

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

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ
по дисциплине «Введение в нереляционные базы данных»
Тема: Скупка электроники

Студент гр. 2304		Ефремов А.Д.
Студент гр. 2304	_____	Затримайлов Д.Р.
Студент гр. 2304	_____	Ишутин О.В.
Студентка гр. 2304	_____	Николаева Т.Д.
Студент гр. 2303	_____	Ламашовский Д.В.
Преподаватель	_____	Заславский М.М.

Санкт-Петербург
2025

ЗАДАНИЕ

Студенты Ефремов А.Д., Затримайлов Д.Р., Ишутин О.В., Николаева Т.Д.

Группа 2304

Студент Ламашовский Д.В.

Группа 2303

Тема проекта: Разработка приложения для скупки электроники

Исходные данные:

Необходимо реализовать приложение для скупки электроники с MongoDB, реализующее интерфейс для скупщика (администратора) и клиента.

Содержание пояснительной записки:

«Содержание», «Введение», «Сценарии использования», «Модель данных», «Разработанное приложение», «Выводы», «Приложения», «Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 03.02.2025

Дата сдачи реферата: 20.05.2025

Дата защиты реферата: 20.05.2025

Студент гр. 2304

Ефремов А.Д.

Студент гр. 2304		Затримайлов Д.Р.
Студент гр. 2304	<hr/>	Ишутин О.В.
Студентка гр. 2304	<hr/>	Николаева Т.Д.
Студент гр. 2303	<hr/>	Ламашовский Д.В.
Преподаватель	<hr/>	Заславский М.М.
	<hr/>	

АННОТАЦИЯ

В рамках данного проекта разработан сервис для скупки электроники с использованием MongoDB и синтетических данных. В системе предусмотрены две роли: клиенты и администраторы (скупщики). Клиенты отправляют заявки на оценку техники, прикрепляя фотографии. Скупщики оценивают электронику по фото или планируют выезд к клиенту для осмотра. Использование MongoDB обеспечивает удобное хранение заявок с разнородными данными. Проект позволяет автоматизировать процесс оценки и скупки техники, улучшая взаимодействие между клиентами и скупщиками. Исходный код доступен для дальнейшего изучения и развития по ссылке: <https://github.com/moevm/nosql1h25-electronics>.

SUMMARY

As part of this project, a service has been developed for buying electronics using MongoDB and synthetic data. The system has two roles: clients and administrators (buyers). Clients send applications for the evaluation of equipment by attaching photos. Buyers evaluate electronics based on photos or plan to visit the customer for inspection. Using MongoDB provides convenient storage of applications with heterogeneous data. The project allows you to automate the process of evaluating and buying equipment, improving the interaction between customers and buyers. The source code is available for further study and development at: <https://github.com/moevm/nosql1h25-electronics>.

СОДЕРЖАНИЕ

Введение	8
1. Сценарии использования	9
1.1. Макет UI	9
1.2. Сценарии использования	9
1.3. СИ №1 «Клиент регистрируется на сайте»	9
1.4. СИ №2 «Клиент хочет оставить заявку»	10
1.5. СИ №3 «Администратор реагирует на заявку»	11
1.6. СИ №4 «Клиент отвечает на заявку»	13
1.7. СИ №5 «Пользователь закрывает заявку»	14
1.8. СИ №6 «Пользователь меняет свои данные»	15
1.9. СИ №7 «Администратор делает массовый импорт»	15
1.10. СИ №8 «Администратор делает массовый экспорт»	16
1.11. СИ №9 «Администратор хочет посмотреть статистику продаж»	17
2. Модель данных	18
2.1. Нереляционная модель	18
2.1.1 Сущности	18
2.1.2 Расчет размера записи	20
2.1.3 Избыточность данных	21
2.1.4 Примеры данных	22
2.1.5 Примеры запросов	24
2.2. Реляционная модель	30
2.2.1 Сущности	30
2.2.2 Расчет размера записи	33
2.2.3 Избыточность данных	35
2.2.4 Примеры данных	36
2.2.5 Примеры запросов	38
2.3. Сравнение моделей	45

2.3.1	Удельный объем информации	45
2.3.2	Запросы по отдельным юзкейсам	46
2.3.3	Вывод	47
3.	Разработанное приложение	49
3.1.	Описание результатов	49
3.2.	Использованные технологии	49
3.3.	Снимки экрана приложения	50
	Заключение	55
	Список использованных источников	56
	Приложение А. Документация по сборке и развертыванию приложения	58
	Приложение Б. Инструкция для пользователя	59
	Приложение В. Нереляционная модель данных	61
	Приложение Г. Реляционная модель данных	62
	Приложение Д. Макет UI	63

ВВЕДЕНИЕ

С актуальностью решаемой проблемы связано существование необходимости автоматизации процесса оценки электроники и взаимодействия между клиентами и скупщиками. Проект направлен на создание платформы, которая упростит подачу заявок и позволит скупщикам оперативно проводить оценку как по фото, так и с выездом к клиенту.

Были поставлены следующие задачи:

- Создать сервис для организации процесса скупки электроники с разделением ролей пользователей на клиентов и администраторов.
- Реализовать функционал подачи клиентами заявок на оценку электроники с загрузкой фотографий и описанием устройств.
- Обеспечить скупщикам возможность просмотра заявок, проведения предварительной оценки по фотографиям.
- Внедрить механизм планирования выездов скупщиков к клиентам для осмотра техники на месте.
- Организовать хранение и обработку данных с использованием MongoDB.

В качестве предлагаемых решений для реализации бекенда выбран фреймворк Django с библиотекой MongoEngine для работы с MongoDB и Django REST Framework для создания API. Фронтенд выполнен на React, что обеспечивает удобный и отзывчивый пользовательский интерфейс. Такое архитектурное решение позволяет эффективно обрабатывать запросы, хранить разнообразные данные и обеспечивать комфортное взаимодействие пользователей с системой.

Качественные требования к решению включают обеспечение безопасности и разграничения доступа для разных ролей пользователей, высокую производительность при работе с базой данных, удобство интерфейса для подачи и обработки заявок, а также возможность масштабирования и дальнейшего развития функционала.

1. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

1.1. Макет UI

Созданный макет приложения см. в приложении Д.

1.2. Сценарий использования

В качестве сценариев использования наша команда выделила следующие:

- СИ №1 «Клиент регистрируется на сайте»;
- СИ №2 «Клиент хочет оставить заявку»;
- СИ №3 «Администратор реагирует на заявку»;
- СИ №4 «Клиент отвечает на заявку»;
- СИ №5 «Пользователь закрывает заявку»;
- СИ №6 «Пользователь меняет свои данные»;
- СИ №7 «Администратор делает массовый импорт»;
- СИ №8 «Администратор делает массовый экспорт»;
- СИ №9 «Администратор хочет посмотреть статистику продаж».

Рассмотрим каждый сценарий использования по-отдельности.

1.3. СИ №1 «Клиент регистрируется на сайте»

Действующее лицо: Клиент

Цель: Создать уникальную учетную запись для доступа к функционалу сайта.

Основной сценарий:

1. Пользователь нажимает кнопку «Регистрация». Система отображает форму регистрации с полями: логин, пароль, фио, номер телефона.
2. Пользователь вводит валидные данные (логин, пароль, ФИО и номер телефона). Логин - последовательность символов, которая идентифицирует конкретного пользователя в системе (минимум 3 символа, максимум 50 символов). Пароль - последовательность символов, которая позволяет пользователю защищать свой аккаунт

(минимум 8 символов, максимум 50 символов). ФИО - Фамилия Имя Отчество пользователя (Минимум 1 символ, максимум 200 символов). Номер телефона - последовательность цифр, соответствующая формату номера телефона (формат «+х xxx xxx xx xx»)

3. Нажимает кнопку «Зарегистрироваться».
4. Система проверяет данные на валидность. Пользователя отправляют на главную страницу сайта.

Альтернативный сценарий «Пользователь ввёл логин, который используется другим пользователем»:

1. Если пользователь ввёл логин, который используется другим пользователем, то на странице регистрации выводится ошибка и пользователь обновляет логин. После пользователь повторяет шаги 3-4.

Альтернативный сценарий «Пользователь ввёл невалидные данные»:

1. Если пользователь ввёл невалидные данные, то на странице регистрации выводится соответствующая ошибка и пользователь обновляет данные. После пользователь повторяет шаги 3-4.

1.4. СИ №2 «Клиент хочет оставить заявку»

Действующее лицо: Клиент

Цель: Оставить заявку на оценку электроники.

Основной сценарий:

1. Клиент вводит данные для авторизации (логин и пароль) и попадает на экран со списком заявок.
2. Клиент нажимает на кнопку «Создать заявку» и попадает на экран с созданием заявки.

3. На экране создания заявки клиент вводит данные о своей заявке. Клиент прикрепляет фото (минимум 1, максимум 10). Клиент вводит название (строковый тип данных, до 50 символов), выбирает категорию (из выпадающего списка), вводит описание (строковый тип данных, до 500 символов), указывает адрес (строковый тип данных, до 500 символов) и предлагает начальную цену (целочисленный тип данных, только положительные значения).
4. После ввода всех данных пользователь нажимает кнопку отправить. Он возвращается к списку заявок, в который добавлена новая заявка.

1.5. СИ №3 «Администратор реагирует на заявку»

Действующее лицо: Администратор

Цель: Ответить на созданную клиентом заявку.

Основной сценарий:

1. Администратор вводит данные для авторизации (логин и пароль) и попадает на экран со списком заявок.
2. Администратор пользуется фильтрами, чтобы найти конкретную заявку. Для данного сценария админ выбирает Статус заявки «Ожидает рассмотрения». Остальные фильтры в данном сценарии являются необязательными.
3. Администратор выбирает из списка заявку, которую он хочет рассмотреть и нажимает на нее, попадая на экран с карточкой товара.
4. Администратор просматривает фотографии и формирует цену. Цена вводится в поле ввода (допустимы только целочисленные положительные значения). Администратор нажимает на кнопку подтверждения.

Альтернативный сценарий «Клиент оставил плохие фотографии в заявке»:

1. Администратор вводит данные для авторизации (логин и пароль) и попадает на экран со списком заявок.

2. Администратор пользуется фильтрами, чтобы найти конкретную заявку. Для данного сценария админ выбирает Статус заявки «Ожидает рассмотрения». Остальные фильтры в данном сценарии являются необязательными.
3. Администратор выбирает из списка заявку, которую он хочет рассмотреть и нажимает на нее, попадая на экран с карточкой товара.
4. Прodelав первые три пункта основного сценария администратор не может сформировать цену на товар. В таком случае он должен предложить пользователю дату выезда. Администратор вводит дату в поле ввода (допустимы только значения типа дата-время). Администратор нажимает кнопку подтверждения.

