МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных

технологий

Специализация 1 –40 01 01 Программное обеспечение информационных

технологий (программирование интернет – изданий)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка базы данных для магазина классических автомобилей»

Выполнил студент Завадский Роман Витальевич

(Ф.И.О.)

Руководитель проекта ассистент Бондарчик Е.Н.

(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В .

(учен. степень, звание, должность, Ф.И.О., подпись)

Консультант: ассистент Бондарчик Е.Н.

(учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер: ассистент Бондарчик Е.Н.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2019

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий   
Кафедра информационных систем и технологий

Утверждаю

Заведующий кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Смелов

подпись инициалы и фамилия

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

**ЗАДАНИЕ**

**к курсовому проектированию**

**по дисциплине** "Базы данных"

Специальность: 1-40 01 01 Программное обеспечение информационных технологий

Группа: 4

Студент: Завадский Роман Витальевич

**Тема: «Разработка базы данных для магазина классических автомобилей»**

**1. Срок сдачи студентом законченной работы**: «30» ноября 2019 г.

**2. Исходные данные к проекту:**

**2.1**. Функционально должны быть выполнены следующие задачи:

* Поиск автомобилей по параметрам;
* Сравнение автомобилей;
* Авторизация и регистрация пользователей;
* Сохранение истории продаж;
* Сортировка автомобилей по ценам и году.

**2.2. Требования.**

* База данных должна быть спроектирована в СУБД Oracle.
* Доступ к данным должен осуществляться только через соответствующие процедуры.
* Количество объектов БД (таблиц, представлений, индексов, пользователей и пр.) регламентируется задачей.
* Должен быть проведен импорт данных из XML файлов, экспорт данных в формат XML.
* Необходимо протестировать производительность базы данных (на таблицах, содержащих не менее 100 000 строк) и внести изменения в структуру в случае необходимости.
* Применить технологию базы данных: мультимедийные типы данных в БД.
* Листинги проекта должны содержать комментарии.

**3. Содержание расчетно-пояснительной записки**

* Введение
* Постановка задачи
* Разработка модели базы данных.
* Разработка необходимых объектов
* Описание процедур импорта и экспорта
* Тестирование производительности
* Описание технологии
* Список используемых источников
* Приложения

**4. Форма представления выполненного курсового проекта:**

* + Теоретическая часть курсового проекта должна быть представлена в формате MS Word.
  + Оформление записки должно быть согласно правилам.
  + Необходимые схемы, диаграммы и рисунки допускается делать в MS Office Visio или копии экрана (интерфейс).
  + Листинги скриптов представляются в приложении.
  + К записке необходимо приложить CD (DVD), который должен содержать: пояснительную записку, листинги и файлы базы данных.

**Календарный план**

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Наименование этапов курсового проекта | Срок выполнения этапов проекта | Примечание |
| 1 | Введение | 25.09.2019 |  |
| 2 | Аналитический обзор литературы по теме проекта | 05.10.2019 |  |
| 3 | Изучение требований, определение вариантов использования | 10.10.2019 |  |
| 4 | Анализ и проектирование модели базы данных. Описание информационных объектов и ограничений целостности. | 15.10.2019 |  |
| 5 | Создание необходимых объектов | 20.10.2019 |  |
| 6 | Импорт и экспорт данных | 26.10.2019 |  |
| 7 | Описание используемой технологии | 30.10.2019 |  |
| 8 | Тестирование производительности | 06.11.2019 |  |
| 9 | Оформление пояснительной записки | 28.11.2019 |  |
| 10 | Сдача проекта | 30.11.2019 |  |

**5. Дата выдачи задания** «10» сентября 2019г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *E. Н. Бондарчик*

(подпись)

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата и подпись студента)

Реферат

ФИО

Подпись

Дата

Лист

4

КП 00.00.ПЗ

Разраб.

*Завадский Р.В.*

Провер.

Бондарчик Е.Н.

Н. контр.

Бондарчик Е.Н.

Утверд.

Бондарчик Е.Н.

Реферат

Лит.

Листов

1

*БГТУ 71171073, 2019*

Пояснительная записка курсового проекта содержит 29 страниц пояснительной записки, 25 иллюстраций, 5 источников литературы, 1 приложение.

Python, Django 2.2.6, oracle

Основная цель курсового проекта: проектирование базы данных для магазина классических автомобилей и изучении мультимедийных типов данных.

В первой главе проводится аналитический обзор литературы по тематике курсового проекта и содержит анализ методов решения поставленных задач, а также примеры отечественных и зарубежных аналогов.

Вторая глава посвящена процессу проектирования системы.

В третьей главе описывается процесс разработки, компоненты и объекты проекта.

В четвертой главе приведено обоснование технических приёмов.

В пятой главе описано тестирование, позволяющее подробно понять интерфейс программного средства.

В заключении приведены результаты проделанной работы.

Содержание

ФИО

Подпись

Дата

Лист

5

КП 00.00.ПЗ

Разраб.

*Завадский Р.В.*

Провер.

Бондарчик Е.Н.

Н. контр.

Бондарчик Е.Н.

Утверд.

Бондарчик Е.Н.

Содержание

Лит.

Листов

1

*БГТУ 71171073, 2019*

[Введение 6](#_Toc27192541)

[1 Аналитический обзор литературы 7](#_Toc27192542)

[2 Модель проекта и теоретические исследования 10](#_Toc27192543)

[3 Разработка программной системы 11](#_Toc27192544)

[3.1 Разработка модели базы данных 11](#_Toc27192545)

[3.2 Индексы 13](#_Toc27192546)

[3.3 Процедуры 13](#_Toc27192547)

[3.4 Триггеры 13](#_Toc27192548)

[3.5 Последовательности 13](#_Toc27192549)

[3.6 Классы 14](#_Toc27192550)

[3.7 Представления 15](#_Toc27192551)

[3.8 Шаблоны 16](#_Toc27192552)

[3.9 Формы 16](#_Toc27192553)

[3.10 Процедуры экспорта и импорта XML 16](#_Toc27192554)

[3.11 Технология мультимедийных типов данных в базе данных 17](#_Toc27192555)

[4 Обоснование технических приемов программирования 19](#_Toc27192556)

[5 Тестирование, проверка работоспособности и анализ данных 21](#_Toc27192557)

[Заключение 27](#_Toc27192558)

[Список литературы 28](#_Toc27192559)

[Приложение А 29](#_Toc27192560)

Введение

ФИО

Подпись

Дата

Лист

6

КП 00.00.ПЗ

Разраб.

*Завадский Р.В.*

Провер.

Бондарчик Е.Н.

Н. контр.

Бондарчик Е.Н.

Утверд.

Бондарчик Е.Н.

Введение

Лит.

Листов

1

*БГТУ 71171073, 2019*

Сегодня практически любой софт, программное средство, приложение не обходится без собственной базы данных. Каждая база данных разрабатывается под конкретное программное обеспечение основываясь на общих подходах и принципах. База данных представляет организованную структуру, предназначенную для хранения, изменения и обработки взаимосвязанной информации. Информация, как правило, хранится в преимущественно больших вопросах, поэтому при разработке базы данных ставится вопрос об её оптимизации для ускорения работы всего приложения.

Современные требования к разработке приложений должны обеспечивать скоростное выполнение запросов к базе данных, обеспечивать безопасность использования и сохранность данных, позволять избегать лазеек для различных кибератак.

В данном проекте база данных разрабатывается для магазина классических автомобилей. Современные приложения типа магазин являются чаще всего кроссплатформенными (далее сайт), а именно интернет-магазинами. Они позволяют множеству пользователям иметь доступ к приложению удалённо, через браузер своего устройства. У каждого пользователя есть возможность создать свой аккаунт и оформить заказ. Администратор же сайта имеет доступ к данным таблиц базы данных, а конкретно он имеет право создавать новые записи, редактировать их, или удалять. Создавать новые таблицы, изменять атрибуты полей, удалять таблицы администратору запрещено.

Поэтому основной целью моего проекта является разработка оптимизированной базы данных совместно с кроссплатформенным приложением типа интернет-магазин на тематику классических автомобилей. Так же, исходя их выбранной технологии «мультимедийные типы данных в базе данных», в проекте реализована возможность работы с изображениями в базе данных, где изображения представляют собой фото продаваемых автомобилей.

При использовании данного программного средства, клиенты смогут просматривать каталог представленных на сайте автомобилей, сортировать и фильтровать их по параметрам, формировать заявки, отслеживать их статус, отслеживать статус заказов. Администратор же приложения непосредственно управляет сайтом, оформляет у компаний-поставщиков поставки автомобилей на склад, имеет доступ к справочнику по моделям авто, который может сам обновлять при необходимости, принимает или отвергает заявки клиентов и следит о выполнении заказов.

Язык разработки проекта – Python. При выполнении курсового проекта используется СУБД Oracle. Так же используется технология Django.

Аналитический обзор литературы

ФИО

Подпись

Дата

Лист

7

КП 01.00.ПЗ

Разраб.

*Завадский Р.В.*

Провер.

Бондарчик Е.Н.

Н. контр.

Бондарчик Е.Н.

Утверд.

Бондарчик Е.Н.

1 Аналитический обзор литературы

Лит.

Листов

3

*БГТУ 71171073, 2019*

Основой для учета, контроля и планирования служат всевозможные картотеки, регистрационные журналы, списки и т.д. Они постепенно накапливаются и обновляются. При большом объеме информации поиск и обобщение необходимых сведений, осуществляемых вручную, представляют собой довольно трудоемкий процесс.

Сейчас же, в век компьютерных технологий и больших прогрессов в этой сфере, на смену картотекам пришли очень удобные, надёжные, быстродействующие компьютерные системы. Для решения задач с использованием компьютерных систем используется специальное программное обеспечение, часто объединенное в крупные информационно-справочные системы.

Постепенно с развитием программного обеспечения ЭВМ появились идеи создания управляющих систем, которые позволяли бы накапливать, хранить и обновлять взаимосвязанные данные по целому комплексу решаемых задач, например, при автоматизации бухгалтерского учета на предприятии. Эти идеи нашли свое воплощение в системах управления базами данных (СУБД). СУБД взаимодействуют не с локальными, а взаимосвязанными по информации массивами, называемыми базами данных. С появлением персональных компьютеров СУБД становятся наиболее популярным средством обработки табличной информации. Они являются инструментальным средством проектирования банков данных при обработке больших объемов информации.

На сегодняшний день существует достаточное количество сайтов, интернет-магазинов, так или иначе связанных с торговлей автомобилями.

Одним из таковых является «abw.by»

Рассмотрим в качестве примера каталог данного сайта на рисунке 1.1.

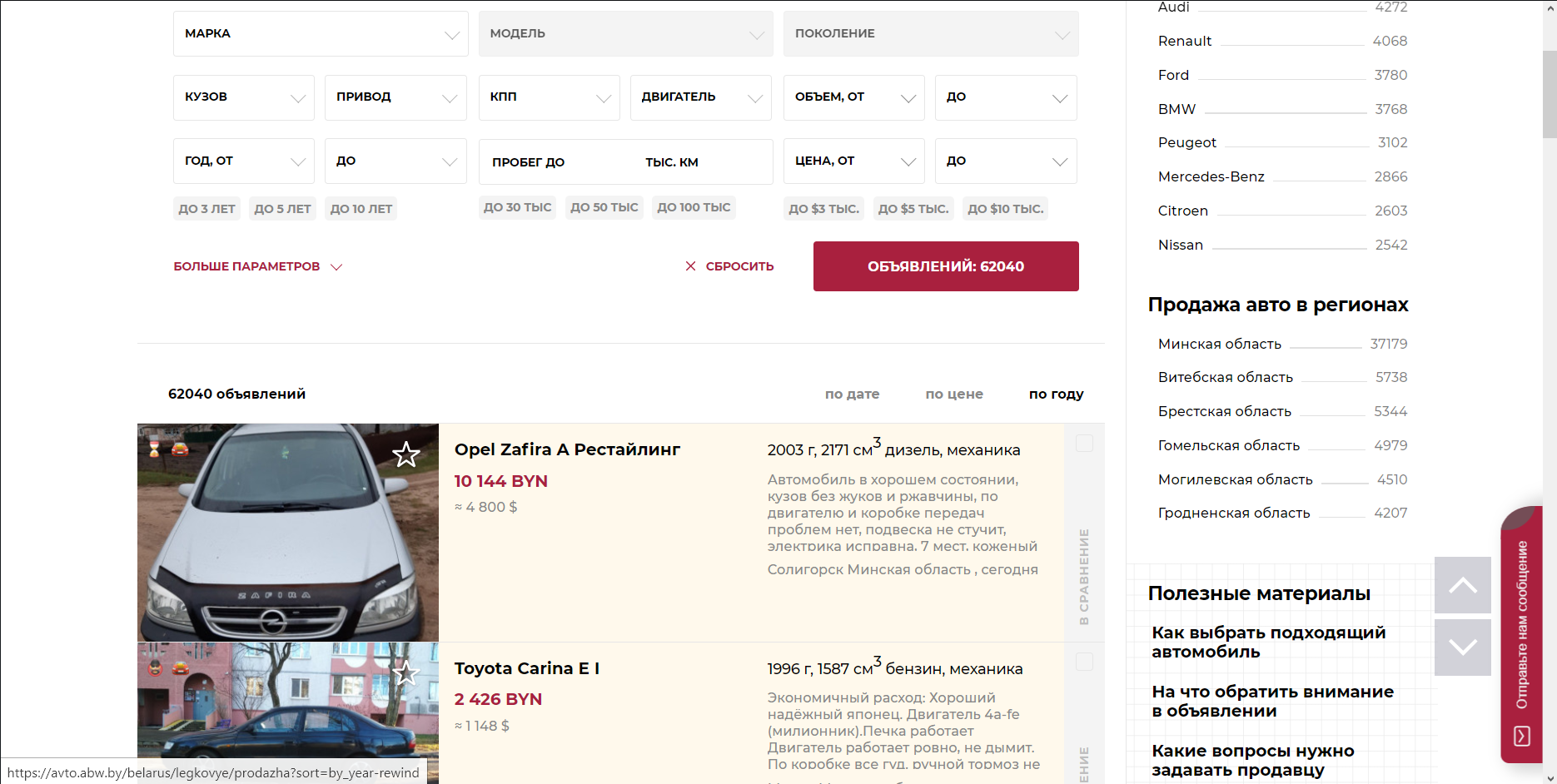


Рисунок 1.1 – Пример каталога «abw.by»

Вкратце об основных возможностях сайта для публикаций объявлений по продаже авто «abw.by»:

* Авторизация и регистрация пользователей;
* Подача и публикация объявлений как о покупке, так и о продаже авто;
* Возможность сортировки и фильтрации объявлений;
* Ведение новостного портала;
* Возможность четко, полно и грамотно описать все стороны и характеристики автомобиля;
* Помещение объявлений в избранное;
* Сравнение объявлений;
* Выдача похожих объявлений;
* Разбиение объявлений по регионам;
* Большое количество параметров фильтрации.

