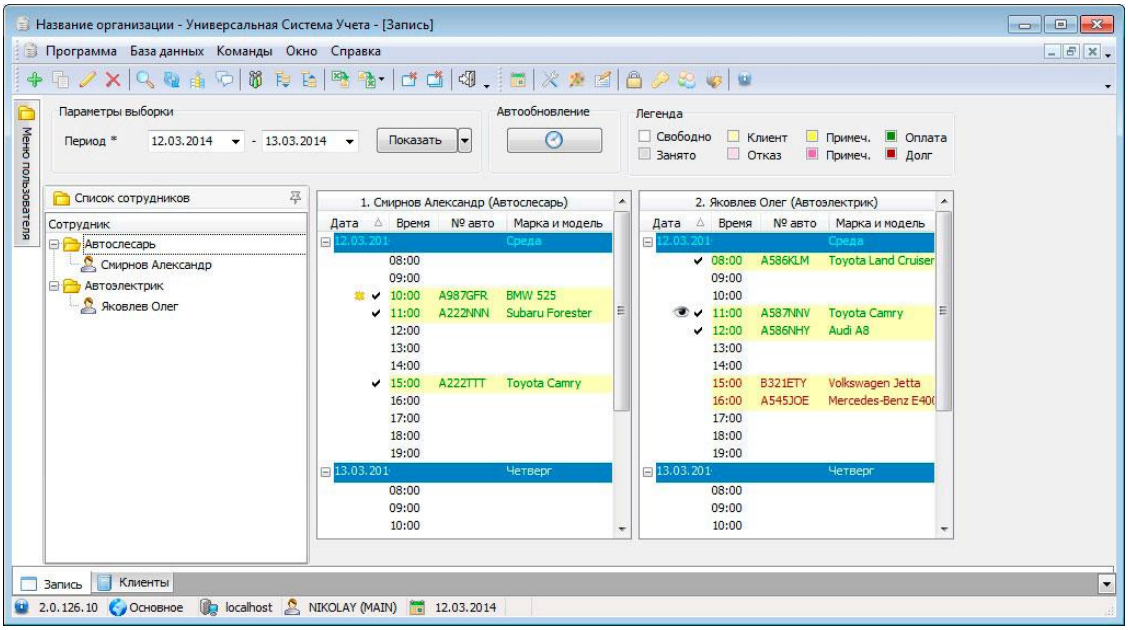

Рис.1.4 – зовнішній вигляд «Універсальної системи обліку»

Після пошуку та аналізу аналогічних систем, сформована порівняльна таблиця розглянутих прикладів та майбутньої інформаційної системи.

Таблиця 1.1 – Порівняльна характеристика аналогів

Назва критерію	Назва системи			
	ПТ-Авто 8.3	Ремонлайн	Універсальна Система Обліку	Дієве СТО
1.Крос-платформовість	-	+	-	+
2. Актуальний дизайн та зручна навігація	-	+	-	+
3. Можливість масштабування	-	-	-	+

1.2 Постановка задачі

Результати проведеного аналітичного огляду доводять актуальність задачі проєктування інформаційних систем керування заявками станції технічного обслуговування. Для розв'язання такої задачі необхідно виконати такі завдання:

1) Визначити структуру бази даних інформаційної системи;

2) Провести вибір засобів програмної реалізації інформаційної системи;

3) Розробити і програмно реалізувати інформаційну системи;

4) Виконати тестування інформаційної системи.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ КЕРУВАННЯ ЗАЯВКАМИ СТАНЦІЇ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ

2.1 Концептуальна модель системи

Система повинна забезпечити ведення електронного обліку автомобілів, що приїзжають на СТО для користування послугами.

Спочатку водій привозить автомобіль, який потребує техогляду, ремонту або інших послуг. Співробітник СТО встановлює мету візиту водія, паралельно додаючи у систему такі дані, як:

- Марка авто
- Номерний знак
- Дата та час візиту
- Опис проблеми або послуги

Дані заносяться зі статусом «Очікує огляду». Взагалі система матиме такі статуси, як:

- «Очікує огляду»
- «Очікує ремонту»
- «Очікує техогляду»
- «Очікування комплектуючих»
- «Техогляд проведено»
- «Відремонтовано»
- «Не підлягає ремонту»

Також система матиме такі типи послуг, як:

- «Техогляд»
- «Ремонт»
- «Розвал сходження»
- «Інше»

Статус заявки змінюється в залежності від ходу виконання послуги. Коли виконання послуги завершено, статус змінюється на «Відремонтовано», «Не підлягає ремонту», «Техогляд проведено», співробітник додає дату та час виконання роботи, помічає заявку як виконану, а також встановлює ціну за виконану послугу.

Модель того, як це відбуватиметься, представлена на рис.2.1.

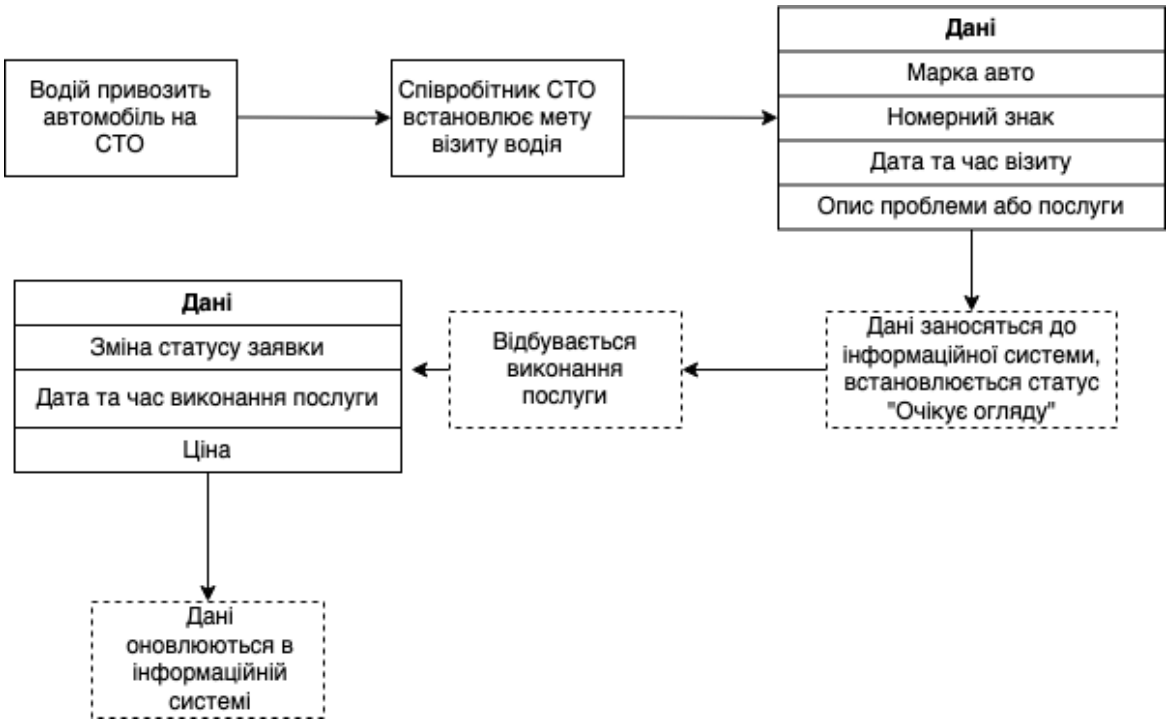

Рис.2.1 Модель процесу керування заявками на станції технічного обслуговування

Варто згадати, що на СТО можуть бути не всі комплектуючі для автомобілів, а тому в ІС варто передбачити те, що ці комплектуючі ми будемо замовляти.

