

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ
по дисциплине «Введение в нереляционные базы данных»
Тема: Каталог объявлений о продаже автомобилей

Студент гр. 0303	_____	Архипов В.А.
Студент гр. 0303	_____	Бодунов П.А.
Студент гр. 0303	_____	Калмак Д.А.
Преподаватель	_____	Заславский М.М.

Санкт-Петербург
2023

ЗАДАНИЕ

Студенты

Архипов В.А.

Бодунов П.А.

Калмак Д.А.

Группа 0303

Тема задания: Каталог объявлений о продаже автомобилей

Исходные данные:

Задача - сервис, позволяющий размещать объявления о покупке или продаже автомобилей, связывать продавцов и покупателей. Необходимые (но недостаточные фичи) - аккаунты продавцов и покупателей, рейтинги и отзывы, страница подробных данных об автомобиле (пробег, номера двигателя ... , марка машины, год, фото). Пользователи - администраторы, клиенты.

Содержание пояснительной записки:

“Содержание”

“Введение”

“Сценарий использования”

“Модель данных”

“Разработанное приложения”

“Заключение”

“Приложение А. Документация по сборке и развертыванию приложения”

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 21.12.2023

Дата сдачи реферата: 21.12.2023

Дата защиты реферата: 21.12.2023

Студент 0303	_____	Архипов В.А.
Студент 0303	_____	Бодунов П.А.
Студент 0303	_____	Калмак Д.А.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В рамках данного курса предполагалось разработать в команде приложение на одну из предложенных тем. Была выбрана тема: Каталог объявлений о продаже автомобилей. Для выполнения задания предлагается использовать СУБД MongoDB.

Найти исходный код и всю дополнительную информацию можно по ссылке: <https://github.com/moevm/nosql2h23-auto-trade>.

SUMMARY

Within the framework of this course it was supposed to develop in a team an application on one of the proposed topics. The topic was chosen: Catalog of car ads for sale. To perform the task it is suggested to use MongoDB DBMS.

You can find the source code and all additional information at the link: <https://github.com/moevm/nosql2h23-auto-trade>.

СОДЕРЖАНИЕ

Введение	6
1. Сценарии использования	7
1.1. Макет UI	7
1.2. Сценарии использования для задачи	7
2. Модель данных	11
2.1. Нереляционная модель данных	11
2.2. Аналог модели данных для SQL СУБД	18
2.3. Сравнение моделей	25
3. Разработанное приложение	27
3.1. Описание	27
3.2. Используемые технологии	27
3.3. Снимки экрана приложения	27
Заключение	32
Список использованных источников	33
Приложение А. Документация по сборке и развертыванию приложения	34

ВВЕДЕНИЕ

Цель работы – изучить один из типов нереляционных баз данных и использовать его в проекте.

Было разработано веб-приложение “Каталог объявлений о продаже автомобилей”, которое позволит хранить информацию об объявлениях, автомобилях, пользователях, а также переписку пользователей.

1. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

1.1. Макет UI

Разработанный макет приложения доступен по ссылке:
https://miro.com/app/board/uXjVNdtQIE8=/?share_link_id=186421136561



Рисунок 1 – Макет UI

1.2. Сценарии использования

Сценарий использования “Поиск автомобиля”.

Действующее лицо: Покупатель

Основной сценарий:

1. Покупатель выставил нужные фильтры: модель, год, цвет, кузов, пробег, двигатель, коробка, привод, руль – на “Главной” для объявлений.
2. Покупатель открывает объявление нажатием на название или фотографию.
3. Покупатель просматривает характеристики автомобиля, приведенные на странице объявления, и фотографию автомобиля в более крупном размере.

4. Автомобиль не понравился и пользователь возвращается к странице с объявлениями на “Главную”.
5. Переход на шаг 1.

Альтернативный сценарий:

1. Покупателю понравился автомобиль.
2. Поиск закончен.

Сценарий использования “Покупка автомобиля”.

Действующее лицо: Покупатель

Основной сценарий:

1. Покупатель нашел автомобиль и хочет его купить - он нажимает “Купить” на странице объявления.
2. Открывается страница для общения с продавцом в рамках этого объявления.
3. Покупатель договаривается о покупке с Продавцом и покупает автомобиль.

Альтернативный сценарий:

1. Покупатель не смог договориться о покупке автомобиля с Продавцом и переходит на “Главную”.

Сценарий использования “Продажа автомобиля”.

Действующее лицо: Продавец

Основной сценарий:

1. Продавец переходит на страницу “Мои объявления”
2. Нажатием на кнопку “Создать объявления” открывается страница для создания объявления.

3. Продавец заполняет информацию об автомобиле: модель, год, цвет, кузов, пробег, двигатель, коробка, привод, руль, цена – и загружает его фотографию.
4. После заполнения Продавец нажимает на кнопку “Создать” и переходит на “Главную”.
5. В личном профиле “Мой аккаунт” Продавец отслеживает сообщения, поступающие по вопросу продажи автомобиля
6. Продавец нажимает на диалог и открывается страница для общения с Покупателем.
7. Продавец не смог договориться о продаже с Покупателем и переходит на “Главную”.
8. Переход на шаг 5.

Альтернативный сценарий:

1. Продавец договорился о продаже с Покупателем и продал автомобиль.
2. После продажи автомобиля Продавец переходит в “Мои объявления” и удаляет объявление автомобиля, который он продал.

Сценарий использования “Валидация объявлений”.

Действующее лицо: Администратор

Основной сценарий:

1. Администратор переходит на страницу “Проверка объявлений”.
2. Администратор выбирает объявление, которое хочет провалидировать.
3. Администратор открывает страницу объявления нажатием на название или фотографию и просматривает информацию об автомобиле: модель, год, цвет, кузов, пробег, двигатель, коробка, привод, руль, цена – и фотографию автомобиля.

4. После проверки Администратор одобряет или отклоняет объявление нажатием на кнопки “Опубликовать” или “Отклонить” соответственно.
5. После нажатия на кнопку Администратор переходит на страницу с объявлениями, ожидающими валидации, и переходит на шаг 2.