Альтернативный сценарий «Пользователь сделал некорректную заявку»:

1. Администратор вводит данные для авторизации (логин и пароль) и попадает на экран со списком заявок.
2. Администратор пользуется фильтрами, чтобы найти конкретную заявку. Для данного сценария админ выбирает Статус заявки «Ожидает рассмотрения». Остальные фильтры в данном сценарии являются необязательными.
3. Администратор выбирает из списка заявку, которую он хочет рассмотреть и нажимает на нее, попадая на экран с карточкой товара.
4. Прodelав первые три пункта основного сценария администратор видит некорректную заявку (заявка не по теме, заполнена данными, не несущими смысла либо содержит логические ошибки (например: товар не соответствует выбранной пользователем категории). Администратор нажимает кнопку «Закрыть заявку». Администратор в всплывающем окне подтверждает свое действие и заявка закрывается.

1.6. СИ №4 «Клиент отвечает на заявку»

Действующее лицо: Клиент

Цель: Дать обратную связь на предложенные администратором цену или время выезда.

Основной сценарий:

1. Клиент вводит данные для авторизации (логин и пароль) и попадает на экран со списком заявок.
2. Клиент видит в списке заявок, что у него есть заявка, имеющая статус «Ожидает подтверждения цены пользователем» или «Ожидает подтверждения даты встречи пользователем». Клиент нажимает на эту заявку и попадает на экран с карточкой товара.
3. Клиенту предлагается на выбор две кнопки: подтвердить цену или предложить свою цену. Рядом со второй кнопкой поле для ввода (тип данных - целочисленный, только положительные значения).
4. Клиент нажимает на кнопку согласиться с ценой, если она его устраивает.

Альтернативный сценарий «Клиент не устроила цена»:

1. Клиент вводит данные для авторизации (логин и пароль) и попадает на экран со списком заявок.
2. Клиент видит в списке заявок, что у него есть заявка, имеющая статус «Ожидает подтверждения цены пользователем» или «Ожидает подтверждения даты встречи пользователем». Клиент нажимает на эту заявку и попадает на экран с карточкой товара.
3. Клиенту предлагается на выбор две кнопки: подтвердить цену или предложить свою цену. Рядом со второй кнопкой поле для ввода (тип данных - целочисленный, только положительные значения).
4. Клиент вводит свою цену в поле ввода и нажимает кнопку рядом с полем ввода.

Альтернативный сценарий «Клиента устроило время выезда»:

1. Клиент вводит данные для авторизации (логин и пароль) и попадает на экран со списком заявок.
2. Клиент видит в списке заявок, что у него есть заявка, имеющая статус «Ожидает подтверждения даты встречи пользователем». Клиент нажимает на эту заявку и попадает на экран с карточкой товара.
3. Клиенту предлагается на выбор две кнопки: подтвердить дату выезда или предложить свою. Рядом со второй кнопкой поле для ввода (тип данных - дата-время). Клиент выбирает, согласен ли он с ценой.
4. Клиент нажимает на кнопку согласиться с датой выезда, если она его устраивает.

Альтернативный сценарий «Клиента не устроило время выезда»:

1. Клиент вводит данные для авторизации (логин и пароль) и попадает на экран со списком заявок.
2. Клиент видит в списке заявок, что у него есть заявка, имеющая статус «Ожидает подтверждения даты встречи пользователем». Клиент нажимает на эту заявку и попадает на экран с карточкой товара.
3. Клиенту предлагается на выбор две кнопки: подтвердить дату выезда или предложить свою. Рядом со второй кнопкой поле для ввода (тип данных - дата-время). Клиент выбирает, согласен ли он с ценой.
4. Клиент вводит своё время в поле ввода и нажимает на кнопку рядом с этим полем ввода.

1.7. СИ №5 «Пользователь закрывает заявку»

Действующее лицо: Пользователь (как администратор, так и клиент)

Цель: Закрывать заявку.

Основной сценарий:

1. Пользователь вводит данные для авторизации и попадает на экран со списком заявок.
2. Пользователь находит свою заявку в списке и нажимает на неё.
3. Пользователь попадает на страницу с карточкой товара и нажимает на кнопку «Закрыть заявку».
4. В всплывающем окне пользователь подтверждает свои намерения. Заявка закрывается.

1.8. СИ №6 «Пользователь меняет свои данные»

Действующее лицо: Пользователь (как администратор, так и клиент)

Цель: Обновить личную информацию в системе для обеспечения актуальности данных.

Основной сценарий:

1. Пользователь входит в систему, используя логин и пароль.
2. Пользователь переходит на страницу «Профиль».
3. Система отображает форму с полями: ФИО, Номер телефона, Роль пользователя(заблокирована для редактирования).
4. Пользователь валидно изменяет своё ФИО и номер телефона.
5. Пользователь нажимает на кнопку «Сохранить»
6. Система обрабатывает запрос и показывает, что данные были успешно сохранены. В аккаунте обновляется дата последнего изменения данных.

Альтернативный сценарий «Пользователь ввёл некорректные данные»:

1. Если пользователь ввёл некорректные данные, то на странице профиля выводится соответствующая ошибка и пользователь обновляет данные. После пользователь повторяет шаги 4-6.

1.9. СИ №7 «Администратор делает массовый импорт»

Действующее лицо: Администратор

Цель: Загрузить сохраненное состояние базы данных в систему.

Основной сценарий:

1. Администратор вводит данные для авторизации (логин и пароль) и попадает на экран со списком заявок.
2. Администратор нажимает на кнопку «Импорт БД». Во всплывающем окне администратор подтверждает своё действие.
3. По завершении импорта данных администратору показывается уведомление об успешном импорте БД.

Альтернативный сценарий «Импорт некорректных данных»:

1. Администратор вводит данные для авторизации (логин и пароль) и попадает на экран со списком заявок.
2. Администратор нажимает на кнопку «Импорт БД». Во всплывающем окне администратор подтверждает своё действие.
3. По завершении импорта данных администратору показывается уведомление о неудачном импорте БД.

1.10. СИ №8 «Администратор делает массовый экспорт»

Действующее лицо: Администратор

Цель: Получить файл с сохраненными данными системы в машинно-читаемом формате (json)

Основной сценарий:

1. Администратор вводит свои данные для авторизации (логин и пароль) и попадает на страницу со списком заявок.
2. Администратор нажимает на кнопку «Экспорт БД».
3. Система обрабатывает запрос.
4. У администратора начинается загрузка файла дампа БД.
5. У администратора файл с дампом БД на диске.

Альтернативный сценарий «У пользователя пропал интернет во время загрузки Дампа БД»:

1. Пользователь исправляет проблему с подключением к сети, а затем повторяет шаги 2-5.

1.11. СИ №9 «Администратор хочет посмотреть статистику продаж»

Действующее лицо: Администратор

Цель: Получить информацию об обороте денежных средств.

Основной сценарий:

1. Администратор вводит данные для авторизации (логин и пароль) и попадает на экран со списком заявок.
2. Администратор нажимает на кнопку «Статистика» и попадает на страницу с графиком, отображающим оборот денежных средств.
3. Администратор изучает график и делает выводы по объему продаж.

2. МОДЕЛЬ ДАННЫХ

2.1. Нереляционная модель

Графическое представление нереляционной модели см. в приложении В.

2.1.1. Сущности

User – содержит данные о зарегистрированных пользователях в системе. Она используется для управления учетными записями и обеспечения безопасности доступа. Поля:

- *_id (ObjectId)* – уникальный идентификатор пользователя;
- *fullname (string)* – полное имя пользователя;
- *role (Role)* – роль пользователя в системе (администратор, пользователь);
- *creation_date (Date)* – дата создания учетной записи;
- *edit_date (Date)* – дата последнего редактирования учетной записи;
- *phone (string)* – номер телефона пользователя;
- *password_hash (string)* – хэшированный пароль пользователя;
- *salt (string)* – соль, используемая для хэширования пароля.

Request – содержит информацию о предложении или объявлении, размещенном пользователем на платформе. Оно включает в себя детали о самом предложении, его категории, местоположении и состоянии. Поля:

- *_id (ObjectId)* – уникальный идентификатор объявления;
- *title (string)* – заголовок объявления/название товара;
- *description (string)* – подробное описание предложения;
- *category (Category)* – категория объявления (*laptop; smartphone; tablet; pc; tv; audio; console; periphery; other*);
- *address (string)* – адрес для осмотра товара;
- *price (number)* – цена предложения;
- *photos (ObjectId[])* – фотографии, прикрепленные к объявлению;
- *statuses (Status[])* – статусы или статус объявления. Массив статусов;

- *user_id (ObjectId)* – идентификатор пользователя, который создал объявление.

Photo – сущность фотографии товара. Поля:

- *_id (ObjectId)* – уникальный идентификатор объекта фотографии товара;
- *data (BinData)* – бинарные данные изображения.

Status – абстрактный класс, описывающий состояние заявки. Поля:

- *timestamp (Date)* – общее поле всех статусов, которое хранит время присвоения конкретного статуса.

Типы статусов:

- *Created* – состояние, которое присваивается при создании заявки;
- *PriceOffer* – состояние, которое присваивается при предложении другой цены. Поля:
 - *price (number)* – поле, хранящее предложенную цену;
 - *user_id (ObjectId)* – поле, хранящее пользователя, который предложил другую цену.
- *PriceAccept* – состояние, которое присваивается при подтверждении цены. Поля:
 - *user_id (ObjectId)* – поле, хранящее пользователя, который подтвердил цену.
- *DateOffer* – состояние, которое присваивается при предложении другой даты. Поля:
 - *date (Date)* – поле, хранящее предложенную дату;
 - *user_id (ObjectId)* – поле, хранящее пользователя, который предложил дату.
- *DateAccept* – состояние, которое присваивается при подтверждении даты. Поля:
 - *user_id (ObjectId)* – поле, хранящее пользователя, который подтвердил дату.

- *Closed* – состояние, которое присваивается при закрытии заявки.

Поля:

- *user_id* (*ObjectId*) – поле, хранящее пользователя, который закрыл заявку;
- *success* (*boolean*) – поле, хранящее положительность / отрицательность решения о закрытии заявки.

2.1.2. Расчет размера записи

User – 185 байт:

- *_id* – 12 байт (размер *ObjectId*);
- *fullname* – 51 байт (в среднем ~50 символов);
- *role* – 7 байт (строка *client*);
- *creation_date* – 8 байт (размер *Date*);
- *edit_date* – 8 байт (размер *Date*);
- *phone* – 17 байт (строка формата «+7(123)456-67-89»);
- *password_hash* – 65 байт (строка SHA-256 хэша);
- *salt* – 17 байт (строка 16 символов).

