Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ
НА ТЕМУ:
Система контроля работы дорожных камер

Студент ИУ7-64Б _____ Новиков М. Р.
(Группа) (Подпись, дата) (Фамилия И. О.)

Руководитель курсовой работы _____ Тассов К. Л.
(Подпись, дата) (Фамилия И. О.)

2020 г.

Содержание

Введение

В настоящее время сеть автомобильных дорог в России стремительно расширяется: увеличивается как протяженность, так и пропускная способность, открываются всё новые и новые трассы [1]. Одним из основных инструментов контроля автомобильного потока являются камеры безопасности дорожного движения [2]. Появившиеся еще в 1960-х годах, к XXI веку дорожные камеры получили широкое распространение и стали цифровыми. Современные дорожные камеры могут быть оснащены сетевым соединением, что позволяет ускорить процесс обработки данных, технического обслуживания и мониторинга.

С развитием компьютерной техники появилась возможность автоматизации множества процессов. Так, например, автоматизация процессов мониторинга и технического обслуживания камер безопасности дорожного движения позволит упростить процессы наблюдения за работоспособностью камер и сервисного обслуживания.

Целью данной курсовой работы являлась разработка приложения, обеспечивающего частичную автоматизацию процессов мониторинга и технического обслуживания камер безопасности дорожного движения.

Достижению цели способствовали следующие поставленные **задачи**:

- анализ предметной области, требований к базе данных и приложению в целом;
- проектирование базы данных и приложения;
- реализация приложения;
- тестирование приложения, исследование его стабильности и быстродействия.

1 Аналитический раздел

1.1 Описание предметной области

Использовать приложение для контроля работы дорожных камер будут сотрудники ГИБДД, а, следовательно, оно должно не требовать от пользователя специальных навыков, быть простым в установке и эксплуатации.

Выделим наиболее важные свойства и характеристики дорожных камер:

- местоположение;
- спецификации (например, тип камеры, производитель и пр.);
- текущее состояние камеры (например, исправное или неисправное).

Кроме того, для каждой камеры могут быть определены данные о сервисном обслуживании: название организации проводящей обслуживание, причины обслуживания, результаты обслуживания и пр.

1.2 Требования к приложению

Проанализировав предметную область курсовой работы, определим требования к приложению.

В качестве общей архитектурной концепции следует выбрать Web-приложение. Оно не привязывает пользователя к конкретной операционной системе, не требует установки и работает в привычной для обычного пользователя среде — браузере.

Для хранения информации о камерах следует использовать базу данных. В приложении должен быть предусмотрен простой механизм для

перехода на другую систему управления базами данных (далее — СУБД).

Так как хранящиеся данные могут отражать различные аспекты мониторинга и технического обслуживания, следует разделить пользователей приложения на группы с различными правами доступа. Каждая группа пользователей должна получать доступ к определенным разделам приложения. Регистрировать новых пользователей и предоставлять права доступа должен администратор приложения. Выделим группы пользователей:

- администраторы;
- аудиторы (занимаются мониторингом дорожных камер и оформлением заявок на сервисное обслуживание);
- юристы (занимаются проверкой документов, например договоров).

В приложении должен быть предусмотрен простой механизм регистрации новых пользователей и создания новых групп пользователей.

1.3 Логическая модель данных

Исходя из описания предметной области и требований к приложению можно сделать вывод, что следует выбрать реляционную модель данных.

1.3.1 Диаграмма «сущность — связь»

На рисунке 1.1 представлена диаграмма «сущность — связь».

1.4 Обзор существующих СУБД

Рассмотрим существующие реляционные системы управления базами данных.

1.4.1 MS SQL

Microsoft SQL Server разрабатывается корпорацией Microsoft с 1989 года. Основным языком запросов в этой СУБД является Transact-SQL — реализация стандарта ANSI/ISO по языку SQL с расширениями. Работает в

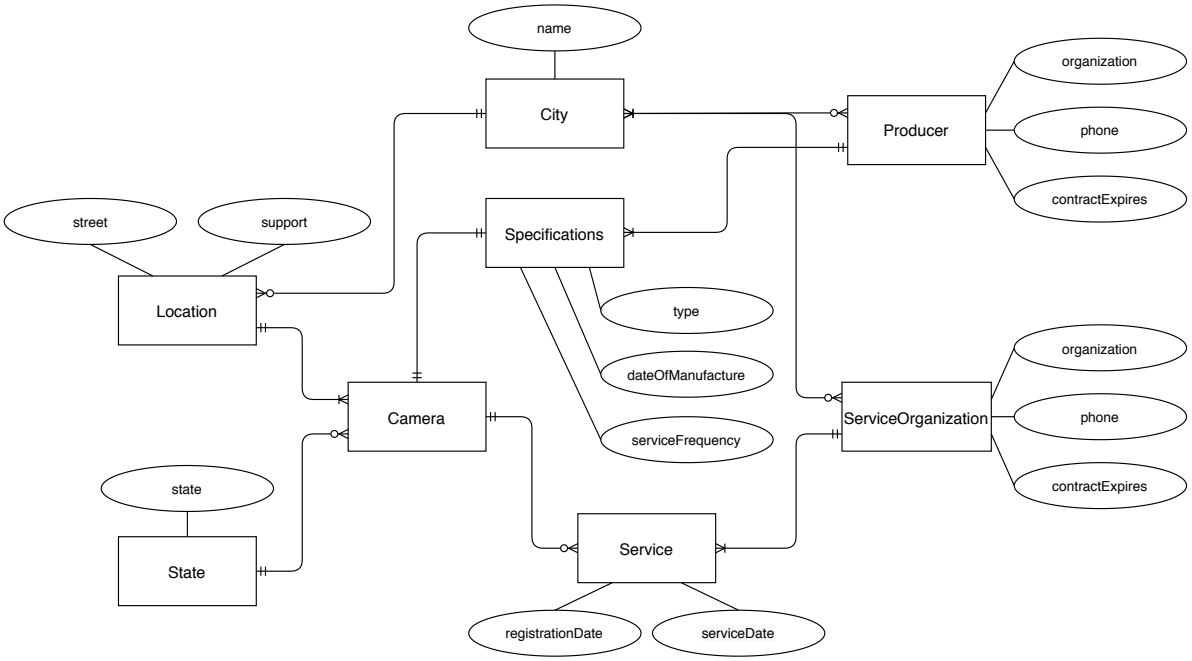

Рис. 1.1: ER-диаграмма

операционных системах UNIX, OS/2, Windows. Является одной из наиболее популярных СУБД и отличается высокой надежностью, производительностью и большим сообществом разработчиков. [3]

Преимущества и недостатки:

+ Легко маштабируемая СУБД.