Так же рассмотрим зарубежный сайт, специализирующийся конкретно на классических авто «old-time.ru», одна из страниц которого представлена на рисунке 1.2.

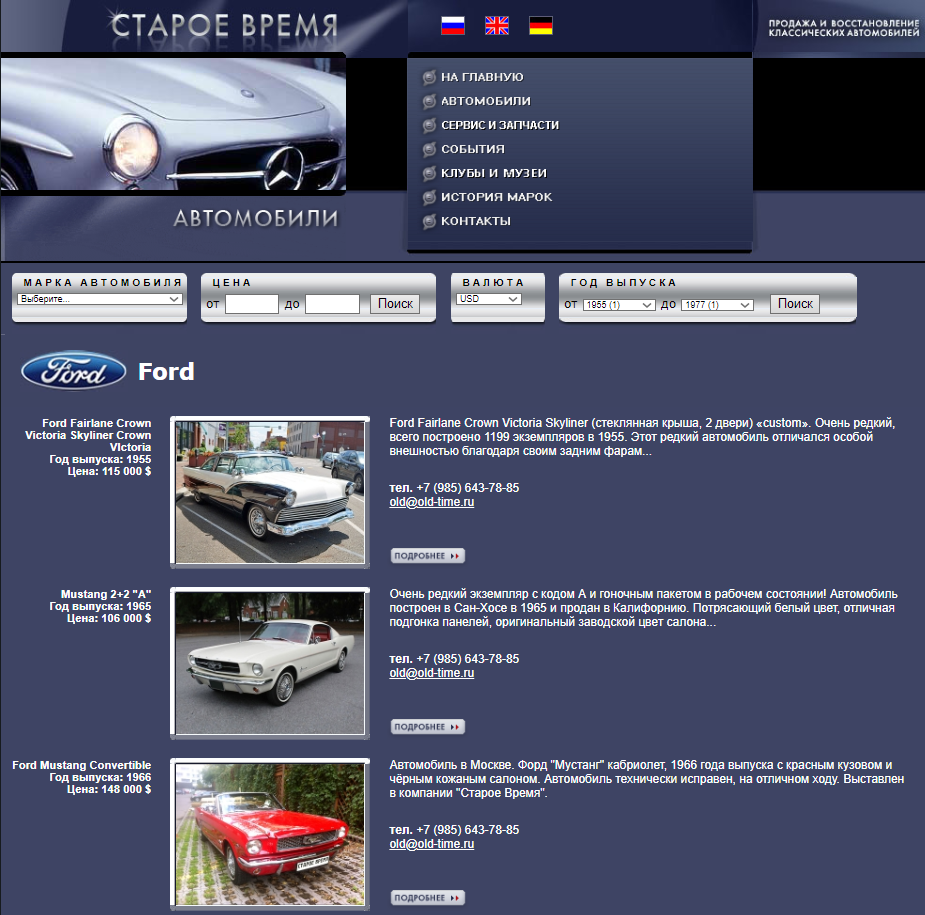


Рисунок 1.2 – Пример сайта «old-time.ru»

И его основные особенности:

* Наличие локализации для 3-х языков;
* Возможность отфильтровать авто;
* Использование в фильтрах разных валют.

Проанализировав отечественных и зарубежные аналоги можно сделать вывод о том, что имеется достаточно много автомагазинов, но конкретно на тематику классических авто лишь единицы и функционал их намного скуднее.

Исходя из приведённых данных были выдвинуты требования к программному средству, о которых говориться в пункте 2.

Модель проекта и теоретические исследования

ФИО

Подпись

Дата

Лист

10

КП 02.00.ПЗ

Разраб.

*Завадский Р.В.*

Провер.

Бондарчик Е.Н.

Н. контр.

Бондарчик Е.Н.

Утверд.

Бондарчик Е.Н.

2 Модель проекта и теоретические исследования

Лит.

Листов

1

*БГТУ 71171073, 2019*

Работа администратора интернет – магазина благодаря грамотно разработанному программному средству может быть в разы ускорена. Для этого важно определить модели и провести исследования. Чем сильнее будет автоматизирован процесс работы с данными – тем эффективнее окажется окончательный результат, а это непосредственно положительно сказывается на прибыли компании от использования приложения. В каждом магазине автомобилей важно, как можно детальнее вести учёт клиентов и связанных с ними заказами, а также хранить полную историю заказов. Каждый клиент может оставлять заявки и смотреть статус их рассмотрения. Приложение позволяет заносить необходимые данные и сведения о клиенте: его имя, паспортные данные, контактную информацию и т.д. При формировании объявления имеется возможность прикрепить к нему фото, в полной мере описать характеристики авто.

Непосредственное управление сайтом осуществляется через администраторскую часть приложения. Она позволяет администратору вести удобное контролирование за всеми поставками, заявками и заказами. Благодаря этой части приложения администратор может проводить анализ данных. Например, сравнивать автомобили по частоте заказа той или иной марки, цены или года выпуска. Система позволит взять под контроль все финансовые операции.

Всё это в свою очередь позволит повысить уровень сервиса качества предоставляемых услуг, что обязательно привлечёт новых клиентов, а вместе с ними и прибыль. Модель приведена на рисунке 2.1.

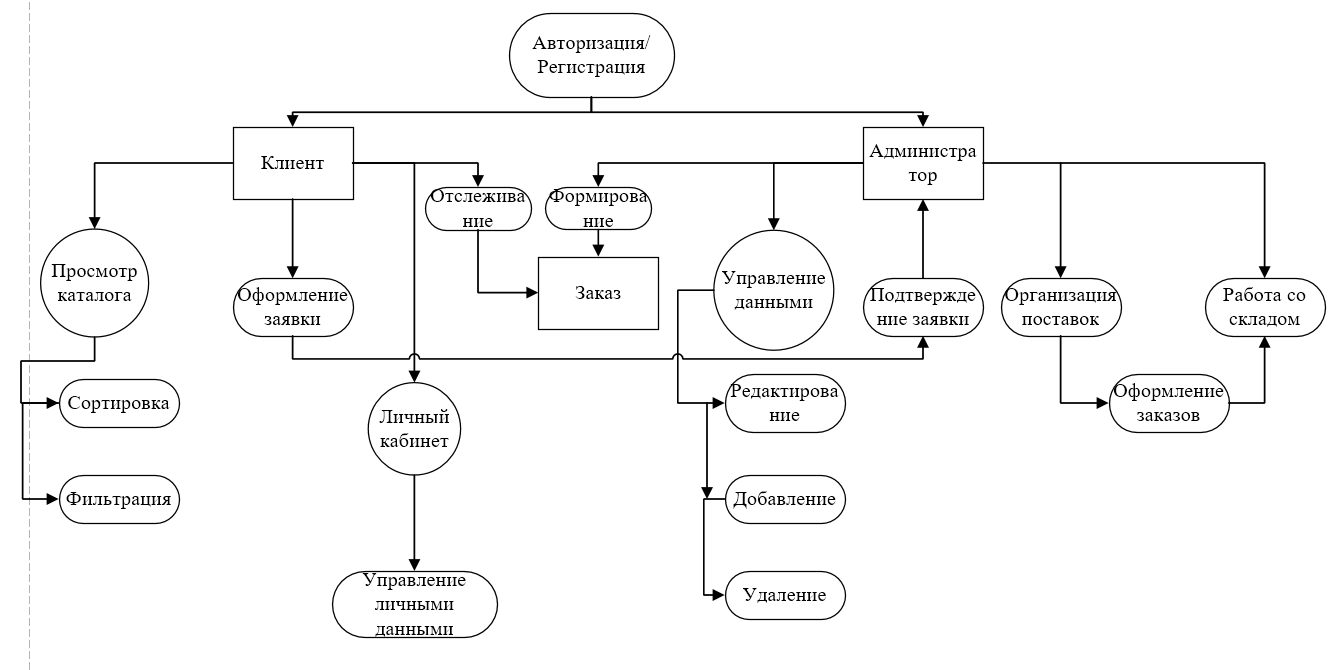


Рисунок 2.1 – Модель приложения

Разработка программной системы

ФИО

Подпись

Дата

Лист

11

КП 03.00.ПЗ

Разраб.

*Завадский Р.В.*

Провер.

Бондарчик Е.Н.

Н. контр.

Бондарчик Е.Н.

Утверд.

Бондарчик Е.Н.

3 Разработка программной системы

Лит.

Листов

8

*БГТУ 71171073, 2019*

Для создания и работы с базой данных используется Oracle СУБД

3.1 Разработка модели базы данных

Для базы данных магазина автомобилей было разработано 8 таблиц. Структура базы данных представлена в приложении А.

В таблице Users хранятся зарегистрированные на сайте пользователи.

Таблица Customers содержит информацию о клиентах. Requests включает в себя всю информацию о заявке, оставленной клиентом. Cars – данные/справочник о различных автомобилях. Orders – таблица, хранящая в себе данные относительно заказов клиентов. Mycars содержит информацию о всех автомобилях, находящихся на складе. Suppliers – информация о поставщиках автомобилей. Deliveries – поставки автомобилей на склад.

Таблица Users состоит из столбцов:

* ID;
* LOGIN – логин пользователя;
* PASSWORD – пароль пользователя;
* USER\_TYPE –тип пользователя.

Таблица Customers состоит из столбцов:

* ID;
* LOGIN\_ID – логин клиента;
* FIO – ФИО клиента;
* PHONE – номер телефона клиента;
* ADRESS – адрес клиента;
* EMAIL – адрес электронной почты клиента;
* PERSON – тип клиента.

Таблица Requests состоит столбцов:

* ID;
* CAR\_ID – зарезервированный авто;
* DATE\_REQUEST – дата заявки;
* CUSTOMER\_ID – клиент;
* REQUEST\_STATUS – статус заявки;
* DELIVERY – включена ли доставка;
* DELIVERY\_ADDRESS – адрес доставки;
* COST – цена.

Таблица Cars состоит из столбцов:

* ID;
* MARK – марка авто;
* MODEL – модель авто;
* CARCASS – типы кузова;
* YEARS\_OF\_PRODUCTIONS – годы выпуска;
* ENGINES – двигатели;
* TRANSMISSIONS – коробки передач;
* DRIVE\_UNITS – типы привода.

Таблица Orders состоит из столбцов:

* ID;
* REQUEST\_ID – заявка;
* ORDER\_STATUS – статус заказа;
* START\_DATE – дата начала выполнения заказа;
* END\_DATE – дата закрытия заказа.

Таблица Mycars состоит из столбцов:

* ID;
* CAR\_ID – заказанный автомобиль;
* CAR\_BODY – тип кузова;
* CAR\_YEAR – год автомобиля;
* ENGINE – двигатель автомобиля;
* TRANSMISSION – тип коробки передач;
* DRIVE\_UNIT – тип привода;
* COLOR – цвет автомобиля;
* IDENTIFICATION\_NUMBER – vin номер автомобиля;
* MILEAGE – пробег;
* COST – цена;
* PHOTOS – название фото авто;
* DATE\_OF\_DELIVERY – дата поставки автомобиля;
* DESCRIPTION – описание автомобиля;
* OPTIONS – опции автомобиля;
* VISIBLE – статус авто, означающий что он зарезервирован или продан;
* SLUG – уникальное обозначение;
* IMAGEB – фото автомобиля.

Таблица Suppliers состоит из столбцов:

* ID;
* NAME – название/имя поставщика;
* ADDRESS – адрес поставщика;
* PHONE – номер телефона поставщика;
* EMAIL – адрес электронной почты.

Таблица Deliveries состоит из столбцов:

* ID;
* SHIPPER\_ID – поставщик;
* CAR\_ID – автомобиль;
* DATE\_ORDER – дата заказа;
* COST – стоимость.

