Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский политехнический университет»**

Факультет: Прикладной математики и механики

Кафедра: Вычислительной математики, механики и биомеханики

Направление: 09.03.02 «Информационные системы и технологии»

Профиль бакалавриата: «Информационные системы и технологии»

**ПРАКТИЧЕСКАЯ РАБОТА**

по дисциплине

**«КОРПОРАТИВНЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ»**

Тема: **«Портфолио»**

Выполнили:

ЦТУ-20-3Б

студенты гр.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кольчерин Александр Алексеевич

Юнусов Василь Равилевич

Шаехов Андрей Евгеньевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ф.И.О.)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*(подпись)*

Принял:

ст. преподаватель, Банников Р.Ю.

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(должность, ФИО руководителя)*

*\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(оценка) (подпись)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(дата)*

**Пермь 2024**

**Оглавление**

[Создание ТЗ на разработку КИС 3](#_Toc163400265)

[1. Введение 3](#_Toc163400266)

[1.1 Назначение 3](#_Toc163400267)

[1.2. Цели 4](#_Toc163400268)

[2. Функциональные требования 4](#_Toc163400269)

[2.1. Регистрация и авторизация 4](#_Toc163400270)

[2.2 Генерация отчетов 4](#_Toc163400271)

[2.3 Обеспечение устойчивости системы 4](#_Toc163400272)

[2.4 Поддержка языков 4](#_Toc163400273)

[3. Требования к дизайну 4](#_Toc163400274)

[4. Требования к надежности и безопасности 5](#_Toc163400275)

[5. Требования к тестированию 5](#_Toc163400276)

[6. Стадии и этапы разработки 5](#_Toc163400277)

[Проектирование хранилища данных КИС 6](#_Toc163400278)

[Проектирование интерфейса КИС 11](#_Toc163400279)

[Проектирование диаграммы классов КИС 16](#_Toc163400280)

[Программная реализация КИС 17](#_Toc163400281)

[Тестирование КИС 22](#_Toc163400282)

[Тестирование 22](#_Toc163400283)

[Тест авторизации 22](#_Toc163400284)

[Тест генерации отчетов 24](#_Toc163400285)

[Тест добавления, удаления и изменения данных 24](#_Toc163400286)

[Приложение 26](#_Toc163400287)

[Код Java приложения 26](#_Toc163400288)

[Код Frontend части приложения 94](#_Toc163400289)

Создание ТЗ на разработку КИС

1. Введение

1.1 Назначение

Техническое задание (ТЗ) разработано для определения требований к созданию веб-приложения для администрирования системы подачи показаний счетчиков. Заказчиком данной системы выступает компания «Геликон Про». Разработчики данной системы – три студента группы ЦТУ-20-3б. Прежде всего, следует подробно описать функционал, который реализовывает администрируемая система.

Данная система должна облегчить процесс подачи показаний в соответствующие организации путём предоставления пользователю возможности организации данного процесса при помощи фотографирования лицевой стороны счетчика и подтверждения правильности показаний в мобильном приложении. Ниже приведен список требований к данному проекту:

* Регистрация и авторизация пользователей;
* Реализация личного кабинета пользователя для предоставления пользователю возможности добавления/редактирования информации своей учетной записи;
* Реализация функционала меню администрирования для корректирования ошибочных данных;
* Предоставление пользователю возможности регистрации ТСЖ, в котором он будет являться управляющим. На данном этапе пользователь должен указать дома и квартиры, которые состоят в этом ТСЖ и далее, когда пользователи будут регистрироваться в системе и указывать квартиры, в которых они проживают, они должны автоматически быть оформлены, как участники данного ТСЖ;
* Обработка функционала пользователей в зависимости от их ролей. На данный момент предполагается использование пяти ролей:
  + Администратор. Имеет возможность корректировки ошибочных данных в системе;
  + Управляющий ТСЖ. Имеет возможность добавления и исключения пользователей в ТСЖ, управляющим которого он является;
  + Владелец квартиры. Имеет возможность регистрации счетчиков и подачи показаний в своей квартире. Также, данная роль позволяет назначать других пользователей арендаторами в своей квартире;
  + Арендатор. Подразумевает возможность подачи показаний в квартире, арендатором которой он является;
  + Пустая роль. Присваивается по умолчанию при регистрации, предполагается, что позже, при присоединении пользователя к ТСЖ, его роль будет изменена на роли владельца или арендатора квартиры.

• Предоставление пользователю возможности сканирования показаний посредством фотографирования лицевой стороны счетчика;

• Предоставление пользователю возможности отслеживания поданных показаний и приблизительная стоимость оплаты услуг за каждый период;

1.2. Цели

* Создать удобную и интуитивно понятную платформу для администрирования системы подачи показаний счетчиков.
* Обеспечить возможность соблюдения целостности данных.

2. Функциональные требования

2.1. Регистрация и авторизация

* Пользователи должны иметь возможность создать учетную запись.
* Система должна обеспечивать проверку подлинности пользователей и защиту от несанкционированного доступа.
* Система должна поддерживать разграничение функционала по ролям:
* Роль администратор – пользователь, имеющий возможность редактирования данных.
* Роль оператор службы поддержки – пользователь, который может просматривать отчеты.

2.2 Генерация отчетов

Система должна генерировать 5 отчетов:

1. Вывод данных показателей приборов учета всех пользователей за месяц
2. Вывод данных показателей приборов учета определенного пользователя за месяц
3. Вывод информации о жильцах, их квартирах по ТСЖ
4. Ввод информации о жильцах определенной квартиры
5. Вывод всех ТСЖ и принадлежащих им ресурсоснабжающих компаний. Этот отчет будет экспортироваться в xls файл.
6. Вывод информации о квартирах, в которых зарегистрирован арендатор.
7. Вывод всех показаний дома определенного месяца.

2.3 Обеспечение устойчивости системы

* Система должна реагировать на исключительные ситуации выводом сообщения об ошибке.
* Система должна корректно валидировать входные данные от пользователя.

Кроме того, администратор и оператор службы поддержки могут корректировать данные в системе, что предоставляет дополнительный фактор устойчивости системы.

2.4 Поддержка языков

Администрируемая система поддерживает русский язык, поэтому требуется поддержка русского языка.

3. **Требования к дизайну**

UI/UX часть приложения должна удовлетворять следующим требованиям:

* интуитивно понятный интерфейс: приложение должно быть легким в использовании и понятным для пользователей всех уровней навыков. Это включает в себя простую навигацию, понятные иконки и интуитивно понятные элементы управления;
* четкая структура и организация: приложение должно иметь четкую структуру и организацию, чтобы пользователи могли легко найти нужные функции и информацию;
* единый дизайн: все элементы интерфейса должны быть оформлены в едином стиле и иметь согласованный дизайн. Это поможет создать единое визуальное впечатление и обеспечить легкость восприятия информации;
* ясные инструкции и подсказки: приложение должно предоставлять ясные инструкции и подсказки, чтобы помочь пользователям понять, как использовать различные функции и выполнять задачи;
* визуальная привлекательность: приложение должно иметь привлекательный дизайн и графику, чтобы привлечь внимание пользователей и создать положительное впечатление;
* удобство использования: приложение должно быть удобным в использовании и предоставлять пользователю возможность выполнять задачи быстро и эффективно;

4. Требования к надежности и безопасности

Пользовательские данные будут защищены от несанкционированного доступа и передачи. Система должна поддерживать парольную аутентификацию. Требуется, чтобы пароли хранились в хешированном виде.

5. Требования к тестированию

* Интеграционное тестирование: необходимо провести тестирование взаимодействия компонентов системы для обнаружения возможных ошибок;
* Ручное тестирование: необходимо провести тестирование для каждого модуля системы.