Request – 639 байт:

- *_id* – 12 байт (размер *ObjectId*);
- *title* – 41 байт (в среднем ~40 символов);
- *description* – 201 байт (в среднем ~200 символов);
- *category* – 11 байт (строка «*smartphone*»);
- *address* – 76 байт (в среднем ~75 символов);
- *price* – 4 байт (размер *Int32*);
- *photos* – 4×12 байт = 48 байт (в среднем ~4 фото, храним как *ObjectId*);
- *statuses* – 6×39 байт = 234 байт (в среднем ~6 статусов, берём размер *DateOffer*: *date* - 8 байт, *user_id* - 12 байт, *timestamp* - 8 байт, *type* - 11 байт);
- *user_id* – 12 байт (размер *ObjectId*).

Photo – 307212 байт:

- *_id* – 12 байт (размер *ObjectId*);
- *data* – 300 кбайт (ограничение на размер файла).

Формула для расчета размера всей модели в байтах с правильным учетом числа фотографий:

$$185N_u + (639 - 48)N_r + (307212 + 12)N_p = 185N_u + 591N_r + 307224N_p$$

где N_u – число пользователей, N_r – число заявок, N_p – число фотографий.

Предположим, что в среднем на одного пользователя приходится по 2 заявки, а на каждую заявку в среднем приходится по 4 фото. Тогда N_r и N_p можно выразить через N_u следующим образом:

$$N_r = 2N_u$$

$$N_p = 4N_r = 8N_u$$

Таким образом можно выразить формулу размера всей модели через одну переменную:

$$185N_u + 591N_r + 307224N_p = 185N_u + 1182N_u + 2457792N_u = 2459159N_u$$

где N_u – число пользователей.

2.1.3. Избыточность данных

Для расчёта размера чистых данных из предложенной модели были убраны идентификаторы, и сама модель подразумевается как иерархический документ.

Формула для расчёта избыточности модели:

$$\begin{aligned} & \frac{185N_u + 591N_r + 307224N_p}{(185 - 12)N_u + (591 - 2 \cdot 12 - 6 \cdot 12)N_r + (307224 - 2 \cdot 12)N_p} \\ &= \frac{185N_u + 591N_r + 307224N_p}{173N_u + 495N_r + 307200N_p}, \end{aligned}$$

где N_u – число пользователей, N_r – число заявок, N_p – число фотографий.

Сделав аналогичные предположения, что $N_r = 2N_u$ и $N_p = 8N_u$, получим формулу для расчёта избыточности модели для одной переменной:

$$\frac{185N_u + 591N_r + 307224N_p}{173N_u + 495N_r + 307200N_p} = \frac{2459159N_u}{2458763N_u} = \frac{2459159}{2458763} \approx 1,0001610566$$

Размер модели растёт линейно по каждому из параметров, что напрямую следует из формул выше.

2.1.4. Примеры данных

Пример 1

Коллекция *users*:

```
[
  {
    "_id": ObjectId("5f4a8a0737d36f6fa78a31ad"),
    "fullname": "Иван Иванов Иванович",
    "password_hash":
"5d540b1fa43cb0fb1c620fe56308a70761987ec867b8fdfaba3b11e188f277d0",
    "salt": "0123456789abcdef",
    "role": "client",
    "phone": "+7(123)456-78-90",
    "creation_date": ISODate("2025-01-01T00:00:00.000Z"),
    "edit_date": ISODate("2025-02-01T00:00:00.000Z")
  },
  {
    "_id": ObjectId("fcacf8074417f4cb52c6d08f4"),
    "fullname": "Петр Петров Петрович",
    "password_hash":
"1e141bffdfe323111fd20563a65437d562aea01c3ac40b1527e81df24b062abd",
    "salt": "123456789abcdef0",
    "role": "admin",
    "phone": "+7(987)654-32-10",
    "creation_date": ISODate("2025-02-01T00:00:00.000Z"),
    "edit_date": ISODate("2025-02-01T00:00:00.000Z")
  }
]
```

Коллекция *requests*:

```
[
  {
    "_id": ObjectId("7d6ad8ce3983d2a5f04ac8d5"),
    "title": "Продажа ноутбука",
    "description": "Хороший ноутбук для работы",
    "address": "Москва, ул. Ленина, 1",
    "category": "laptop",
    "price": 50000,
    "photos": [ObjectId("c3372e60cbb804d4cbefb34d"),
ObjectId("ece8880f8c4c0a5cdfd7931a")],
    "user_id": ObjectId("5f4a8a0737d36f6fa78a31ad"),
    "statuses": [
      {
        "type": "created",
        "timestamp": ISODate("2025-01-01T00:00:00.000Z")
      }
    ]
  }
]
```

```

    },
    {
      "type": "price_offer",
      "timestamp": ISODate("2025-01-02T00:00:00.000Z"),
      "price": 45000,
      "user_id": ObjectId("fcdf8074417f4cb52c6d08f4")
    }
  ]
}
]

```

Коллекция *photos*:

```

[
  {
    "_id": ObjectId("c3372e60cbb804d4cbefb34d"),
    "data": BinData(0, "aGVsbG93b3JsZA==")
  },
  {
    "_id": ObjectId("ece8880f8c4c0a5cdfd7931a"),
    "data": BinData(0, "aGVsbG93b3JsZA==")
  }
]

```

Пример 2

Коллекция *users*:

```

[
  {
    "_id": ObjectId("eb47ae23fad58db1228987bb"),
    "fullname": "Иван Иванов Иванович",
    "password_hash":
"5d540b1fa43cb0fb1c620fe56308a70761987ec867b8fdfab3b11e188f277d0",
    "salt": "0123456789abcdef",
    "role": "client",
    "phone": "+7(123)456-78-90",
    "creation_date": ISODate("2025-01-01T00:00:00.000Z"),
    "edit_date": ISODate("2025-02-01T00:00:00.000Z")
  },
  {
    "_id": ObjectId("7a33965ce9d2e017a6a57ddd"),
    "fullname": "Петр Петров Петрович",
    "password_hash":
"1e141bf1dfe323111fd20563a65437d562aea01c3ac40b1527e81df24b062abd",
    "salt": "123456789abcdef0",
    "role": "admin",
    "phone": "+7(987)654-32-10",
    "creation_date": ISODate("2023-02-01T00:00:00.000Z"),
    "edit_date": ISODate("2023-02-01T00:00:00.000Z")
  }
]

```

Коллекция *requests*:

```

[
  {

```

```

    "_id": ObjectId("05587e684e4912e242449eb0"),
    "title": "Продажа ноутбука",
    "description": "Хороший ноутбук для работы",
    "address": "Москва, ул. Ленина, 1",
    "category": "laptop",
    "price": 50000,
    "photos": [ObjectId("3f1da63f3c8411d0f027f166"),
ObjectId("f6196a4d2a5b8b65a233eede")],
    "user_id": ObjectId("eb47ae23fad58db1228987bb"),
    "statuses": [
      {
        "type": "created",
        "timestamp": ISODate("2023-01-01T00:00:00.000Z")
      },
      {
        "type": "price_offer",
        "timestamp": ISODate("2023-01-02T00:00:00.000Z"),
        "price": 45000,
        "user_id": ObjectId("7a33965ce9d2e017a6a57ddd")
      }
    ]
  }
]

```

Коллекция *photos*:

```

[
  {
    "_id": ObjectId("3f1da63f3c8411d0f027f166"),
    "data": BinData(0, "aGVsbG93b3JsZA==")
  },
  {
    "_id": ObjectId("f6196a4d2a5b8b65a233eede"),
    "data": BinData(0, "aGVsbG93b3JsZA==")
  }
]

```

2.1.5. Примеры запросов

Фильтрация заявок

Применяет к таблицам заявок фильтры: даты последнего изменения статуса (от и до), статус заявки, ФИО клиента, участие админа в разрешении заявки, категория, название, описание. Запрос составлен так, чтобы можно было убрать неиспользуемые фильтры.

```

db.requests.aggregate([
  { $addFields: { last_status: { $arrayElemAt: [ "$statuses", -1 ] } }},
  { $match: { "last_status.timestamp": { $gte: new Date('2017-11-01') } }}, // дата изменение статуса (от)
  { $match: { "last_status.timestamp": { $lte: new Date('2028-01-01') } }}, // дата изменения статуса (до)
  { $match: { "last_status.type": "finished" } }, // статус заявки

```



```

    { $match: { statuses: { $elemMatch: { user_id:
ObjectId("5f4a8a0737d36f6fa78a31ad") }}}}, // участвовал в разрешении
    { $match: { category: "smartphone" }}, // категория
    { $match: { title: { $regex: "apple", $options: "i" } }}, // имя
    { $match: { description: { $regex: "apple", $options: "i" } }}, //
описание
    { $lookup: { from: "users", localField: "user_id", foreignField:
"_id", as: "user_info" }}, // ФИО клиента
    { $match: { "user_info.fullname": { $regex: "артем", $options: "i"
}}}, // ФИО клиента
    { $project: { title: 1, category: 1, "last_status.type": 1,
"last_status.timestamp": 1 } }},
  ])

```

Число запросов: 1.

Число задействованных коллекций: 2 (*requests, users*).

Клиент регистрируется на сайте

Проверка на то, что пользователя с таким телефоном нет:

```

db.users.findOne({
  "phone": "+7(999)123-45-67"
})

```

Вставка данных пользователя:

```

db.users.insertOne({
  "fullname": "Сергеев Сергей Сергеевич",
  "password_hash":
"143ef4a014832ce5ffed740b5f33bf4da0d9516d3c3f6f9cc231ee4495936287",
  "salt": "abcdef0123456789",
  "role": "client",
  "phone": "+7(999)123-45-67",
  "creation_date": new ISODate(),
  "edit_date": new ISODate()
})

```

Число запросов: 2.

Число задействованных коллекций: 1 (*requests*).

Клиент хочет оставить заявку

Вставка фотографий в таблицу photos:

```

db.photos.insertMany([
  {
    data: new BinData(0, "aGVsbG93b3JsZA==")
  },
  {
    data: new BinData(0, "SGVsbG8gV29ybGQh")
  }
]);

```

Вставка заявки в таблицу requests:

```
db.requests.insertOne({
  "title": "Ноутбук Lenovo ThinkPad X1",
  "description": "Состояние отличное, 2023 года выпуска",
  "address": "Москва, ул. Тверская, 15",
  "category": "laptop",
  "price": 75000,
  "photos": [
    ObjectId("665f8d3e1c9d440000aabcde"),
    ObjectId("665f8d3e1c9d440000aabcdf")
  ],
  "user_id": ObjectId("5f4a8a0737d36f6fa78a31ad"),
  "statuses": [
    {
      "type": "created",
      "timestamp": new ISODate()
    }
  ]
})
```

Число запросов: 2.