+ Автоматизированные административные задачи: управление блокировкой, управление память и пр.

+ Поддержка большинства инструкций языка SQL.

+ Простая интеграция с другими продуктами Microsoft.

– Зависимость от операционной среды.

– Высокая цена.

1.4.2 MySQL

MySQL — свободная реляционная СУБД, разрабатываемая Oracle. Программный продукт распространяется по лицензии GNU General Public License и коммерческой лицензии. MySQL обеспечивает поддержку большого количества типов таблиц. [4]

Преимущества и недостатки:

+ Легко маштабируемая СУБД.
+ Высокая производительность за счет упрощения некоторых стандартов.
+ Поддержка большинства инструкций языка SQL.
+ Существуют версии, распространяемые по лицензии GNU.
− Заложены некоторые ограничения функционала, которые иногда необходимы в особо требовательных приложениях.
− Уступает другим СУБД по надежности.

1.4.3 PostgreSQL

Свободная система управления базами данных PostgreSQL существует для множества UNIX-подобных операционных систем и Microsoft Windows. Часто используется при разработке веб-сайтов. В отличии от других СУБД, поддерживает некоторые важные объектно-ориентированные и реляционные функции баз данных. [5]

Преимущества и недостатки:

+ Легко маштабируемая СУБД.
+ Поддержка множества сторонних библиотек, инструментов для проектирования и управления данными.
+ Поддержка большинства инструкций языка SQL.
+ Открытый исходный код.
− Производительность в некоторых задачах ниже, чем у MySQL.

1.4.4 SQLite

SQLite — это встраиваемая СУБД (библиотека). Используется преимущественно в небольших проектах и приложениях, по умолчанию встроена

в мобильные приложения на платформе Android. База данных использует для своей работы единственный файл и не предоставляет сетевого интерфейса. [6]

Преимущества и недостатки:

+ Повышение производительности за счет хранения базы данных на стороне клиента.

+ Хорошо переносимая, так как представляет из себя единственный файл.

+ Хорошо подходит для небольших приложений.

− Накладывает множество ограничений на разработчика.

Выводы

В аналитическом разделе определена задача и предметная область курсовой работы. Исходя из анализа предметной области определены требования к приложению, в том числе требования, направленные на упрощение процесса модификации. Определена логическая модель базы данных, построена диаграмма «сущность — связь». Рассмотрены наиболее популярные системы управления базами данных.

2 Конструкторский раздел

2.1 Проектирование базы данных

В аналитическом разделе расчетно-пояснительной записки на основе данных о предметной области была построена ER-диаграмма. Для проектирования базы данных преобразуем диаграмму «сущность — связь» в схему базы данных на основе реляционной модели данных.

Выделим следующие таблицы:

- Камера (Camera) — основная таблица базы данных, содержит внешние ключи к таблицам Местоположение, Спецификации, Состояние.
- Местоположение (Location) — таблица местоположений дорожных камер, содержит внешний ключ к таблице Город.
- Город (City) — таблица городов.
- Спецификации (Specification) — таблица технических данных о камерах, содержит внешний ключ к таблице Производитель.
- Производитель (Producer) — таблица производителей камер, содержит внешний ключ к таблице Город.
- Состояние (State) — таблица состояний камер.
- Сервис (Service) — таблица, хранящяя данные о сервисных обслуживаниях камер, содержит внешние ключи к таблицам Камера и Сервисная организация.
- Сервисная организация (ServiceOrganization) — таблица организаций, предоставляющих сервисное обслуживание камер, содержит внешний ключ к таблице Город.

На рисунке 2.1 представлена диаграмма базы данных.

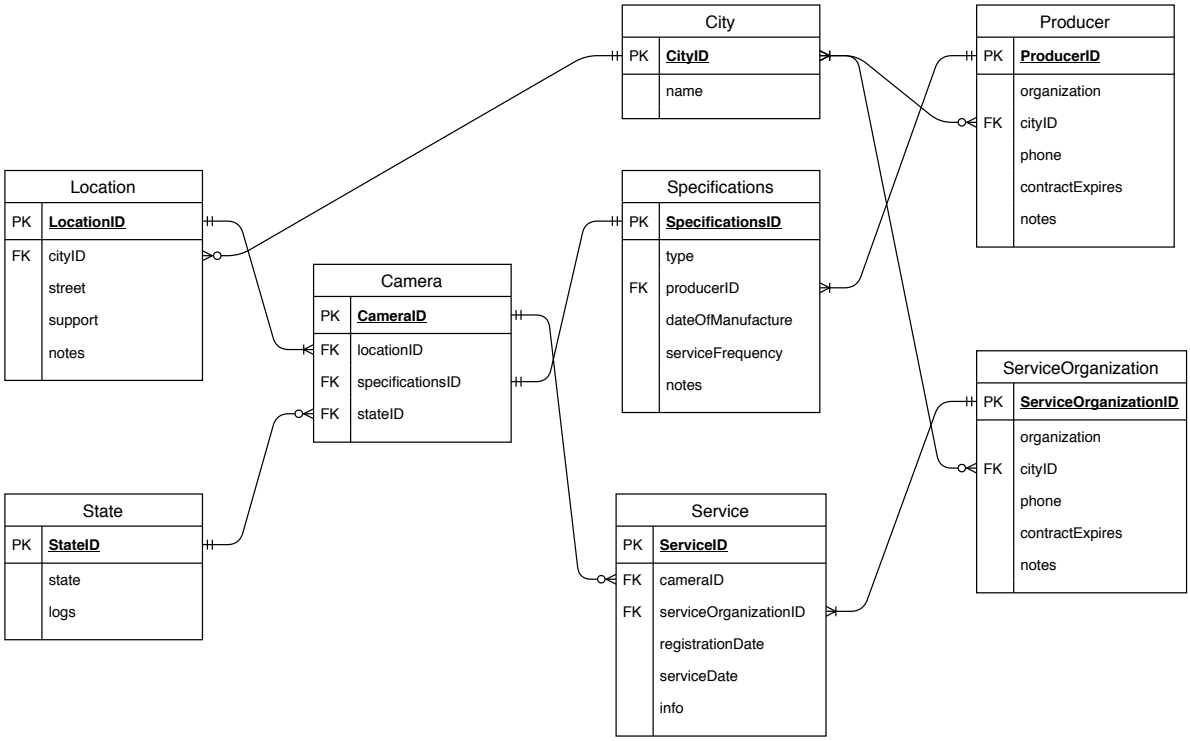

Рис. 2.1: Диаграмма базы данных

2.2 Архитектура приложения

Для проектирования приложения применим архитектурый шаблон проектирования Model-View-Controller (далее — MVC). [7]