3.2 Индексы

При назначении столбца в качестве первичного колюча Oracle автоматически создаёт индекс по этому столбцу. Соответственно в базе данных имеются следующие индексы: CARS\_PK, CUSTOMERS\_PK, DELIVERIES\_PK, ORDERS\_PK, REQUESTS\_PK, SUPPLIERS\_PK, USERS\_PK, MYCARS\_PK.

3.3 Процедуры

Хранимая процедура – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. В проекте имеется 24 процедуры, по 3 на каждую таблицу, а именно процедуры создания, изменения и удаления. Все используемые в курсовом проекте процедуры приведены в приложении А.

3.4 Триггеры

Триггеры представляют специальный тип хранимой процедуры, которая вызывается автоматически при выполнении определенного действия. В курсовом проекте имеется 8 триггеров для каждой из таблиц, предназначенных для обращения к последовательностям при добавлении в таблицы новых строк. Пример одного из таких триггеров приведён на рисунке 3.2

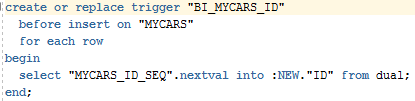


Рисунок 3.2 – Пример триггера

3.5 Последовательности

Последовательности – это объект базы данных, который генерирует целые числа в соответствии с правилами, установленными во время его создания. Всего в курсовом проекте 8 последовательностей для каждой из таблиц, применяющихся для автоматического увеличения значения поля ID. Пример одной из таких последовательностей приведён на рисунке 3.3.

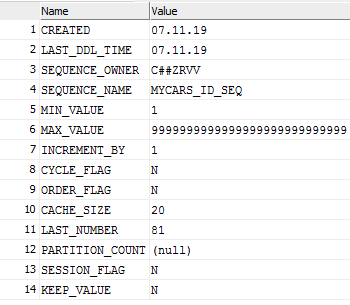


Рисунок 3.3 – Пример последовательности

3.6 Классы

Классы, представленные в курсовом проекте, являются объектами модели. Каждому классу соответствует одна из таблиц в нашей базе данных.

Диаграмма классов представлена на рисунке 3.4. Классы модели приведены в приложении А.

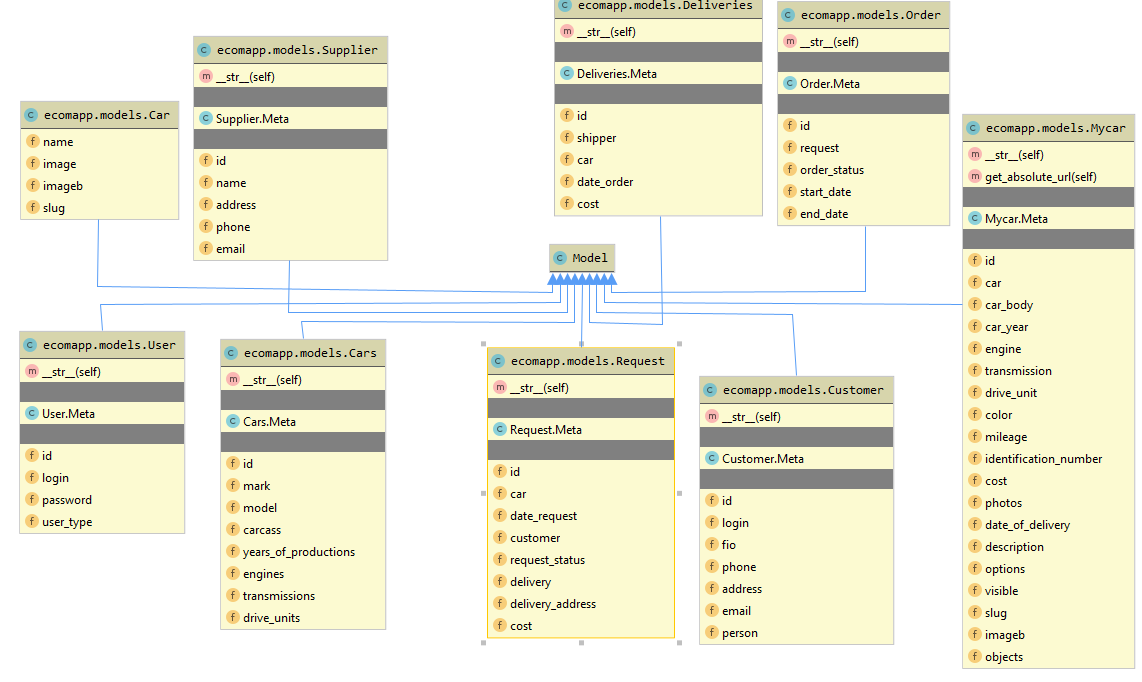


Рисунок 3.4 – Диаграмма классов

При создании проекта первым делом мы создали базу данных, а затем создавались модели. Однако в случае необходимости изменений в моделях использовались миграции, которые автоматически вносили изменения в базу данных.



Рисунок 3.5 – Пример миграции

3.7 Представления

Центральным моментом любого веб-приложения является обработка запроса, который отправляет пользователь. В Django за обработку запроса отвечают представления или views. По сути представления представляют функции обработки, которые принимают данные запроса в виде объекта request и генерируют некоторый результат, который затем отправляется пользователю.

В курсовом проекте всего 12 представлений.

* base\_view – отвечает за отображение информации на главной странице;
* order\_by\_view – отвечает за сортировку автомобилей;
* filter\_view – отвечает за фильрацию автомобилей;
* mycar\_view – отвечает за отображение информации об автомобиле;
* request\_view – отвечает за отображение заявок клиентов;
* add\_to\_request\_view – отвечает за добавление заявок от клиентов;
* remove\_from\_request\_view – отвечает за удаление заявок от клиентов;
* change\_request\_view – позволяет клиентам изменять параметры доставки в своей заявке;
* order\_view – отвечает за отображение заказов клиентов.

Все представления находятся в приложении А.

3.8 Шаблоны

Шаблоны (template) отвечают за формирование внешнего вида приложения. Они предоставляют специальный синтаксис, который позволяет внедрять данные в код HTML. Шаблоны же непосредственно связаны с представлениями.

В курсовом проекте всего 7 шаблонов.

* account – личный кабинет;
* base – главная страница;
* login – логин форма;
* mycar – страница объявления;
* order – таблица заказов;
* registration – форма регистрации;
* request – таблица заявок.

Некоторые из шаблонов представлены в приложении А.

3.9 Формы

Django предоставляет специальные возможности по работе с формами, которые позволяют определять функциональноть форм в одном месте и использовать многократно в разных местах, упрощают валидацию данных, помогают связывать формы с моделями и многое другое.

Каждая форма определяется в виде отдельного класса, который расширяет класс forms.Form. Классы размещаются внутри проекта, где они используются. Нередко они помещаются в отдельный файл, или могут размещаться внутри уже имеющихся в приложении файлов.

В курсовом проекте всего 3 формы.

* LoginForm – форма авторизации;
* RegistrationForm – форма регистрации;
* CustomerForm – форма личного кабинета.

3.10 Процедуры экспорта и импорта XML

В курсовой работе реализованы процедуры экспорта данных в xml файл из базы данных и импорта данных из xml файла в базу данных. При выполнении данных процедур импорт и экспорт осуществляется для одной из указанных таблиц.

Код создания процедуры экспорта данных таблицы в xml представлен в приложении А.

Код процедуры импорта данных из xml для таблицы Users представлен на рисунке 3.6.

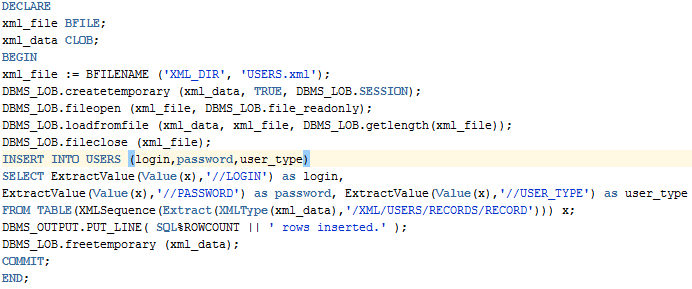


Рисунок 3.6 – Процедура импорта данных

3.11 Технология мультимедийных типов данных в базе данных

Технология мультимедийных типов данных в курсовом проекте отражена в виде хранимых изображений автомобилей. Изображения хранятся в таблице Mycars в столбце imageb, что представлено на рисунке 3.7.

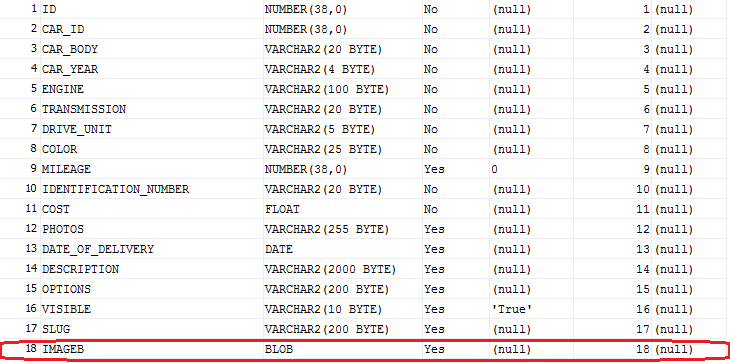


Рисунок 3.7 – Столбец с хранимыми в базе изображениями

Для хранения в поле изображений в СУБД применяется тип BLOB. Тип BLOB представляет из себя массив двоичных данных. Для того что бы преобразовать изображение в массив байт используется функция read(), представленная на рисунке 3.8.

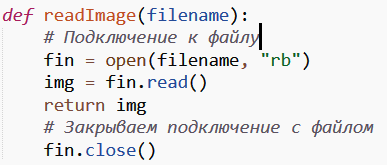


Рисунок 3.8 – Функция преобразования изображения в массив байт

Далее полученный массив байт записывается в соответствующее поле в базе данных. В поле базы данных данный массив отображается как (BLOB), однако при двойном клике мышью по данному полю появляется окно, представленное на рисунке 3.9, в котором можно посмотреть его параметры, а также представить массив в качестве изображения или текста.

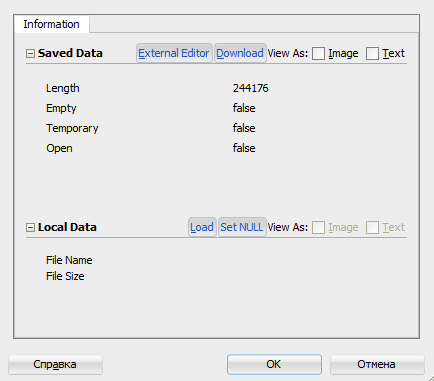


Рисунок 3.9 – Окно с параметрами

У хранения файлов в базе данных есть один несомненный плюс – шифрование.

Однако у хранения файлов на сервере есть ряд преимуществ:

* Скорость. Пожалуй, одно из самых важных преимуществ. Дело в том, что при хранении в БД нам нужно произвести больше действий для вывода изображения, чем из файловой системы.
* Универсальность. Обычные изображения можно использовать для множества приложений. Бинарные данные из базы необходимо сначала конвертировать.
* Простота реализации. Закачать файл на сервер и сохранить ссылку на него в таблицу легче, чем реализовывать логику сохранения бинарных данных в ту же таблицу. А простота – это скорость разработки и внесения изменений. Время – деньги.

Обоснование технических приемов программирования

ФИО

Подпись

Дата

Лист

19

КП 04.00.ПЗ

Разраб.

*Завадский Р.В.*

Провер.

Бондарчик Е.Н.

Н. контр.

Бондарчик Е.Н.

Утверд.

Бондарчик Е.Н.

4 Обоснование технических приёмов программирования

Лит.

Листов

2

*БГТУ 71171073, 2019*

При проектировании нашего программного средства мы использовали технологию Django. Данная технология представляет из себя фреймворк для создания веб-приложений на языке Python, использующий шаблон MVC. Архитектура MVC позволяет разработчику работать с визуальным представлением и бизнес-логикой приложения отдельно. При работе с Django чаще используют термин MVT – Model-View-Template или модель-представление-шаблон. Компоненты MVT можно использовать независимо друг от друга. Схема архитектуры MVT представлена на рисунке 4.1.

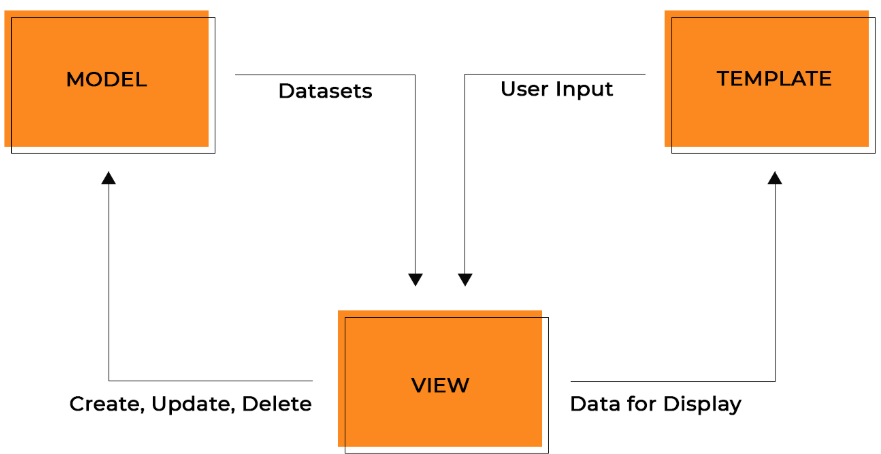


Рисунок 4.1 – схема архитектуры MVT в Django

Для работы с базой данных Django использует собственный ORM, в котором модель данных описывается классами Python, и по ней генерируется схема базы данных. ORM автоматически передаёт данные из БД, например в объекты, которые используются в коде приложения. ORM ускоряет разработку прототипов и готовых веб-приложений на Django. Также ORM позволяет быстро переключаться между базами данных с минимальными изменениями кода.