Число задействованных коллекций: 2 (*requests*, *photos*).

Администратор реагирует на заявку

Основной сценарий:

Получение фотографий, для предварительного осмотра:

```
db.photos.find({
  _id: { $in: db.requests.findOne({ _id:
ObjectId("665f9a3e1c9d440000aabce3") }).photos }
});
```

Обновление заявки, с предложением цены:

```
db.requests.updateOne({
  _id: ObjectId("665f9a3e1c9d440000aabce3")
}, {
  $push: {
    statuses: {
      type: "price_offer",
      timestamp: new ISODate(),
      price: 80000,
      user_id: ObjectId("fc4f8074417f4cb52c6d08f4")
    }
  }
});
```

Число запросов: 2.

Число задействованных коллекций: 2 (*requests*, *photos*).

Альтернативный сценарий «Клиент оставил плохие фотографии в заявке»

Получение фотографий, для предварительного осмотра:

```
db.photos.find({
  _id: { $in: db.requests.findOne({ _id:
ObjectId("665f9a3e1c9d440000aabce3") }).photos }
});
```

Обновление заявки, с предложением даты осмотра:

```
db.requests.updateOne({
  _id: ObjectId("665f9a3e1c9d440000aabce3")
}, {
  $push: {
    statuses: {
      type: "date_offer",
      timestamp: new ISODate(),
      date: ISODate("2025-03-15T00:00:00.000Z"),
      user_id: ObjectId("fcac8074417f4cb52c6d08f4")
    }
  }
});
```

Число запросов: 2.

Число задействованных коллекций: 2 (*requests*, *photos*).

Альтернативный сценарий «Пользователь сделал некорректную заявку»

Администратор закрывает заявку:

```
db.requests.updateOne({
  _id: ObjectId("665f9a3e1c9d440000aabce3")
}, {
  $push: {
    statuses: {
      type: "closed",
      timestamp: new ISODate(),
      success: false,
      user_id: ObjectId("fcac8074417f4cb52c6d08f4") // ID администратора
    }
  }
});
```

Число запросов: 1.

Число задействованных коллекций: 1 (*requests*).

Клиент отвечает на заявку

Основной сценарий:

Пользователь соглашается с ценой:

```

db.requests.updateOne({
  _id: ObjectId("665f9a3e1c9d440000aabce3")
}, {
  $push: {
    statuses: {
      type: "price_accept",
      timestamp: new ISODate(),
      user_id: ObjectId("5f4a8a0737d36f6fa78a31ad") // ID клиента
    }
  }
});

```

Число запросов: 1.

Число задействованных коллекций: 1 (*requests*).

Альтернативный сценарий «Клиент не устроила цена»

Пользователь предлагает свою цену:

```

db.requests.updateOne({
  _id: ObjectId("665f9a3e1c9d440000aabce3")
}, {
  $push: {
    statuses: {
      type: "price_offer",
      timestamp: new ISODate(),
      price: 70000, // Предложенная клиентом цена
      user_id: ObjectId("5f4a8a0737d36f6fa78a31ad") // ID клиента
    }
  }
});

```

Число запросов: 1.

Число задействованных коллекций: 1 (*requests*).

Альтернативный сценарий «Клиента устроило время выезда»

Клиент соглашается с временем выезда, которое ему назначили:

```

db.requests.updateOne({
  _id: ObjectId("665f9a3e1c9d440000aabce3")
}, {
  $push: {
    statuses: {
      type: "date_accept",
      timestamp: new ISODate(),
      user_id: ObjectId("5f4a8a0737d36f6fa78a31ad") // ID клиента
    }
  }
});

```

Число запросов: 1.

Число задействованных коллекций: 1 (*requests*).

Альтернативный сценарий «Клиента не устроило время выезда»

Пользователь предлагает свое время выезда:

```
db.requests.updateOne({
  _id: ObjectId("665f9a3e1c9d440000aabce3")
}, {
  $push: {
    statuses: {
      type: "date_offer",
      timestamp: new ISODate(),
      date: ISODate("2025-03-15T00:00:00.000Z"), // Предложенная клиентом
цена
      user_id: ObjectId("5f4a8a0737d36f6fa78a31ad") // ID клиента
    }
  }
});
```

Число запросов: 1.

Число задействованных коллекций: 1 (*requests*).

Пользователь закрывает заявку

```
db.requests.updateOne({
  _id: ObjectId("665f9a3e1c9d440000aabce3")
}, {
  $push: {
    statuses: {
      type: "closed",
      timestamp: new ISODate(),
      success: false,
      user_id: ObjectId("fcdf8074417f4cb52c6d08f4") // ID пользователя
    }
  }
});
```

Число запросов: 1.

Число задействованных коллекций: 1 (*requests*).

Пользователь меняет свои данные

Пользователь меняет свой номер телефона и ФИО:

```
db.users.updateOne({ _id: ObjectId("id пользователя") }, { $set: { phone:
"новый номер телефона", fullname: "новое ФИО" } })
```

Число запросов: 1.

Число задействованных коллекций: 1 (*users*).

Администратор делает массовый импорт

Пользователь с правами администратора производит массовый импорт данных:

```
mongorestore --drop --gzip --archive='имя архива'
```

Администратор делает массовый экспорт

Пользователь с правами администратора производит массовый экспорт данных:

```
mongodump --db electronics --gzip --archive > "backup_$(date +%Y-%m-%d-%T)".gz
```

Администратор хочет посмотреть статистику продаж

Пользователь с правами администратора запрашивает продажи:

```
db.requests.aggregate([
  { $project: { price: 1, lastStatus: { $arrayElemAt: [ "$statuses", -1 ] } } },
  { $match: { "lastStatus.type": "finished", "lastStatus.success": true } },
  { $group: { _id: { $dateToString: { format: "%Y-%m-%d", date: "$lastStatus.timestamp" } }, sum: { $sum: "$price" } } },
  { $sort: { _id: 1 } }
]);
```

Число запросов: 1.

Число задействованных коллекций: 1 (*requests*).

2.2. Реляционная модель

Графическое представление реляционной модели см. в приложении Г.

2.2.1. Сущности

User – содержит данные о зарегистрированных пользователях в системе. Она используется для управления учетными записями и обеспечения безопасности доступа. Поля:

- *id (int)* – уникальный идентификатор пользователя;
- *fullname (text)* – полное имя пользователя;
- *role (Role)* – роль пользователя (*admin*, *client*);

- *creation_date (timestamp)* – дата создания учетной записи;
- *edit_date (timestamp)* – дата последнего редактирования учетной записи;
- *phone (text)* – номер телефона пользователя;
- *password_hash (text)* – хэшированный пароль пользователя;
- *salt (text)* – соль, используемая для хэширования пароля.

Request – содержит информацию о заявке. Поля:

- *id (int)* – уникальный идентификатор заявки;
- *title (text)* – заголовок заявки;
- *description (text)* – подробное описание заявки;
- *category (Category)* – категория объявления (*laptop; smartphone; tablet; pc; tv; audio; console; periphery; other*);
- *address (text)* – адрес для осмотра товара;
- *user_id (int)* – идентификатор пользователя, который создал объявление.

Photo – фотография товара. Поля:

- *id (int)* – уникальный идентификатор объекта фотографии товара;
- *data (blob)* – бинарные данные изображения;
- *request_id (int)* – идентификатор заявки, к которой относится фотография.

CreatedStatus – описывает состояние заявки при создании. Поля:

- *id (int)* – уникальный идентификатор;
- *timestamp (timestamp)* – время присвоения статуса;
- *initial_price (int)* – цена, указанная при создании заявки;
- *request_id (int)* – идентификатор заявки.

PriceOfferStatus – описывает состояние заявки. Присваивается при предложении другой цены. Поля:

- *id (int)* – уникальный идентификатор;
- *timestamp (timestamp)* – время присвоения статуса;
- *price (int)* – предложенная цена;

- *request_id (int)* – идентификатор заявки;
- *user_id (int)* – идентификатор пользователя, предложившего цену.

PriceAcceptStatus – описывает состояние заявки. Присваивается при подтверждении цены. Поля:

- *id (int)* – уникальный идентификатор;
- *timestamp (timestamp)* – время присвоения статуса;
- *request_id (int)* – идентификатор заявки;
- *user_id (int)* – идентификатор пользователя, подтвердившего цену.

DateOfferStatus – описывает состояние заявки. Присваивается при предложении даты. Поля:

- *id (int)* – уникальный идентификатор;
- *timestamp (timestamp)* – время присвоения статуса;
- *date (timestamp)* – предложенная дата;
- *request_id (int)* – идентификатор заявки;
- *user_id (int)* – идентификатор пользователя, предложившего дату.

DateAcceptStatus – описывает состояние заявки. Присваивается при подтверждении даты. Поля:

- *id (int)* – уникальный идентификатор;
- *timestamp (timestamp)* – время присвоения статуса;
- *request_id (int)* – идентификатор заявки;
- *user_id (int)* – идентификатор пользователя, подтвердившего цену.

ClosedStatus – описывает состояние заявки. Присваивается при подтверждении даты. Поля:

- *id (int)* – уникальный идентификатор;
- *timestamp (timestamp)* – время присвоения статуса;
- *success (boolean)* – положительность/отрицательность решения о закрытии заявки;
- *request_id (int)* – идентификатор заявки;
- *user_id (int)* – идентификатор пользователя, подтвердившего цену.

2.2.2. Расчет размера записи

User – 174 байт:

- *id* – 4 байт (размер *int*);
- *fullname* – 51 байт (в среднем ~50 символов);
- *role* – 4 байт (размер *int*);
- *creation_date* – 8 байт (размер *timestamp*);
- *edit_date* – 8 байт (размер *timestamp*);
- *phone* – 17 байт (строка формата «+7(123)456-78-90»);
- *password_hash* – 65 байт (строка SHA-256 хэша);
- *salt* – 17 байт (строка 16 символов).

Request – 330 байт:

- *id* – 4 байт (размер *int*);
- *title* – 41 байт (в среднем ~40 символов);
- *description* – 201 байт (в среднем ~200 символов);
- *category* – 4 байта (размер *int*);
- *address* – 76 байт (в среднем ~75 символов);
- *user_id* – 4 байта (размер *int*).

Photo – 307208 байт:

- *id* – 4 байта (размер *int*);
- *data* – 300 кбайт (ограничение на размер файла);
- *request_id* – 4 байта (размер *int*).

CreatedStatus – 20 байт:

- *id* – 4 байт (размер *int*);
- *timestamp* – 8 байт (размер *timestamp*);
- *initial_price* – 4 байта (размер *int*);
- *request_id* – 4 байта (размер *int*).