6. Стадии и этапы разработки

На стадии «Технический (и рабочий) проект» должны быть выполнены перечисленные ниже этапы работ:

* Проектирование хранилища. Форма отчётности – отчёт. Срок выполнения – не позднее 13.02.2024.
* Проектирование интерфейса. Форма отчётности – отчёт. Срок выполнения – не позднее 20.02.2024.
* Разработка диаграммы классов. Форма отчётности – отчёт. Срок выполнения – не позднее 27.02.2024.
* Программная реализация. Форма отчётности – отчёт и демонстрация заявленного

функционала ИС. Срок выполнения – не позднее 19.03.2024.

* Тестирование. Форма отчётности – отчёт. Срок выполнения – не позднее 30.03.2024.

Проектирование хранилища данных КИС

Предметная область

Основной сущностью данной системы является пользователь. Сущность содержит основную информацию о пользователе, логин, пароль, ФИО и номер телефона. Каждый пользователь связан с квартирой, посредством промежуточной таблицы, которая содержит роль жильца данной квартиры. Сущность ТСЖ так же привязана к пользователю. К данной сущности привязан адрес, к которому уже привязана квартира. К каждой квартире привязан счетчик, который содержит информацию о данном счетчике, для его идентификации. История показаний счетчика является отдельной сущностью. Она содержит в себе показания приборов учета, которые занес пользователь и приблизительную стоимость услуг. Так же к счетчику привязана сущность «ресурс» (газ, вода, электричество) и «цена» на данный ресурс. К ресурсу привязан провайдер, который предоставляет услуги.

Логическая модель

Логическая модель данных корпоративной информационной системы представлена на рисунке 1.

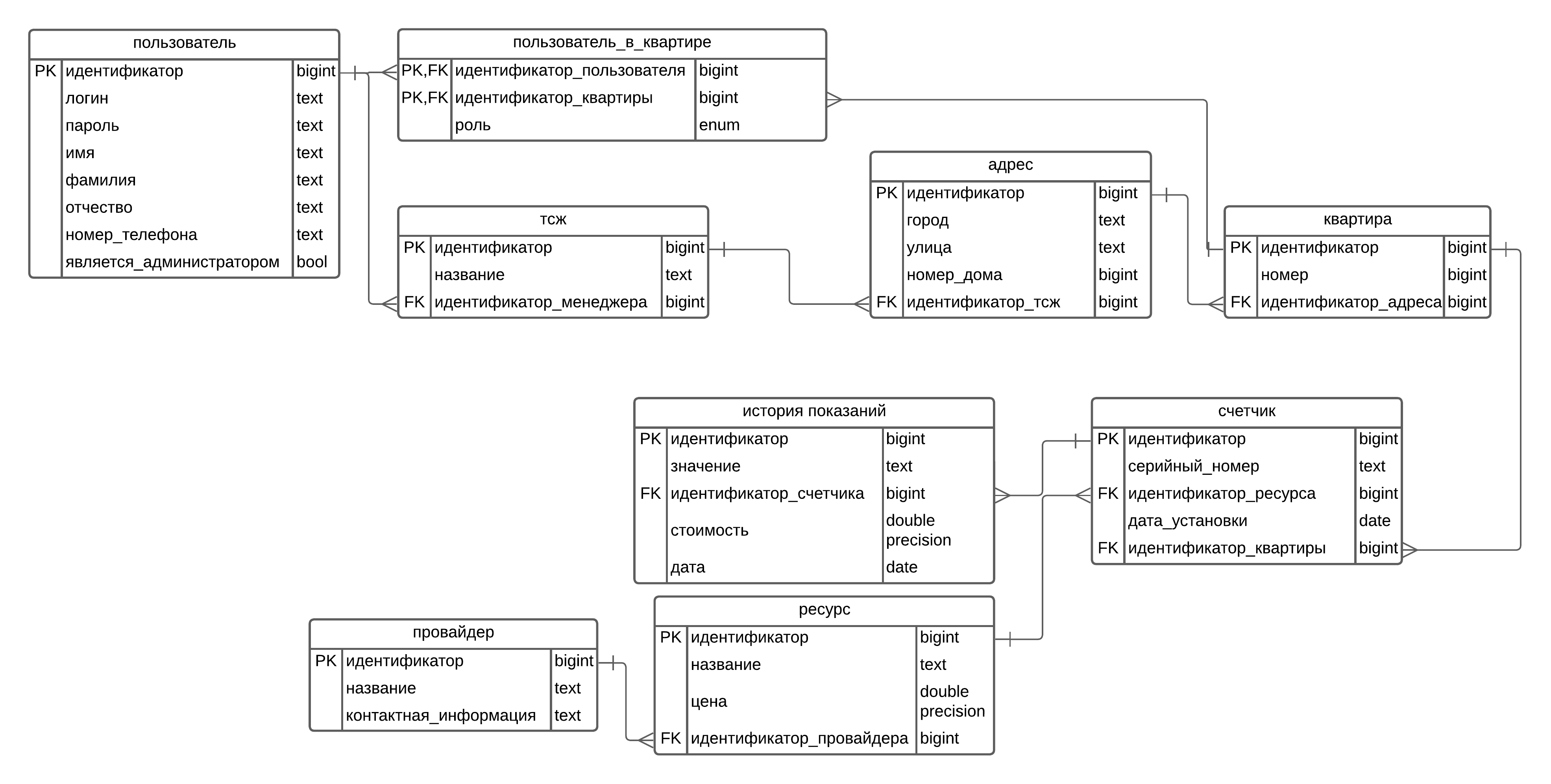


Рисунок 1. Логическая модель системы.

**Обоснование выбранных типов данных:**

Все текстовые поля базы данных хранятся в типе text, который неограничен по длине. Такое решение было принято в связи с тем, чтобы в дальнейшем не возникло никаких проблем, связанных с физическими ограничениями базы данных, хотя такое решение является не самым оптимальным.

Поля таблицы «Пользователь» как правило хранятся в типах text. Это связано с тем, чтобы не ограничивать пользователя в длине его данных. Поле «является\_администратором» может содержать всего два значения, поэтому было выбран тип данных bool.

Поле «серийный номер» таблицы «Счетчик» и поле «значение» таблицы «История показаний» были выбраны типа text, для того, чтобы система была максимально универсальной и могла хранить серийные номера и значения счетчиков любых типов и форматов.

Поле «контактная\_информация» таблицы «Провайдер» хранится в типе text, так как эта контактная информация может содержать в себе хоть номер телефона, хоть адрес, хоть какую-либо другую информацию, полезную для пользователя.

Таблицы «Адрес» и «Квартира» разбиты на две разные по причине того, чтобы не хранить всю информацию об адресе для каждой квартиры. Кроме того, так более удобно добавлять новые квартиры, в уже зарегистрированный дом, когда дом уже зарегистрирован.

Физическая модель

Физическая модель данных корпоративной информационной системы представлена на рисунке 2.