2.2.1 Архитектурный шаблон MVC

MVC представляет из себя схему разделения данных и бизнес-логики приложения, пользовательского интерфейса и управляющей логики на три независимых компонента: модель, представление и контроллер. Такой подход позволяет изолировать данные и управляющую логику, независимо разрабатывать, тестировать, поддерживать и модифицировать компоненты. Схема шаблона MVC представлена на рисунке 2.2.

Модель

Модель представляет собой данные и методы для работы с данными. В модели выполняются запросы к базе данных, бизнес-логика. Этот компонент

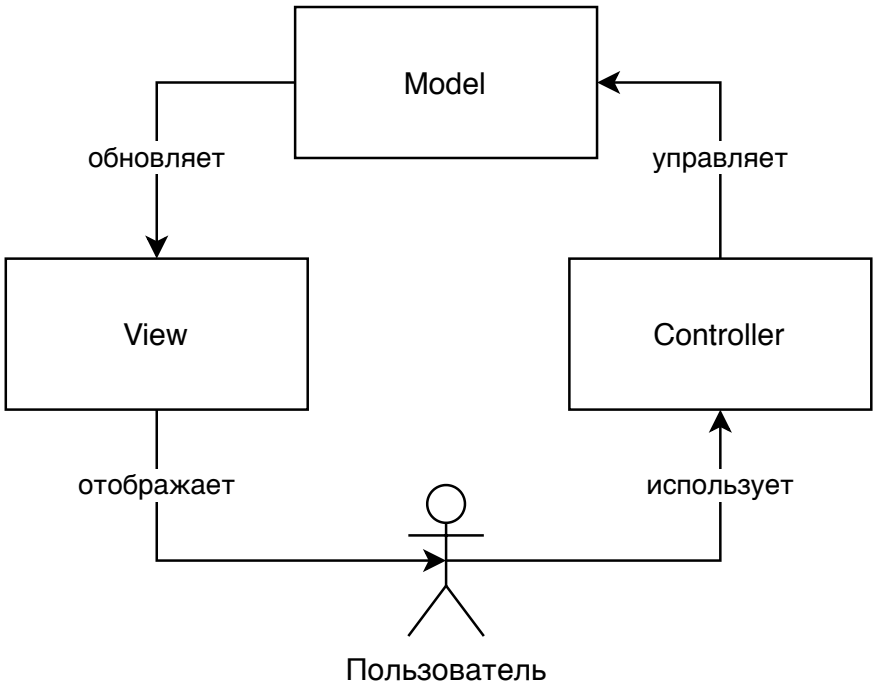

Рис. 2.2: Схема шаблона проектирования MVC

разрабатывается таким образом, чтобы отвечать на запросы контроллера, изменять свое внутреннее состояние и не зависеть от представлений.

Представление

Представление получает данные модели и отображает их пользователю. Представление не обрабатывает данные.

Контроллер

Контроллер является связующим компонентом — интерпретирует действия пользователя, оповещая модель об изменениях, которые необходимо внести.

2.2.2 UML-диаграммы классов

На рисунках 2.3–2.4 представлены UML-диаграммы классов приложения.

2.2.3 UML-диаграмма компонентов приложения

На рисунке 2.5 представлена UML-диаграмма модулей приложения.