Для цього ми розширимо нашу концептуальну модель, в якій передбачимо облік заявок на замовлення комплектуючих. Результат ви можете переглянути на рис.2.2.

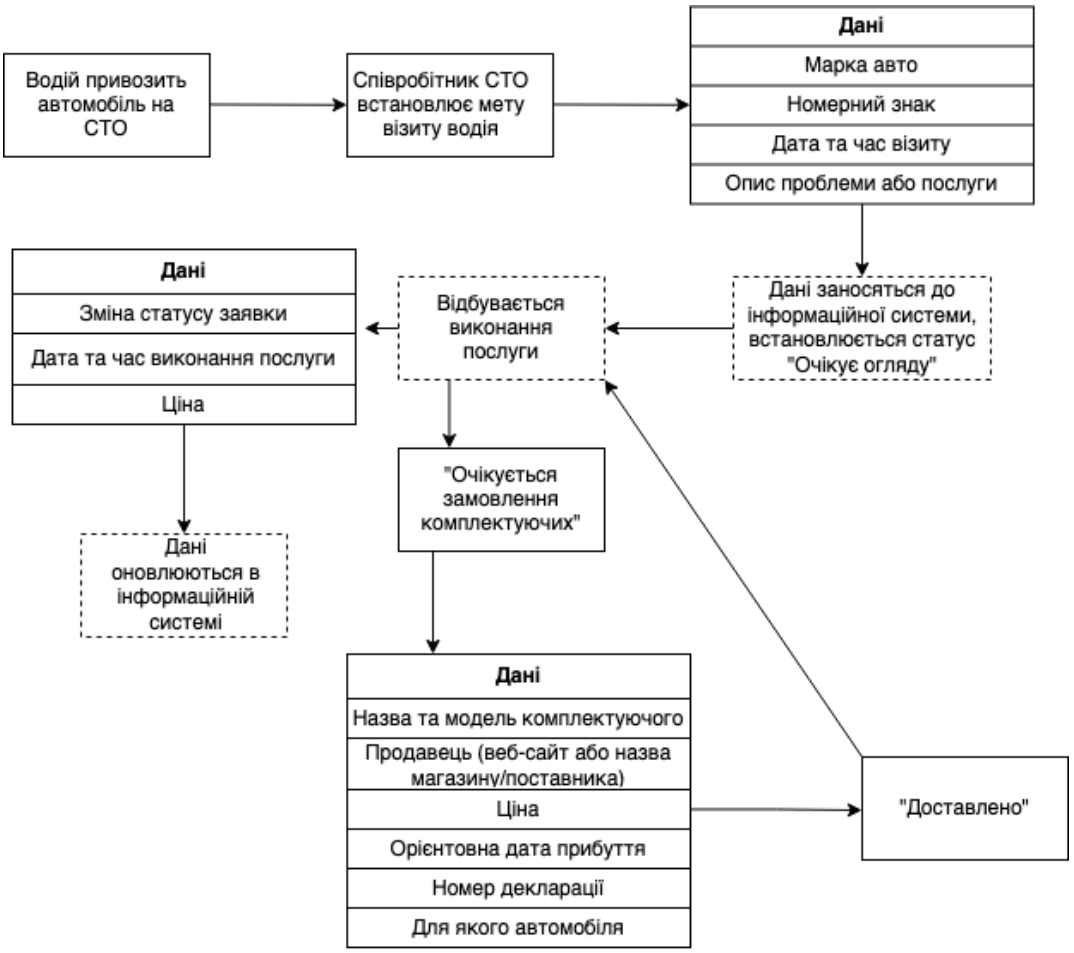

Рис.2.2 – Модель процесу керування заявками на СТО разом з замовленням потрібних комплектуючих

Дані, які заноситимуться в ІС про модель:

- Назва та модель комплектуючого;
- Продавець (веб-сайт, назва магазину або поставника);
- Кількість;
- Ціна;
- Орієнтовна дата прибуття;
- Для якого замовлення;
- Номер декларації доставки;
- Статус доставки;

Статуси доставки будуть такими:

- «Очікується відправлення»

- «Відправлено»
- «Доставлено до місця призначення»

Завершивши створення концептуальної моделі ІС, переходимо до проектування бази даних системи.

2.2 Проектування бази даних системи

Опираючись на концептуальну модель ІС, було спроектовано базу даних, яка буде основою інформаційної системи. Схему таблиць бази даних представлено на рис.2.2. Вона містить в собі такі таблиці, як applications («Заявки»), applicationtypes («Типи послуг»), applicationstatus («Статуси заявок»), detailsstatus («Статуси доставки деталей») та detailsforapps («Деталі для заявки»).

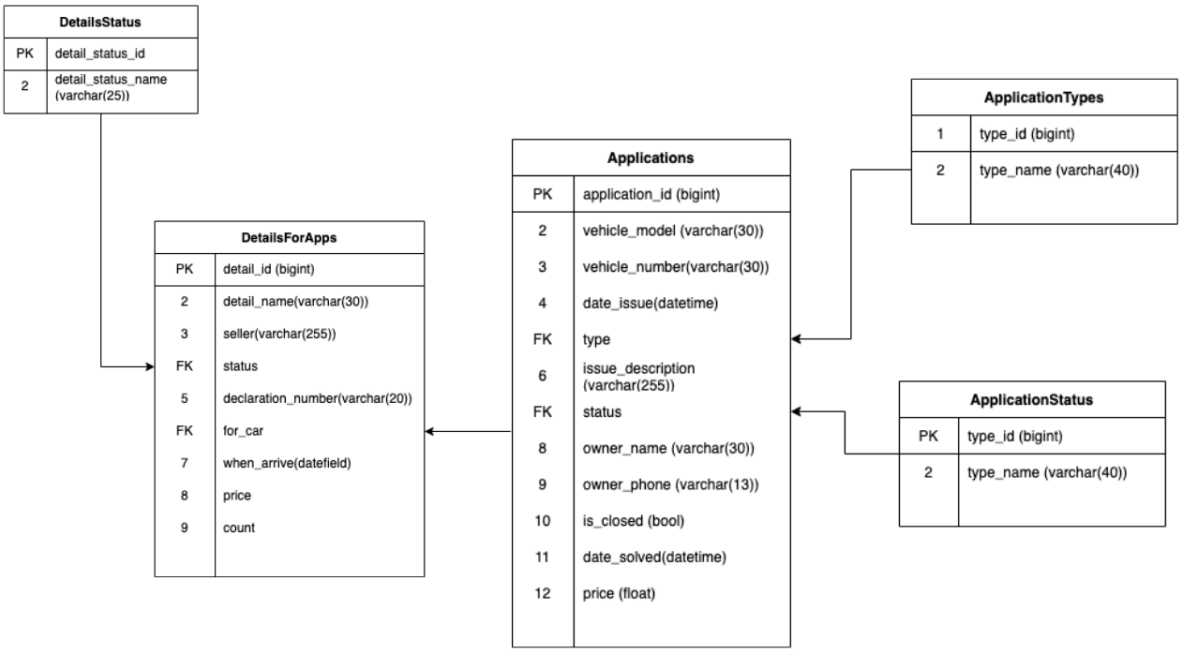

Рис.2.2 Схема бази даних інформаційної системи керування заявками станції технічного обслуговування