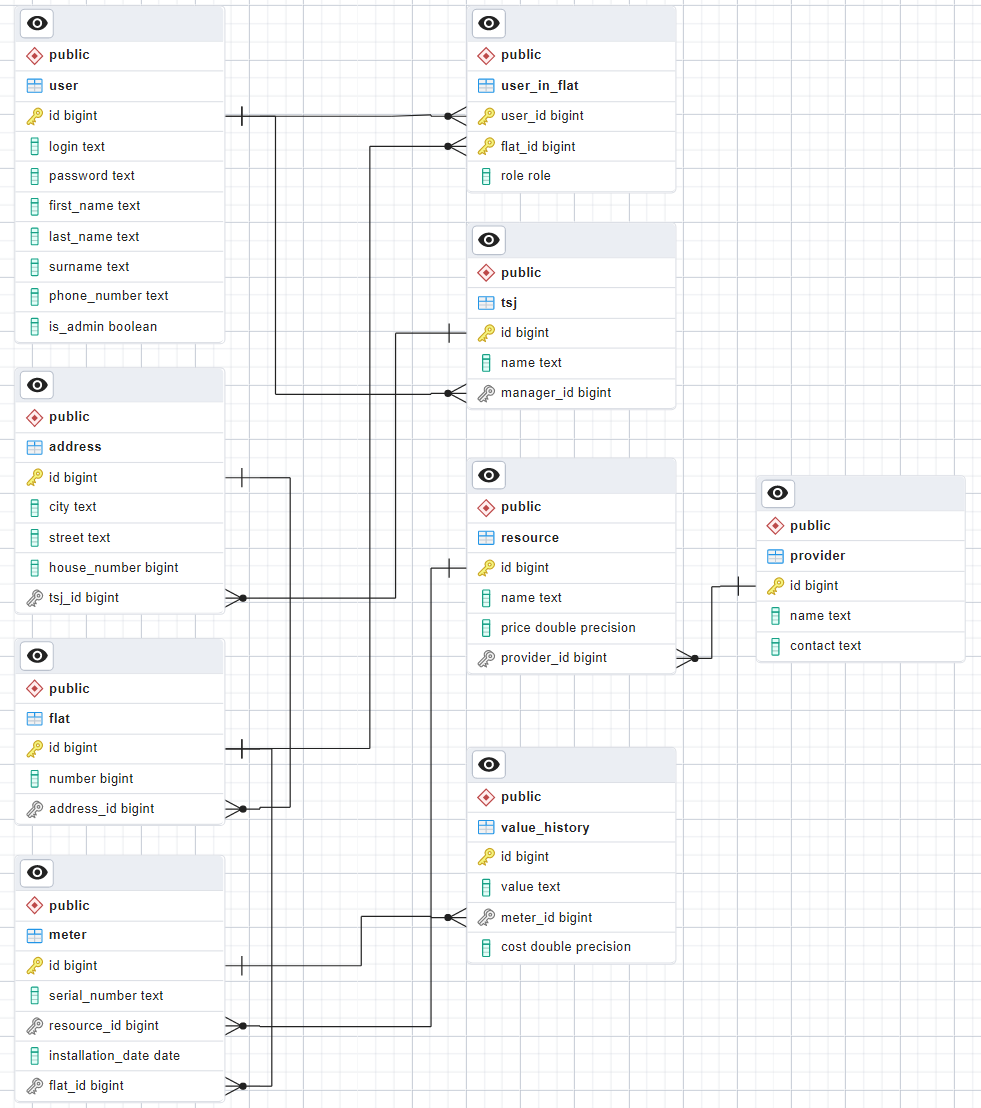


Рисунок 2. Физическая модель системы.

Содержимое БД

На рисунке 3 изображено содержимое таблицы «user».

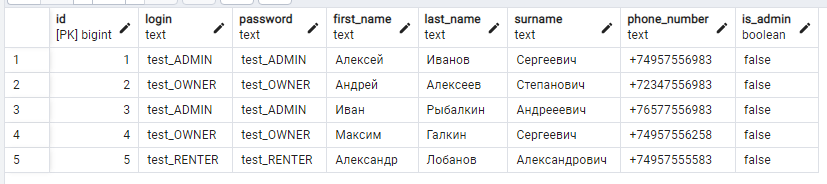


Рисунок 3. Содержимое таблицы «user».

На рисунке 4 изображено содержимое таблицы «flat».

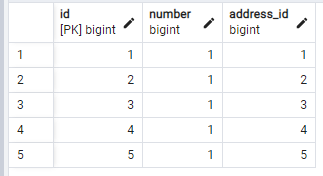


Рисунок 4. Содержимое таблицы «flat».

На рисунке 5 изображено содержимое таблицы «tsj».

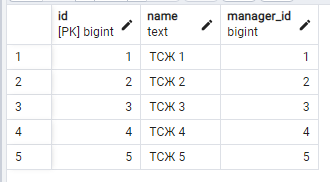


Рисунок 5. Содержимое таблицы «tsj».

На рисунке 6 изображено содержимое таблицы «meter».

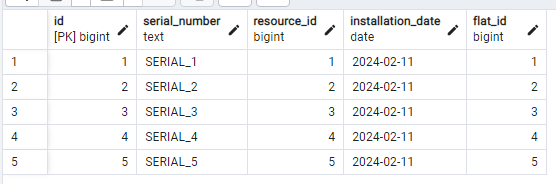


Рисунок 6. Содержимое таблицы «meter».

На рисунке 7 изображено содержимое таблицы «value\_history».

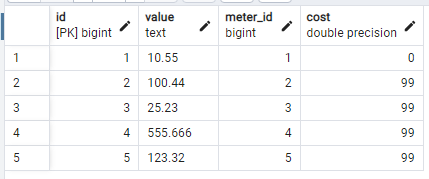


Рисунок 7. Содержимое таблицы «value\_history».

На рисунке 8 изображено содержимое таблицы «provider».

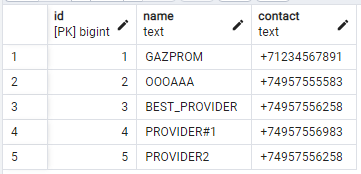


Рисунок 8. Содержимое таблицы «provider».

На рисунке 9 изображено содержимое таблицы «resource».

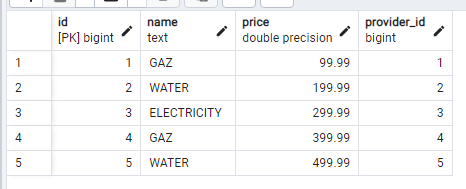


Рисунок 9. Содержимое таблицы «resource».

На рисунке 10 изображено содержимое таблицы «user\_id\_flat».

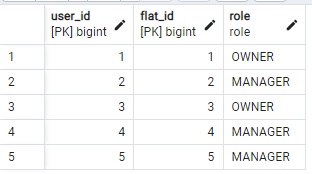


Рисунок 10. Содержимое таблицы «user\_id\_flat».

Проектирование интерфейса КИС

Интерфейс

Интерфейс спроектирован на веб-платформе Figma. В течение разработки дизайн может изменяться.

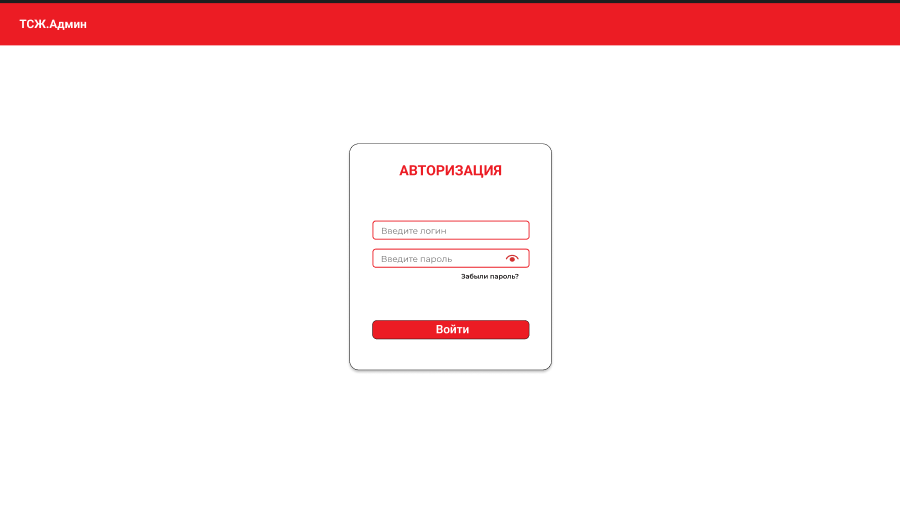
На рисунке 11 представлена форма авторизации.

Рисунок 11. Форма авторизации.

На рисунке 12 показан вид приложения от лица сотрудника ТСЖ.