Django определяет модель как источник информации о данных, в которых содержатся ключевые поля и поведение данных. Одна модель указывает на одну таблицу в базе данных. Django поддерживает базы данных PostgreSQL, MySQL, SQLite и Oracle.

Так как для курсового проекта бы выбрали базу данных Oracle, необходимо стало выбрать технологию, для взаимодействия с ней. Django как раз поддерживает Oracle. Для этого нам пришлось дополнительно установить драйвер Python cx\_Oracle. После этого выполняем подключение, которое представлено на рисунке 4.2.



Рисунок 4.2. – Подключение Oracle к Django

Django представляет удобную утилиту inspectdb, которая может создавать модели по существующей базе данных.

Модели содержат информацию о данных. Эти данные представлены атрибутами или полями. Поскольку модель представляет собой простой класс, она ничего не знает о других уровнях Django. Взаимодействие между уровнями происходит через API.

Модель отвечает за бизнес-логику, методы, свойства и другие элементы, связанные с манипуляцией данными. Также модели позволяют разработчикам создавать, читать, обновлять и удалять объекты в базе данных.

Представление решает три задачи: принимает HTTP-запросы, реализует бизнес-логику, отправляет HTTP-ответ в ответ на запросы. То есть представление получает данные от модели и предоставляет шаблонам доступ к этим данным или предварительно обрабатывает данные и затем предоставляет к ним доступ шаблонам.

Шаблоны представляют собой файлы с HTML-кодом, с помощью которого отображаются данные. Содержимое файлов может быть статическим или динамическим. Шаблоны не содержат бизнес-логики. Поэтому они только отображают данные.

Тестирование, проверка работоспособности и анализ данных

ФИО

Подпись

Дата

Лист

21

КП 05.00.ПЗ

Разраб.

*Завадский Р.В.*

Провер.

Бондарчик Е.Н.

Н. контр.

Бондарчик Е.Н.

Утверд.

Бондарчик Е.Н.

5 Тестирование, проверка работоспособности и анализ данных

Лит.

Листов

6

*БГТУ 71171073, 2019*

В ходе тестирования моего приложения были проведены тесты на:

* валидацию авторизации и регистрации;
* добавление, удаление, изменение данных;
* доступ клиентов в часть администратора;
* импорт и экспорт XML;
* загрузка изображения;

Тестируем для начала окно авторизации и регистрации.

Введём для начала неверный пароль. Обработка продемонстрирована на рисунке 5.1.

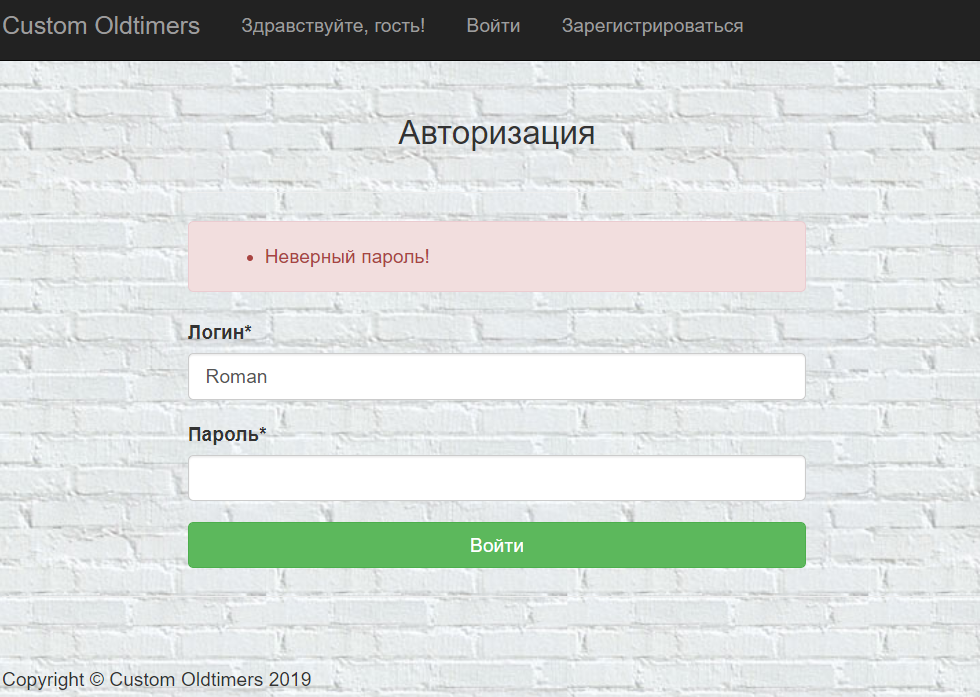


Рисунок 5.1 – Тестирование авторизации

Протестируем регистрацию. Попробуем зарегистрировать уже имеющегося в приложении пользователя. Обработка исключения представлена на рисунке 5.2.

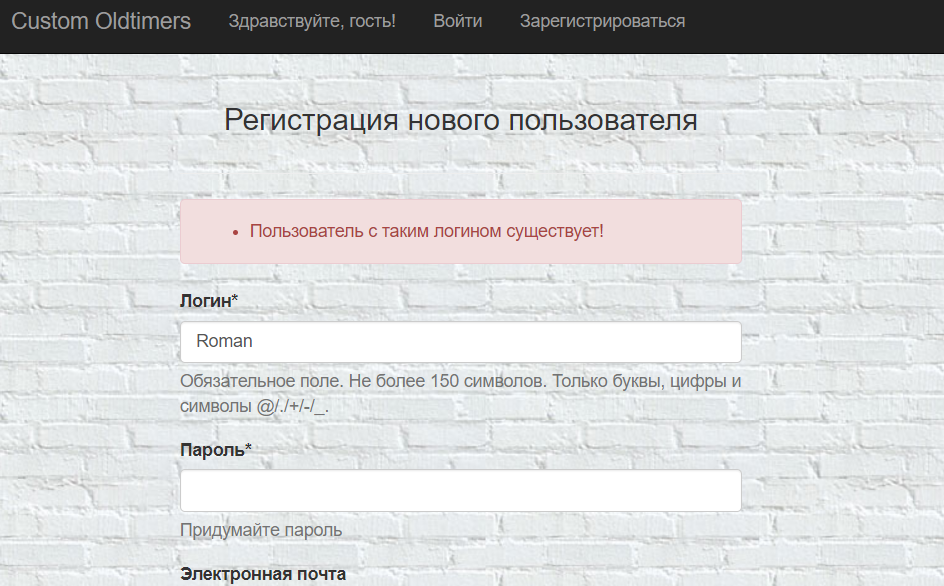


Рисунок 5.2 – Тестирование регистрации

Попробуем оставить при регистрации пустые поля. Обработка представлена на рисунке 5.3.

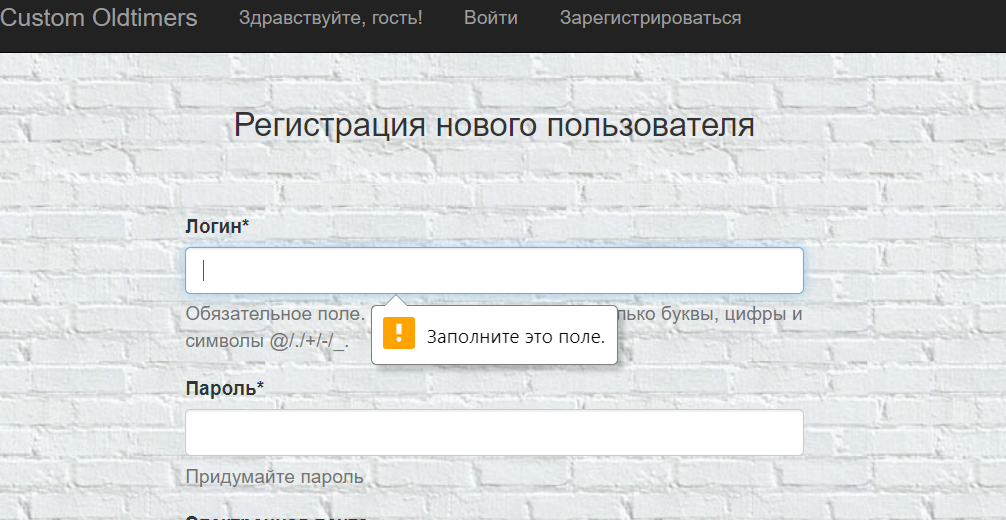


Рисунок 5.3 – Тестирование регистрации на пустое значение

Перейдем к тестированию работоспособности приложения. Протестируем возможность пользователя оставить заявку на интересующий его автомобиль. При клике пользователя на объявление открывается страница с описанием автомобиля, однако кнопка «Оставить заявку» отсутствует. Вместо этого, пользователю предлагается войти или зарегистрироваться, что продемонстрировано на рисунке 5.4.



Рисунок 5.4 – Требование для создания заявки пользователю

После авторизации пользователя на странице появится кнопка «Оставить заявку». При клике на неё пользователь попадает на страницу с его заявками, это представлено на рисунке 5.5.

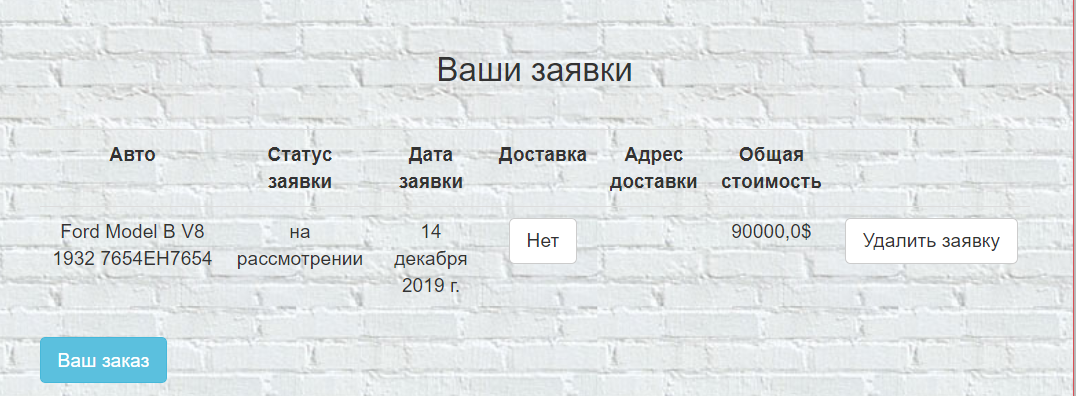


Рисунок 5.5 – Таблица с заявками пользователя

Далее войдём в администраторскую часть приложения, для этого на главной странице добавим в адресной строке /admin. После входа проверим наличие созданной пользователем заявки, что представлено на рисунке 5.6.

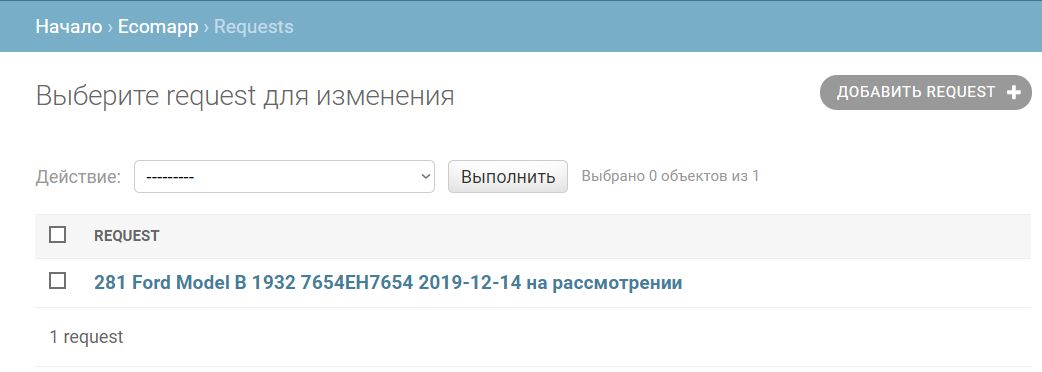


Рисунок 5.6 – Просмотр заявок администратором

Далее протестируем доступ клиентов в часть администратора. Тест представлен на рисунке 5.7.

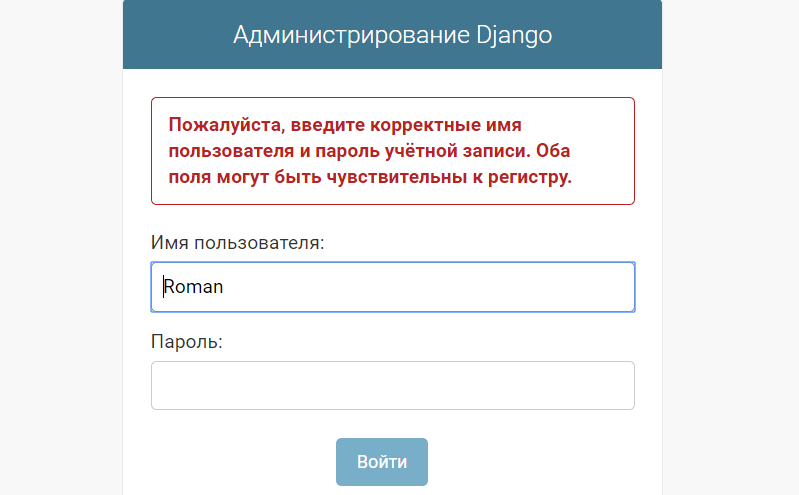


Рисунок 5.7 – Доступ клиентов в часть администратора

Клиент не имеет доступа к части администратора.