У базі даних для інформаційних систем буде 4 зовнішніх ключі, а саме:

- Type у таблиці Applications, який посилатиметься на обрану комірку з ApplicationTypes;
- Status у таблиці Applications, який посилатиметься на обрану комірку з ApplicationStatus;
- Status у таблиці DetailsForApps, який посилатиметься на обрану комірку з DetailStatus;
- For_car у таблиці DetailsForApps, що посилатиметься на обрану комірку з Applications;

Це зроблено для того, щоб в ІС співробітник у декілька простих кроків міг обрати статус виконання послуги, тип, а також статус доставки комплектуючих та замовлення, для якого йдуть ці комплектуючі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Вибір засобів програмної реалізації

Розглянемо обрані засоби для програмної реалізації інформаційної системи керування заявками.

Python – мова програмування, що є інтерпретованою та об’єктно-орієнтованою високого рівня зі строгою динамічною типізацією. Завдяки синтаксису даної мови програмування, вона широко використовується не лише в різноманітних видах розробки, а також у автоматизації рутинних процесів.

В даному проекті ми використовуватимемо Python для розробки клієнтської частини системи, для цього використаємо її веб-фреймворк Django.

Django – високорівнений веб-фреймворк, що базується на мові Python, що дозволяє будувати сайти з однієї або декількох частин, які рекомендується робити модульними. Django дозволяє швидко розробити потрібний веб-застосунок та забезпечує просте масштабування, якщо в цьому є потреба.

Django має в собі готовий каркас для майбутнього веб-застосунку, а популярність цієї мови забезпечує користувачів численною кількістю плагінів, які також спрощують розробку сайту.

За замовчуванням, Django працює з СУБД SQLite, яка також використовуватиметься на проекті.

SQLite – компактна реляційна СУБД, що втілена у вигляді бібліотеки та має в собі реалізації стандарту SQL-92.

SQLite можна використовувати для тестування або компактних застосунків, в чому допомагає Django, адже завдяки вбудованому Object-Relational Mapping (ORM), моделювати базу даних та додавати функції для роботи з нею можна не відходячи від синтаксису Python.

3.2 Розробка клієнтської частини системи

Для початку роботи з Django, ми повинні мати на комп’ютері встановлений Python => 3.6 та активоване віртуальне середовище.

Якщо ці кроки виконані, у віртуальному середовищі ми прописуємо такий набір команд:

- pip3 install Django – встановлення фреймворку у віртуальному середовищі
- django-admin startproject sto_system – автоматичне створення майбутнього проекту
- python manage.py startapp sto_cms – створення основного модуля проекту

Коли ми виконали ці кроки, не варто забути додати назву нашого застосунку sto_cms, який ми створили, у пункт INSTALLED_APPS файлу settings.py, що знаходиться у директорії sto_system. Це потрібно для того, щоб Django при запуску сервера запустив усі модулі, які ви створили. Після виконання цього кроку, фрагмент коду виглядатиме так:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'sto_cms'
]
```

Щоб створити базу даних, а також забезпечити безперебійний зв’язок між застосунком і базою даних. за допомогою класів у файлі models.py, що знаходиться за директорією sto_cms ми прописуємо моделі наших таблиць у БД, спираючись на схему бази даних.

Моделі виглядатимуть наступним чином:

```
from django.db import models
```

```
class ApplicationTypes(models.Model):
    type_id = models.BigAutoField(primary_key=True, null=False)
    type_name = models.CharField(max_length=40, null=False)


class ApplicationStatus(models.Model):
    status_id = models.BigAutoField(primary_key=True, null=False)
    status_name = models.CharField(max_length=25, null=False)

class DetailStatus(models.Model):
    detail_status_id = models.BigAutoField(primary_key=True, null=False)
    detail_status_name = models.CharField(max_length=25, null=False)


class Applications(models.Model):
    application_id = models.BigAutoField(primary_key=True, null=False)
    vehicle_model = models.CharField(max_length=30)
    vehicle_number = models.CharField(max_length=30)
    date_issue = models.DateTimeField(null=False)
    type = models.ForeignKey(ApplicationTypes, on_delete=models.CASCADE)
    issue_description = models.CharField(max_length=255)
    status = models.ForeignKey(ApplicationStatus, on_delete=models.CASCADE)
    owner_name = models.CharField(max_length=30, null=False)
    owner_phone = models.CharField(max_length=13, null=False)
    is_closed = models.BooleanField(null=False, default=False)
    date_solved = models.DateTimeField(null=True, blank=True)
    price = models.DecimalField(max_digits=10, decimal_places=2, null=True,
blank=True)

class DetailsForApps(models.Model):
    detail_id = models.BigAutoField(primary_key=True, null=False)
    detail_name = models.CharField(max_length=30, null=False)
    seller = models.CharField(max_length=255, null=False)
    status = models.ForeignKey(DetailStatus, on_delete=models.CASCADE)
    declaration_number = models.CharField(max_length=20)
    for_car = models.ForeignKey(Applications, on_delete=models.CASCADE)
    when_arrive = models.DateTimeField()
    price = models.DecimalField(max_digits=10, decimal_places=2, default=1)
    count = models.IntegerField(default=1)
```

Після того, як ми окреслили моделі нашої майбутньої бази даних, ми виконуємо дві команди у директорії нашого проекту:

- python manage.py makemigrations – автоматично генерує вихідний код для створення або редагування нашої бази даних на основі моделей, які ми вже прописали;

- python manage.py migrate – автоматично створює або редагує базу даних згідно утворених міграцій;

Так як у цьому проекті ми працюємо з SQLite, то після виконання другої команди з'являється файл db.sqlite3 у директорії нашого проекту.

Так як Django з самого початку має інструментарій для створення базових шаблонів, нам достатньо знати як поєднати нашу базу даних та комірки. Для цього ми редагуємо файл admin.py і тепер вихідний код виглядатиме так:

```
from django.contrib import admin
from sto_cms.models import Applications, ApplicationStatus, ApplicationTypes,
DetailStatus, DetailsForApps


@admin.register(Applications)
class ApplicationsAdmin(admin.ModelAdmin):
    list_display = ("vehicle_model",
                    "vehicle_number",
                    "type",
                    "status",
                    "date_issue",
                    "is_closed",
                    "date_solved",
                    "price")


@admin.register(ApplicationStatus)
class ApplicationStatusAdmin(admin.ModelAdmin):
    list_display = ("status_id",
                    "status_name")


@admin.register(ApplicationTypes)
class ApplicationTypesAdmin(admin.ModelAdmin):
    list_display = ("type_id",
                    "type_name")


@admin.register(DetailStatus)
class DetailStatusAdmin(admin.ModelAdmin):
    list_display = ("detail_status_id",
```

```
                    "detail_status_name")


@admin.register(DetailsForApps)
class DetailsForAppsAdmin(admin.ModelAdmin):
    list_display = ("detail_id",
                    "detail_name",
                    "seller",
                    "status",
                    "declaration_number",
                    "for_car",
                    "when_arrive",
                    "price",
                    "count")
```

Що в цьому випадку відбувається? Розглянемо покроково вихідний код, а саме список імпортованого коду та одну з функцій.