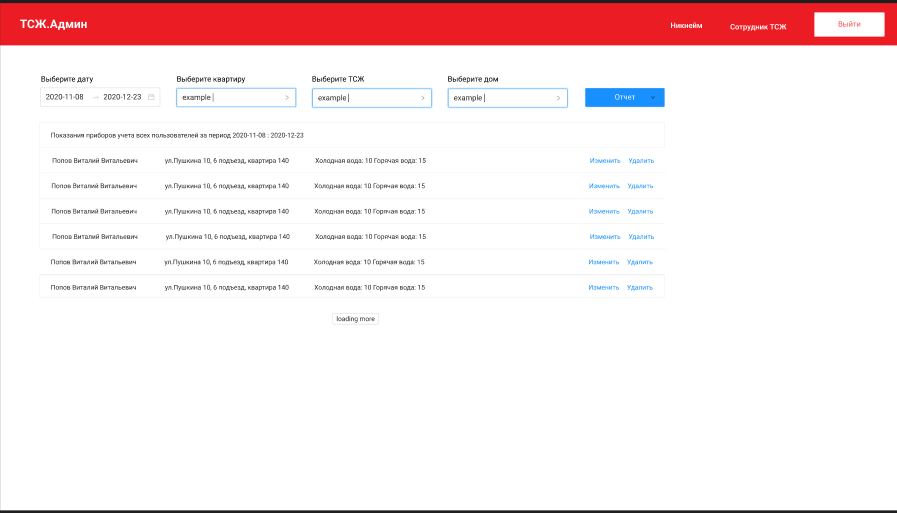


Рисунок 12. Вид приложения от лица сотрудника ТСЖ.

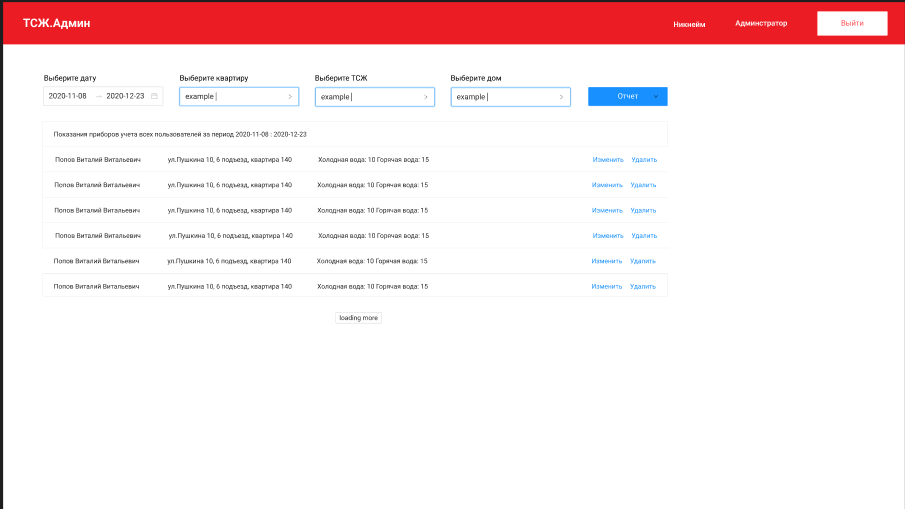


Рисунок 13. Вид приложения от лица администратора.

На рисунке 14 и 15 показано отличие администратора от сотрудника ТСЖ. У администратора есть доступ ко всем ТСЖ, а у сотрудника ТСЖ будет доступ только к дому, квартире, ТСЖ к которому он привязан.

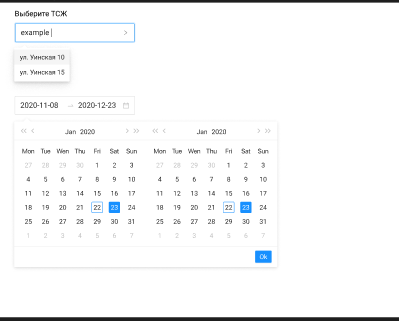
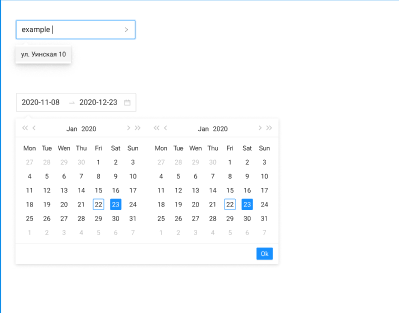
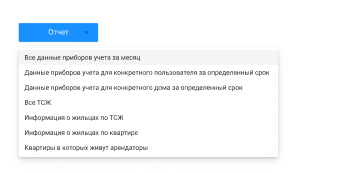


Рисунок 14. Вид окна выбора от лица администратора.



Рисунок 15. Вид окна выбора от лица сотрудника ТСЖ.

На рисунке 16 показана кнопка и выпадающее окно с выбором типа отчета.

Рисунок 16. Кнопка и выпадающее окно с выбором типа отчета.

На рисунке 17 показано изменение показаний.

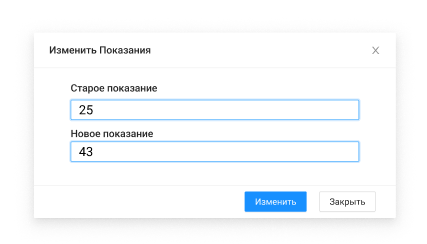


Рисунок 17. Меню изменения показаний.

На рисунках 18, 19 представлены разные виды отчетов.

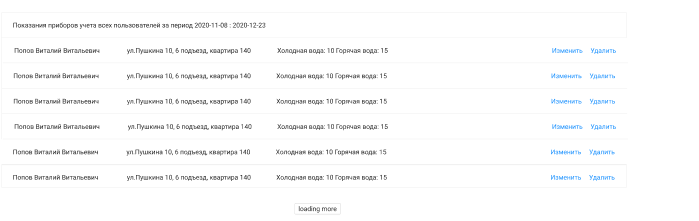


Рисунок 18. Вид отчета, который выводится в виде списка и имеет возможность изменения данных.

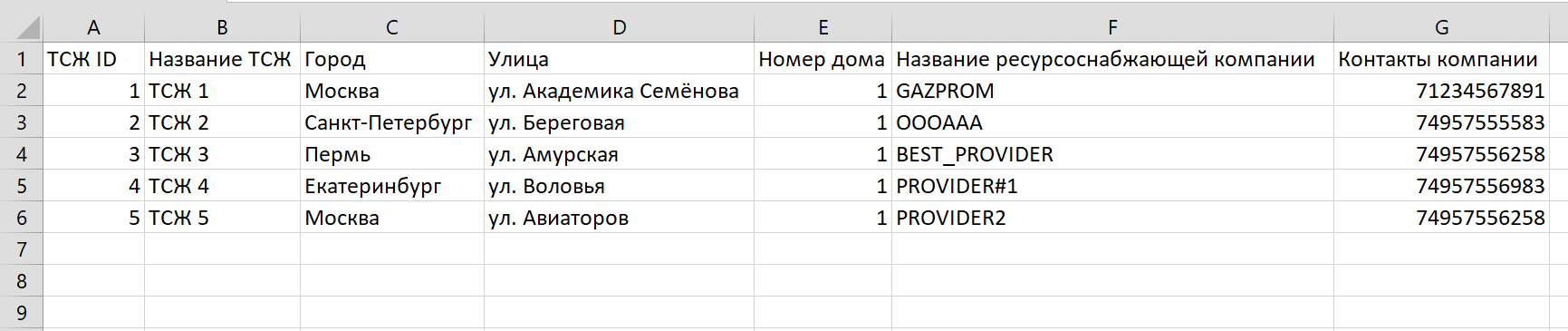


Рисунок 19. Excel-файл с примером отчёта.

На рисунках 20 и 21 представлена реакция на ошибочный ввод и неполный ввод.