Альтернативный сценарий:

1. После нажатия на кнопку Администратор переходит на страницу с объявлениями и объявлений на проверку больше нет.
2. Валидация закончена.

2. МОДЕЛЬ ДАННЫХ

2.1. Нереляционная модель данных

Графическое представление модели на рис. 2.

user_id	login	password	user_status	name	rating	create_date	reviews: [name, mark, text, date]	dialogs: [dialog_id, ad_id, messages: [user_id, text, timestamp]]	ads: [ad_id, photo, brand, model, year, color, body, mileage, engine, transmission, drive, helm, price, create_date, edit_date, view, status]
---------	-------	----------	-------------	------	--------	-------------	---	--	---

Рисунок 2 - Модель данных NoSQL

Описание назначений коллекций, типов данных и сущностей

Коллекция Users

- user_id типа ObjectId для хранения идентификационного номера пользователя
- login типа string для логина пользователя
- password типа string для пароля соответствующего пользователя
- user_status типа string для хранения статуса пользователя.
- name типа string для хранения имени пользователя
- rating типа float для средней оценки пользователя по отзывам
- create_date типа string для даты создания аккаунта
- структура reviews для отзывов, состоящая из трех полей:
 - name типа string для хранения имени пользователя, который написал отзыв
 - mark типа int для хранения оценки
 - text типа string для текста отзыва
 - date типа string для даты создания отзыва
- структура dialogs для диалогов, состоящая из трех полей:
 - dialog_id типа ObjectId для хранения идентификатора диалога
 - ad_id типа ObjectId для хранения идентификационного номера

- структура messages для сообщений в диалоге, состоящая из двух полей:
 - user_id типа ObjectId для хранения идентификационного номера пользователя, отправившего сообщение
 - text типа string для хранения текста сообщения
 - timestamp типа string для времени создания сообщения
- структура ads для объявлений, состоящая из 17 полей:
 - ad_id типа ObjectId для хранения идентификационного номера
 - photo типа string для хранения пути до файла с фотографиями
 - brands типа string для хранения марки автомобиля
 - model типа string для хранения модели автомобиля
 - year типа int для года выпуска автомобиля
 - color типа string для цвета автомобиля
 - body типа string для вида кузова автомобиля
 - mileage типа int для пробега автомобиля
 - engine типа string для типа двигателя автомобиля
 - transmission типа string типа коробки передач автомобиля
 - drive типа string для хранения типа привода автомобиля
 - helm типа string для хранения типа руля автомобиля
 - price типа int для хранения цены автомобиля
 - create_date типа string для хранения даты создания объявления
 - edit_date типа string для хранения даты последнего редактирования объявления
 - view типа int для хранения количества просмотров объявления
 - status типа string для хранения статуса проверки объявления

Оценка удельного объема информации, хранимой в модели

- user_id – ObjectId 12 байт
- login – string 10 символов – 20 байт

- password – string 20 символов – 40 байт
- user_status - string 12 символов - 24 байта
- name – string 10 символов – 20 байт
- rating – double – 8 байт
- create_date – string 10 символов – 20 байт
- reviews – четыре поля: 2 string по 10 символов, int 4 байта, string 50 символов – 144 байта
- dialogs – четыре поля: 2 ObjectId по 12 байт, структура messages: один ObjectId 12 байт, две строки типа string по 40 байт каждая – 116 байтов
- ads – 4 int по 4 байта, 1 ObjectId - 12 байт, 12 string по 10 символов – 268 байт

Неизменная по размеру область памяти (user_id, login, password, user_status, name, rating, create_date): 144 байт

Создание объявления подразумевает в дальнейшем диалог с покупателем. Среднее число сообщений в диалоге шесть. Итого имеем $268 + 24 + 92 \cdot 6 = 844$ байт.

Покупка автомобиля подразумевает диалог с покупателем и отзыв о покупке. Среднее число сообщений в диалоге шесть. Итого имеем $24 + 92 \cdot 6 + 124 = 700$ байт.

За основной элемент примем объявление, его количество будет N . С объявлением идет отзыв, диалог между двумя пользователями по шесть сообщений на каждого. Память одного пользователя равна неизменной части, константа. Поэтому её можно отбросить. Исходя из предыдущих расчетов: $844 + 700 - 24$ (из-за повторного диалога) = 1520 байта

Таким образом, $1520 \cdot N$.

Избыточность модели

Фактический объем модели определяется формулой $1520 \cdot N$.

Избыточными данными примем дублирующуюся информацию, а именно повторяющийся `ad_id` типа `ObjectId`, поле `name` типа `string` 10 символов и `user_id` типа `ObjectId`, использующийся в структуре `messages`.

“Чистый” объем данных: $1520 - 12 - (20 + 12) \cdot 12$ (из-за 12 сообщений) = 1124 байта.

Таким образом, $1124 \cdot N$.

Отношение между фактическим объемом модели и “чистым” объемом данных: $\frac{1520 \cdot N}{1124 \cdot N} = 1.352$

Направление роста модели при увеличении количества объектов каждой сущности

Модель состоит из одной коллекции. Рост одного поля не влияет на рост других. При добавлении нового пользователя в систему создаётся новый документ в коллекции `Users` с пустыми полями `reviews`, `dialogs` и `ads`. При отправлении первого сообщения у второго участника диалога в документе в поле `dialogs` записывается идентификатор диалога. При создании отзыва на пользователя `user` в документе `user` в поле `reviews` добавляется новая запись – оставленный отзыв. При создании пользователем нового объявления в поле `ads` данного пользователя добавляется созданное объявление.