PriceOfferStatus – 24 байт:

- *id* – 4 байт (размер *int*);
- *timestamp* – 8 байт (размер *timestamp*);
- *price* – 4 байта (размер *int*);

- *request_id* – 4 байта (размер *int*);
- *user_id* – 4 байта (размер *int*).

PriceAcceptStatus – 20 байт:

- *id* – 4 байт (размер *int*);
- *timestamp* – 8 байт (размер *timestamp*);
- *request_id* – 4 байта (размер *int*);
- *user_id* – 4 байта (размер *int*).

DateOfferStatus – 28 байт:

- *id* – 4 байт (размер *int*);
- *timestamp* – 8 байт (размер *timestamp*);
- *date* – 8 байт (размер *timestamp*);
- *request_id* – 4 байта (размер *int*);
- *user_id* – 4 байта (размер *int*).

DateAcceptStatus – 20 байт:

- *id* – 4 байт (размер *int*);
- *timestamp* – 8 байт (размер *timestamp*);
- *request_id* – 4 байта (размер *int*);
- *user_id* – 4 байта (размер *int*).

ClosedStatus – 21 байт:

- *id* – 4 байт (размер *int*);
- *timestamp* – 8 байт (размер *timestamp*);
- *success* – 1 байт (размер *boolean*);
- *request_id* – 4 байта (размер *int*);
- *user_id* – 4 байта (размер *int*).

Category и *Role* являются *enum*. В данном случае рассматриваем хранение *enum* PostgreSQL. Тут каждая запись включает OID перечисления, порядок сортировки и текстовую метку.

Category – 170 байт:

- «*laptop*» – 19 байт (6 символов, $4 + 4 + 4 + 6 + 1$);
- «*smartphone*» – 24 байт (11 символов, $4 + 4 + 4 + 11 + 1$);

- «*tablet*» – 18 байт (5 символов, $4 + 4 + 4 + 5 + 1$);
- «*pc*» – 15 байт (2 символа, $4 + 4 + 4 + 2 + 1$);
- «*tv*» – 15 байт (2 символа, $4 + 4 + 4 + 2 + 1$);
- «*audio*» – 18 байт (5 символов, $4 + 4 + 4 + 5 + 1$);
- «*console*» – 20 байт (7 символов, $4 + 4 + 4 + 7 + 1$);
- «*periphery*» – 23 байт (10 символов, $4 + 4 + 4 + 10 + 1$);
- «*other*» – 18 байт (5 символов, $4 + 4 + 4 + 5 + 1$).

Role – 37 байт:

- «*admin*» – 18 байт (5 символов, $4 + 4 + 4 + 5 + 1$);
- «*client*» – 19 байт (6 символов, $4 + 4 + 4 + 6 + 1$).

Размер модели в байтах зависит от числа пользователей и числа заявок. На одну заявку в среднем приходится 6 статусов (по одному каждого типа), а также 4 фотографии. Поэтому формула расчета размера всей модели в байтах будет:

$$37 + 170 + 174N_u + (330 + 4 \cdot 307208 + 21 + 20 + 28 + 20 + 24 + 20)N_r = 2078 + 174N_u + 1229295N_r,$$

где N_u – число пользователей, N_r – число заявок.

Будем считать, что в среднем на одного пользователя приходится по 2 заявки. Тогда формула преобразится к следующему виду и будет зависеть от одной переменной:

$$207 + 174N_u + 1229295 \cdot 2 \cdot N_u = 207 + 2458765N_u,$$

где N_u – число пользователей.

2.2.3. Избыточность данных

Для расчёта размера чистых данных из модели были убраны идентификаторы.

Формула для расчёта избыточности модели:

$$\frac{207+174N_u+1229279N_r}{(174-4)N_u+((330-8)+4\cdot(307208-8)+9+8+16+8+12+12)N_r} = \frac{207+174N_u+1229279N_r}{170N_u+1229187N_r},$$

где N_u – число пользователей, N_r – число заявок.

Аналогично, приведем ее к зависимости от одной переменной:

$$\frac{207+2458732N_u}{75+(174-4)N_u+((330-8)+4\cdot(307208-8)+9+8+16+8+12+12)\cdot 2\cdot N_u} = \frac{207+2458732N_u}{75+2458544N_u},$$

где N_u – число пользователей.

Размер модели растет линейно по каждому из параметров. В модели отсутствует дублирование данных, следовательно, каждая из сущностей учитывается один раз.

2.2.4. Примеры данных

Пример данных представлен в таблицах 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9.

Таблица 3.1 - пример данных таблицы *users*

id	fullname	role	creation_date	edit_date	phone	password_hash	salt
1	Волкова Ирина Сергеевна	admin	2025-03-05 16:26:17	2025-03-05 16:26:17	+7(987) 654-32-10	abcdef ...	1234567 890abcdef
2	Николаев Всеволод Александрович	client	2025-03-05 16:26:17	2025-03-05 16:26:17	+7(911) 222-33-44	abcdef ...	fedcba9 876543210
3	Ашина Полина Викторовна	client	2025-03-05 16:26:17	2025-03-05 16:26:17	+7(900) 123-45-67	abcdef ...	ghijklm nopqrstuv

Таблица 3.2 - пример данных таблицы *requests*

id	title	description	category	address	user_id
1	iPhone	Состояние почти новое. Ремонтов не было.	smartphone	ул. Савушкина, д. 111	2
2	Игровая приставка	PlayStation 5, два геймпада, не пользовался	console	пр. Культуры, д. 26к1	3
3	Продам монитор	27 дюймов, IPS, 144 Гц	periphery	Большой Сампсониевский проспект, 69к3	2

Таблица 3.3 - пример данных таблицы *photos*

id	data	request_id
1	\x696d6167655f646174615f31	1
2	\x696d6167655f646174615f32	1
3	\x696d6167655f646174615f33	2
4	\x696d6167655f646174615f34	2
5	\x696d6167655f646174615f35	3

Таблица 3.4 - пример данных таблицы *created_statuses*

id	timestamp	initial_price	request_id
1	2025-03-06 19:44:00.976041	90000	1
2	2025-03-06 19:44:00.976041	50000	2
3	2025-03-06 19:44:00.976041	20000	3

Таблица 3.5 - пример данных таблицы *price_offer_statuses*

id	timestamp	price	request_id	user_id
1	2025-03-06 19:44:00.978075	45000	2	1
2	2025-03-06 19:44:00.978075	47000	2	2
3	2025-03-06 19:44:00.978075	18000	3	1

Таблица 3.6 - пример данных таблицы *price_accept_statuses*

id	timestamp	request_id	user_id
1	2025-03-06 19:44:00.980461	3	2

Таблица 3.7 - пример данных таблицы *date_offer_statuses*

id	timestamp	date	request_id	user_id
1	2025-03-06 19:44:00.992065	2025-03-10 12:00:00	1	1

Таблица 3.8 - пример данных таблицы *date_accept_statuses*

id	timestamp	request_id	user_id
----	-----------	------------	---------

1	2025-03-06 19:44:00.994326	1	2
---	----------------------------	---	---

Таблица 3.9 - пример данных таблицы *closed_statuses*

id	timestamp	success	user_id	request_id
1	2025-03-06 19:44:00.978075	true	1	1
2	2025-03-06 19:44:00.978075	false	1	2

2.2.5. Примеры запросов

Фильтрация заявок

```

SELECT DISTINCT r.*
FROM requests r
LEFT JOIN LATERAL (
    SELECT request_id, MAX(timestamp) as last_update, 'created' as
status FROM created_status GROUP BY request_id
    UNION ALL
    SELECT request_id, MAX(timestamp), 'price_offer' FROM
price_offer_status GROUP BY request_id
    UNION ALL
    SELECT request_id, MAX(timestamp), 'price_accept' FROM
price_accept_status GROUP BY request_id
    UNION ALL
    SELECT request_id, MAX(timestamp), 'date_offer' FROM
date_offer_status GROUP BY request_id
    UNION ALL
    SELECT request_id, MAX(timestamp), 'date_accept' FROM
date_accept_status GROUP BY request_id
    UNION ALL
    SELECT request_id, MAX(timestamp), 'closed' FROM closed_status
GROUP BY request_id
) latest_status ON r.id = latest_status.request_id
LEFT JOIN users u ON r.user_id = u.id
LEFT JOIN LATERAL (
    SELECT user_id FROM (
    SELECT request_id, user_id FROM price_offer_status
    UNION ALL
    SELECT request_id, user_id FROM price_accept_status
    UNION ALL
    SELECT request_id, user_id FROM date_offer_status
    UNION ALL
    SELECT request_id, user_id FROM date_accept_status
    UNION ALL
    SELECT request_id, user_id FROM closed_status
    ) AS status_union WHERE status_union.request_id = r.id LIMIT 1
) status_users ON true
WHERE
    (latest_status.last_update BETWEEN '2025-03-01' AND '2025-03-10')

```

```

AND (latest_status.status = 'closed')
AND (u.fullname ILIKE '%Николаев %')
AND (status_users.user_id = 1)
AND (r.category = 'smartphone')
AND (r.title ILIKE '%iPhone%')
AND (r.description ILIKE '%Ремонт%');

```

На данном примере использовали все доступные фильтры: отфильтровали все заявки, последнее изменение статуса которых было с 01.03.25 по 10.03.25, последним статусом является closed, созданные пользователем с фамилией Николаев, в разрешении которых участвовал администратор с id 1, категорией заявки является smartphone, в названии есть iPhone, а в описании Ремонт. Результат запроса представлен в таблице 3.10.

Таблица 3.10 - результат запроса

id	title	description	category	address	user_id
1	iPhone 13 Pro	Состояние почти новое. Ремонт не было.	smartphone	ул. Савушкина, д. 111	2

Число запросов: 13 (с учетом всех подзапросов).

Число задействованных коллекций: 8 (*requests, users, created_status, price_offer_status, price_accept_status, date_offer_status, date_accept_status, closed_status*).

Клиент регистрируется на сайте

```

SELECT EXISTS (
    SELECT 1 FROM users WHERE phone = '+7(999)123-45-67'
) AS user_exists;

```

Первый запрос проверяет наличие пользователя с данным номером телефона в базе данных. Если возвращено false, то добавляем запись вторым запросом:

```

INSERT INTO users (fullname, role, phone, password_hash, salt)
VALUES ('Шилов Алексей Владимирович', 'client', '+7(999)123-45-67',
'abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz0123456789ww',
'1234567890abcdef');

```

Результат запроса представлен в таблице 3.11.