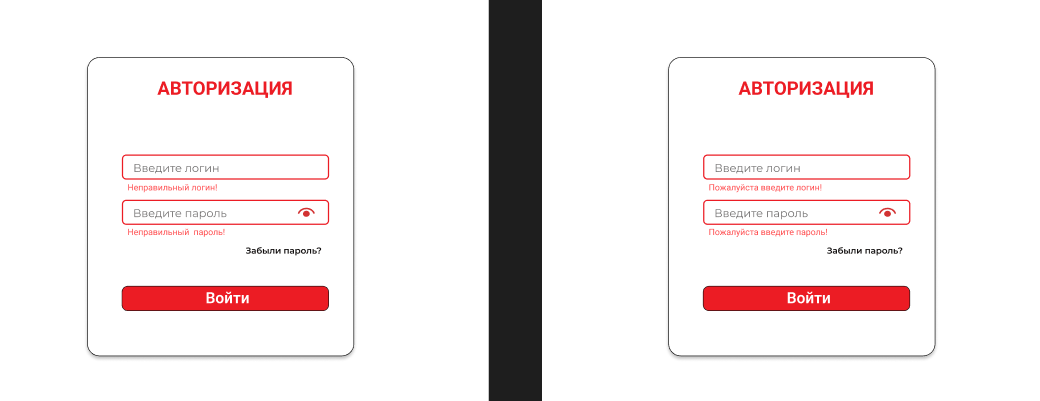


Рисунок 20. Реакция на ошибочный ввод при авторизации.

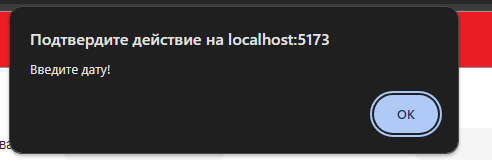


Рисунок 21. Реакция на отсутсвие ввода при выборе квартиры, ТСЖ или дома.

Проектирование диаграммы классов КИС

Диаграмма классов

Далее предоставлены диаграммы классов, созданные с помощью инструмента yFiles.

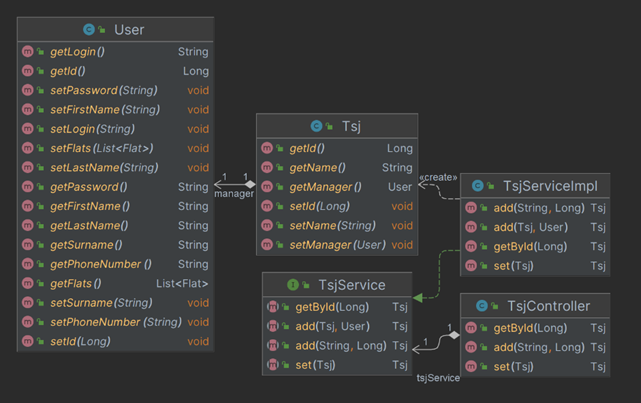


Рисунок 22. Диаграмма классов, связанных с пользователями и ТСЖ

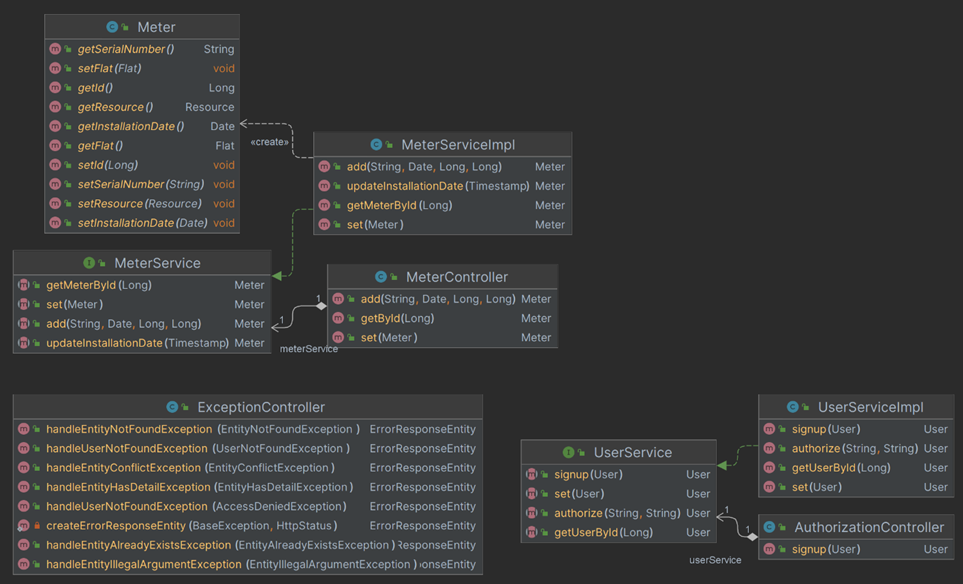


Рисунок 23. Диаграмма классов, связанных со счетчиками, авторизацией и обработкой исключений.

Описание диаграммы классов

Класс User – представляет сущность пользователя. Он содержит методы get и set для его полей.

Класс UserServiceImpl – отвечает за добавление, получение и изменения информации о пользователе.

Класс AuthorizationController – принимает http запросы и взаимодействует с UserServiceImpl.

Класс Meter – представляет сущность счетчика. Он содержит методы get и set для его полей.

Класс MeterServiceImpl – содержит логику работы со счетчиками, позволяет получать информацию и изменять ее.

Класс MeterController – принимает http запросы и взаимодействует с MeterServiceImpl.

Класс Tsj – представляет сущность ТСЖ. Он содержит методы get и set для его полей.

Класс TsjServiceImpl – содержит логику работы с ТСЖ, позволяет получать информацию и изменять ее.

Класс TsjController – принимает http запросы и взаимодействует с TsjServiceImpl.

Класс ExeptionController – предназначен для того, чтобы при возникновении исключений приложение отправляло статус о сбое.

Программная реализация КИС

Цель работы

В ходе работы должна быть реализована корпоративная информационная система по администрированию системы подачи показаний счетчиков. КИС должна поддерживать добавление/удаление и изменение показаний счетчиков, а также вывод 7 отчетов. Доступ к системе должен выполняться с помощью авторизации пользователей.

Реализация

Для реализации программного кода backend был выбран язык программирования Java в связке с фреймворком Spring. В качестве среды разработки была выбрана IntelliJ IDEA. Для реализации программного кода frontend части приложения был выбран язык TypeScript и библиотека React.

Работа с приложением начинается с авторизации.