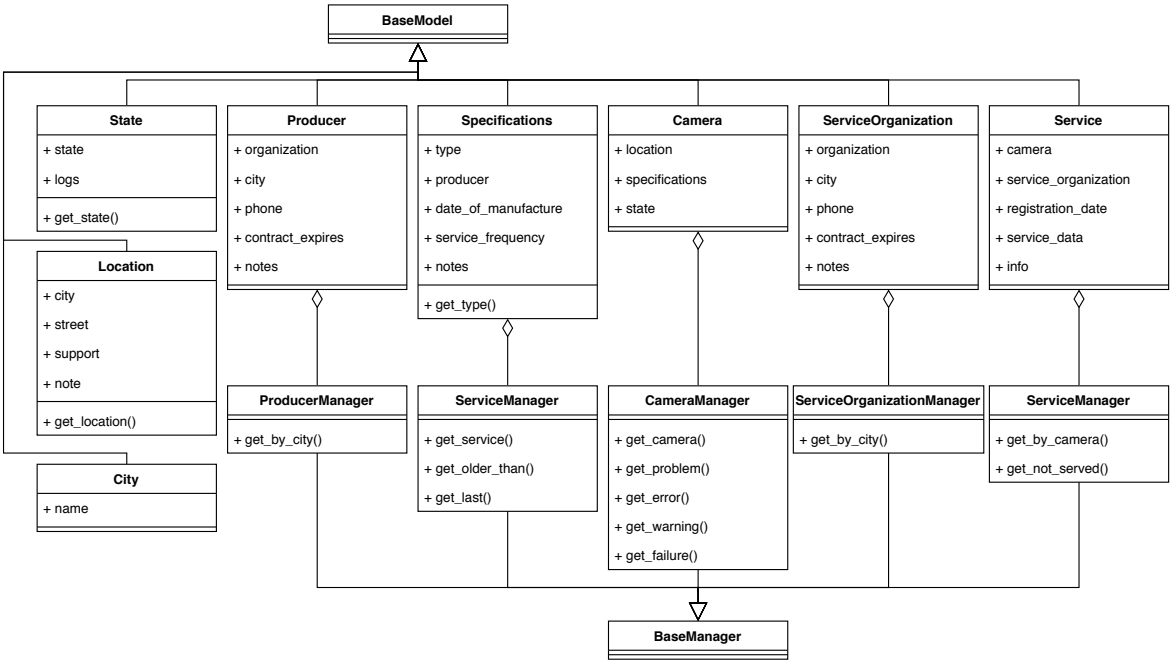

Рис. 2.3: UML-диаграмма классов `Model` и `Manager`

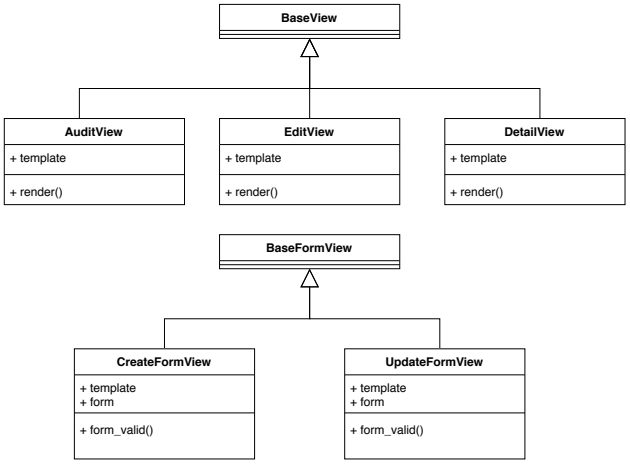

Рис. 2.4: UML-диаграмма классов `View` и `FormView`

Выводы

В конструкторском разделе была спроектирована база данных и архитектура приложения, построены ULM-диаграммы классов приложения и UML-диаграмма компонентов приложения.

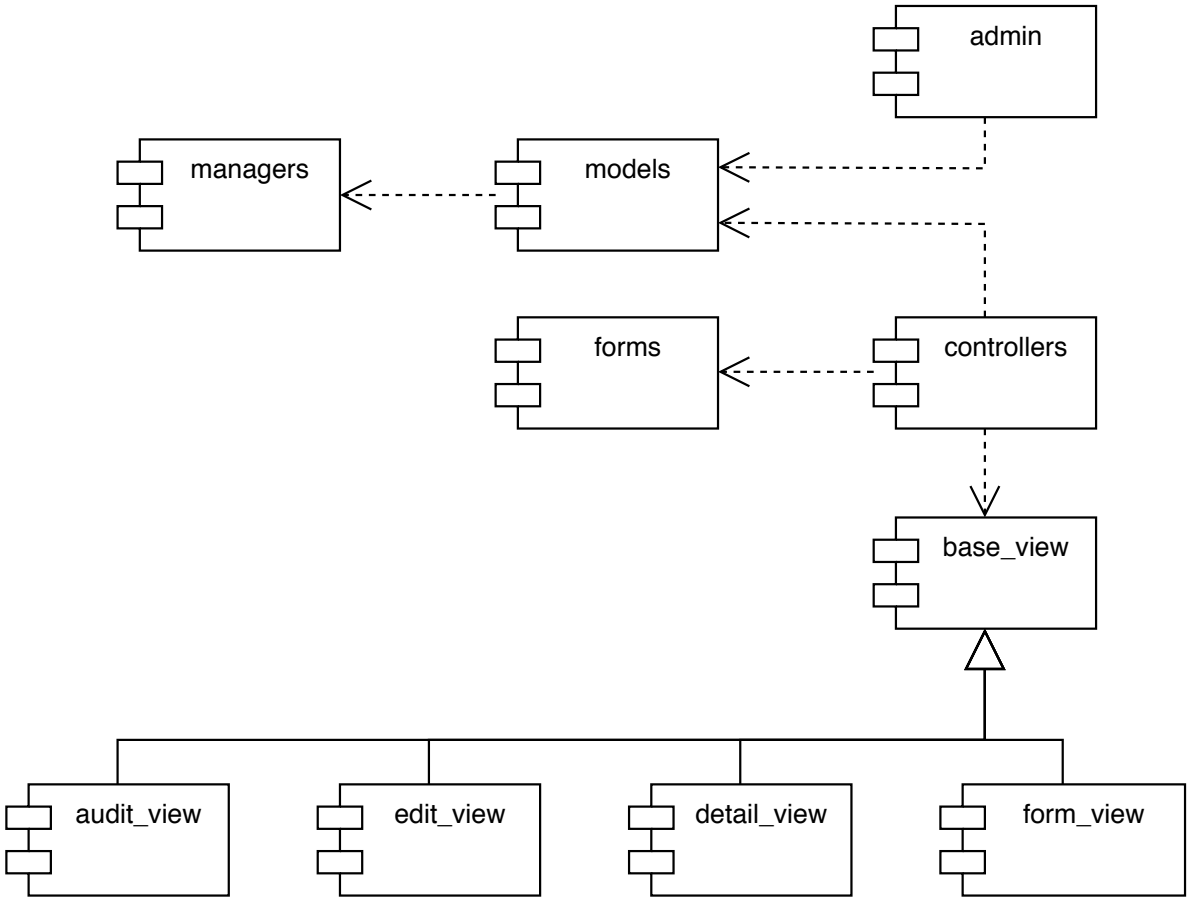

Рис. 2.5: UML-диаграмма компонентов приложения

3 Технологический раздел

3.1 Выбор языка программирования, среды и инструментов разработки

В качестве языка программирования был выбран язык Python. Данный язык программирования активно используется в веб-разработке, имеет множество сторонних библиотек и фреймворков, что облагчает разработку. [8]

В качестве фреймворка для разработки был выбран Django [9], так как он содержит мощные инструменты для разработки web-приложений и является одним из самых популярных фреймворков. Для верстки шаблонов сайта был выбран фреймворк Bootstrap [10].

В качестве интегрированной среды разработки был выбран PyCharm. Эта среда поддерживает Django, содержит встроенный статический анализатор кода, множество встроенных инструментов упрощающих разработку и отдадку приложения. [11]

В качестве СУБД будет использована PostgreSQL (SQLite в процессе разработки и отладки приложения). [5] [6]

3.2 Реализация архитектуры программы

3.2.1 Реализация шаблонов проектирования

В основе Django лежит шаблон проектирования MVC, который во фреймворке называется Model-View-Template, где Model — модель, являющаяся фактически ORM-сущностью, View — контроллер, Template — представление. Бизнес-логику в Django принято выделять в отдельный компонент.