Таблица 3.11 - результат запроса

id	fullname	role	creation_date	edit_date	phone	password_hash	salt

1	Волкова Ирина Сергеевна	admin	2025-03-05 16:26:17	2025-03-05 16:26:17	+7(987) 654-32-10	abcdef ...	1234567 890abcd ef
2	Николаев Всеволод Александрови ч	client	2025-03-05 16:26:17	2025-03-05 16:26:17	+7(911) 222-33-44	abcdef ...	fedcba9 8765432 10
3	Ашина Полина Викторовна	client	2025-03-05 16:26:17	2025-03-05 16:26:17	+7(900) 123-45-67	abcdef ...	ghijklm nopqrstu v
4	Шилов Алексей Владимирови ч	client	2025-03-06 20:39:45.5 77729	2025-03-06 20:39:45 .577729	+7(999) 123-45-67	abcdef ...	1234567 890abcd ef

Число запросов: 2.

Число задействованных коллекций: 1 (*users*).

Клиент хочет оставить заявку

```

WITH new_request AS (
  INSERT INTO requests (title, description, category, address,
user_id)
  VALUES (
    'Samsung Galaxy S23 Ultra',
    'Телефон в идеальном состоянии с гарантией',
    'smartphone',
    'Санкт-Петербург, Невский пр-т, 100',
    2
  )
  RETURNING id
),
insert_photos AS (
  INSERT INTO photos (data, request_id)
  SELECT data, (SELECT id FROM new_request)
  FROM (VALUES
    (decode('U1NhbxN1bmc=', 'base64')),
    (decode('R2FsYXh5UzIz', 'base64'))
  ) AS t(data)
  RETURNING request_id
)
INSERT INTO created_status (initial_price, request_id)
VALUES (
  85000,

```



```
(SELECT id FROM new_request)
);
```

Этот запрос добавляет новую заявку с фотографиями и создает запись в `created_status`. Результат запроса представлен в таблице 3.12.

Таблица 3.12 - результат запросов

id	title	description	category	address	user_id	photo_id	photo_data_base64	initial_price
4	Samsung Galaxy S23 Ultra	Телефон в идеальном состоянии с гарантией	smartphone	Санкт-Петербург, Невский пр-т, 100	2	6	U1NhbXN1bm c=	85000
4	Samsung Galaxy S23 Ultra	Телефон в идеальном состоянии с гарантией	smartphone	Санкт-Петербург, Невский пр-т, 100	2	7	R2FsYXh5UzIz	85000

Число запросов: 1.

Число задействованных коллекций: 3 (*requests, photos, created_status*).

Администратор реагирует на заявку

Основной сценарий

Получение фотографий, для предварительного осмотра:

```
SELECT p.id, encode(p.data, 'base64') AS photo_data_base64
FROM photos p
WHERE p.request_id = (
    SELECT id
    FROM requests
    WHERE id = 1
);
```

Обновление заявки, с предложением цены:

```
INSERT INTO price_offer_status (price, request_id, user_id)
VALUES (
    80000,
    1,
    1
);
```

Число запросов: 2.

Число задействованных коллекций: 3 (*requests, photos, price_offer_status*).

Альтернативный сценарий «Клиент оставил плохие фотографии в заявке»

Получение фотографий, для предварительного осмотра:

```
SELECT p.id, encode(p.data, 'base64') AS photo_data_base64
FROM photos p
WHERE p.request_id = (
    SELECT id
    FROM requests
    WHERE id = 1
);
```

Обновление заявки, с предложением даты осмотра:

```
INSERT INTO date_offer_status (date, request_id, user_id)
VALUES (
    '2025-03-15 00:00:00',
    1,
    1
);
```

Число запросов: 2.

Число задействованных коллекций: 3 (*requests*, *photos*, *date_offer_status*).

Альтернативный сценарий «Пользователь сделал некорректную заявку»

Администратор закрывает заявку:

```
INSERT INTO closed_status (success, user_id, request_id)
VALUES (
    FALSE,
    1,
    1
);
```

Число запросов: 1.

Число задействованных коллекций: 1 (*closed_status*).

Клиент отвечает на заявку

Основной сценарий

Пользователь соглашается с ценой:

```
INSERT INTO price_accept_status (request_id, user_id)
VALUES (
    1,
    2
);
```

Число запросов: 1.

Число задействованных коллекций: 1 (*price_accept_status*).

Альтернативный сценарий «Клиент не устроила цена»

Пользователь предлагает свою цену:

```
INSERT INTO price_offer_status (price, request_id, user_id)
VALUES (
    70000,
    1,
    2
);
```

Число запросов: 1.

Число задействованных коллекций: 1 (*price_offer_status*).

Альтернативный сценарий «Клиента устроило время выезда»

Клиент соглашается с временем выезда, которое ему назначили:

```
INSERT INTO date_accept_status (request_id, user_id)
VALUES (
    1,
    2
);
```

Число запросов: 1.

Число задействованных коллекций: 1 (*date_accept_status*).

Альтернативный сценарий «Клиента не устроило время выезда»

Пользователь предлагает свое время выезда:

```
INSERT INTO date_offer_status (date, request_id, user_id)
VALUES (
    '2025-03-15 00:00:00',
    1,
    2
);
```

Число запросов: 1.

Число задействованных коллекций: 1 (*date_offer_status*).

Пользователь закрывает заявку

```
INSERT INTO closed_status (success, user_id, request_id)
VALUES (
    FALSE, -- Неуспешное закрытие
    1,
    1
);
```

Число запросов: 1.

Число задействованных коллекций: 1 (*closed_status*).

Пользователь меняет свои данные

Пользователь меняет свой номер телефона и ФИО:

```
UPDATE users
SET phone = '+7(999)123-45-67',    -- Новый номер телефона
    fullname = 'Шилов Алексей Викторович' -- Новое ФИО
WHERE id = 2;
```

Число запросов: 1.

Число задействованных коллекций: 1 (*users*).

Администратор делает массовый импорт

Пользователь с правами администратора производит массовый импорт

данных:

```
pg_dump -U username -h hostname -p port -d database_name -F c -f
backup.dump
```

Администратор делает массовый экспорт

Пользователь с правами администратора производит массовый экспорт

данных:

```
pg_restore -U username -h hostname -p port -d database_name --clean --if-
exists backup.dump
```

Администратор хочет посмотреть статистику продаж

Пользователь с правами администратора запрашивает продажи:

```
WITH latest_price AS (
  SELECT DISTINCT ON (request_id)
    request_id,
    price,
    timestamp
  FROM price_offer_status
  ORDER BY request_id, timestamp DESC
)
SELECT
  cs.timestamp AS closed_timestamp,
  COALESCE(lp.price, c.initial_price) AS final_price
FROM closed_status cs
LEFT JOIN latest_price lp ON cs.request_id = lp.request_id
LEFT JOIN created_status c ON cs.request_id = c.request_id
WHERE cs.success = true;
```

Число запросов: 2.

Число задействованных коллекций: 3 (*closed_status*, *created_status*, *price_offer_status*).

2.3. Сравнение моделей

2.3.1. Удельный объем информации

Сущность *User*:

- SQL – 174 байт;
- NoSQL – 185 байт.

В SQL может требоваться немного меньше памяти из-за более строгих типов данных.

Сущность *Request*:

- SQL – 330 байт;
- NoSQL – 639 байт.
- NoSQL требует больше памяти из-за своей схемы документного хранения и избыточности данных. SQL более оптимизирован для хранения табличных данных.

Сущность *Photo*: в обоих случаях размер одинаковый, поскольку размер зависит от конкретного файла, а не от структуры базы данных.

Сущность *Status*:

- SQL – требует создания множества отдельных таблиц, размер которых варьируется от 20 до 28 байт. В среднем у каждого заказа существует около 6 статусов, поэтому для одного заказа потребуется ~144 байт;
- NoSQL – не требует создания дополнительных таблиц. Хранится как отдельное поле в сущности *Request*. В среднем занимает 234 байта.

SQL необходимо для хранения статусов в среднем меньше памяти по сравнению с NoSQL моделью, однако реляционная модель требует создания большого количества дополнительных таблиц.

Пояснения для выбора:

- SQL:
 - Подходит для операций, где важна строгая структура данных и размер. Исключительная оптимизация для хранения больших объемов табличной информации и быстрая обработка запросов;
 - Хорошо подходит для случаев, когда данные имеют четкие связи и структурированность.
- NoSQL:
 - Лучше подходит для хранения неструктурированных данных и больших объемов информации с возможными изменениями схемы;
 - Хорошо работает для приложений, где скорость развертывания и гибкость важнее строгой схемы (например, в реальном времени, быстро меняющиеся данные).

При прочих равных, модель NoSQL будет занимать чуть больше памяти, чем SQL. Если учитывать объем дополнительных индексов, таблиц связи, а также необходимость использования множества дополнительных таблиц для неструктурированной сущности Status, то объем памяти в реляционной модели увеличится, поэтому в контексте данной задачи выгодней использовать NoSQL модель.

2.3.2. Запросы по отдельным юзкейсам

NoSQL Модель:

- В случае с запросами в NoSQL мы видим, что для фильтрации или обновления данных, администратор делает запрос к коллекции users, чтобы изменить данные (один запрос);
- Запрос для получения статистики с использованием агрегации также требует только один запрос к коллекции requests;

- Если количество объектов (например, заявок) увеличивается, NoSQL может более эффективно обрабатывать большие объемы данных в одном запросе, так как часто используются более простые операции.

SQL Модель:

- Запросы в реляционных моделях могут потребовать больше затрат, так как необходимо использовать JOIN для объединения данных из различных таблиц;
- Процесс обновления информации требует отдельного запроса к каждому из соответствующих столбцов в таблице пользователей (например, два запроса для обновления имени и телефона). Это увеличивает количество запросов, необходимых для выполнения операций;
- Если пришла заявка, которая включает сложные запросы с JOIN, SQL запросы могут тысячу раз обращаться к нескольким таблицам, что ведет к увеличению времени выполнения и сложности запросов.

Пояснение выбора модели:

С точки зрения количества запросов, модель NoSQL имеет преимущество в случае работы с большими объемами данных, предлагая возможность обновления, фильтрации и агрегации с меньшим количеством запросов и меньшими затратами по времени. В реляционных базах данных сложные запросы часто требуют уже нескольких этапов, что приводит к большему количеству операций и времени выполнения. Поэтому в контексте нашего проекта выгодней использовать NoSQL модель, предполагая, что рано или поздно наш проект получит большие охваты аудитории.

2.3.3. Вывод

Объем данных:

Модели SQL и NoSQL имеют разные архитектурные принципы, что приводит к различным затратам на хранение. NoSQL имеет чуть больший объем,

но при этом данная модель значительно упрощает использование неструктурированных данных, что сильно оптимизирует масштабирование и упрощает использование системы.