```
from django.contrib import admin
```

Ми імпортуємо модуль для адміністраторської панелі admin, який вже вбудований у Django.

```
from sto_cms.models import Applications, ApplicationStatus, ApplicationTypes,
DetailStatus, DetailsForApps
```

Ми імпортуємо класи моделей, які ми самотужки створили у файлі models.py

```
@admin.register(Applications)
```

В даному випадку використаний декоратор @admin.

Декоратор – це структурний паттерн проектування, що надає можливість динамічно додавати об’єктам нову функціональність [8]

В цьому випадку з декоратору використовується функція register, яка зареєструє нашу модель з БД.

```
class ApplicationsAdmin(admin.ModelAdmin):
    list_display = ("vehicle_model",
                    "vehicle_number",
                    "type",
                    "status",
                    "date_issue",
                    "is_closed",
                    "date_solved",
                    "price")
```

Ми оголошуємо клас ApplicationsAdmin, який наслідується від admin.ModelAdmin.

Після цього ми прописуємо назви тих комірок, які хочемо бачити у загальному списку.

Для того, щоб отримати доступ до нашого сайту, ми повинні створити користувача. Django також дозволяє зробити це у декілька кроків, за допомогою команди python manage.py createsuperuser, яку прописуємо у терміналі в директорії нашого проекту.

Після виконання команди, в нас запитується спочатку логін, потім e-mail, а потім пароль. В нашому випадку це буде зв’язка admin:admin, адже для тестування повинні бути якомога простіші дані для входу.

Таким чином, ми створили базову інформаційну систему керування заявок на станції технічного обслуговування. Тепер переходимо до тестування.

3.3. Тестування інформаційної системи керування заявками СТО

У терміналі переходимо до директорії нашого проекту і запускаємо нашу систему наступною командою:

- python manage.py runserver

Якщо ми не допустили помилок у коді, то нас зустрічає наступне:

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 1, 2021 - 15:46:19
Django version 3.2.4, using settings 'sto_system.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