3.2.2 Компоненты приложения

Выделим четыре компонента: компонент доступа к данным, компонент бизнес-логики, компонент графического интерфейса пользователя, компонент, связывающий бизнес-логику и графический интерфейс пользователя.

Полные листинги с исходным кодом основных классов представлены в приложениях.

Компонент доступа к данным

Данный компонент представляет из себя классы `django.db.models.Models` (далее — модели), данные в которых соответствуют атрибутам таблиц в базе данных. Модели содержат методы для обработки данных на уровне строки, например метод `get_state()`, возвращающий словесное описание состояния камеры. На листинге 3.1 представлена модель `State`, соответствующая таблице Состояние (State) в базе данных. Класс также содержит метаинформацию для упрощения работы в приложении.

Листинг 3.1: Модель `State`

```
class State(models.Model):
    STATES = (
        (0, 'OK'),
        (1, 'WARNING'),
        (2, 'ERROR'),
        (3, 'FAILURE'),
    )

    state = models.IntegerField(choices=STATES, verbose_name='состояние')
    logs = models.TextField(verbose_name='журнал')

    def get_state(self):
        return self.get_state_display()

    def __str__(self):
        return f'{self.get_state()}'

    class Meta:
        verbose_name = 'состояние'
        verbose_name_plural = 'состояния'
```

Для работы с данными на уровне таблицы используются классы, называемые менеджерами. Эти классы наследуются от `django.db.models.Managers` и содержат методы для доступа к данным. На листинге 3.2 приведен пример менеджера.

Листинг 3.2: Менеджер `CameraManager`

```
class CameraManager(models.Manager):
    def get_camera(self, camera_id):
        try:
            q = self.get(pk=camera_id)
        except self.DoesNotExist:
            q = None

        return q

    def get_problem(self):
        return self.exclude(state__state='OK')

    def get_error(self):
        return self.filter(state__state='ERROR')

    def get_warning(self):
        return self.filter(state__state='WARNING')

    def get_failure(self):
        return self.filter(state__state='FAILURE')

    def filter_pk(self, pk):
        return self.filter(pk__in=pk)
```

Компонент бизнес-логики

Компонент бизнес-логики состоит из классов, которые было решено называть сервисами (services). На листинге 3.3 представлен класс, контролирующий работу камер. Метод этого класса возвращает список камер, состояние которых неисправно и для которых, вместе с тем, еще не было оформлено сервисное обслуживание.

Листинг 3.3: Сервис `NotServicedCamerasController`

```
class NotServicedCamerasController:
    @staticmethod
    def get_cameras_need_service():
        not_serviced_cameras = Service.objects.get_not_served().values()
```

```
5           not_serviced_cameras_id = [i.get('pk') for i in
                not_serviced_cameras]
6           return
                Camera.objects.get_problem().filter_pk(not_serviced_cameras_id)
```

Компонент, связывающий бизнес-логику и интерфейс

Классы-контроллеры в Django наследуются от класса `django.views.generic.DefaultView`. При этом существуют классы для выполнения типичных задач представления данных: для отображения списка объектов — `ListView`, для отображения информации о конкретном объекте — `DetailView` и др. На листинге 3.4 представлен класс `AuditDetailView`. В нем, в частности, переопределен метод `get_context_data()` для передачи в компонент интерфейса дополнительной информации об обслуживании камеры.

Листинг 3.4: Контроллер `AuditDetailView`

```
1  class AuditDetailView(DetailView):
2      model = Camera
3      template_name = 'app/audit_detail.html'
4      context_object_name = 'camera'
5
6      def get_context_data(self, **kwargs):
7          context = super().get_context_data()
8          context['service'] =
               Service.objects.get_by_camera(pk=self.kwargs['pk'])
9          return context
```

Компонент интерфейса

Компонент интерфейса представляет из себя набор `HTML`-страниц, использующих шаблонизатор Django, который позволяет использовать шаблоны для генерации конечных страниц. Шаблонизатор позволяет, в частности, переиспользовать код и ускоряет верстку Web-приложения. Данные передаются из контроллера в качестве параметров на страницу.

3.3 Интерфейс приложения

Интерфейс приложения представляет из себя страницу, в шапке которой расположено название приложения, информация об авторизированном пользователе и кнопка выхода из системы.

В основной части приложения находится навигационное меню, в котором расположены ссылки на доступные пользователю страницы приложения. Ниже расположено название текущей страницы и непосредственно контект страницы, например таблицы с данными. Имеются также кнопки, например для добавления новых записей.

В нижней части страницы находится служебная информация и ссылка на административную панель сайта.

На рисунках 3.1–3.2 представлен интерфейс приложения.