Протестируем импорт и экспорт XML. Вызовем процедуру экспорта TABLE\_TO\_XML\_FILE. Выберем в качестве экспортируемой таблицу Users. В результате получим файл USERS.xml с содержимым, представленным на рисунке 5.8.

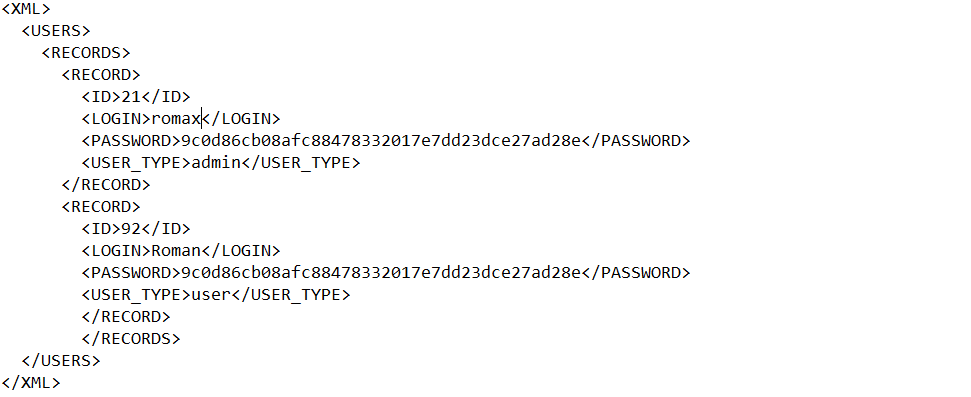


Рисунок 5.8 – Файл экспорта XML

Добавим ко всем полям login 1 и выполним процедуру импорт файла XML в базу данных для таблицы Users. В результате добавятся новые пользователи в таблице Users, что представлено на рисунке 5.9.

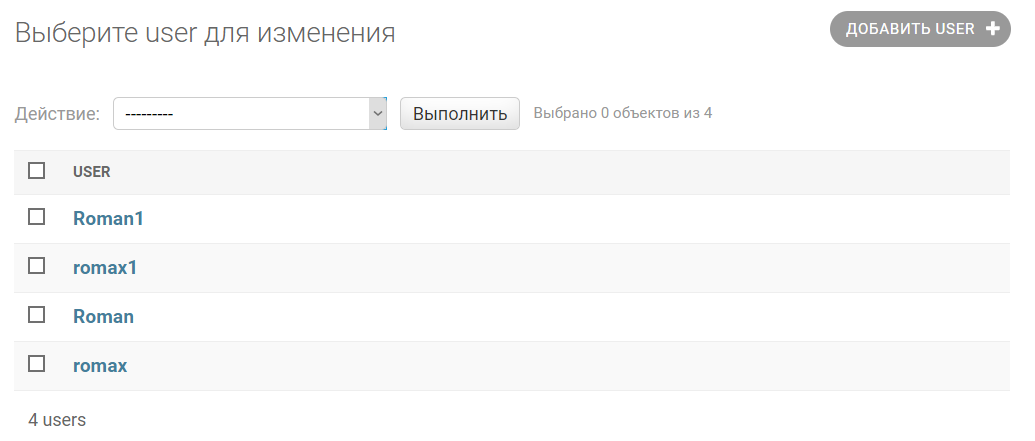


Рисунок 5.9 – Результат выполнения импорта XML данных

Проверим возможность загрузки изображения. Для этого зайдём в часть администратора и создадим новый Mycar автомобиль. Заполним все необходимые поля и выберем изображение, что представлено на рисунке 5.10.



Рисунок 5.10 – Выбор изображения

После этого нажмем кнопку «сохранить». Перейдем в нашу базу данных и найдем добавленный автомобиль. Кликнем по полю imageb, и выберем представление информации в виде изображения. Результат предоставлен на рисунке 5.11.

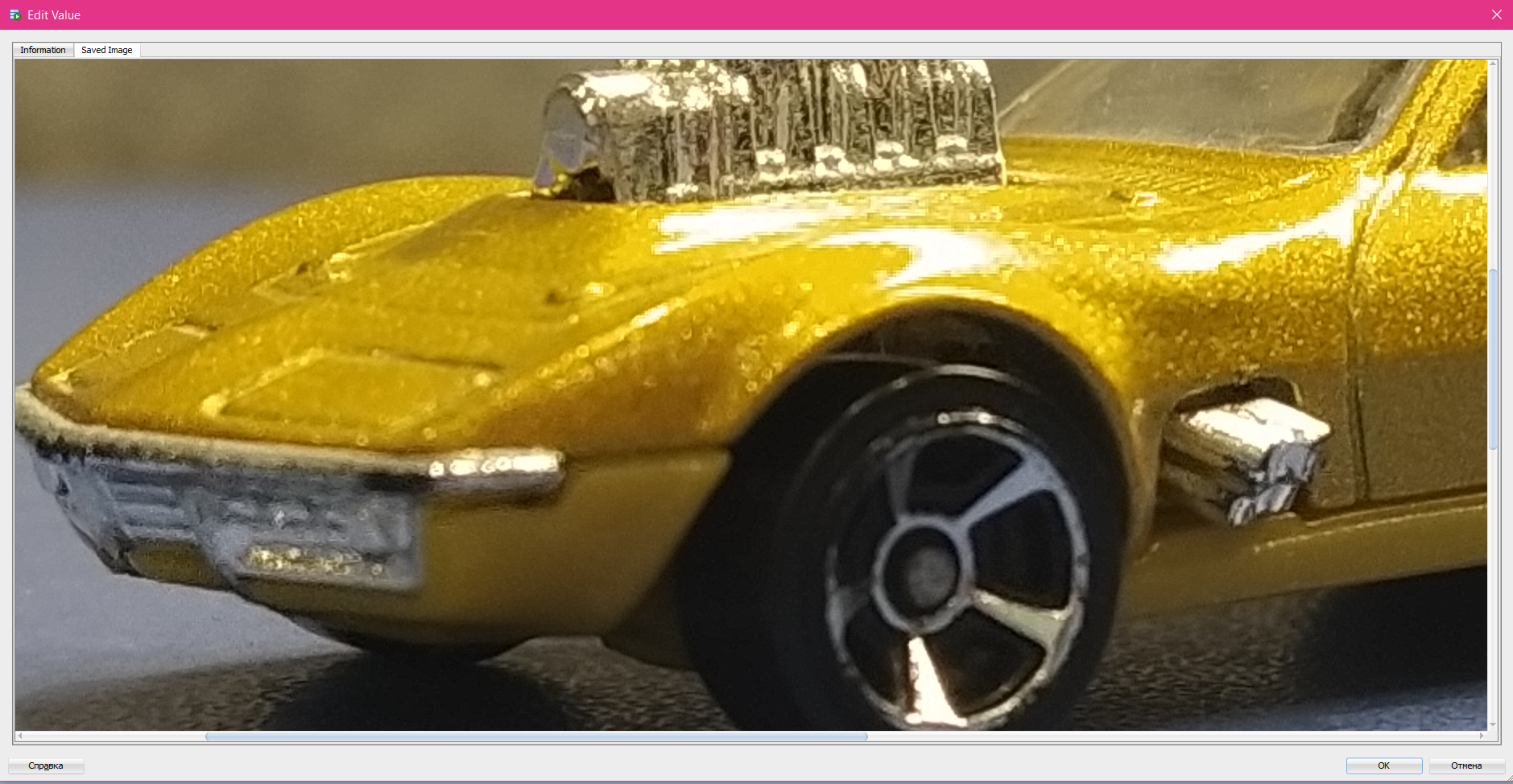


Рисунок 5.11 – Загруженное в базу данных изображение

После этого проверим отображение фото на сайте. Перейдем на сайт и найдем его, что представлено на рисунке 5.12.

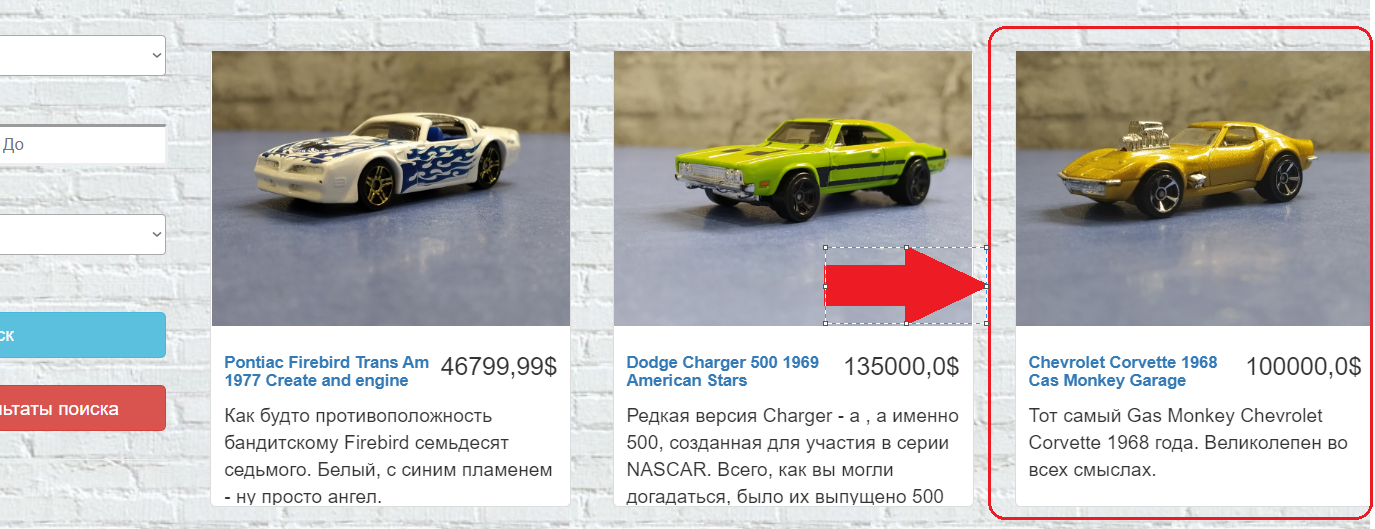


Рисунок 5.12 – Отображение изображения на сайте

Проведя тестирование таким образом мы установили, что сайт выполняет свою основную функцию, а именно позволяет пользователям заказывать классические автомобили. Технология использования изображений в качестве мультимедийных данных работает успешно. Также пользователи не имеют доступа к неразрешённому им функционалу.

Заключение

ФИО

Подпись

Дата

Лист

27

КП 00.00.ПЗ

Разраб.

*Завадский Р.В.*

Провер.

Бондарчик Е.Н.

Н. контр.

Бондарчик Е.Н.

Утверд.

Бондарчик Е.Н.

Заключение

Лит.

Листов

1

*БГТУ 71171073, 2019*

В процессе решения поставленной задачи была достигнута поставленная цель по созданию базы данных для программного средства «Магазин классических автомобилей», которая в совокупности с приложением формирует полноценный сайт для работы в сфере торговли автомобилями. Основой целью курсового проекта стало формирование навыков проектирования базы данных для дальнейшей интеграции с приложением. При разработке выполнены следующие пункты:

* Регистрация и авторизация пользователей;
* Разделение приложение на пользовательскую часть и часть администратора;
* Отслеживание статуса заявки и заказа;
* Реализована технология мультимедийных типов данных в базе данных;
* Реализована сортировка и фильтрация автомобилей;
* Хранение истории продаж;
* Импорт и экспорт XML.

В курсовом проекте были реализованы следующие задачи:

* Конфигурирование программного обеспечения;
* Создание базы данных;
* Создание интерфейса;
* Реализация функций работы приложения;
* Тестирование программного продукта.

Данный проект по сравнению с прямыми аналогами имеет больший функционал и удобный интерфейс. По сравнению с другими интернет – магазинами не уступает по многим параметрам.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объёме.

Список литературы

ФИО

Подпись

Дата

Лист

28

КП 00.00.ПЗ

Разраб.

*Завадский Р.В.*

Провер.

Бондарчик Е.Н.

Н. контр.

Бондарчик Е.Н.

Утверд.

Бондарчик Е.Н.

Заключение

Лит.

Листов

1

*БГТУ 71171073, 2019*

1. METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: <https://metanit.com> Дата доступа: 04.10.2019

2. Хеллман, Даг. Стандартная библиотека Python 3: справочник с примерами. 2-е изд. / Хеллман, Даг. – СПб: ООО “Диалектика”, 2019. – 1376 с.