З цього ми можемо зрозуміти, що система запустилася успішно без будь-яких помилок, версію Django що використовується на проекті, а також за якою адресою та на якому порті знаходиться наш проект.

Ми заходимо у браузер, переходимо за посиланням яке вказано у повідомленні вище та бачимо привітальну сторінку, яка вказує на те, що все прошло успішно.

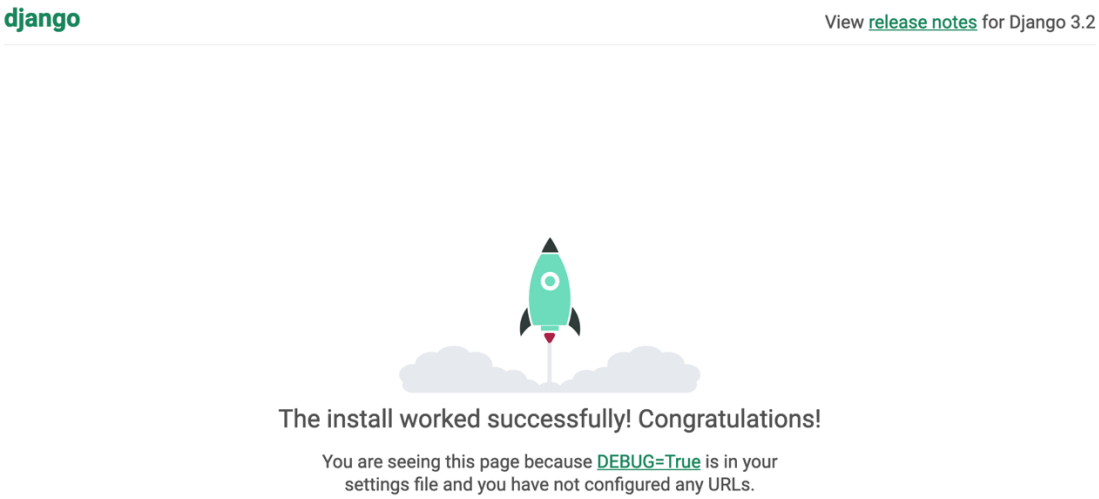

Рис.3.1 – привітальна сторінка з повідомленням про успішний запуск проекту на Django

Для того, щоб потрапити до нашої системи, потрібно додати до нашого посилання /admin. Нас зустрічає сторінка, в якій ми повинні ввести свій логін та пароль. В нашому випадку це admin:admin.

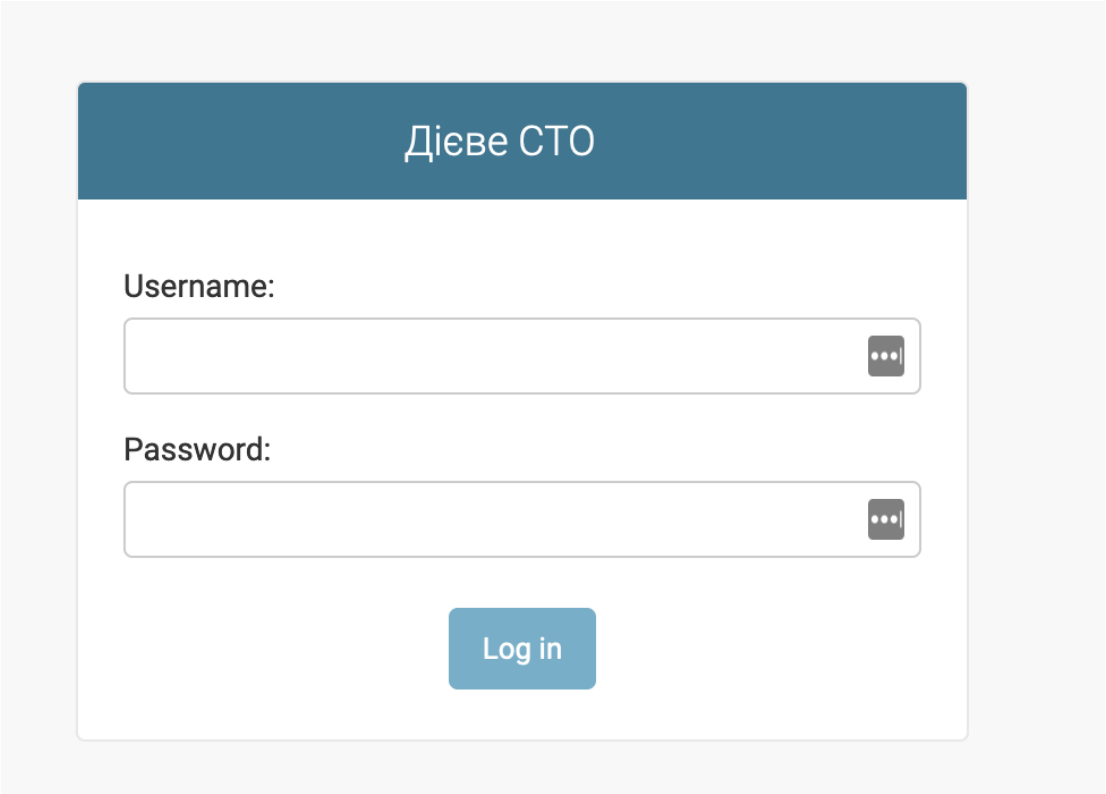

Рис.3.2. – сторінка входу у систему «Дієве СТО»

Коли ми натискаємо на кнопку «Log in», нас перенаправляє на сторінку системи, де ми бачимо:

- «шапку» сайту, яка показує назву системи, хто зараз у системі, кнопка «Зміни пароля» та кнопка виходу;
- Блок з таблицями у нашій БД – «Authentication and authorization» (таблиці з користувачами системи) та «sto_cms» (таблиці нашої системи, яка зберігає інформацію про заявки, матеріали тощо);
- Блок Recent Actions («Останні дії»), що показує детальний розбір того, що було нещодавно змінено або додано;

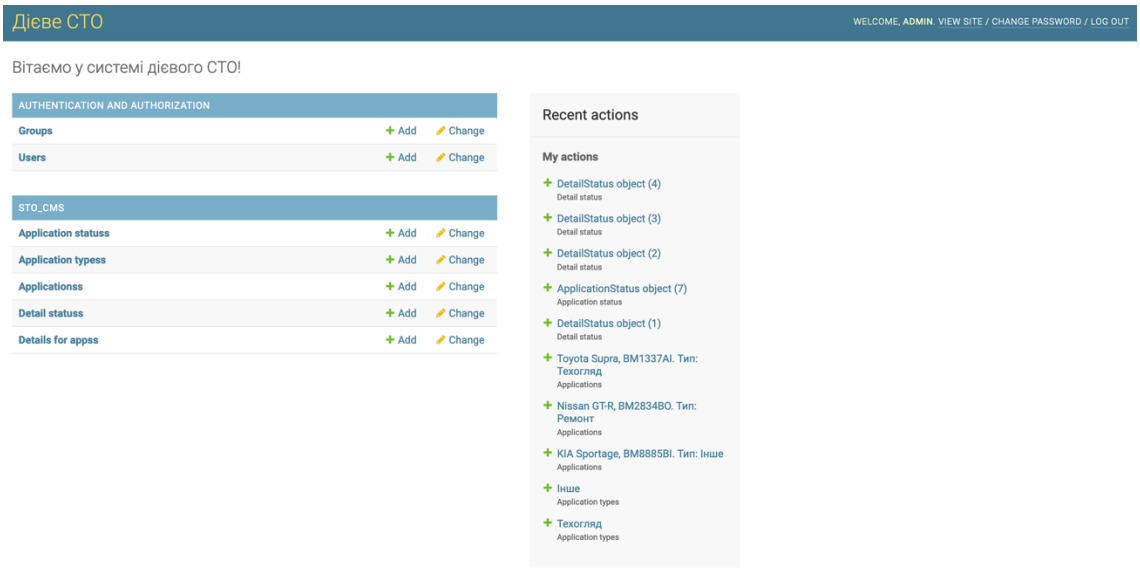

Рис.3.4 – головна сторінка системи «Дієве СТО»

Заповнимо такі таблиці, як Application status, application types та details status. Для цього використаємо ті дані, які вже були описані при створенні концептуальної моделі.

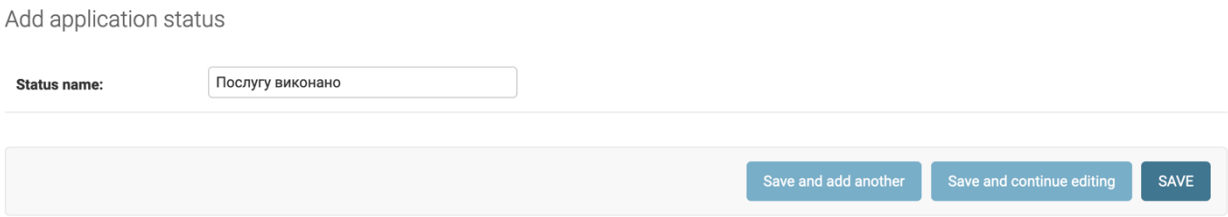

Рис.3.4. – приклад заповнення однієї з таблиць «Application status».

Натискаємо на кнопку Save and add another, щоб додати інші статуси. Результати заповнення даними можна переглянути на рис.3.5, рис.3.6, рис.3.7.

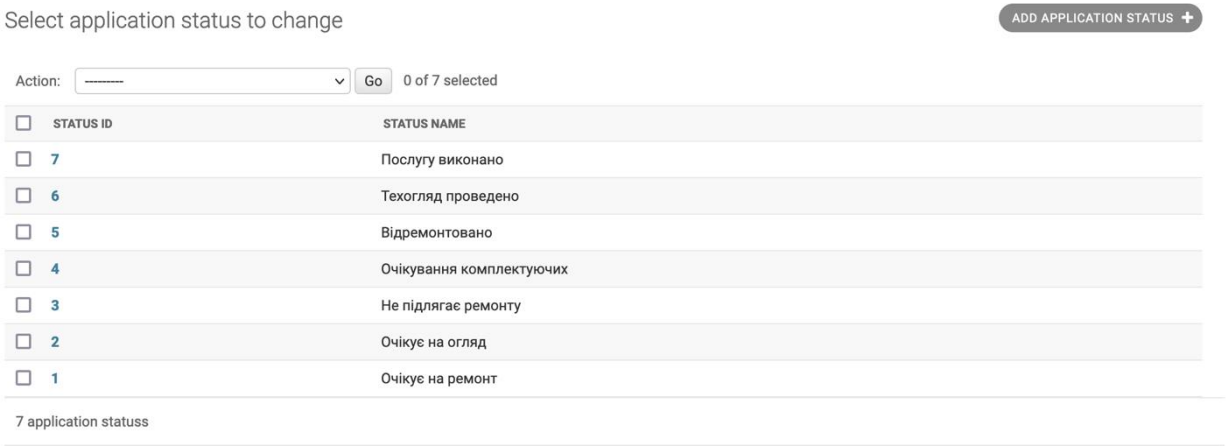

Рис.3.5. – заповнена таблиця зі статусами виконання послуги

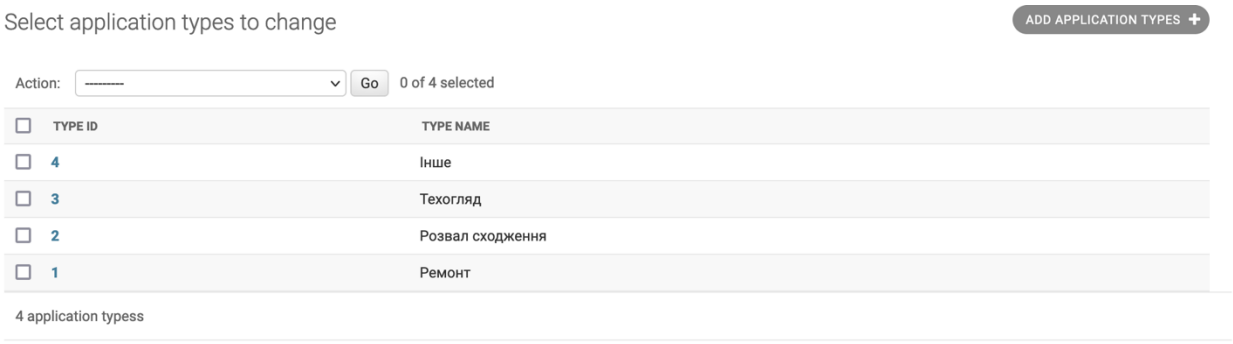

Рис.3.6 – заповнена таблиця з типами послуг

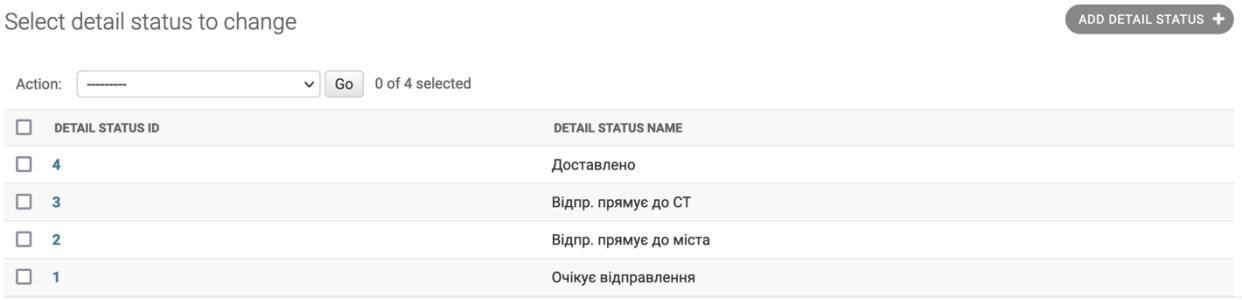

Рис.3.7 – заповнена таблиця зі статусами доставки комплектуючих