Запросы к модели, с помощью которых реализуются сценарии использования

Текст запросов

Поиск автомобиля

- Поиск с фильтром:

```
db.users.find({ads.model: "Mercedes", ads.status : "Опубликован"})
```

- Для отображения страницы с объявлением требуется вся информация об объявлении:

```
db.users.find({ads.ad_id : 1})
```

Покупка автомобиля

- Поиск диалога между продавцом и покупателем по конкретному объявлению:

```
db.users.find({dialogs.ad_id : 7, user_id : 4})
```

- Если такой диалог существует, то необходимо получить все сообщения из него:

```
db.users.find({dialogs.dialog_id : 11})
```

- Если такого диалога нет, то создадим его и зафиксируем первое отправленное сообщение:

```
db.users.updateOne(
  {user_id: 2},
  {
    $push: {
      dialogs: {
        ad_id: 3,
        messages: [text: "Обмен на стаю собак интересен?", timestamp:
"2022-03-29T12:04:06Z"]
      }
    }
  }
)
```

```
db.users.updateOne(
  {user_id: 1},
  {
    $push: {
      dialogs: {
        ad_id: 3
      }
    }
  }
)
```

Продажа автомобиля

- **Объявления пользователя:**

```
db.users.find({user_id: 777})
```

- **Добавление объявления:**

```
db.users.updateOne(  
  {user_id: 777},  
  {  
    $push: {  
      ads: {  
        photo: `./cars_photos/sellBestCarEver`,  
        brand: Mercedes,  
        model: AMG-GT,  
        year: 2018,  
        color: черный,  
        body: седан,  
        mileage: 10000,  
        engine: бензин,  
        transmission: автомат,  
        drive: задний,  
        helm: левый,  
        price: 10000000  
        create_date: "08.08.2019"  
        edit_date: NULL  
        view: 0  
        status: "Проверка"  
      }  
    }  
  }  
)
```

- На странице “Мой аккаунт” продавец должен видеть диалоги, поэтому нужен запрос на все диалоги продавца:

```
db.users.find({user_id : 777})
```

- Для диалога между продавцом и покупателем нужно получить все сообщения, поэтому нужен запрос на все сообщения для данного диалога:

```
db.users.find({dialogs.dialog_id : 1})
```

Валидация объявлений

- Для раздела “Проверка объявлений” требуется запрос на все объявления со статусом “Проверка”:

```
db.users.find({ads.status: "Проверка"})
```

- Для отображения страницы с объявлением требуется вся информация об объявлении:

```
db.users.find({ads.ad_id: 1})
```

- При нажатии “Опубликовать” объявление меняет статус с “Проверка” на “Опубликовано”:

```
db.users.updateOne(
  {ads.ad_id: 1},
  {$set: {ads.status: "Опубликовано"}}
)
```

Количество запросов для совершения юзкейсов

- Поиск автомобиля: 2 запроса
- Покупка автомобиля: 2/3 запроса в зависимости от существования диалога
- Продажа автомобиля: 4 запроса
- Валидация объявлений: 3 запроса

Количество задействованных коллекций

- Одна коллекция: Users.

Пример данных представлен на рис. 3.

1	masterpiece	nosqlsbetterthansql	Пользователь	Анастасия	4.5	2022-03-29	reviews: [Рудольф: 5, Девушка супер. Машина тоже ничего!, 2022-03-29]	dialogs: [1, 1, messages: [1, Обмен на стаю собак интересней!, 2022-03-29T12:04:06Z]]	ads: [1, .cars_photos/sellBestCarEver, Mercedes, AMG-GT, 2018, черный, седан, 10000, бензин, автомат, задний, левый, 10000000, 08-08-2019, NULL, 808, Проверка]
---	-------------	---------------------	--------------	-----------	-----	------------	---	---	---

Рисунок 3 - Пример данных NoSQL

2.2. Аналог модели данных для SQL СУБД

Графическое представление модели изображено на рис. 4.

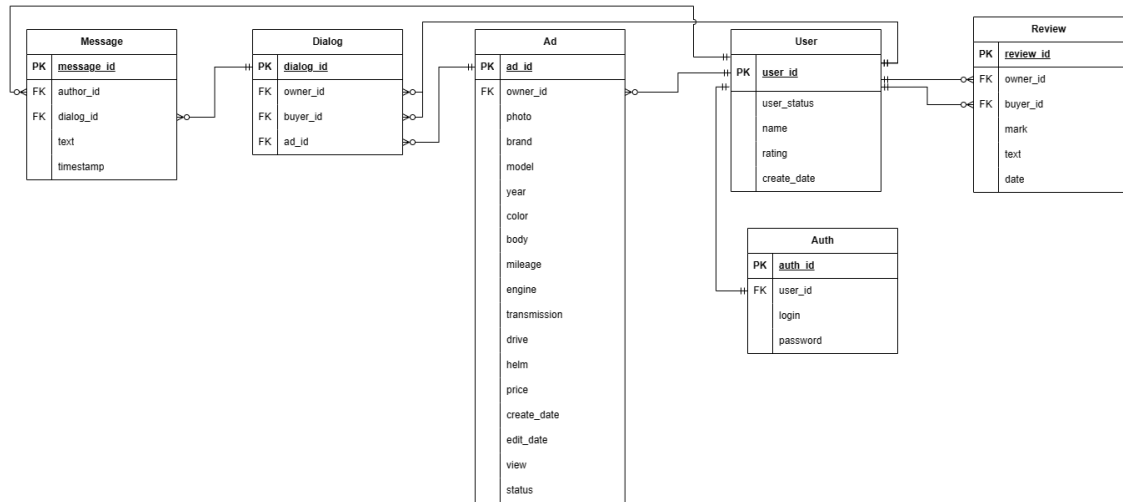


Рисунок 4 – модель реляционной базы данных

Описание назначений коллекций, типов данных и сущностей

User - таблица пользователей.

Содержит:

- user_id типа int для хранения идентификационного номера пользователя
- name типа text для хранения имени пользователя
- rating типа float для средней оценки пользователя по отзывам
- create_date типа text для даты создания аккаунта
- user_status типа text для хранения статуса пользователя (администратор, обычный пользователь)

Auth - таблица авторизации.

Содержит:

- auth_id типа int для хранения идентификационного номера авторизационных данных

- user_id типа int для хранения идентификационного номера пользователя
- login типа text для логина пользователя
- password типа text для пароля соответствующего пользователя

Review - таблица отзывов.