3. djbook.ru [Электронный ресурс] / Режим доступа: <https://djbook.ru> Дата доступа: 11.10.2019

4. Oracle-dba.ru [Электронный ресурс] / Режим доступа: <https://oracle-dba.ru> Дата доступа 17.10.2019

5. stackoverflow.com [Электронный ресурс] / Режим доступа: <https://stackoverflow.com> Дата доступа: 26.10.2019

Приложение А

Листинг класса Cars

|  |
| --- |
| class Cars(models.Model):  id = models.AutoField(primary\_key=True)  mark = models.CharField(max\_length=30)  model = models.CharField(max\_length=60)  carcass = models.CharField(max\_length=100)  years\_of\_productions = models.CharField(max\_length=10)  engines = models.TextField(max\_length=1000)  transmissions = models.CharField(max\_length=50)  drive\_units = models.CharField(max\_length=20)  class Meta:  managed = False  db\_table = 'cars'  ordering = ['mark', 'model']  def \_\_str\_\_(self):  return (self.mark + ' ' + self.model) |

Листинг класса Customer

|  |
| --- |
| class Customer(models.Model):  id = models.AutoField(primary\_key=True)  login = models.ForeignKey('User', on\_delete=models.CASCADE)  fio = models.CharField(max\_length=200, blank=True, null=True)  phone = models.CharField(max\_length=20, blank=True, null=True)  address = models.CharField(max\_length=200, blank=True, null=True)  email = models.EmailField(max\_length=100, blank=True, null=True)  person = models.CharField(max\_length=23, blank=True, null=True, choices= (('физическое лицо', 'физическое лицо'),('частный предприниматель','частный предприниматель')))  class Meta:  managed = False  db\_table = 'customers'  def \_\_str\_\_(self):  return (self.fio) |

Листинг класса Deliveries

|  |
| --- |
| class Deliveries(models.Model):  id = models.AutoField(primary\_key=True)  shipper = models.ForeignKey('Supplier', on\_delete=models.CASCADE)  car = models.ForeignKey('Cars', on\_delete=models.CASCADE)  date\_order = models.DateField(blank=True, null=True)  cost = models.FloatField(blank=True, null=False)  class Meta:  managed = False  db\_table = 'deliveries' |

Листинг класса User

|  |
| --- |
| class User(models.Model):  id = models.AutoField(primary\_key=True)  login = models.CharField(unique=True, max\_length=20)  password = models.CharField(max\_length=200)  user\_type = models.CharField(max\_length=10, blank=True, null=True, choices= (('user', 'user'),('admin','admin')))  class Meta:  managed = False  db\_table = 'users'  def \_\_str\_\_(self):  return self.login |

Листинг класса Mycar

|  |
| --- |
| class Mycar(models.Model):  id = models.AutoField(primary\_key=True)  car = models.ForeignKey('Deliveries', on\_delete=models.CASCADE)  car\_body = models.CharField(max\_length=20, choices=MYCAR\_BODY\_CHOICES)  car\_year = models.CharField(max\_length=4, choices=Years)  engine = models.CharField(max\_length=100)  transmission = models.CharField(max\_length=20)  drive\_unit = models.CharField(max\_length=5, choices=(('RWD','RWD'),('AWD','AWD'),('FWD','FWD')))  color = models.CharField(max\_length=25)  mileage = models.BigIntegerField(blank=True, null=True, default = 0)  identification\_number = models.CharField(max\_length=20)  cost = models.FloatField(blank=True)  photos = models.ImageField()  date\_of\_delivery = models.DateField(blank=True, null=True)  description = models.TextField(max\_length=2000, blank=True, null=True)  options = models.CharField(max\_length=200, blank=True, null=True)  visible = models.CharField(max\_length=10, default = 'True' , choices=(('True','True'),('False','False')))  slug = models.SlugField(blank = True)  imageb = models.BinaryField()  objects = MycarManager()  class Meta:  managed = False  db\_table = 'mycars'  def \_\_str\_\_(self):  return(self.car.car.mark + ' ' + self.car.car.model + ' ' + self.options + ' ' + self.car\_year + ' ' + self.identification\_number)  def get\_absolute\_url(self):  return reverse('mycar\_detail', kwargs={'mycar\_slug': self.slug}) |

Листинг класса Request

|  |
| --- |
| class Request(models.Model):  id = models.AutoField(primary\_key=True)  car = models.ForeignKey('Mycar', on\_delete=models.CASCADE)  date\_request = models.DateField()  customer = models.ForeignKey('Customer', on\_delete=models.CASCADE)  request\_status = models.CharField(max\_length=20, choices=(('на рассмотрении','на рассмотрении'), ('принята', 'принята'), ('отклонена', 'отклонена')))  delivery = models.CharField(max\_length=10, choices=(('Нет','Нет'), ('Да', 'Да')))  delivery\_address = models.CharField(max\_length=200, blank=True, null=True)  cost = models.FloatField(blank=True, null=True)  class Meta:  managed = False  db\_table = 'requests'  def \_\_str\_\_(self):  return(str(self.id) + ' ' + self.car.car.car.mark + ' ' + self.car.car.car.model + ''  + self.car.car\_year + ' ' + self.car.identification\_number + ' ' + str(self.date\_request) + ' ' + self.request\_status) |

Листинг класса Order

|  |
| --- |
| class Order(models.Model):  id = models.AutoField(primary\_key=True)  request = models.ForeignKey('Request', on\_delete=models.CASCADE)  order\_status = models.CharField(max\_length=20, choices=(('выполняется','выполняется'),('готов','готов')))  start\_date = models.DateField(blank=True, null=True)  end\_date = models.DateField(blank=True, null=True)  class Meta:  managed = False  db\_table = 'orders'  def \_\_str\_\_(self):  return(str(self.id) + ' ' + self.request.customer.fio + ' ' + self.request.car.car.car.mark + ' ' +  self.request.car.car.car.model + ' ' + self.request.car.car\_year  +' '+ self.order\_status) |

Листинги представлений

|  |
| --- |
| def base\_view(request):  requests = Request.objects.all()  order\_by\_cost = request.GET.get('order\_by\_cost')  order\_by\_cost\_distinct= request.GET.get('order\_by\_cost\_distinct')  order\_by\_year=request.GET.get('order\_by\_year')  order\_by\_year\_distinct=request.GET.get('order\_by\_year\_distinct')  news\_order = request.GET.get('news\_order')  first\_cost = request.GET.get('first\_cost')  last\_cost = request.GET.get('last\_cost')  first\_year = request.GET.get('first\_year')  last\_year = request.GET.get('last\_year')  filter\_mark = request.GET.get('filter\_mark')  filter\_model = request.GET.get('filter\_model')  filter\_body = request.GET.get('filter\_body')  filter\_color = request.GET.get('filter\_color')  filter\_engine = request.GET.get('filter\_engine')  filter\_KPP1 = request.GET.get('filter\_KPP1')  filter\_KPP2 = request.GET.get('filter\_KPP2')  filter\_KPP = ''  if filter\_KPP1 == None and filter\_KPP2 == None:  filter\_KPP = ''  if filter\_KPP1 == 'МКПП' and filter\_KPP2 == None:  filter\_KPP = 'МКПП'  if filter\_KPP1 == None and filter\_KPP2 == 'АКПП':  filter\_KPP = 'АКПП'  if filter\_KPP1 == 'МКПП' and filter\_KPP2 == 'АКПП':  filter\_KPP = ''  filter\_unit = request.GET.get('filter\_unit')  first\_mileage = request.GET.get('first\_mileage')  last\_mileage = request.GET.get('last\_mileage')  filter\_supplier = request.GET.get('filter\_supplier')  mycars = Mycar.objects.all(order\_by\_cost, order\_by\_cost\_distinct, order\_by\_year, order\_by\_year\_distinct, news\_order, first\_cost, last\_cost, first\_year, last\_year, filter\_mark, filter\_model, filter\_body, filter\_color, filter\_engine, filter\_KPP, filter\_unit, first\_mileage, last\_mileage, filter\_supplier)  filterset = Mycar.objects.all(None, None, None, None, "WordOrder", first\_cost, last\_cost, first\_year, last\_year, filter\_mark, filter\_model, filter\_body, filter\_color, filter\_engine, filter\_KPP, filter\_unit, first\_mileage, last\_mileage, filter\_supplier)  markset = ''  modelset = ''  colorset = ''  engineset = ''  supplierset = ''  for mark in filterset:  markset = markset + mark.car.car.mark + '^'  markset = markset.rstrip('^')  massmarks = markset.split('^')  massmarks = list(set(massmarks))  massmarks = sorted(massmarks)  for model in filterset:  modelset = modelset + model.car.car.model + '^'  modelset = modelset.rstrip('^')  massmodels = modelset.split('^')  massmodels = list(set(massmodels))  massmodels = sorted(massmodels)  for color in filterset:  colorset = colorset + color.color + '^'  colorset = colorset.rstrip('^')  masscolors = colorset.split('^')  masscolors = list(set(masscolors))  masscolors = sorted(masscolors)  for engine in filterset:  engineset = engineset + engine.engine + '^'  engineset = engineset.rstrip('^')  massengines = engineset.split('^')  massengines = list(set(massengines))  massengines = sorted(massengines)  for supplier in filterset:  supplierset = supplierset + supplier.car.shipper.name + '^'  supplierset = supplierset.rstrip('^')  masssuppliers = supplierset.split('^')  masssuppliers = list(set(masssuppliers))  masssuppliers = sorted(masssuppliers)    context = {  'mycars': mycars,  'massmarks': massmarks,  'massmodels': massmodels,  'masscolors': masscolors,  'massengines': massengines,  'masssuppliers': masssuppliers,  'requests': requests  }  return render(request, 'base.html', context)  def add\_to\_request\_view(request, mycar\_slug):  requests = Request.objects.all()  mycar = Mycar.objects.get(slug = mycar\_slug)  new\_item, \_ = Request.objects.get\_or\_create(car = mycar, cost=mycar.cost, date\_request = datetime.today() , customer= Customer.objects.get(login = User.objects.get(login = request.user.username)) , request\_status = "на рассмотрении", delivery="Нет")  if new\_item not in requests:  mycar.visible = 'False'  mycar.save()  requests, \_ = Request.objects.get\_or\_create(car = mycar, cost= mycar.cost, date\_request = datetime.today() , customer= Customer.objects.get(login = User.objects.get(login = request.user.username)), request\_status = "на рассмотрении", delivery="Нет")  return HttpResponseRedirect('/requests/')  def remove\_from\_request\_view(request, mycar\_slug):  requests = Request.objects.all()  mycar = Mycar.objects.get(slug = mycar\_slug)  for requests\_mass in requests:  if requests\_mass.car == mycar:  del\_request = Request.objects.get(car = mycar)  del\_request.delete()  mycar.visible = 'True'  mycar.save()  return HttpResponseRedirect('/requests/')  def change\_request\_delivery(request):  requests = Request.objects.all()  delivery = request.GET.get('delivery')  request\_id = request.GET.get('request\_id')  requests = Request.objects.get(id = int(request\_id))  requests.delivery = delivery  if delivery == 'Да' :  requests.delivery\_address = requests.customer.address  requests.cost\_with\_delivery = requests.car.cost \* 1.03  requests.cost = requests.cost\_with\_delivery  if delivery == 'Нет' :  requests.delivery\_address = ''  requests.cost\_with\_delivery = requests.car.cost  requests.cost = requests.cost\_with\_delivery  requests.save()  return JsonResponse({'delivery': requests.delivery, 'delivery\_address': requests.delivery\_address, 'cost\_with\_delivery': requests.cost\_with\_delivery})  def order\_view(request):  requests = Request.objects.all()  current\_requests = Request.objects.filter(customer = Customer.objects.get(login = User.objects.get(login = request.user.username)))  orders = Order.objects.none()  for r in current\_requests:  if r.request\_status == 'принята':  orders |= Order.objects.filter(request = r)  context = {  'requests': requests,  'orders': orders  }  return render(request, 'order.html', context)  def account\_view(request):  form = CustomerForm(request, request.POST or None)  if form.is\_valid():  login = form.cleaned\_data['login']  email = form.cleaned\_data['email']  phone = form.cleaned\_data['phone']  fio = form.cleaned\_data['fio']  address = form.cleaned\_data['address']  person = form.cleaned\_data['person']  password = form.cleaned\_data['password\_new']  current\_customer = Customer.objects.get(login = User.objects.get(login = request.user.username))  current\_user = User.objects.get(login = request.user.username)  current\_main\_user = MainUser.objects.get(username = request.user.username)  if password:  hashpass = hashlib.sha1(password.encode('utf-8')).hexdigest()  current\_main\_user.set\_password(password)  current\_user.password = hashpass  request.user.username = login  current\_main\_user.username = login  current\_user.login = login  current\_customer.email = email  current\_customer.phone = phone  current\_customer.fio = fio  current\_customer.address = address  current\_customer.person = person  current\_main\_user.save()  current\_customer.save()  current\_user.save()  if password:  return HttpResponseRedirect(reverse('base'))  context = {  'form': form  }  return render(request, 'account.html', context)  def registration\_view(request):  form = RegistrationForm(request.POST or None)  if form.is\_valid():  new\_user = form.save(commit = False)  username = form.cleaned\_data['username']  password = form.cleaned\_data['password']  email = form.cleaned\_data['email']  phone = form.cleaned\_data['phone']  fio = form.cleaned\_data['fio']  address = form.cleaned\_data['address']  person = form.cleaned\_data['person']  new\_user.username = username  new\_user.set\_password(password)  new\_user.email = email  new\_user.save()  hashpass = hashlib.sha1(password.encode('utf-8')).hexdigest()  #new\_user\_alternative, \_ = User.objects.get\_or\_create(login = username, password = hashpass, user\_type = 'user')  new\_user\_alternative = User()  new\_user\_alternative.login = username  new\_user\_alternative.password = hashpass  new\_user\_alternative.user\_type = 'user'  new\_user\_alternative.save()  #new\_customer, \_ = Customer.objects.get\_or\_create(login = new\_user\_alternative, fio = fio, phone = phone, address = address, person = person, email = email)  if form.is\_valid():  new\_customer = Customer()  new\_customer.login = User.objects.get(login = username)  new\_customer.fio = fio  new\_customer.phone = phone  new\_customer.address = address  new\_customer.person = person  new\_customer.email = email  new\_customer.save()  login\_user = authenticate(username=username, password = password)  if login\_user:  login(request, login\_user)  return HttpResponseRedirect(reverse('base'))  context = {  'form': form  }  return render(request, 'registration.html', context)  def login\_view(request):  form = LoginForm(request.POST or None)  if form.is\_valid():  username = form.cleaned\_data['username']  password = form.cleaned\_data['password']  login\_user = authenticate(username = username, password = password)  if login\_user:  login(request, login\_user)  return HttpResponseRedirect(reverse('base'))  context = {  'form': form  }  return render(request, 'login.html', context)  def order\_by\_view(request):  requests = Request.objects.all()  order\_by\_cost = request.GET.get('order\_by\_cost')  order\_by\_cost\_distinct= request.GET.get('order\_by\_cost\_distinct')  order\_by\_year=request.GET.get('order\_by\_year')  order\_by\_year\_distinct=request.GET.get('order\_by\_year\_distinct')  news\_order = request.GET.get('news\_order')  mycars = Mycar.objects.all(order\_by\_cost, order\_by\_cost\_distinct, order\_by\_year, order\_by\_year\_distinct, news\_order)  return JsonResponse({'order\_by\_cost': order\_by\_cost, 'order\_by\_cost\_distinct': order\_by\_cost\_distinct, 'order\_by\_year': order\_by\_year, 'order\_by\_cost\_distinct': order\_by\_cost\_distinct, 'news\_order': news\_order})  def filter\_view(request):  requests = Request.objects.all()  order\_by\_cost = request.GET.get('order\_by\_cost')  order\_by\_cost\_distinct= request.GET.get('order\_by\_cost\_distinct')  order\_by\_year=request.GET.get('order\_by\_year')  order\_by\_year\_distinct=request.GET.get('order\_by\_year\_distinct')  news\_order = request.GET.get('news\_order')  first\_cost = request.GET.get('first\_cost')  last\_cost = request.GET.get('last\_cost')  first\_year = request.GET.get('first\_year')  last\_year = request.GET.get('last\_year')  filter\_mark = request.GET.get('filter\_mark')  filter\_model = request.GET.get('filter\_model')  filter\_body = request.GET.get('filter\_body')  filter\_color = request.GET.get('filter\_color')  filter\_engine = request.GET.get('filter\_engine')  filter\_KPP1 = request.GET.get('filter\_KPP1')  filter\_KPP2 = request.GET.get('filter\_KPP2')  filter\_KPP = ''  if filter\_KPP1 == None and filter\_KPP2 == None:  filter\_KPP = ''  if filter\_KPP1 == 'МКПП' and filter\_KPP2 == None:  filter\_KPP = 'МКПП'  if filter\_KPP1 == None and filter\_KPP2 == 'АКПП':  filter\_KPP = 'АКПП'  if filter\_KPP1 == 'МКПП' and filter\_KPP2 == 'АКПП':  filter\_KPP = ''  filter\_unit = request.GET.get('filter\_unit')  first\_mileage = request.GET.get('first\_mileage')  last\_mileage = request.GET.get('last\_mileage')  filter\_supplier = request.GET.get('filter\_supplier')  mycars = Mycar.objects.all(order\_by\_cost, order\_by\_cost\_distinct, order\_by\_year, order\_by\_year\_distinct, news\_order, first\_cost, last\_cost, first\_year, last\_year, filter\_mark, filter\_model, filter\_body, filter\_color, filter\_engine, filter\_KPP, filter\_unit, first\_mileage, last\_mileage, filter\_supplier)  filterset = Mycar.objects.all(None, None, None, None, "WordOrder", first\_cost, last\_cost, first\_year, last\_year, filter\_mark, filter\_model, filter\_body, filter\_color, filter\_engine, filter\_KPP, filter\_unit, first\_mileage, last\_mileage, filter\_supplier)  return JsonResponse({'first\_cost': first\_cost, 'last\_cost': last\_cost, 'first\_year': first\_year, 'last\_year': last\_year, 'filter\_mark': filter\_mark, 'filter\_model': filter\_model, 'filter\_body': filter\_body, 'filter\_color': filter\_color, 'filter\_engine': filter\_engine, 'filter\_KPP1': filter\_KPP1, 'filter\_KPP2': filter\_KPP2, 'filter\_unit': filter\_unit, 'first\_mileage': first\_mileage, 'last\_mileage': last\_mileage, 'filter\_supplier': filter\_supplier })  def mycar\_view(request, mycar\_slug):  requests = Request.objects.all()  mycar = Mycar.objects.get(slug = mycar\_slug)  #bdata = mycar.imageb  context = {  'mycar': mycar,  'requests': requests  }  return render(request, 'mycar.html', context)  def request\_view(request):  #requests = Request.objects.all()  requests = Request.objects.filter(customer = Customer.objects.get(login = User.objects.get(login = request.user.username)))  context = {  'requests': requests  }  return render(request, 'request.html', context) |