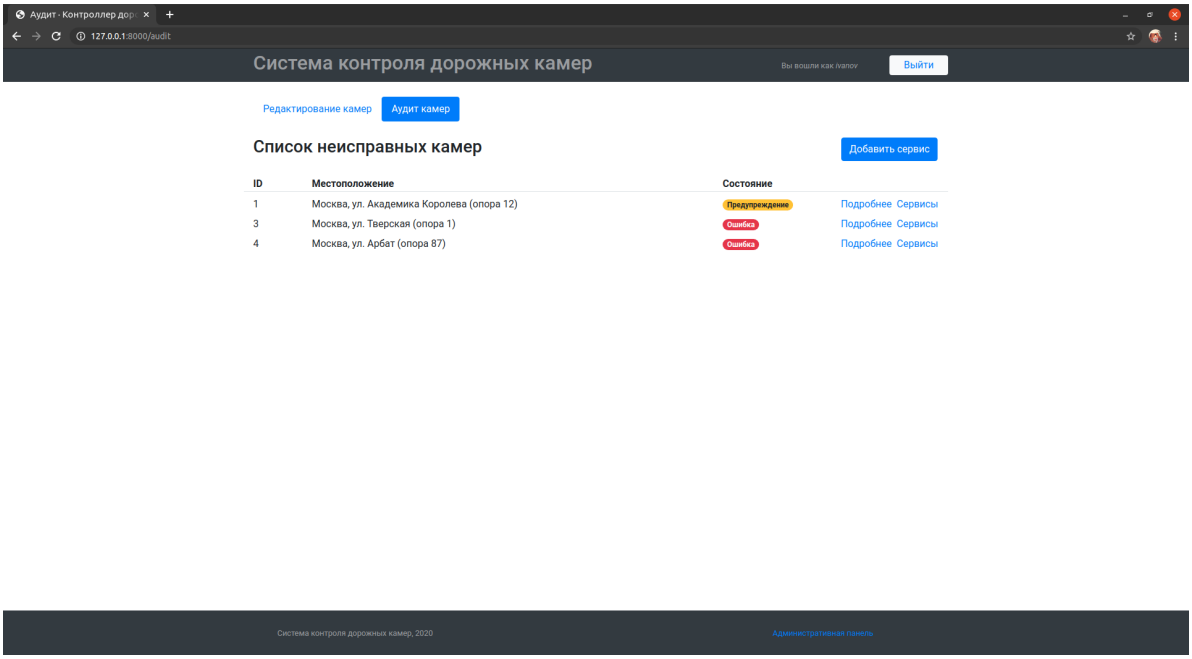

Рис. 3.1: Интерфейс страницы «Аудит камер»

3.4 Отладка и тестирование приложения

Для процесса отладки программы использовался отладчик и статический анализатор кода, встроенные в среду разработки PyCharm.

Программа успешно прошла все тесты и показала высокую стабильность.

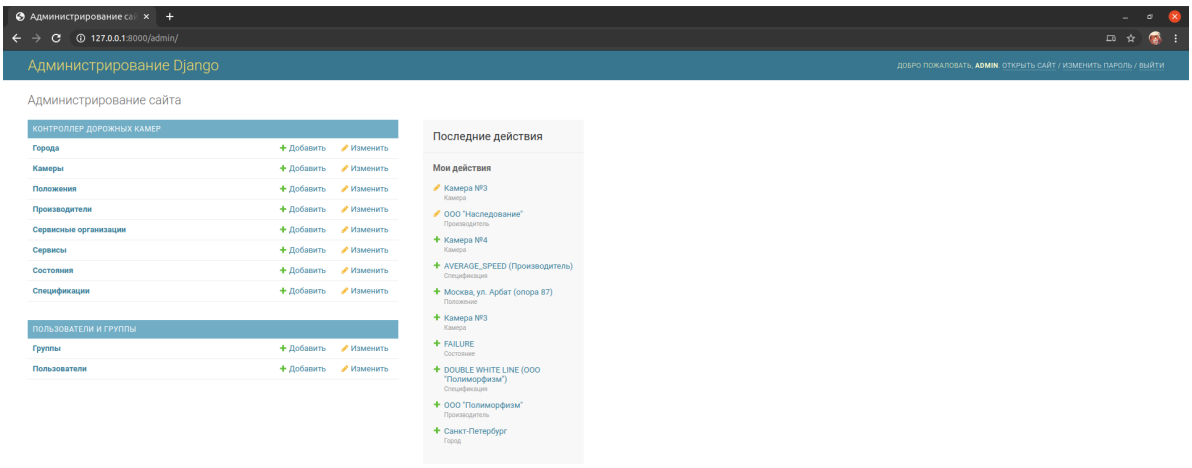

Рис. 3.2: Интерфейс административной панели

Выводы

В технологическом разделе выбран язык и инструменты разработки, реализовано Web-приложение и проведено его успешное тестирование.

4 Исследовательский раздел

4.1 Технические характеристики тестирующего ПК

Тестирование приложения производилось на локальном сервере, запущенном на персональном компьютере со следующими техническими характеристиками:

- Процессор Intel Core i5-6200U с тактовой частотой 2.3 ГГц.
- Оперативная память объемом 6 Гб.
- SSD со скоростью чтения и записи 540 МБ/с.
- Операционная система Ubuntu 20.04.

4.2 Исследование времени добавления записей в базу данных

Для исследования производительность базы данных использовался модуль Faker, позволяющий генерировать случайные данные различных типов. [12]

С помощью модуля Faker было подготовлено в общей сложности 10000 записей для добавления в различные таблицы базы данных. Данные загружались группами, начиная с 1000 записей, с шагом в 1000 записей. В результате исследования получен график, представленный на рисунке 4.1. Полученные данные о времени добавления записей в базу данных были усреднены.