Содержит:

- review типа int для хранения идентификационного номера отзыва
- owner_id типа int для хранения идентификационного номера пользователя, которому отзыв предназначается
- buyer_id типа int для хранения идентификационного номера пользователя, которым отзыв написан
- mark типа int для оценки в отзыве
- text типа text для текста отзыва
- date типа text для даты создания отзыва

Ad – таблица объявлений.

Содержит:

- ad_id типа int для хранения идентификационного номера
- owner_id типа int для хранения идентификационного номера владельца объявления
- photo типа text для хранения пути до файла с фотографиями
- brand типа text для хранения марки автомобиля
- model типа text для хранения модели автомобиля
- year типа int для года выпуска автомобиля
- color типа text для цвета автомобиля
- body типа text для вида кузова автомобиля

- mileage типа int для пробега автомобиля
- engine типа text для типа двигателя автомобиля
- transmission типа text типа коробки передач автомобиля
- drive типа text для типа привода автомобиля
- helm типа text для расположения руля автомобиля
- price типа int для хранения цены автомобиля
- create_date типа text для хранения даты создания объявления
- edit_date типа text для хранения даты последнего редактирования объявления
- view типа int для хранения количества просмотров объявления
- status типа text для хранения статуса объявления (опубликовано, ожидает модерации)

Message – таблица сообщений.

Содержит:

- message_id типа int для хранения идентификатора сообщения
- author_id типа int для хранения идентификатора автора сообщения
- dialog_id типа int для хранения идентификатора диалога, к которому принадлежит данное сообщение
- text типа text для хранения текста сообщения
- timestamp типа text для времени создания сообщения

Dialog – таблица диалогов.

Содержит:

- dialog_id типа int для хранения идентификатора диалога
- owner_id типа int для хранения идентификатора владельца автомобиля

- buyer_id типа int для хранения идентификатора человека, которых хочет обсудить покупку авто с продавцом
- ad_id типа int для хранения идентификатора объявления, по которому происходит беседа

Оценка удельного объема информации, хранимой в модели

Auth – два int по 4 байта - 8 байт, два text один на 10 символов, другой на 20 символов, один символ весит 2 байт: 20 и 40 байт. Суммарно 68 байт

User – int 4 байта, float 4 байта, два text на 10 символов - 40 байт, text на 12 символов - 24 байта. Суммарно 72 байта

Review – четыре int по 4 байта - 16 байт, text на 50 символов 100 байт, text на 10 символов 20 байт. Суммарно 136 байта

Ad - шесть int по 4 байта - 24 байта, двенадцать text по 10 символов 240 байт. Суммарно 264 байта

Dialog – четыре int по 4 байта – 16 байт.

Message – три int по 4 байта - 12 байт, и один text средней длины 20 символов – 40 байт, и text 20 символов - 40 байт. Суммарно 92 байта.

Неизменная по размеру область памяти: поскольку пользователь имеет данные для входа в систему, то при создании пользователя автоматически создаётся запись в таблице User и в таблице Auth – по одной записи и там, и там. Итого имеем $68 + 72 = 140$ байт.

Создание объявления подразумевает в дальнейшем диалог с покупателем. Среднее число сообщений в диалоге шесть. Итого имеем $264 + 16 + 552 = 832$ байта. Покупка автомобиля подразумевает диалог с покупателем и отзыв о покупке. Среднее число сообщений в диалоге шесть. Итого имеем $6 * 92 + 136 + 16 = 704$ байт.

За основной элемент примем объявление, его количество будет N. С объявлением идет отзыв, диалог между двумя пользователями по шесть сообщений на каждого. Память одного пользователя равна неизменной части, константа. Поэтому её можно отбросить. Исходя из предыдущих расчетов: $832 + 704 - 16$ (из-за повторного диалога) = 1520 байта

Таким образом, $1520 * N$.

Избыточность модели

Фактический объем модели определяется формулой $1520 * N$.

Избыточными данными примем дублирующуюся информацию, а именно внешние ключи (FK) в сущностях Ad, Dialog, Message, Review. Всего внешних ключей 8, которые имеют тип int.

“Чистый” объем данных: $1520 - 6 * 4 - 2 * 4 * 12$ (из-за 12 сообщений) = 1400 байт

Таким образом, $1400 * N$.

Отношение между фактическим объемом модели и “чистым” объемом данных: $\frac{1520N}{1400N} = 1.086$

Направление роста модели при увеличении количества объектов каждой сущности

При создании пользователя User у него есть личные регистрационные данные, то есть растет Auth. В остальных таблицах рост одной сущности не влияет на рост других.

Запросы к модели, с помощью которых реализуются сценарии использования

Поиск автомобиля

- Поиск с фильтром:

```
SELECT * FROM Ad
WHERE model = "Mercedes" AND status = "Опубликовано"
```

- Для отображения страницы с объявлением требуется вся информация об объявлении:

```
SELECT * FROM Ad
WHERE ad_id = 1
```

Покупка автомобиля

- Поиск диалога между продавцом и покупателем по конкретному объявлению:

```
SELECT dialog_id FROM Dialog
WHERE buyer_id = 4 AND ad_id = 7
```

- Если такой диалог существует, то необходимо получить все сообщения из него:

```
SELECT * FROM Message
WHERE dialog_id = 11
```

- Если такого диалога нет, то создадим его и зафиксируем первое отправленное сообщение:

```
INSERT INTO Dialog(owner_id, buyer_id, ad_id)
VALUES (1, 2, 3);
```

```
INSERT INTO Message(author_id, dialog_id, text, timestamp)
VALUES (1, (SELECT dialog_id FROM Dialog WHERE owner_id = 1 AND buyer_id = 2 AND
ad_id = 3), "Обмен на стаю собак интересен?", "2022-03-29T12:04:06Z");
```