Для того, щоб перевірити працездатність таблиць із заявками та комплектуючими, ми також повинні заповнити їх потрібними даними.

Переходимо на головну сторінку та на пункті «Applications» обираємо кнопку «Add». Нас зустрічає форма, яка представлена на рис.3.8.

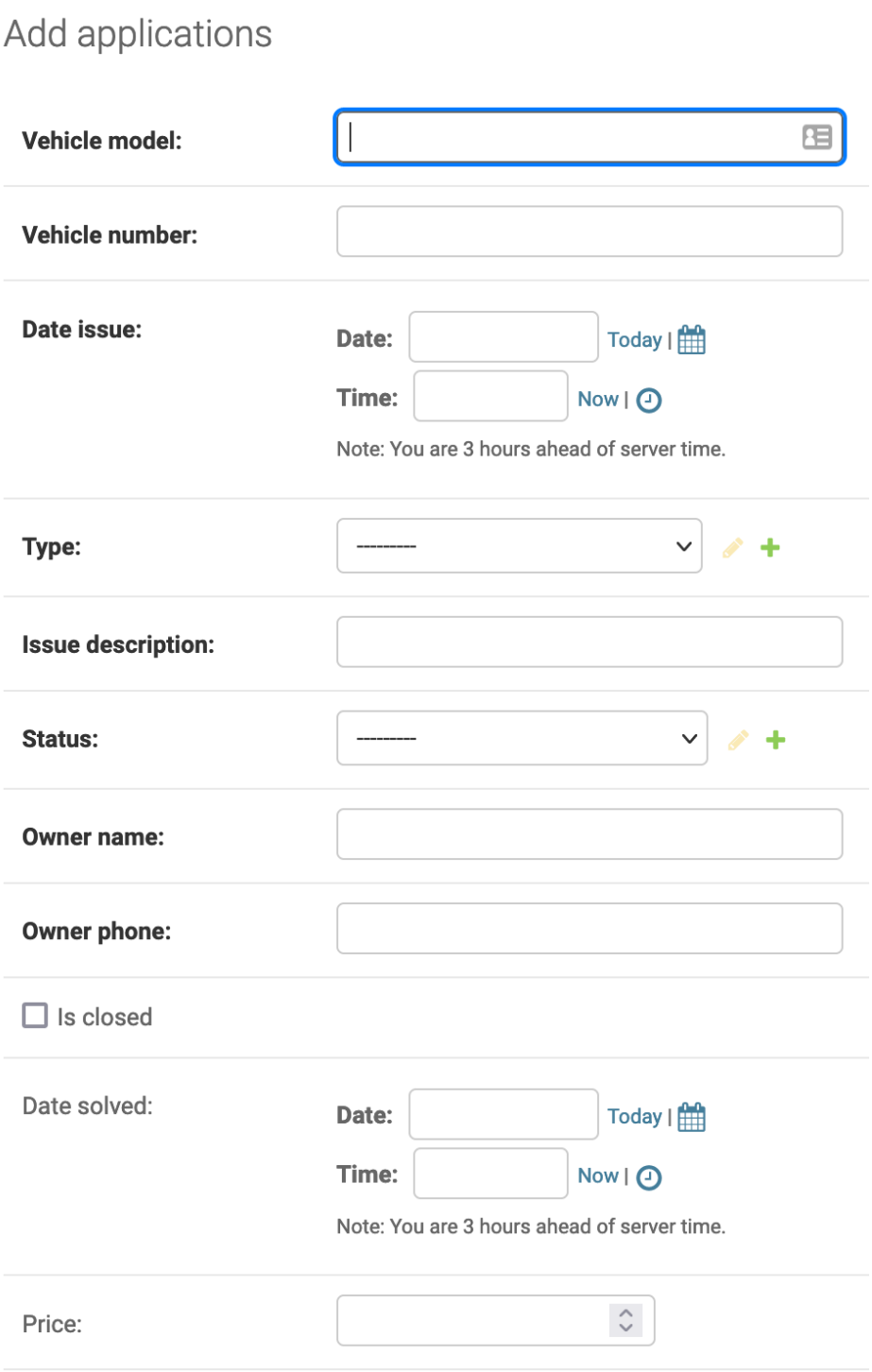

Рис.3.8 – форма заповнення заявки

Варто зазначити, що для полів Date Solved («Дата, коли ремонт завершено») та Price («Ціна»), застосовано параметр blank=True, що дозволяє залишити поля пустими. Це є логічним, адже дата виконання послуги та ціна виставляються вже по факту. Окрім цього, створений об’єкт можна буде без

проблем змінювати, адже кожного разу потрібно оновлювати статус виконання послуги.

Після заповнення різноманітними даними, результат можна переглянути на рис.3.9.

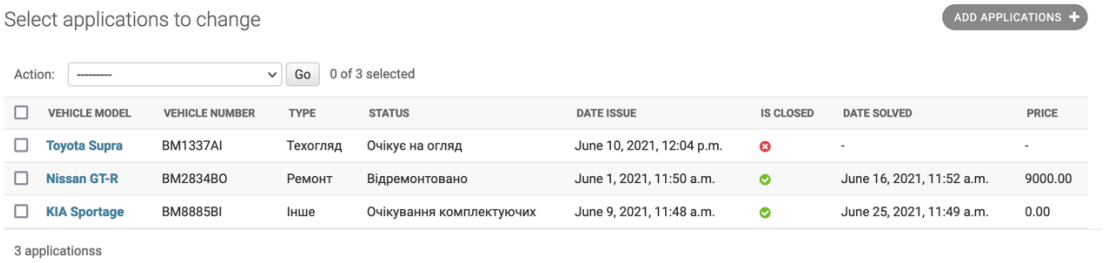

Рис.3.9 – таблиця заявок після заповнення тестовими даними

Як бачимо, заповнення та відображення даних відбувається коректно.

Продовжуємо заповнення даними, але вже іншої таблиці – DetailsForApps («Комплектуючі для заявок»). На рис.3.10 представлена форма заповнення.

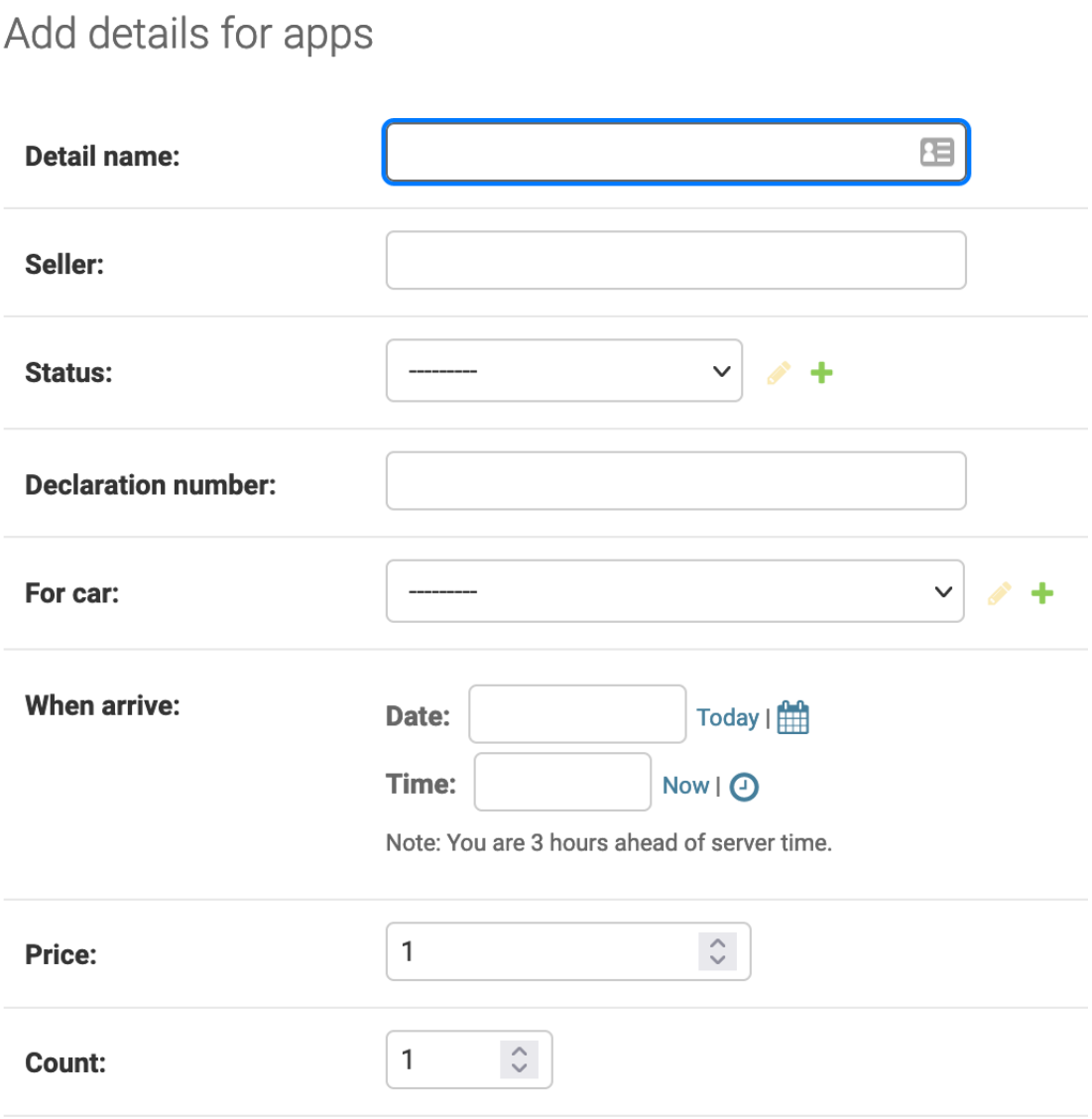

Рис.3.10 – форма заповнення даних про деталь, яку було замовлено

На рис.3.11 представлено результат заповнення даними.

DETAIL NAME	SELLER	STATUS	DECLARATION NUMBER	FOR CAR	WHEN ARRIVE	PRICE	COUNT
Пр.фари для Nissan GT-R	https://www.avtoradosti.com.ua/p/160566.html	Доставлено	72352352362782372344	Nissan GT-R, BM2834BO. Тип: Ремонт	June 14, 2021, 4:53 p.m.	11375.00	2
фарба для авто металік Mixon	https://prom.ua/p617887796-aerozolnaya-kraska-dlya.html?	Очікує відправлення	72532672832832642426	KIA Sportage, BM8885BI. Тип: Інше	June 30, 2021, 6 p.m.	140.00	5
Літні покришки Gislaved Ultra	https://infoshina.com.ua/gislaved-ultra-speed-2-205-55-r16-91v	Очікує відправлення	11648384348726326234	Toyota Supra, BM1337AI. Тип: Техогляд	June 23, 2021, 4:49 p.m.	1359.00	1

Рис.3.11 – заповнена таблиця DetailsForApps («Деталі для заявок»).

Як бачимо, відображення та заповнення даними також працює коректно.

Виходячи з результатів тестування, ми бачимо що система має:

- мінімалістичний інтуїтивний інтерфейс;