Листинг шаблона mycar

|  |
| --- |
| {% extends 'base.html' %}  {% block content %}  <div class = 'col-sm-4'>  <img src = '{{ mycar.photos.url }}' style='width: 100%; height: :500px;'>  </div>  <div class = 'col-sm-8'>  <h3 class = 'text-center'>{{ mycar.car.car }} {{ mycar.options }} {{ mycar.car\_year}} {{ mycar.car.shipper }} </h3>  <p>{{ mycar.description }}</p>  <hr>  <p>Год выпуска: {{ mycar.car\_year }}</p>  <p>Двигатель: {{ mycar.engine }}</p>  <p>Трансмиссия: {{ mycar.transmission }}</p>  <p>Привод: {{ mycar.drive\_unit }}</p>  <p>Пробег: {{ mycar.mileage }} км</p>  <p>Идентификационный номер: {{ mycar.identification\_number }}</p>  <p>Цена: <strong> {{ mycar.cost }}$</strong></p>  {% if request.user.is\_authenticated %}  <a href = '{% url "add\_to\_request" mycar\_slug=mycar.slug %}'><button class='btn btn-danger'>Оставить заявку</button></a>  {% else %}  <p class="text-center">Заявки могут оставлять только авторизированные пользователи. Пожалуйста, <a href = '{% url "login" %}'>войдите</a> или <a href = '{% url "registration" %}'>зарегистрируйтесь</a>.</p>  {% endif %}  </div>  {% endblock content %} |