Запросы:

NoSQL предоставляет более простые и экономичные запросы, в то время как SQL требует большей сложности и число операций увеличивается с ростом объектов. Особенно из-за не структурированности некоторых сущностей.

Общий вывод:

Таким образом, для плавного масштабирования и работы с большим объемом данных модель NoSQL может оказаться предпочтительнее, чем SQL.

3. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

3.1. Описание результатов

Разработанное решение представляет из себя приложение с Web-интерфейсом. Для запуска приложения была создана конфигурация docker compose. В конфигурации используются следующие контейнеры:

- *db* – контейнер с базой данных MongoDB;
- *backend* – контейнер с бэкендом, собирается из папки с проектом бэкенда;
- *frontend* – контейнер на основе образа nginx. Собирается из папки с проектом фронтенда. Выполняет две функции:
 - раздача статических файлов фронтенд части Web-приложения;
 - проксирование запросов к API на контейнер *backend*.

3.2. Используемые технологии

Репозиторий проекта состоит из двух подпроектов: проект бэкенд сервера и проект фронтенд части Web-приложения.

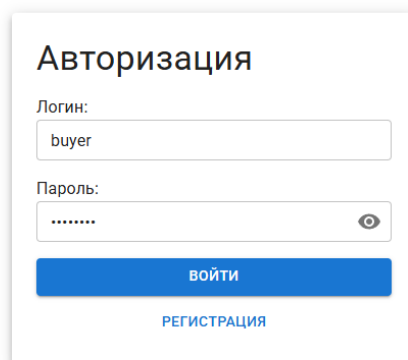
Для создания бэкенда был использован фреймворк django. Для использования готовых функций запроса к API бэкенда был настроен Swagger.

Для создания фронтенда была использована библиотека React. В качестве языка программирования был выбран TypeScript для чистоты и безопасности кода. В качестве сборщика был выбран бандлер Vite, так как данный бандлер предоставляет возможность создать проект с выбранной конфигурацией (React + Typescript) и предоставляет готовые средства для разработки и сборки проекта без дополнительной его настройки. В качестве UI kit был выбран MUI, так как макет приложения изначально разрабатывался с учетом его использования.

Для сборки приложения была настроена сборка docker образа двух проектов. Для его развёртывания была создана конфигурация docker compose (см. п. 4.1).

3.3. Снимки экрана приложения

Снимки экрана разработанного приложения представлены на рисунках 1-9.



Авторизация

Логин:

buyer

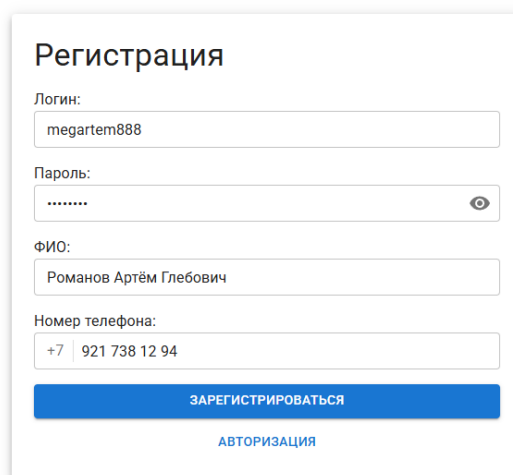
Пароль:

.....

ВОЙТИ

РЕГИСТРАЦИЯ

Рисунок 1 - страница авторизации пользователя



Регистрация

Логин:

megartem888

Пароль:

.....

ФИО:

Романов Артём Глебович

Номер телефона:

+7 921 738 12 94

ЗАРЕГИСТРИРОВАТЬСЯ

АВТОРИЗАЦИЯ

Рисунок 2 - страница регистрации пользователя

Список заявок

[ПЕРЕЙТИ В ПРОФИЛЬ](#)
[СОЗДАТЬ ЗАЯВКУ](#)
[ВЫХОД](#)

Дата изменения статуса (от):

Дата изменения статуса (до):

Статус заявки: Всё

Категория товара: Всё

Сортировать по: Категории

[ОБНОВИТЬ СПИСОК](#)


Создатель	Название	Категория	Последний статус	Дата назначения статуса
Орлова Елена Владимировна	телевизор с изогнутым экраном	Телевизор	Дата встречи подтверждена	10.05.2025
Орлова Елена Владимировна	телевизор ретро из советского союза	Телевизор	Предложена дата встречи	03.05.2025

1-2 из 2 < >

Рисунок 3 - страница пользователя списка заявок с примененными фильтрами

[В ГЛАВНОЕ МЕНЮ](#)

Карточка товара



телевизор ретро из советского союза

бабушкин телевизор, знаю что такое ценится сейчас

Категория: Телевизор

Цена: 5000 P

Адрес: ул. Савушкина, д. 111, парадная 1

[ЗАКРЫТЬ ЗАЯВКУ](#)

История операций

- +

27.04.2025, 23:27:08

Заявка создана
- 📅

03.05.2025, 00:08:03

Скупщиком была предложена дата встречи 03.05.2025 04:00
- 🗨️

Скупщик предложил дату встречи 03.05.2025 04:00

 - Подтвердить дату: ☒
 - Предложить дату:

Рисунок 4 - страница заявки с запросом пользователю подтвердить дату встречи

Профиль

ФИО:

Номер телефона:

Роль:

СОХРАНИТЬ ИЗМЕНЕНИЯ

ОТМЕНА

- 07:13 20.04.2025
Аккаунт **создан**
- 17:31 20.05.2025
Данные аккаунта **изменены**

Рисунок 5 - страница пользователя изменения информации об аккаунте

Список заявок

ПЕРЕЙТИ В ПРОФИЛЬ **СОЗДАТЬ ЗАЯВКУ** **ВЫХОД**

Дата изменения статуса (от):

Дата изменения статуса (до):

Статус заявки:

Категория товара:


Сортировать по:

ОБНОВИТЬ СПИСОК

Создатель	Дата назначения статуса
Орлова Елена Владимировна	19.05.2025
Орлова Елена Владимировна	03.05.2025
Орлова Елена Владимировна	10.05.2025

Добавление товара

Изображения
 5o4rcInQskM.jpg



Название

Категория

Описание

Адрес

Предложенная цена (руб.)

СОЗДАТЬ ОТМЕНА

1-3 из 3

Рисунок 6 - окно создания заявки

Список заявок

ПЕРЕЙТИ В ПРОФИЛЬ
ЭКСПОРТ БД
ИМПОРТ БД
СТАТИСТИКА
ВЫХОД

Дата изменения статуса (от):

Дата изменения статуса (до):

Статус заявки: Всё

Клиент:

☐ Участвовал в разрешении

Категория товара: Всё

ПРИМЕНИТЬ ФИЛЬТРЫ

Создатель	Название	Категория	Последний статус	Дата назначения статуса
Турова Алиса Анатольевна	телефон филипс раскладушка	Смартфон	Предложена дата встречи	26.04.2025
Турова Алиса Анатольевна	Монитор SunWind 23.8" SUN-M24BA101 — IZHARD	Прочее	Создан	22.04.2025

11-12 из 12

Рисунок 7 - страница скупщика списка заявок с применёнными фильтрами

Профиль

ФИО:

Номер телефона:

Роль:

СОХРАНИТЬ ИЗМЕНЕНИЯ

ОТМЕНА

+

07:12 20.04.2025
Аккаунт **создан**

Рисунок 8 - страница скупщика изменения информации об аккаунте

[В ГЛАВНОЕ МЕНЮ](#)

Статистика по заявкам

☒ Участвовал в разрешении

Название содержит

Категория

Адрес

ФИО пользователя

Мин. цена

Макс. цена

Мин. дата изменения
дд.мм.гггг

Макс. дата изменения
дд.мм.гггг

Последний статус

[СБРОСИТЬ ФИЛЬТРЫ](#)

Ось X
Дата изменения

Ось Y
Цена

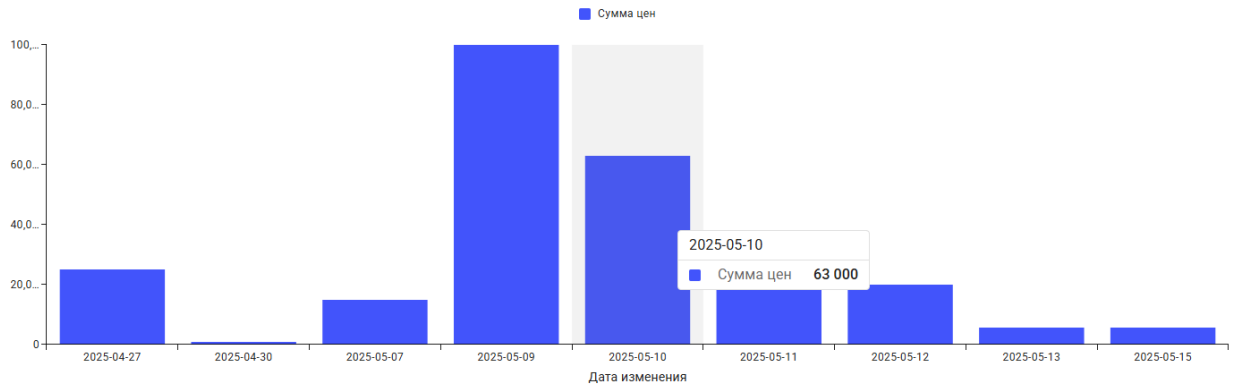


Рисунок 9 - страница скупщика со статистикой

ЗАКЛЮЧЕНИЕ

В процессе выполнения индивидуального домашнего задания были достигнуты следующие результаты:

1. Разработан сервис для организации процесса скупки электроники с разделением ролей пользователей на клиентов и администраторов;
2. Реализованы все сценарии использования приложения;
3. Реализована нереляционная модель хранения данных на основе СУБД MongoDB;
4. Реализовано развертывание приложения с использованием Docker.