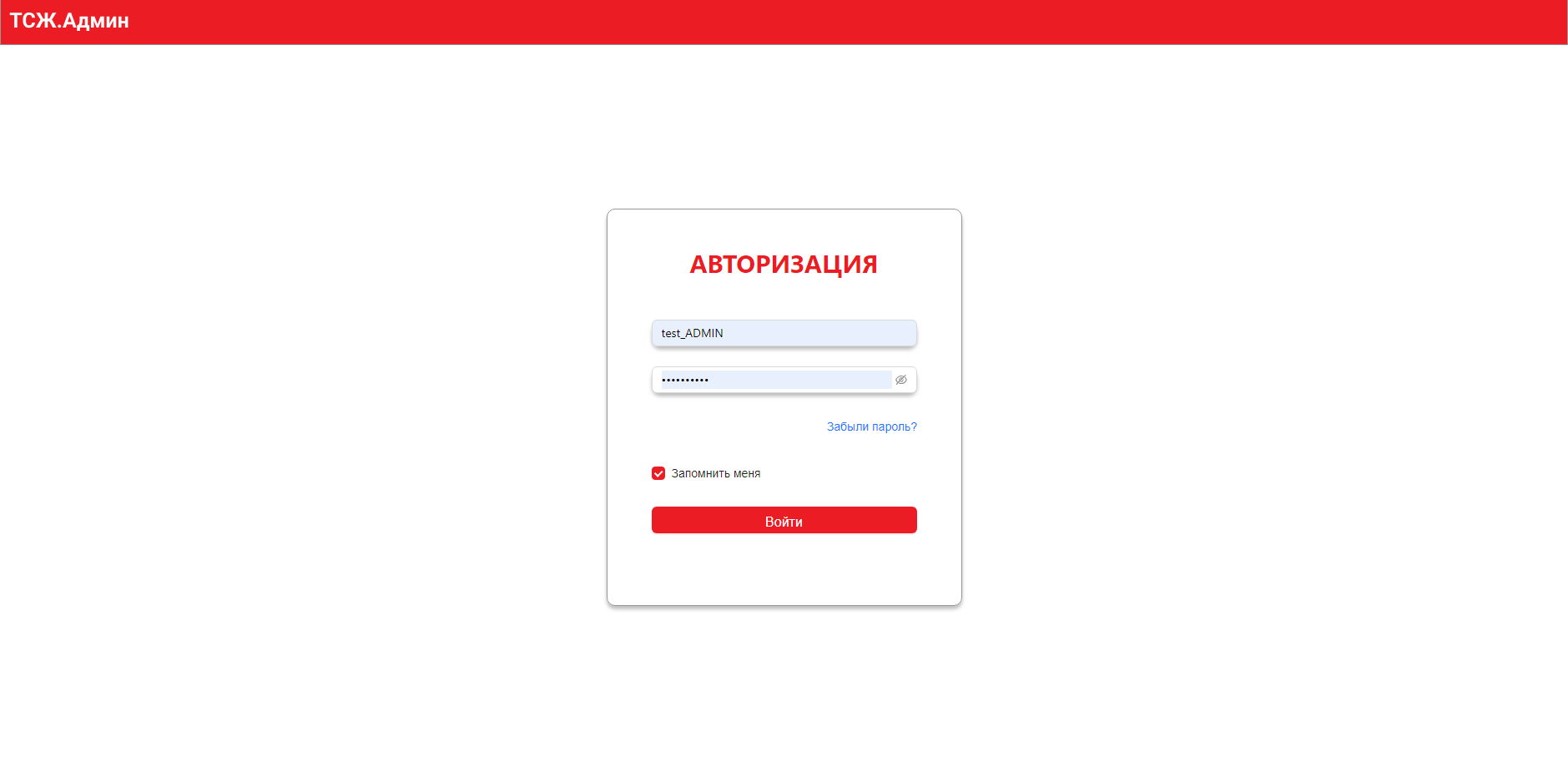


Рисунок 24. Страница авторизации.

После ввода данных необходимо нажать кнопку «Войти», пользователь авторизуется в системе, после чего откроется пустое окно с отчетами. Это продемонстрировано на рисунке 25.

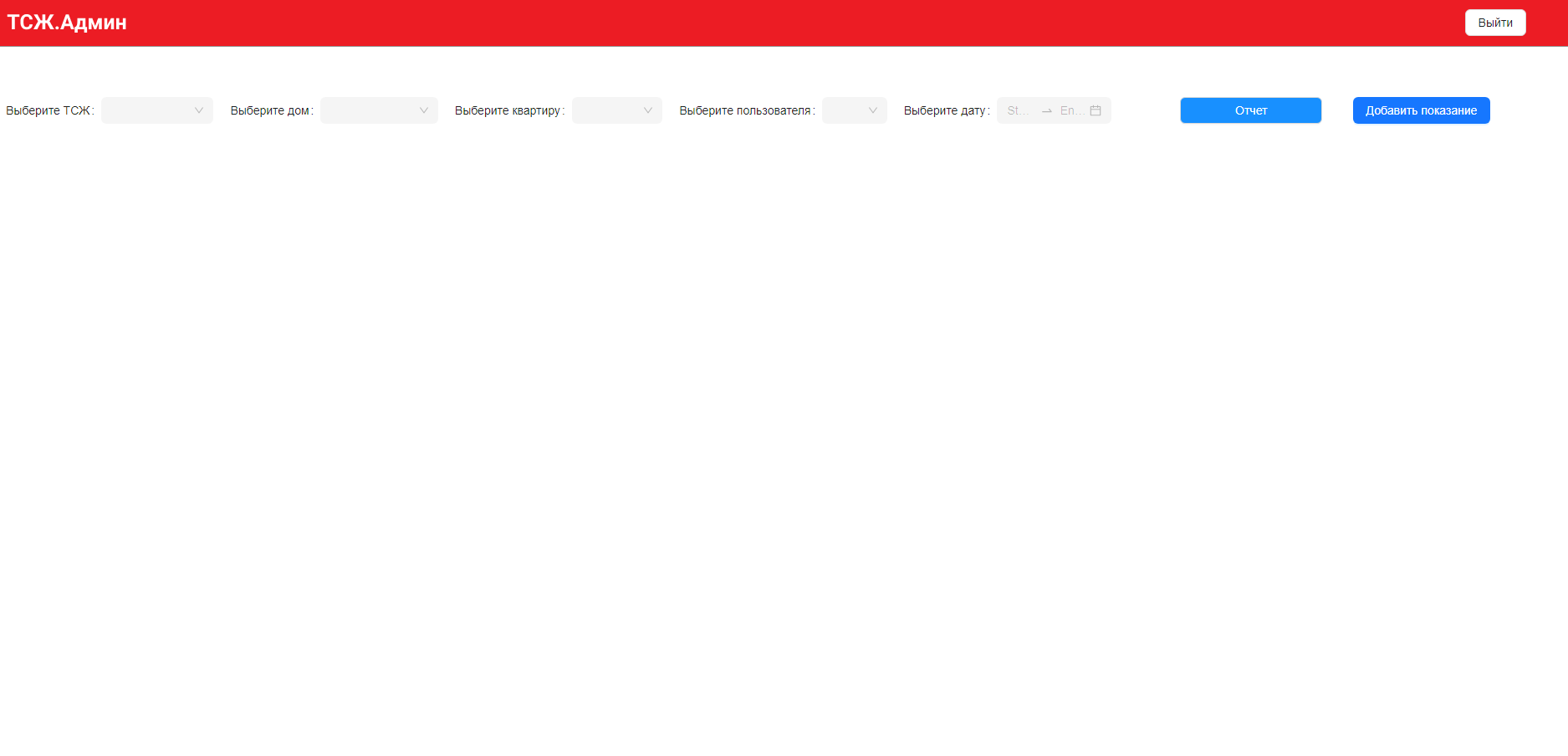


Рисунок 25. Пустое окно с отчетами.

Далее пользователь выбирает данные для формирования отчета и выбирает его форму при помощи нажатия на кнопку «Отчет».

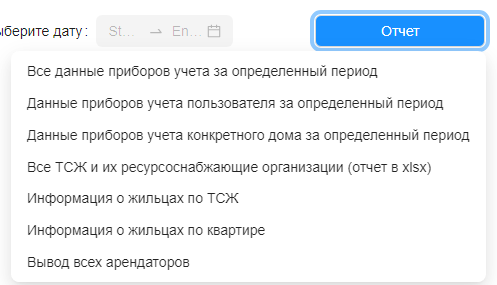


Рисунок 26. Разновидности форм отчета.

На рисунке 27 показан результат отчета по выводу всех показаний счетчиков. При помощи кнопки «Изменить» можно редактировать данные о значениях в показаниях. Кнопка «Удалить» удаляет выбранное показание счетчика.