Продажа автомобиля

- **Объявления пользователя:**

```
SELECT * FROM Ad
WHERE owner_id = 777
```

- **Добавление объявления:**

```
INSERT Ad(owner_id, photo, brand, model, year, color, body, mileage, engine,
transmission, drive, helm, price, create_date, edit_date, view, status)
VALUES (777,    "./cars_photos/sellBestCarEver",    "Mercedes",    "AMG-GT",    2018,
"черный",    "седан",    10000,    бензин,    "автомат",    "задний",    "левый",    10000000,
"08.08.2019", NULL, 0, "Проверка")
```

- На странице “Мой аккаунт” продавец должен видеть диалоги, поэтому нужен запрос на все диалоги продавца:

```
SELECT * FROM Dialog
WHERE owner_id = 777 or buyer_id = 777
```

- Для диалога между продавцом и покупателем нужно получить все сообщения, поэтому нужен запрос на все сообщения для данного диалога:

```
SELECT * FROM Message
WHERE dialog_id = 1
```

Валидация объявлений

- Для раздела “Проверка объявлений” требуется запрос на все объявления со статусом “Проверка”:

```
SELECT * FROM Ad
WHERE status = "Проверка"
```

- Для отображения страницы с объявлением требуется вся информация об объявлении:

```
SELECT * FROM Ad
WHERE ad_id = 1
```

- При нажатии “Опубликовать” объявление меняет статус с “Проверка” на “Опубликовано”:

```
UPDATE Ad
SET status = "Опубликовано"
WHERE ad_id = 1
```


Количество запросов для совершения юзкейсов

- Поиск автомобиля: 2 запроса
- Покупка автомобиля: 2/4 запроса в зависимости от существования диалога
- Продажа автомобиля: 4 запроса
- Валидация объявлений: 3 запроса

Количество задействованных коллекций

- Три сущности: Ad, Dialog и Message.

Пример данных представлен на рисунке 5.

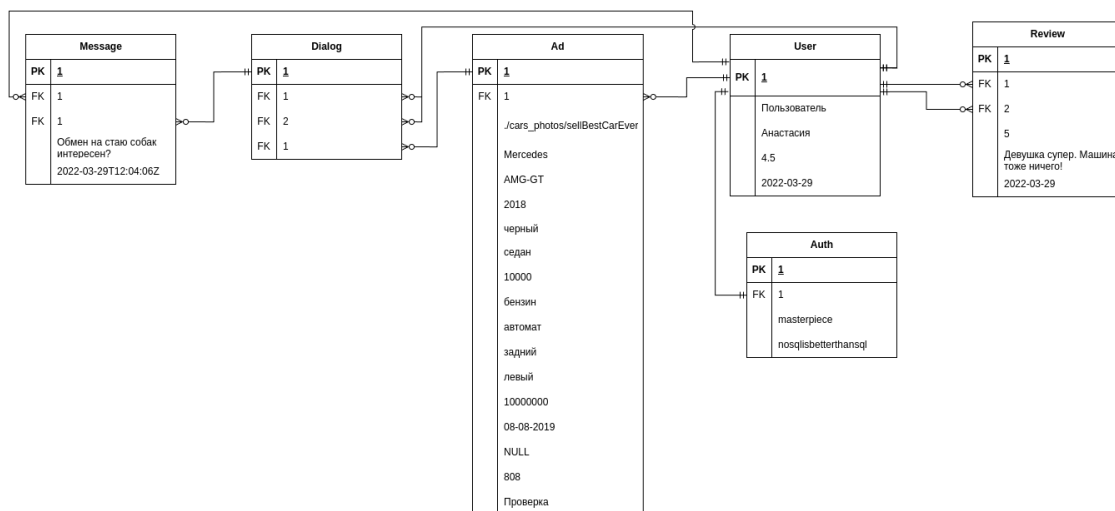


Рисунок 5 – пример данных реляционной СУБД

2.3. Сравнение моделей

Удельный объем информации:

За основной элемент было принято объявление, его количество N .

Полученная формула в NoSQL: $1520 \cdot N$.

В SQL: $1520 \cdot N$.

Данные объемы равны.

Однако чистый объем данных в NoSQL занимает 1124 байта, а в SQL - 1400 байта, то есть чистая модель данных в SQL занимает в $\frac{1400}{1124} = 1.25$ раз больше.

Запросы по отдельным юзкейсам:

Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров

- Поиск автомобиля: 2 запроса - количество запросов одинаково
- Покупка автомобиля: 2/3 и 2/4 запроса в зависимости от существования диалога, то есть NoSQL при отсутствии диалога работает на один запрос меньше
- Продажа автомобиля: 4 запроса - количество запросов одинаково
- Валидация объявлений: 3 запроса - количество запросов одинаково

Количество задействованных коллекций:

- В SQL задействовано три сущности, а в NoSQL задействована одна коллекция.

3. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

3.1. Описание

Backend реализован при помощи фреймворка express на базе nodejs на языке программирования JavaScript с использованием базы данных MongoDB. Для обработки собственных post-запросов и удаления данных использован jQuery. Для загрузки и редактирования фотографий использован multer.

Frontend реализован при помощи шаблонизатора Pug и стилей CSS.

3.2. Используемые технологии

База данных: MongoDB

Backend: nodejs, express, JavaScript, jQuery, multer.

Frontend: Pug, CSS.

Контейнеры:

Контейнер с базой данных: образ mongo:7.0.3

Контейнер с утилитой для работы с базой данных: образ mongo-express:1.0.0-20

Контейнер приложения: node:21-alpine

3.3. Снимки экрана приложения

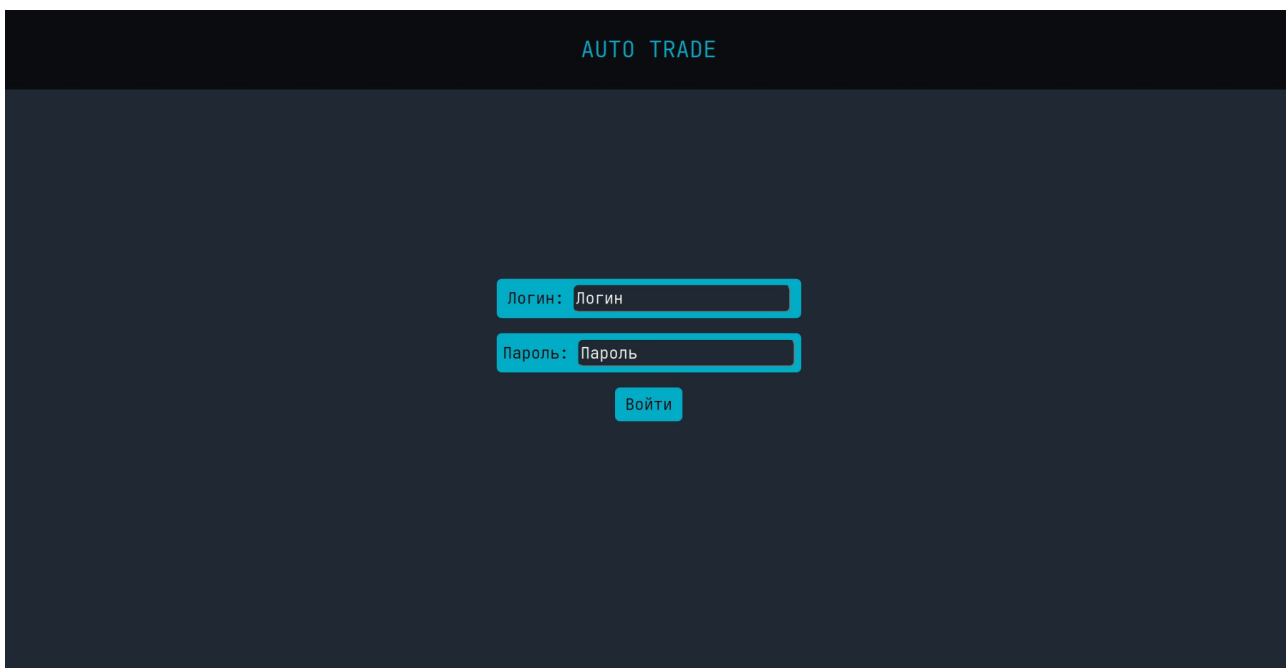


Рисунок 6 – Страница авторизации

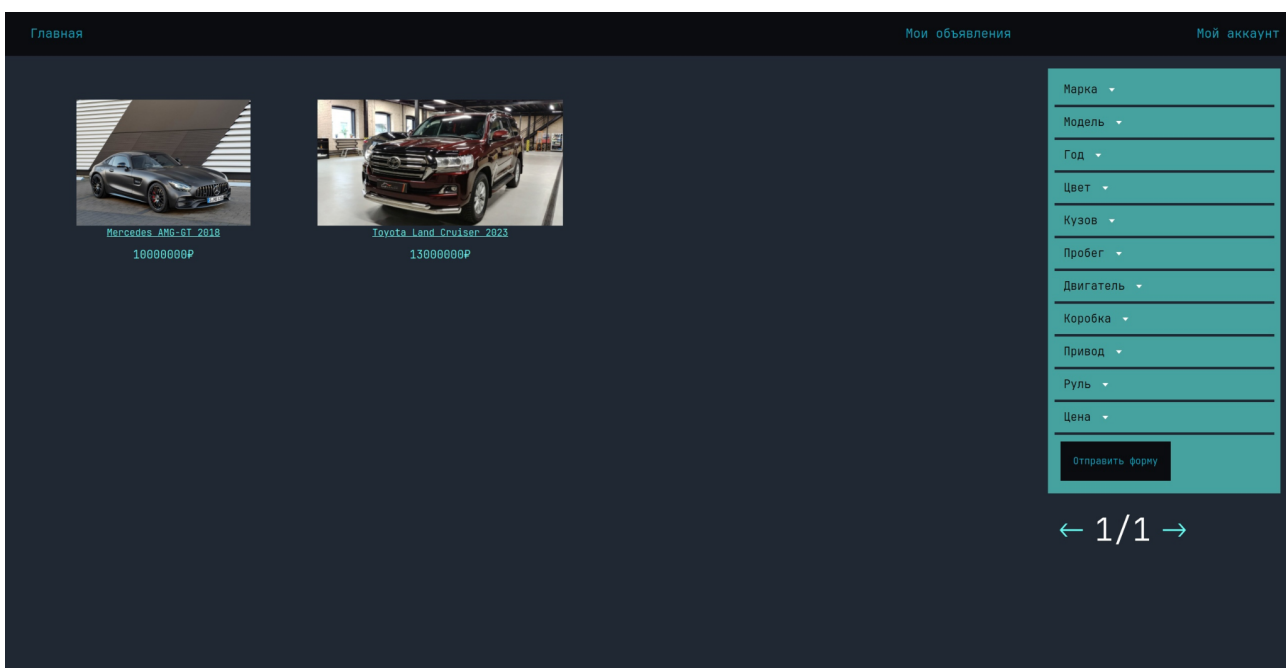


Рисунок 7 – Список опубликованных объявлений

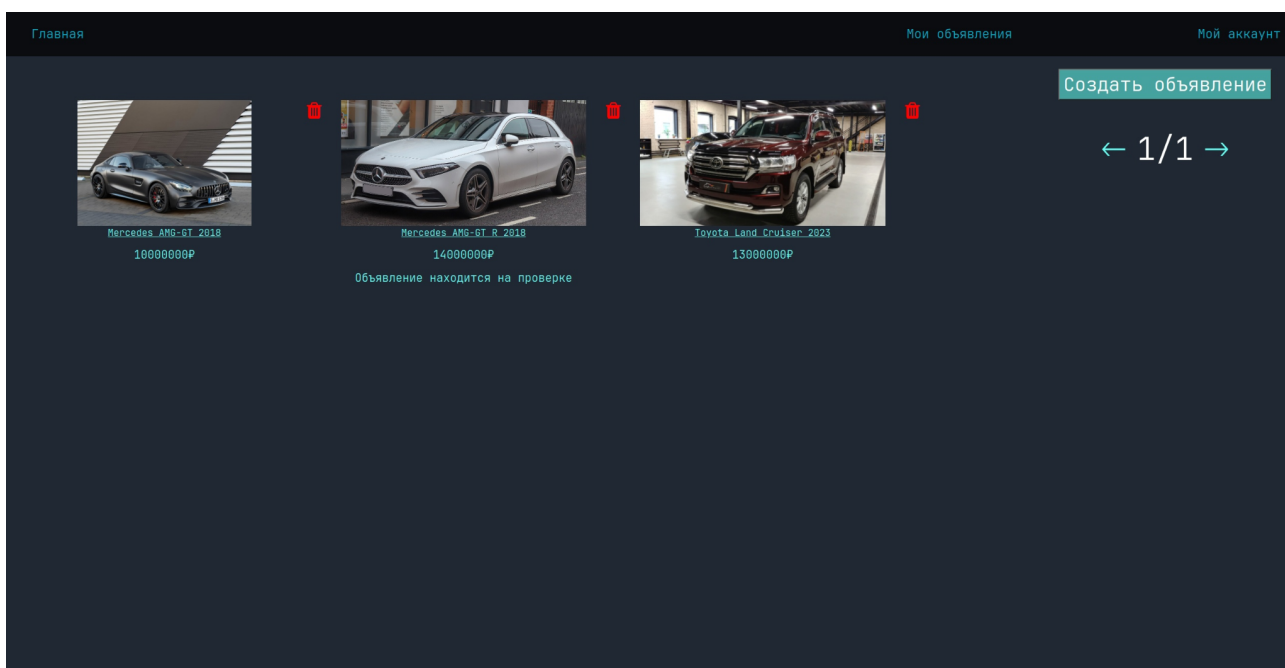


Рисунок 8 – Мои объявления

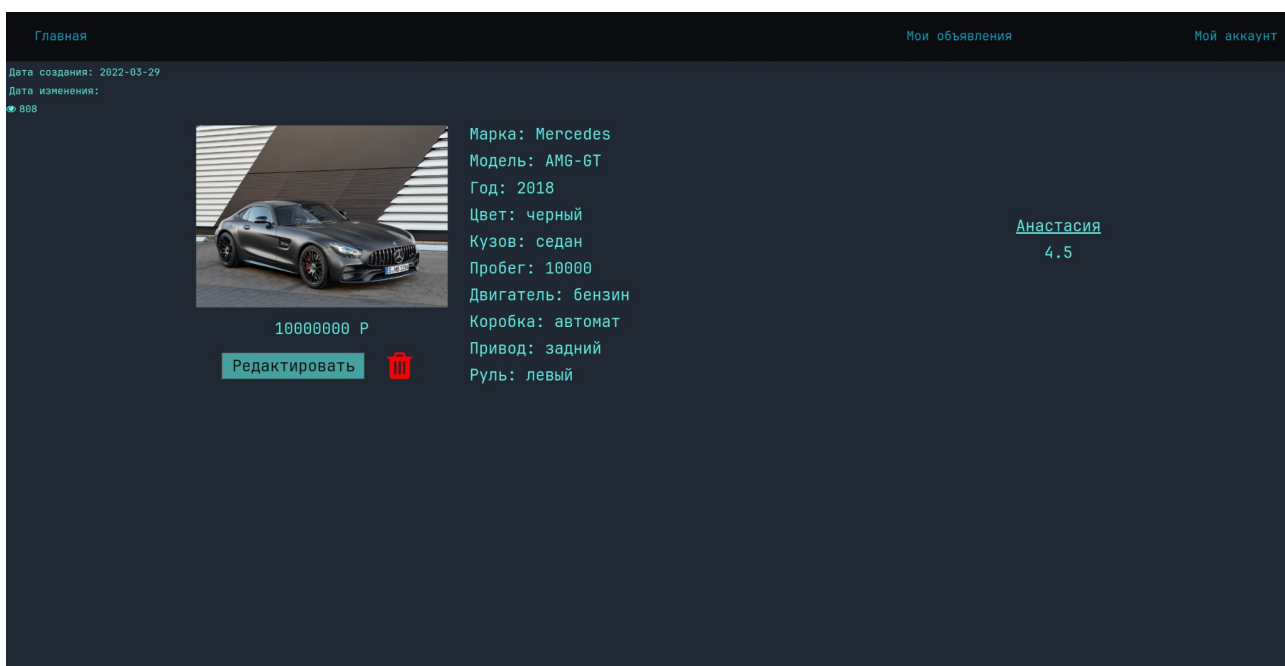



Рисунок 9 – Страница объявления

Главная Мои объявления Мой аккаунт



Загрузить другое фото: Выберите файл Файл не выбран

Марка: Mercedes

Модель: AMG-GT

Год: 2018

Цвет: черный

Кузов: седан

Пробег: 10000

Двигатель: бензин

Коробка: автомат

Привод: задний

Руль: левый

Стоимость: 10000000

Обновить объявление

Рисунок 10 – Страница редактирования объявления

Главная Мои объявления Мой аккаунт

Фото: Выберите файл Файл не выбран

Марка:

Модель:

Год:

Цвет:

Кузов:

Пробег:

Двигатель:

Коробка:

Привод:

Руль:

Стоимость:

Создать объявление

Рисунок 11 – Страница создания объявления

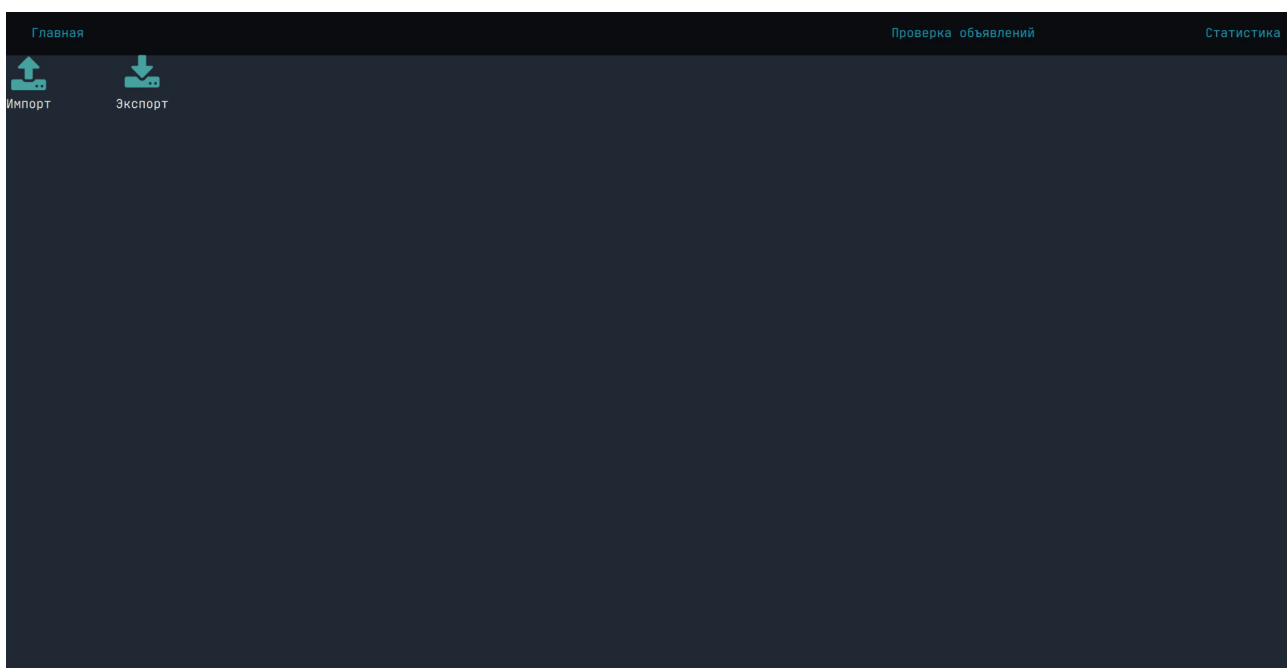


Рисунок 12 – Страница статистика с импортом и экспортом

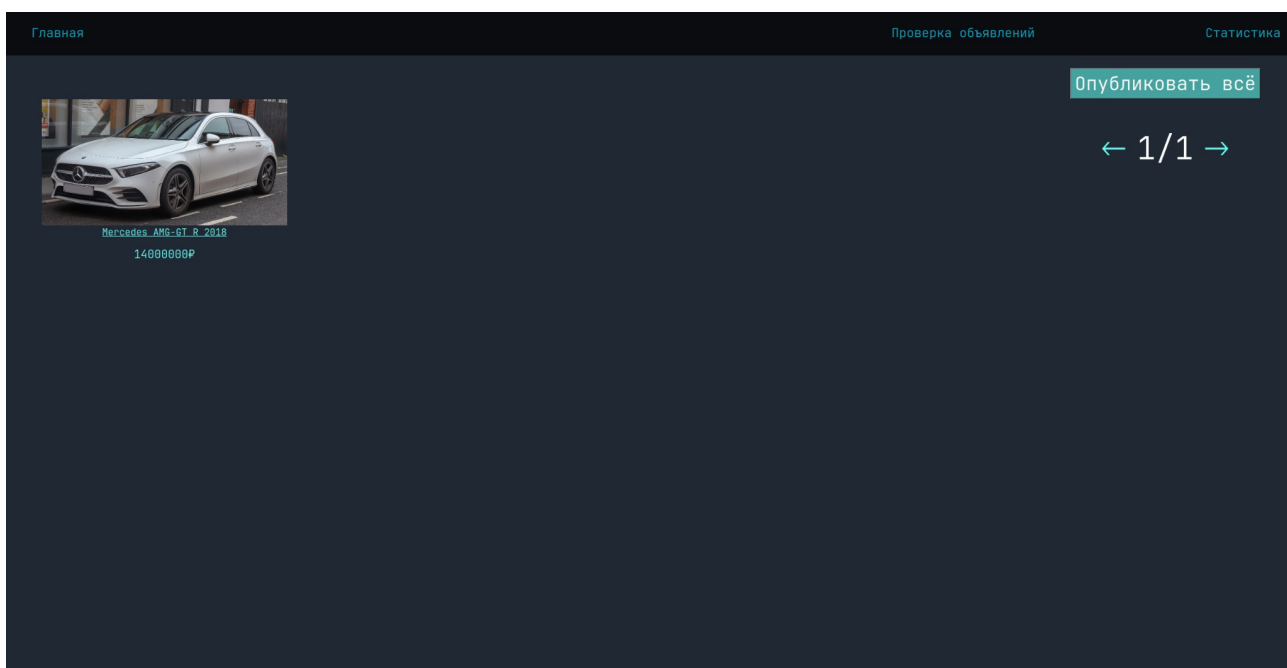


Рисунок 13 – Страница проверки объявлений администратором

ЗАКЛЮЧЕНИЕ

В результате работы было разработано веб-приложение “Каталог объявлений о продаже автомобилей”, которое позволяет хранить информацию об объявлениях, автомобилях, пользователях, а также переписку продавцов с покупателями. Пользователи сайта могут покупать, продавать автомобили, а администраторы – контролировать объявления, просматривать статистику и выполнять массовый импорт и экспорт.

а. Недостатки и пути для улучшения полученного решения

Добавление асинхронной фильтрации.

б. Будущее развитие решения

Добавление страницы для создания отзывов.

Добавление страницы для диалогов пользователей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Репозиторий веб-приложения: [электронный ресурс]. URL: <https://github.com/moevm/nosql2h23-auto-trade/tree/main> (дата обращения 21.12.2023)
2. MongoDB The Developer Data Platform: [электронный ресурс]. URL: <https://www.mongodb.com/> (дата обращения 21.12.2023)

ПРИЛОЖЕНИЕ А

ДОКУМЕНТАЦИЯ ПО СБОРКЕ И РАЗВЕРТЫВАНИЮ ПРИЛОЖЕНИЯ

Сборка приложения:

`docker compose build` или `docker-compose build`

Запуск приложения:

`docker compose up` или `docker-compose up`