- коректно працюючі зовнішні ключі, за рахунок яких у декілька кроків обираються потрібні дані для створення нового об’єкта, а відповідно ми бачимо що до чого має відношення;
- швидкодія за рахунок простого інтерфейсу;

ВИСНОВКИ

В роботі було розв'язано задачу розробки інформаційної системи керування заявками станції технічного обслуговування. Для розв'язання такої задачі було виконано такі завдання:

1) Створено концептуальну модель інформаційної системи;

2) Визначено структуру бази даних інформаційної системи;

3) Проведено вибір засобів програмної реалізації інформаційної системи;

4) Розроблено і програмно реалізовано інформаційну систему керування заявками СТО, що можна використовувати для малого бізнесу;

Надалі планується масштабування системи, а саме:

- автоматичний підрахунок ціни згідно кількості замовлених комплектуючих;
- автоматична зміна статусу після доставки деталей;
- створення інших ролей у інформаційній системі, які дозволятимуть не відходячи від робочого місця майстру надати інформацію про зміну статуса;
- розмежування інформаційної системи по підрозділах СТО, що дозволить використовувати цей продукт і на середньомасштабній СТО;

СПИСОК ЛІТЕРАТУРИ

1. Connolly R., Hoar R. Fundamentals of Web Development, 2nd Edition. – Pearson Education, 2018. – 1248 p.
2. ПТ-Авто 8.3 – офіційний веб-сайт [Електронний ресурс] – режим доступу: http://pt-auto.com.ua
3. Ремонлайн – офіційний веб-сайт [Електронний ресурс] – режим доступу: https://remonline.ua/ru/autoservice/
4. Універсальна система обліку – система управління на СТО [Електронний ресурс] – режим доступу: http://usu.kz/sistema_upravleniya_na_sto.php
5. Python 3.9.5 – Документація [Електронний ресурс] – режим доступу: https://docs.python.org/3/
6. Django 3.2.4 – Документація [Електронний ресурс] – режим доступу: https://docs.djangoproject.com/en/3.2/releases/3.2.4/
7. SQLite – офіційний веб-сайт [Електронний ресурс] – режим доступу: https://www.sqlite.org/index.html
8. Декоратори – Refactoring Guru [Електронний ресурс] – режим доступу: https://refactoring.guru/uk/design-patterns/decorator

Додаток А

Вихідний код проекту

sto_system

settings.py