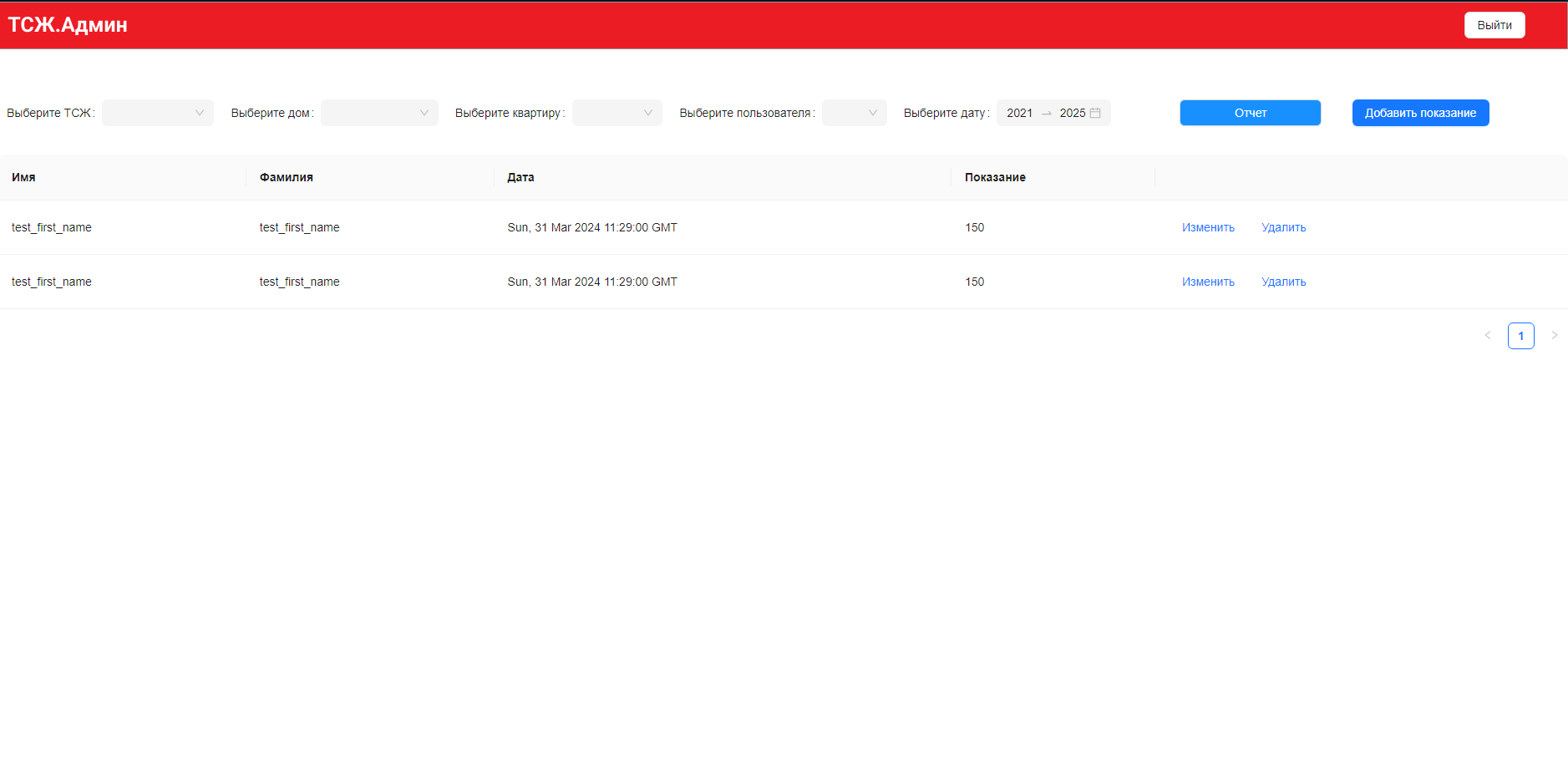


Рисунок 27. Отчет по выводу всех показаний счетчиков.

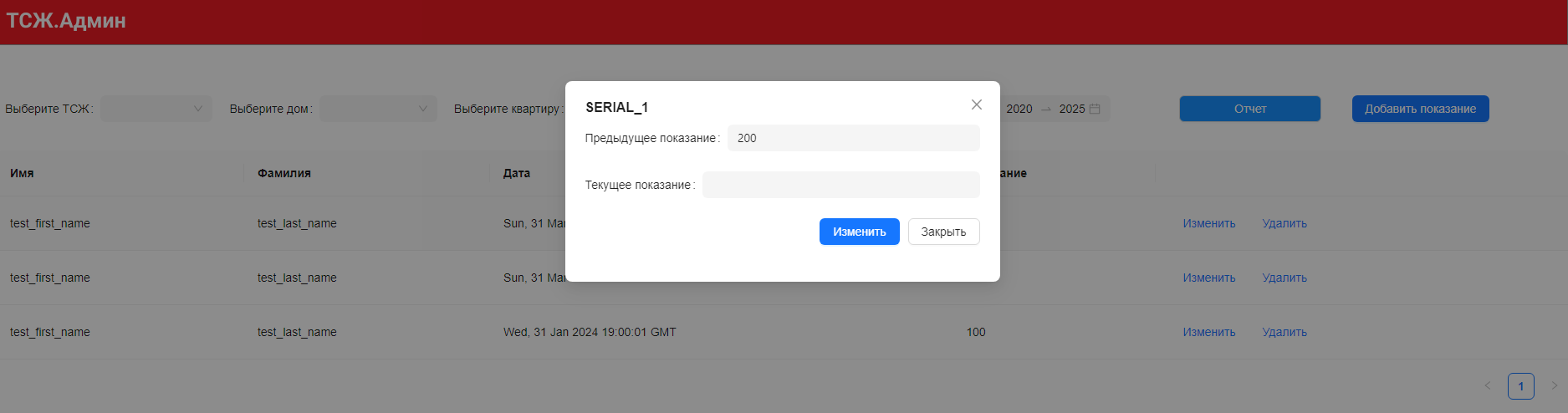


Рисунок 28. Окно редактирования значений в показании счетчика.

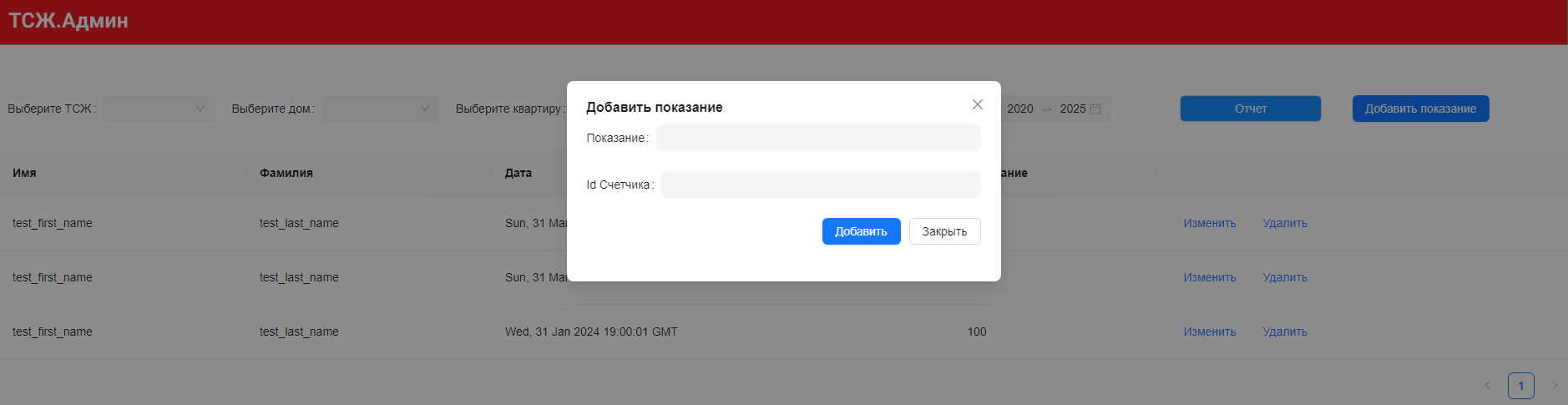


Рисунок 29. Окно добавления значений в показания счетчика.

При выборе вида отчета «Все ТСЖ и их ресурсоснабжающие организации (отчет в xlsx)» открывается окно выбора места сохранения отчета в формате xlsx. Это продемонстрировано на рисунке 30.

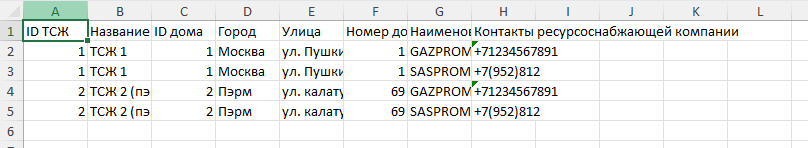


Рисунок 30. Отчет с выводом всех ТСЖ и их ресурсоснабжающих организаций в Excel.