Листинг процедур

|  |
| --- |
| CREATE OR REPLACE PROCEDURE INSERTCUSTOMER(p\_userid IN CUSTOMERS."LOGIN\_ID"%TYPE,  p\_fio IN CUSTOMERS."FIO"%TYPE,  p\_phone IN CUSTOMERS."PHONE"%TYPE,  p\_address IN CUSTOMERS."ADDRESS"%TYPE,  p\_email IN CUSTOMERS."EMAIL"%TYPE,  p\_person IN CUSTOMERS."PERSON"%TYPE) IS  BEGIN  INSERT INTO CUSTOMERS(LOGIN\_ID, FIO, PHONE, ADDRESS, EMAIL, PERSON) VALUES (p\_userid, p\_fio, p\_phone, p\_address, p\_email, p\_person);  COMMIT;  END;  --провектка  DECLARE  --переменные если хочешь просмотреть результат  BEGIN  --INSERTCUSTOMER(заполняешь все поля);  -- ну и DBMS\_output  END;  CREATE OR REPLACE PROCEDURE DELETECUSTOMER(p\_customerid IN CUSTOMERS."ID"%TYPE) IS  BEGIN  DELETE FROM CUSTOMERS WHERE CUSTOMERS."ID" = p\_customerid;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE UPDATECUSTOMER(p\_userid IN CUSTOMERS."LOGIN\_ID"%TYPE,  p\_fio IN CUSTOMERS."FIO"%TYPE,  p\_phone IN CUSTOMERS."PHONE"%TYPE,  p\_address IN CUSTOMERS."ADDRESS"%TYPE,  p\_email IN CUSTOMERS."EMAIL"%TYPE,  p\_person IN CUSTOMERS."PERSON"%TYPE) IS  BEGIN  UPDATE CUSTOMERS SET CUSTOMERS."FIO" = p\_fio, CUSTOMERS."PHONE" = p\_phone, CUSTOMERS."ADDRESS" = p\_address, CUSTOMERS."EMAIL" = p\_email, CUSTOMERS."PERSON" = p\_person  WHERE CUSTOMERS."LOGIN\_ID" = p\_userid;  COMMIT;  END;  --CAR  CREATE OR REPLACE PROCEDURE INSERTCAR(p\_mark IN CARS."MARK"%TYPE,  p\_model IN CARS."MODEL"%TYPE,  p\_carcass IN CARS."CARCASS"%TYPE,  p\_years\_of\_productions IN CARS."YEARS\_OF\_PRODUCTIONS"%TYPE,  p\_engines IN CARS."ENGINES"%TYPE,  p\_transmissions IN CARS."TRANSMISSIONS"%TYPE,  p\_drive\_units IN CARS."DRIVE\_UNITS"%TYPE) IS  BEGIN  INSERT INTO CARS(MARK, MODEL, CARCASS, YEARS\_OF\_PRODUCTIONS, ENGINES, TRANSMISSIONS, DRIVE\_UNITS ) VALUES (p\_mark, p\_model, p\_carcass, p\_years\_of\_productions, p\_engines, p\_transmissions, p\_drive\_units);  COMMIT;  END;  CREATE OR REPLACE PROCEDURE DELETECAR(p\_carid IN CARS."ID"%TYPE) IS  BEGIN  DELETE FROM CARS WHERE CARS."ID" = p\_carid;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE UPDATECAR(p\_carid IN CARS."ID"%TYPE,  p\_mark IN CARS."MARK"%TYPE,  p\_model IN CARS."MODEL"%TYPE,  p\_carcass IN CARS."CARCASS"%TYPE,  p\_years\_of\_productions IN CARS."YEARS\_OF\_PRODUCTIONS"%TYPE,  p\_engines IN CARS."ENGINES"%TYPE,  p\_transmissions IN CARS."TRANSMISSIONS"%TYPE,  p\_drive\_units IN CARS."DRIVE\_UNITS"%TYPE) IS  BEGIN  UPDATE CARS SET CARS."MARK" = p\_mark, CARS."MODEL" = p\_model, CARS."CARCASS" = p\_carcass, CARS."YEARS\_OF\_PRODUCTIONS" = p\_years\_of\_productions, CARS."ENGINES" = p\_engines, CARS."TRANSMISSIONS" = p\_transmissions, CARS."DRIVE\_UNITS" = p\_transmissions  WHERE CARS."ID" = p\_carid;  COMMIT;  END;  --DELIVER  CREATE OR REPLACE PROCEDURE INSERTDELIVER(p\_shipperid IN DELIVERIES."SHIPPER\_ID"%TYPE,  p\_carid IN DELIVERIES."CAR\_ID"%TYPE,  p\_date\_order IN DELIVERIES."DATE\_ORDER"%TYPE,  p\_cost IN DELIVERIES."COST"%TYPE) IS  BEGIN  INSERT INTO DELIVERIES(SHIPPER\_ID, CAR\_ID, DATE\_ORDER, COST ) VALUES (p\_shipperid, p\_carid, p\_date\_order, p\_cost);  COMMIT;  END;  CREATE OR REPLACE PROCEDURE DELETEDELIVER(p\_deliverid IN DELIVERIES."ID"%TYPE) IS  BEGIN  DELETE FROM DELIVERIES WHERE DELIVERIES."ID" = p\_deliverid;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE UPDATEDELIVER(p\_deliverid IN DELIVERIES."ID"%TYPE,  p\_shipperid IN DELIVERIES."SHIPPER\_ID"%TYPE,  p\_carid IN DELIVERIES."CAR\_ID"%TYPE,  p\_date\_order IN DELIVERIES."DATE\_ORDER"%TYPE,  p\_cost IN DELIVERIES."COST"%TYPE) IS  BEGIN  UPDATE DELIVERIES SET DELIVERIES."SHIPPER\_ID" = p\_shipperid, DELIVERIES."CAR\_ID" = p\_carid, DELIVERIES."DATE\_ORDER" = p\_date\_order, DELIVERIES."COST" = p\_cost  WHERE DELIVERIES."ID" = p\_deliverid;  COMMIT;  END;  --MYCAR  CREATE OR REPLACE PROCEDURE INSERTMYCAR(p\_carid IN MYCARS."CAR\_ID"%TYPE,  p\_car\_body IN MYCARS."CAR\_BODY"%TYPE,  p\_car\_year IN MYCARS."CAR\_YEAR"%TYPE,  p\_engine IN MYCARS."ENGINE"%TYPE,  p\_transmission IN MYCARS."TRANSMISSION"%TYPE,  p\_drive\_unit IN MYCARS."DRIVE\_UNIT"%TYPE,  p\_color IN MYCARS."COLOR"%TYPE,  p\_mileage IN MYCARS."MILEAGE"%TYPE,  p\_identification\_number IN MYCARS."IDENTIFICATION\_NUMBER"%TYPE,  p\_cost IN MYCARS."COST"%TYPE,  p\_photos IN MYCARS."PHOTOS"%TYPE,  p\_date\_of\_delivery IN MYCARS."DATE\_OF\_DELIVERY"%TYPE,  p\_description IN MYCARS."DESCRIPTION"%TYPE,  p\_options IN MYCARS."OPTIONS"%TYPE,  p\_visible IN MYCARS."VISIBLE"%TYPE,  p\_slug IN MYCARS."SLUG"%TYPE,  p\_imageb IN MYCARS."IMAGEB"%TYPE) IS  BEGIN  INSERT INTO MYCARS(CAR\_ID, CAR\_BODY, CAR\_YEAR, ENGINE, TRANSMISSION, DRIVE\_UNIT, COLOR, MILEAGE, IDENTIFICATION\_NUMBER, COST, PHOTOS, DATE\_OF\_DELIVERY, DESCRIPTION, OPTIONS, VISIBLE, SLUG, IMAGEB)  VALUES (p\_carid, p\_car\_body, p\_car\_year, p\_engine, p\_transmission, p\_drive\_unit, p\_color, p\_mileage, p\_identification\_number, p\_cost, p\_photos, p\_date\_of\_delivery, p\_description, p\_options, p\_visible, p\_slug, p\_imageb);  COMMIT;  END;  CREATE OR REPLACE PROCEDURE DELETEMYCAR(p\_carid IN MYCARS."ID"%TYPE) IS  BEGIN  DELETE FROM MYCARS WHERE MYCARS."ID" = p\_carid;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE UPDATEMYCAR(p\_carid IN MYCARS."ID"%TYPE,  p\_car\_id IN MYCARS."CAR\_ID"%TYPE,  p\_car\_body IN MYCARS."CAR\_BODY"%TYPE,  p\_car\_year IN MYCARS."CAR\_YEAR"%TYPE,  p\_engine IN MYCARS."ENGINE"%TYPE,  p\_transmission IN MYCARS."TRANSMISSION"%TYPE,  p\_drive\_unit IN MYCARS."DRIVE\_UNIT"%TYPE,  p\_color IN MYCARS."COLOR"%TYPE,  p\_mileage IN MYCARS."MILEAGE"%TYPE,  p\_identification\_number IN MYCARS."IDENTIFICATION\_NUMBER"%TYPE,  p\_cost IN MYCARS."COST"%TYPE,  p\_photos IN MYCARS."PHOTOS"%TYPE,  p\_date\_of\_delivery IN MYCARS."DATE\_OF\_DELIVERY"%TYPE,  p\_description IN MYCARS."DESCRIPTION"%TYPE,  p\_options IN MYCARS."OPTIONS"%TYPE,  p\_visible IN MYCARS."VISIBLE"%TYPE,  p\_slug IN MYCARS."SLUG"%TYPE,  p\_imageb IN MYCARS."IMAGEB"%TYPE) IS  BEGIN  UPDATE MYCARS SET MYCARS."CAR\_ID" = p\_car\_id, MYCARS."CAR\_BODY" = p\_car\_body, MYCARS."CAR\_YEAR" = p\_car\_year, MYCARS."ENGINE" = p\_engine, MYCARS."TRANSMISSION" = p\_transmission, MYCARS."DRIVE\_UNIT" = p\_drive\_unit,  MYCARS."COLOR" = p\_color, MYCARS."MILEAGE" = p\_mileage, MYCARS."IDENTIFICATION\_NUMBER" = p\_identification\_number, MYCARS."COST" = p\_cost, MYCARS."PHOTOS" = p\_photos, MYCARS."DATE\_OF\_DELIVERY" = p\_date\_of\_delivery,  MYCARS."DESCRIPTION" = p\_description, MYCARS."OPTIONS" = p\_options, MYCARS."VISIBLE" = p\_visible, MYCARS."SLUG" = p\_slug, MYCARS."IMAGEB" = p\_imageb  WHERE MYCARS."ID" = p\_carid;  COMMIT;  END;  --ORDER  CREATE OR REPLACE PROCEDURE INSERTORDER(p\_request\_id IN ORDERS."REQUEST\_ID"%TYPE,  p\_order\_status IN ORDERS."ORDER\_STATUS"%TYPE,  p\_start\_date IN ORDERS."START\_DATE"%TYPE,  p\_end\_date IN ORDERS."END\_DATE"%TYPE) IS  BEGIN  INSERT INTO ORDERS(REQUEST\_ID, ORDER\_STATUS, START\_DATE, END\_DATE ) VALUES (p\_request\_id, p\_order\_status, p\_start\_date, p\_end\_date);  COMMIT;  END;  CREATE OR REPLACE PROCEDURE DELETEORDER(p\_orderid IN ORDERS."ID"%TYPE) IS  BEGIN  DELETE FROM ORDERS WHERE ORDERS."ID" = p\_orderid;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE UPDATEORDER(p\_orderid IN ORDERS."ID"%TYPE,  p\_request\_id IN ORDERS."REQUEST\_ID"%TYPE,  p\_order\_status IN ORDERS."ORDER\_STATUS"%TYPE,  p\_start\_date IN ORDERS."START\_DATE"%TYPE,  p\_end\_date IN ORDERS."END\_DATE"%TYPE) IS  BEGIN  UPDATE ORDERS SET ORDERS."REQUEST\_ID" = p\_request\_id, ORDERS."ORDER\_STATUS" = p\_order\_status, ORDERS."START\_DATE" = p\_start\_date, ORDERS."END\_DATE" = p\_start\_date  WHERE ORDERS."ID" = p\_orderid;  COMMIT;  END;  --REQUEST  CREATE OR REPLACE PROCEDURE INSERTREQUEST(p\_car\_id IN REQUESTS."CAR\_ID"%TYPE,  p\_date\_request IN REQUESTS."DATE\_REQUEST"%TYPE,  p\_customer\_id IN REQUESTS."CUSTOMER\_ID"%TYPE,  p\_request\_status IN REQUESTS."REQUEST\_STATUS"%TYPE,  p\_delivery IN REQUESTS."DELIVERY"%TYPE,  p\_delivery\_address IN REQUESTS."DELIVERY\_ADDRESS"%TYPE,  p\_cost IN REQUESTS."COST"%TYPE) IS  BEGIN  INSERT INTO REQUESTS(CAR\_ID, DATE\_REQUEST, CUSTOMER\_ID, REQUEST\_STATUS, DELIVERY, DELIVERY\_ADDRESS, COST)  VALUES (p\_car\_id, p\_date\_request, p\_customer\_id, p\_request\_status, p\_delivery, p\_delivery\_address, p\_cost);  COMMIT;  END;  CREATE OR REPLACE PROCEDURE DELETEREQUEST(p\_requestid IN REQUESTS."ID"%TYPE) IS  BEGIN  DELETE FROM REQUESTS WHERE REQUESTS."ID" = p\_requestid;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE UPDATEREQUEST(p\_requestid IN REQUESTS."ID"%TYPE,  p\_car\_id IN REQUESTS."CAR\_ID"%TYPE,  p\_date\_request IN REQUESTS."DATE\_REQUEST"%TYPE,  p\_customer\_id IN REQUESTS."CUSTOMER\_ID"%TYPE,  p\_request\_status IN REQUESTS."REQUEST\_STATUS"%TYPE,  p\_delivery IN REQUESTS."DELIVERY"%TYPE,  p\_delivery\_address IN REQUESTS."DELIVERY\_ADDRESS"%TYPE,  p\_cost IN REQUESTS."COST"%TYPE) IS    BEGIN  UPDATE REQUESTS SET REQUESTS."CAR\_ID" = p\_car\_id, REQUESTS."DATE\_REQUEST" = p\_date\_request, REQUESTS."CUSTOMER\_ID" = p\_customer\_id, REQUESTS."REQUEST\_STATUS" = p\_request\_status, REQUESTS."DELIVERY" = p\_delivery,  REQUESTS."DELIVERY\_ADDRESS" = p\_delivery\_address, REQUESTS."COST" = p\_cost  WHERE REQUESTS."ID" = p\_requestid;  COMMIT;  END;  --SUPPLIERS  CREATE OR REPLACE PROCEDURE INSERTSUPPLIER(p\_name IN SUPPLIERS."NAME"%TYPE,  p\_address IN SUPPLIERS."ADDRESS"%TYPE,  p\_phone IN SUPPLIERS."PHONE"%TYPE,  p\_email IN SUPPLIERS."EMAIL"%TYPE) IS  BEGIN  INSERT INTO SUPPLIERS(NAME, ADDRESS, PHONE, EMAIL ) VALUES (p\_name, p\_address, p\_phone, p\_email);  COMMIT;  END;  CREATE OR REPLACE PROCEDURE DELETESUPPLIER(p\_supplierid IN SUPPLIERS."ID"%TYPE) IS  BEGIN  DELETE FROM SUPPLIERS WHERE SUPPLIERS."ID" = p\_supplierid;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE UPDATESUPPLIER(p\_supplierid IN SUPPLIERS."ID"%TYPE,  p\_name IN SUPPLIERS."NAME"%TYPE,  p\_address IN SUPPLIERS."ADDRESS"%TYPE,  p\_phone IN SUPPLIERS."PHONE"%TYPE,  p\_email IN SUPPLIERS."EMAIL"%TYPE) IS    BEGIN  UPDATE SUPPLIERS SET SUPPLIERS."NAME" = p\_name, SUPPLIERS."ADDRESS" = p\_address, SUPPLIERS."PHONE"= p\_phone, SUPPLIERS."EMAIL" = p\_email  WHERE SUPPLIERS."ID" = p\_supplierid;  COMMIT;  END;  --USER  CREATE OR REPLACE PROCEDURE INSERTUSER(p\_login IN USERS."LOGIN"%TYPE,  p\_password IN USERS."PASSWORD"%TYPE,  p\_user\_type IN USERS."USER\_TYPE"%TYPE) IS  BEGIN  INSERT INTO USERS(LOGIN, PASSWORD, USER\_TYPE ) VALUES (p\_login, p\_password, p\_user\_type);  COMMIT;  END;  CREATE OR REPLACE PROCEDURE DELETEUSER(p\_userid IN USERS."ID"%TYPE) IS  BEGIN  DELETE FROM USERS WHERE USERS."ID" = p\_userid;  COMMIT;  END;  CREATE OR REPLACE PROCEDURE UPDATEUSER(p\_userid IN USERS."ID"%TYPE,  p\_login IN USERS."LOGIN"%TYPE,  p\_password IN USERS."PASSWORD"%TYPE,  p\_user\_type IN USERS."USER\_TYPE"%TYPE) IS  BEGIN  UPDATE USERS SET USERS."LOGIN" = p\_login, USERS."PASSWORD" = p\_password, USERS."USER\_TYPE" = p\_user\_type  WHERE USERS."ID" = p\_userid;  COMMIT;  END;  create or replace directory XML\_DIR as 'd:\Dokuments\XML'  --EXPORT XML  create or replace procedure table\_to\_xml\_file(table\_name in varchar2) as  ctx dbms\_xmlgen.ctxhandle;  clb clob;  file utl\_file.file\_type;  buffer varchar2(32767);  position pls\_integer := 1;  chars pls\_integer := 32767;  begin  ctx := dbms\_xmlgen.newcontext('select \* from "' || table\_name || '"');  dbms\_xmlgen.setrowsettag(ctx, 'RECORDS');  dbms\_xmlgen.setrowtag(ctx, 'RECORD');  select xmlserialize(document  xmlelement("XML",  xmlelement(evalname(table\_name),  dbms\_xmlgen.getxmltype(ctx)))  indent size = 2)  into clb  from dual;  dbms\_xmlgen.closecontext(ctx);  file := utl\_file.fopen('XML\_DIR', table\_name || '.xml', 'w', 32767);  while position < dbms\_lob.getlength(clb) loop  dbms\_lob.read(clb, chars, position, buffer);  utl\_file.put(file, buffer);  utl\_file.fflush(file);  position := position + chars;  end loop;  utl\_file.fclose(file);  commit;  end table\_to\_xml\_file;  --IMPORT  DECLARE  xml\_file BFILE;  xml\_data CLOB;  BEGIN  xml\_file := BFILENAME ('XML\_DIR', 'USERS.xml');  DBMS\_LOB.createtemporary (xml\_data, TRUE, DBMS\_LOB.SESSION);  DBMS\_LOB.fileopen (xml\_file, DBMS\_LOB.file\_readonly);  DBMS\_LOB.loadfromfile (xml\_data, xml\_file, DBMS\_LOB.getlength(xml\_file));  DBMS\_LOB.fileclose (xml\_file);  INSERT INTO USERS (login,password,user\_type)  SELECT ExtractValue(Value(x),'//LOGIN') as login,  ExtractValue(Value(x),'//PASSWORD') as password, ExtractValue(Value(x),'//USER\_TYPE') as user\_type  FROM TABLE(XMLSequence(Extract(XMLType(xml\_data),'/XML/USERS/RECORDS/RECORD'))) x;  DBMS\_OUTPUT.PUT\_LINE( SQL%ROWCOUNT || ' rows inserted.' );  DBMS\_LOB.freetemporary (xml\_data);  COMMIT;  END; |