```
import os
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-
@j403=o9#ek0q#v$iz1j%=a13#zoi4kx6_ftxegply*m8fjk8='

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'sto_cms'
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
```

```
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'sto_system.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')]
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'sto_system.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}


# Password validation
# https://docs.djangoproject.com/en/3.2/ref/settings/#auth-password-
validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
```

```
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/3.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.2/howto/static-files/

STATIC_URL = '/static/'

# Default primary key field type
# https://docs.djangoproject.com/en/3.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Urls.py

```
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),
]

admin.site.site_header = "Дієве СТО"
admin.site.site_title = "Дієве СТО"
admin.site.index_title = "Вітаємо у системі дієвого СТО!"
```

Wsgi.py

```
import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'sto_system.settings')

application = get_wsgi_application()
```

Asgi.py

```
import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'sto_system.settings')

application = get_asgi_application()
```

sto_cms

models.py

```
from django.db import models


class ApplicationTypes(models.Model):
    type_id = models.BigAutoField(primary_key=True, null=False)
    type_name = models.CharField(max_length=40, null=False)

    def __str__(self):
        return f"{self.type_name}"


class ApplicationStatus(models.Model):
    status_id = models.BigAutoField(primary_key=True, null=False)
    status_name = models.CharField(max_length=25, null=False)

    def __str__(self):
        return f"{self.status_name}"


class DetailStatus(models.Model):
    detail_status_id = models.BigAutoField(primary_key=True, null=False)
    detail_status_name = models.CharField(max_length=25, null=False)

    def __str__(self):
```

```
        return f"{self.detail_status_name}"


class Applications(models.Model):
    application_id = models.BigAutoField(primary_key=True, null=False)
    vehicle_model = models.CharField(max_length=30)
    vehicle_number = models.CharField(max_length=30)
    date_issue = models.DateTimeField(null=False)
    type = models.ForeignKey(ApplicationTypes, on_delete=models.CASCADE)
    issue_description = models.CharField(max_length=255)
    status = models.ForeignKey(ApplicationStatus, on_delete=models.CASCADE)
    owner_name = models.CharField(max_length=30, null=False)
    owner_phone = models.CharField(max_length=13, null=False)
    is_closed = models.BooleanField(null=False, default=False)
    date_solved = models.DateTimeField(null=True, blank=True)
    price = models.DecimalField(max_digits=10, decimal_places=2, null=True,
blank=True)

    def __str__(self):
        return f"{self.vehicle_model}, {self.vehicle_number}. Тип:
{self.type}"


class DetailsForApps(models.Model):
    detail_id = models.BigAutoField(primary_key=True, null=False)
    detail_name = models.CharField(max_length=30, null=False)
    seller = models.CharField(max_length=255, null=False)
    status = models.ForeignKey(DetailStatus, on_delete=models.CASCADE)
    declaration_number = models.CharField(max_length=20)
    for_car = models.ForeignKey(Applications, on_delete=models.CASCADE)
    when_arrive = models.DateTimeField()
    price = models.DecimalField(max_digits=10, decimal_places=2, default=1)
    count = models.IntegerField(default=1)

    def __str__(self):
        return f"{self.detail_id}, {self.detail_name}, Прибуде:
{self.when_arrive}"
```

apps.py

```
from django.apps import AppConfig


class StoCmsConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'sto_cms'
```

admin.py

```
from django.contrib import admin
from sto_cms.models import Applications, ApplicationStatus, ApplicationTypes,
DetailStatus, DetailsForApps


@admin.register(Applications)
class ApplicationsAdmin(admin.ModelAdmin):
    list_display = ("vehicle_model",
                    "vehicle_number",
                    "type",
                    "status",
                    "date_issue",
                    "is_closed",
                    "date_solved",
                    "price")


@admin.register(ApplicationStatus)
class ApplicationStatusAdmin(admin.ModelAdmin):
    list_display = ("status_id",
                    "status_name")


@admin.register(ApplicationTypes)
class ApplicationTypesAdmin(admin.ModelAdmin):
    list_display = ("type_id",
                    "type_name")


@admin.register(DetailStatus)
class DetailStatusAdmin(admin.ModelAdmin):
    list_display = ("detail_status_id",
                    "detail_status_name")


@admin.register(DetailsForApps)
class DetailsForAppsAdmin(admin.ModelAdmin):
    list_display = ("detail_id",
                    "detail_name",
                    "seller",
                    "status",
                    "declaration_number",
                    "for_car",
                    "when_arrive",
                    "price",
                    "count")
```

Manage.py

```
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'sto_system.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```