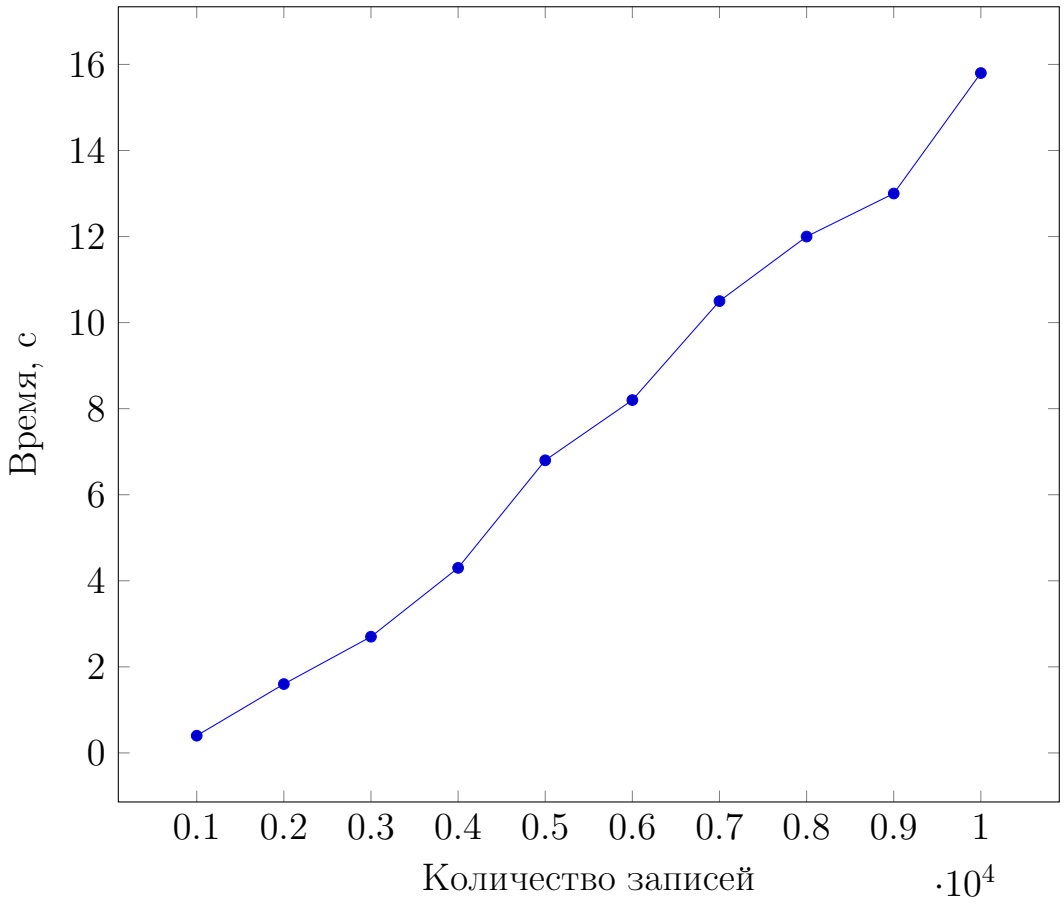

Рис. 4.1: Результаты исследования

Время добавления в базу данных 10000 записей можно считать приемлемым. Можно предположить, что на практике именно такой объем данных и будет единовременно загружаться в систему.

Выводы

В экспериментальном разделе расчетно-пояснительной записки было исследовано время загрузки данных в систему.

Заключение

В результате работы над курсовым проектом были выполнены всем поставленные задачи:

- проанализирована предметная область, требования к базе данных и приложению в целом;
- спроектированы база данных и приложение;
- реализовано приложение;
- протестировано приложения, исследована его стабильность и быстродействие.

Можно предложить следующие варианты модификации программного продукта:

- добавление новых групп пользователе и страниц приложения;
- добавление других методов поиска информации в приложении;
- интеграция с другими приложениями и сервисами.

Список использованных источников

[1] Носарев А. В. Автомобильные дороги России. *Межотраслевой Альманах*, (30), 2010.

[2] Willis Hendrikz Wilson, C and Bellamy Le Brocque. Speed cameras for the prevention of road traffic injuries and deaths. *The Cochrane Library*, (10), 2010.

[3] Microsoft. Документация СУБД MS SQL. URL: https://docs.microsoft.com/ru-ru/sql/?view=sql-server-ver15 (дата обращения: 27.05.2020).

[4] Oracle Corporation. Документация СУБД MySQL. URL: https://dev.mysql.com/doc/ (дата обращения: 27.05.2020).

[5] The PostgreSQL Global Development Group. Документация СУБД Postgresql. URL: https://www.postgresql.org/docs/ (дата обращения: 27.05.2020).

[6] Документация СУБД SQLite. URL: https://www.sqlite.org/docs.html (дата обращения: 27.05.2020).

[7] Стив Бурбек. *Программирование приложений в Smalltalk-80.* 1998.

[8] Python Software Foundation. Документация языка программирования Python. URL: https://docs.python.org/3/ (дата обращения: 27.05.2020).

[9] Django Software Foundation. Документация фреймворка Django. URL: https://docs.djangoproject.com/en/3.0/ (дата обращения: 27.05.2020).

[10] Bootstrap team. Документация фреймворка Bootstrap. URL: https://getbootstrap.com/docs/4.5/getting-started/introduction/ (дата обращения: 27.05.2020).

[11] JetBrains s.r.o. Документация интегрированной среды разработки PyCharm. URL: https://www.jetbrains.com/ru-ru/pycharm/documentation/ (дата обращения: 27.05.2020).

[12] Документация модуля Faker. URL: https://faker.readthedocs.io/en/master/ (дата обращения: 27.05.2020).