Разработанное решение полностью соответствует поставленным целям. В качестве направлений для развития решения можно выделить:

1. Добавление адаптивного дизайна для приложения;
2. Добавление возможности обмена сообщениями между клиентом и администратором;
3. Добавление функционала для добавления дополнительных фотографий после создания заявки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Репозиторий с разработанным приложением // GitHub. URL: <https://github.com/moevm/nosql1h25-electronics> (дата обращения: 16.05.2025)
2. Документация по API и основным функциям библиотеки React // React Documentation. URL: <https://react.dev/reference/react> (дата обращения: 16.05.2025)
3. Документация и полный перечень компонентов библиотеки Material UI с примерами использования // Material UI. URL: <https://mui.com/material-ui/all-components/> (дата обращения: 16.05.2025)
4. Документация по библиотеке React Hook Form для управления формами в React-приложениях // React Hook Form. URL: <https://react-hook-form.com/docs> (дата обращения: 16.05.2025)
5. Документация по маршрутизации в React-приложениях с использованием React Router // React Router. URL: <https://reactrouter.com/home> (дата обращения: 16.05.2025)
6. Документация по работе с переменными окружения и режимами сборки в инструменте Vite // Vite. URL: <https://vite.dev/guide/env-and-mode.html> (дата обращения: 16.05.2025)
7. Документация по синтаксису и возможностям Dockerfile для создания контейнеров // Docker Documentation. URL: <https://docs.docker.com/reference/dockerfile/> (дата обращения: 16.05.2025)
8. Документация по веб-фреймворку Django для разработки на Python // Django Documentation. URL: <https://docs.djangoproject.com/en/5.1/> (дата обращения: 16.05.2025)
9. Документация по MongoEngine - ODM для MongoDB на Python // MongoEngine Documentation. URL: <https://docs.mongoengine.org/> (дата обращения: 16.05.2025)

10.Документация по Simple JWT для аутентификации JWT в Django REST Framework // Simple JWT Documentation. URL: <https://django-rest-framework-simplejwt.readthedocs.io/en/latest/> (дата обращения: 16.05.2025)

ПРИЛОЖЕНИЕ А

ДОКУМЕНТАЦИЯ ПО СБОРКЕ И РАЗВЕРТЫВАНИЮ ПРИЛОЖЕНИЯ

В рамках задания был настроен Docker compose. С его помощью осуществляется запуск приложения. Необходимо выполнить следующие действия:

1. Склонируйте репозиторий с помощью команды

```
git clone https://github.com/moevm/nosql1h25-electronics
```

2. Перейдите в папку проекта

```
cd nosql1h25-electronics
```

3. Запустите проект с помощью команды

```
docker compose up --build
```

4. Приложение доступно по адресу <http://localhost:8000/> или <http://127.0.0.1:8000/>

ПРИЛОЖЕНИЕ Б

ИНСТРУКЦИЯ ДЛЯ ПОЛЬЗОВАТЕЛЯ

Инструкция для клиента:

1. Регистрация и вход

Зарегистрируйтесь, указав логин, пароль, ФИО и номер телефона.

После регистрации войдите в систему с помощью логина и пароля.

2. Подача заявки на оценку

На главной странице нажмите “Создать заявку”. Заполните форму с информацией о вашем устройстве: название, категория, описание, адрес, предложенная цена. Прикрепите 1 или более фотографий.

Отправьте заявку.

3. Просмотр статуса заявки

На главной странице вы можете видеть список своих заявок с текущим статусом и комментариями скупщиков. Для просмотра более подробной информации о заявке (например, динамики изменения статуса) можно нажать на заявку, тем самым перейти на страницу заявки.

4. Подтверждение оценки и выезда

При назначении скупщиком цены или даты выезда, вы на странице заявки можете подтвердить или отклонить предложение. При отказе необходимо указать устраивающую вас цену или дату выезда.

5. Закрытие заявки

В любой момент времени, пока ваша заявка не считается закрытой, вы можете закрыть ее. В этом случае сделка не будет заключена. Для этого перейдите на страницу заявки и нажмите на “Закрыть заявку”.

6. Редактирование профиля

На главной странице нажмите “Перейти в профиль”. Измените телефон или ФИО, нажмите на “Сохранить изменения”.

Инструкция для администратора (скупщика):

1. Вход в систему

Войдите в систему с помощью логина и пароля

2. Просмотр заявок

На главной странице отображается список заявок от клиентов.

Предусмотрена фильтрация заявок по дате изменения статуса, статусу заявки, клиенту, категории, описанию, названию. Кроме того, возможно выбрать только заявки, в которых участвовали вы.

3. Предварительная оценка по фото

На главной странице нажмите на необходимую заявку. Так вы перейдете на ее страницу. Ознакомьтесь с фотографиями и описанием техники. Вы можете предложить свою цену или запланировать дату выезда.

4. Обновление статуса заявки

На странице заявки вы можете обновить ее статус в зависимости от этапа обработки. Вы можете предложить свою цену, согласиться на цену клиента, а также предложить дату выезда или согласиться на дату выезда, предложенную клиентом.

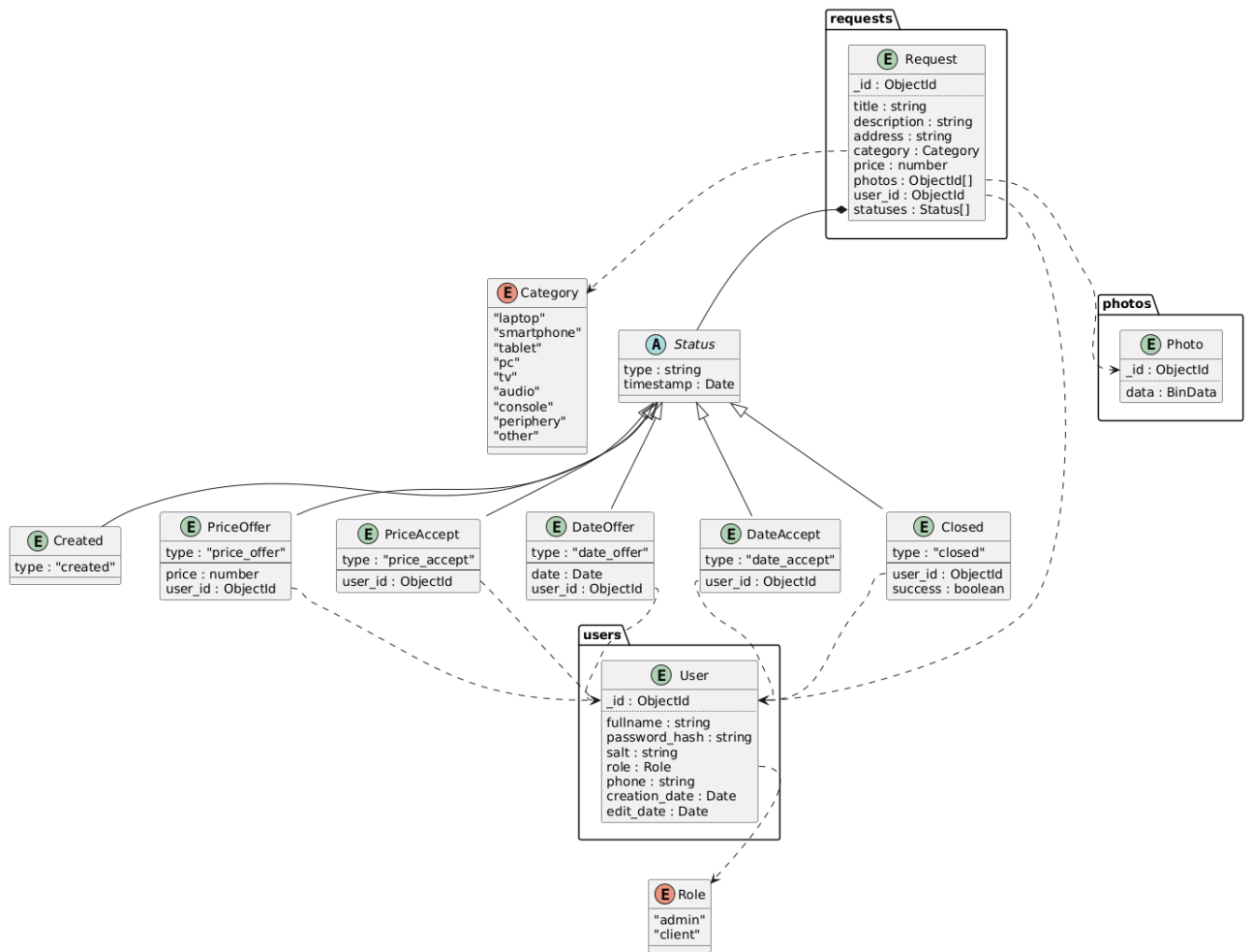
5. Заккрытие заявки

На любом этапе на странице заявки вы можете ее закрыть. Для этого нажмите на “Заккрыть заявку”. В этом случае сделка не будет заключена. Если выезд состоялся, необходимо подтвердить рассмотрение заявки. Для этого нажмите на “Подтвердить” в статусах.

6. Редактирование профиля

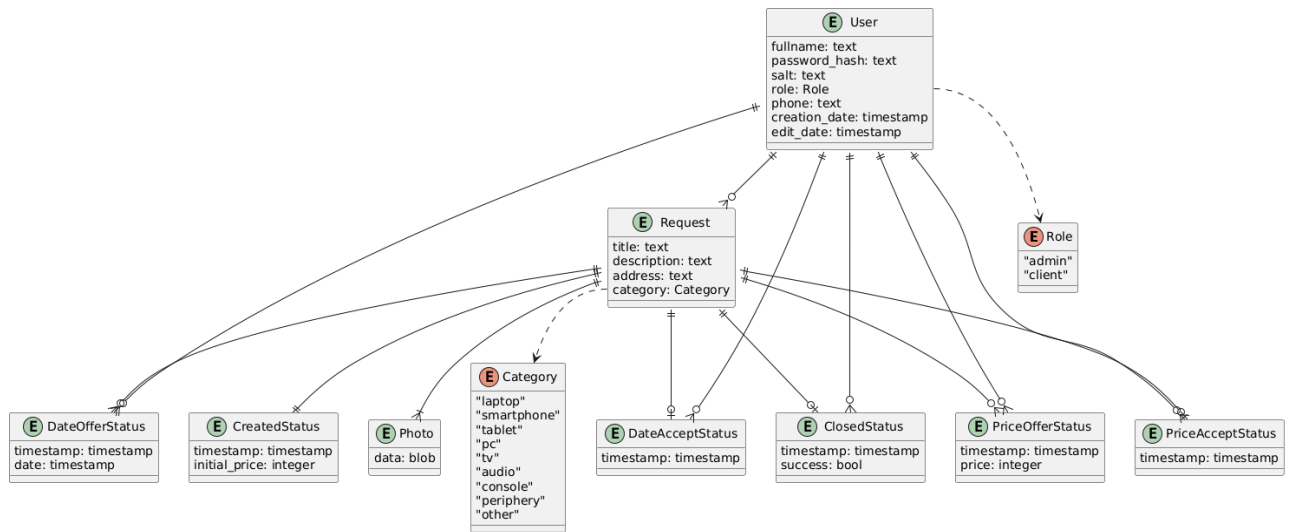
На главной странице нажмите “Перейти в профиль”. При необходимости обновите ФИО и телефон и нажмите на кнопку “Сохранить изменения”.

ПРИЛОЖЕНИЕ В НЕРЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ



ПРИЛОЖЕНИЕ Г

РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ



ПРИЛОЖЕНИЕ Д МАКЕТ UI