Приложение А. Модели

Листинг 4.1: Модели приложения

```
from django.db import models

from app.managers import SpecificationsManager, CameraManager,
    ServiceManager


class City(models.Model):
    name = models.CharField(max_length=30, verbose_name='название')

    def __str__(self):
        return self.name

    class Meta:
        verbose_name = 'город'
        verbose_name_plural = 'города'


class Location(models.Model):
    city = models.ForeignKey(City, on_delete=models.CASCADE,
       verbose_name='город')
    street = models.CharField(max_length=100, verbose_name='улица')
    support = models.IntegerField(verbose_name='номер опоры')
    notes = models.TextField(verbose_name='заметки')

    def get_location(self):
        if self.support:
            return f'{self.city}, {self.street} (опора {self.support})'
        else:
            return f'{self.city}, {self.street}'

    def __str__(self):
        return self.get_location()

    class Meta:
        verbose_name = 'положение'
        verbose_name_plural = 'положения'
```

```
35
36
37 class State(models.Model):
38     STATES = (
39         (0, 'OK'),
40         (1, 'WARNING'),
41         (2, 'ERROR'),
42         (3, 'FAILURE'),
43     )
44
45     state = models.IntegerField(choices=STATES, verbose_name='состояние')
46     logs = models.TextField(verbose_name='журнал')
47
48     def get_state(self):
49         return self.get_state_display()
50
51     def __str__(self):
52         return f'{self.get_state()}'
53
54     class Meta:
55         verbose_name = 'состояние'
56         verbose_name_plural = 'состояния'
57
58
59 class Producer(models.Model):
60     organization = models.CharField(max_length=100,
          verbose_name='название␣организации')
61     city = models.ManyToManyField(City, verbose_name='город')
62     phone = models.CharField(max_length=10, verbose_name='телефон')
63     contract_expires = models.DateField(verbose_name='срок␣действия␣
          договора')
64     notes = models.TextField(verbose_name='заметки')
65
66     def __str__(self):
67         return self.organization
68
69     class Meta:
70         verbose_name = 'производитель'
71         verbose_name_plural = 'производители'
72
73
74 class Specifications(models.Model):
75     TYPES = (
76         (0, 'SPEED'),
77         (1, 'AVERAGE_SPEED'),
78         (2, 'RED␣LIGHT'),
79         (3, 'DOUBLE␣WHITE␣LINE'),
80         (4, 'BUS␣LANE'),
```

```
81          (5, 'TOLLBOOTH'),
82          (6, 'LEVEL␣CROSSING'),
83          (7, 'CONGESTION␣CHARGE'),
84      )
85
86      type = models.IntegerField(choices=TYPES, verbose_name='тип')
87      producer = models.ForeignKey(Producer, on_delete=models.CASCADE,
            verbose_name='производитель')
88      date_of_manufacture = models.DateField(verbose_name='дата␣
            производства')
89      service_frequency = models.IntegerField(verbose_name='частота␣
            сервисного␣обслуживания')
90      notes = models.TextField(verbose_name='заметки')
91
92      objects = SpecificationsManager()
93
94      def get_type(self):
95          return self.get_type_display()
96
97      def __str__(self):
98          return f'{self.get_type()}␣({self.producer.organization})'
99
100     class Meta:
101         verbose_name = 'спецификация'
102         verbose_name_plural = 'спецификации'
103
104
105 class Camera(models.Model):
106     location = models.ForeignKey(Location, on_delete=models.CASCADE,
            verbose_name='местоположение')
107     specifications = models.OneToOneField(Specifications,
            on_delete=models.CASCADE, verbose_name='спецификации')
108     state = models.ForeignKey(State, on_delete=models.CASCADE,
            verbose_name='состояние')
109
110     objects = CameraManager()
111
112     def __str__(self):
113         return f'Камера␣{self.id}'
114
115     class Meta:
116         verbose_name = 'камера'
117         verbose_name_plural = 'камеры'
118
119
120 class ServiceOrganization(models.Model):
121     organization = models.CharField(max_length=100,
            verbose_name='название␣организации')
```

```
122     city = models.ManyToManyField(City, verbose_name='город')
123     phone = models.CharField(max_length=10, verbose_name='телефон')
124     contract_expires = models.DateField(verbose_name='срок действия
            договора')
125     notes = models.TextField(verbose_name='заметки')
126 
127     def __str__(self):
128         return self.organization
129 
130     class Meta:
131         verbose_name = 'сервисная организация'
132         verbose_name_plural = 'сервисные организации'
133 
134 
135 class Service(models.Model):
136     camera = models.ForeignKey(Camera, on_delete=models.CASCADE,
            verbose_name='ID камеры')
137     service_organization = models.ForeignKey(ServiceOrganization,
            on_delete=models.CASCADE,
138                                              verbose_name='сервисная
                                                 организация')
139     registration_date = models.DateField(auto_now_add=True)
140     service_data = models.DateField(verbose_name='дата сервиса')
141     info = models.TextField(verbose_name='информация')
142 
143     objects = ServiceManager()
144 
145     def __str__(self):
146         return f'сервис {self.id}'
147 
148     class Meta:
149         verbose_name = 'сервис'
150         verbose_name_plural = 'сервисы'
```

Приложение Б. Менеджеры

Листинг 4.2: Менеджеры приложения

```
import datetime

from django.db import models


class ProducerManager(models.Manager):
    def get_by_city(self, city):
        return self.filter(city__name=city)


class SpecificationsManager(models.Manager):
    def get_older_than(self, term):
        return self.filter(date_of_manufacture__lte=datetime.date.today()
            - term)

    def get_last(self):
        return self.order_by('-date_of_manufacture')


class CameraManager(models.Manager):
    def get_camera(self, camera_id):
        try:
            q = self.get(pk=camera_id)
        except self.DoesNotExist:
            q = None

        return q

    def get_problem(self):
        return self.exclude(state__state='OK')

    def get_error(self):
        return self.filter(state__state='ERROR')

    def get_warning(self):
        return self.filter(state__state='WARNING')
```

```
36
37      def get_failure(self):
38          return self.filter(state__state='FAILURE')
39
40      def filter_pk(self, pk):
41          return self.filter(pk__in=pk)
42
43
44  class ServiceOrganizationManager(models.Manager):
45      def get_by_city(self, city):
46          return self.filter(city__name=city)
47
48
49  class ServiceManager(models.Manager):
50      def get_record(self, service_id):
51          try:
52              q = self.get(pk=service_id)
53          except self.DoesNotExist:
54              q = None
55
56          return q
57
58      def get_by_camera(self, pk):
59          return self.filter(camera=pk)
60
61      def get_latest_by_camera(self, pk):
62          return self.filter(camera=pk).order_by('-service_data')
63
64      def get_not_served(self):
65          return self.filter(service_data__lte=datetime.date.today())
```

Приложение В. Представления

Листинг 4.3: Представления приложения

```
from django.shortcuts import get_object_or_404
from django.urls import reverse_lazy
from django.views.generic import ListView, DetailView, CreateView
from .forms import ServiceForm

from .models import Camera, Service


class CameraListView(ListView):
    model = Camera
    template_name = 'app/editing.html'
    context_object_name = 'cameras'


class CameraDetailView(DetailView):
    model = Camera
    template_name = 'app/detail.html'
    context_object_name = 'camera'


class CameraStateListView(ListView):
    model = Camera
    template_name = 'app/audit.html'
    context_object_name = 'cameras'


class ServiceListView(ListView):
    model = Service
    template_name = 'app/services.html'
    context_object_name = 'services'

    camera = None

    def get_queryset(self):
        self.camera = get_object_or_404(Camera, pk=self.kwargs['camera'])
        return Service.objects.get_latest_by_camera(self.camera)
```

```
37
38      def get_context_data(self, **kwargs):
39          context = super().get_context_data(**kwargs)
40          context['camera'] = self.camera
41          return context
42
43
44  class AuditDetailView(DetailView):
45      model = Camera
46      template_name = 'app/audit_detail.html'
47      context_object_name = 'camera'
48
49      def get_context_data(self, **kwargs):
50          context = super().get_context_data()
51          context['service'] =
                Service.objects.get_by_camera(pk=self.kwargs['pk'])
52          return context
53
54
55  class ServiceCreateView(CreateView):
56      model = Service
57      template_name = 'app/audit.html'
58      form_class = ServiceForm
59      success_url = reverse_lazy('/')
60
61      def get_context_data(self, **kwargs):
62          kwargs['cameras'] = Camera.objects.all()
63          return super().get_context_data(**kwargs)
```