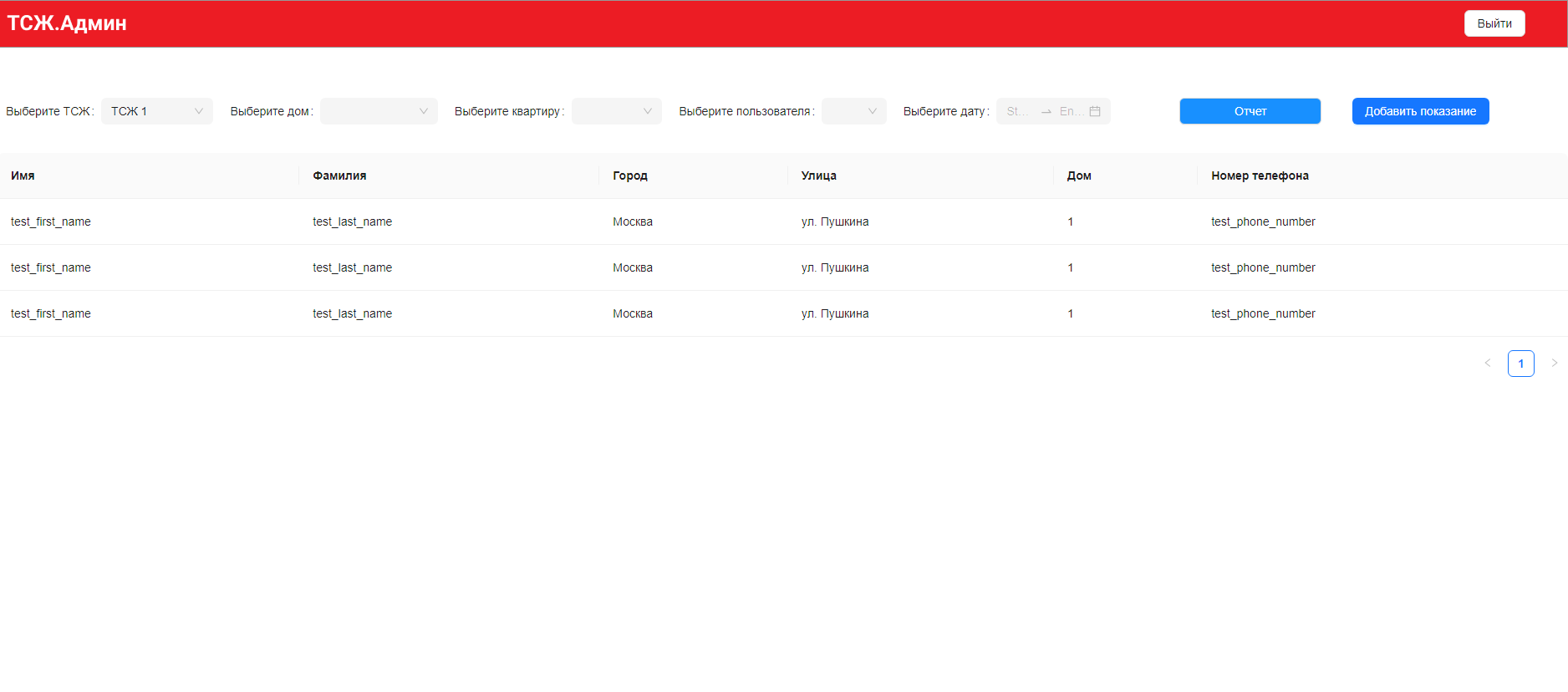


Рисунок 31. Отчет с выводом информации о жильцах по выбранному ТСЖ.

Для получения отчета с данными приборов учета пользователя за определенный период необходимо выбрать пользователя и указать временной период, за который необходимо получить информацию.

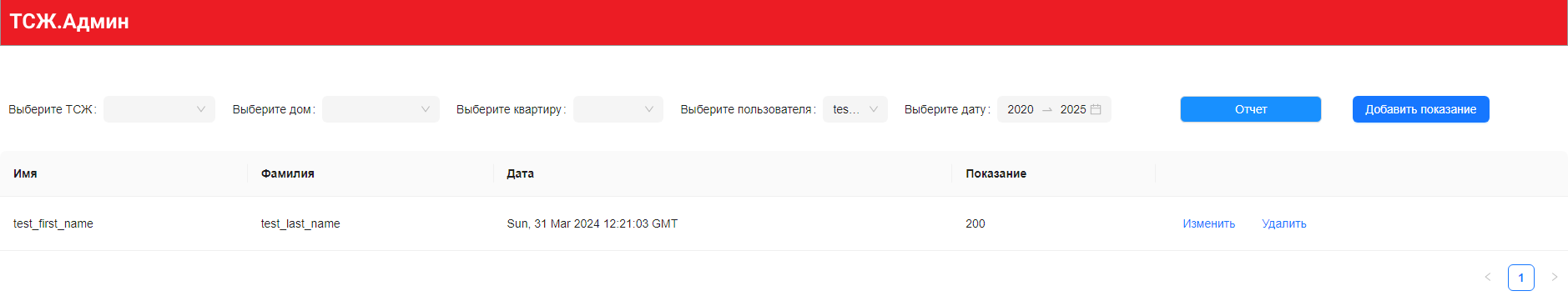


Рисунок 32. Отчет с выводом данных приборов учета пользователя за определенный период.

Для получения отчета с данными приборов учета выбранного дома за определенный период необходимо выбрать из списка необходимый дом и указать временной период, за который необходимо получить информацию.

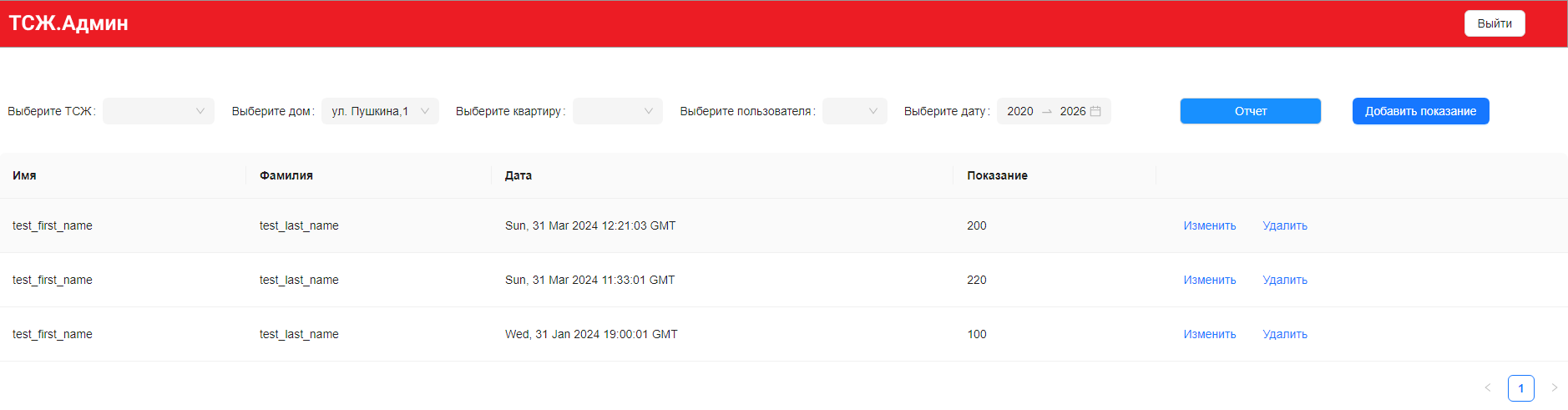


Рисунок 33. Отчет с выводом данных приборов учета выбранного дома за определенный период.

На рисунке 34 продемонстрирован вывод отчета с информацией о жильцах, прикрепленных с конкретному ТСЖ. Для получения отчета необходимо выбрать нужную ТСЖ из списка.

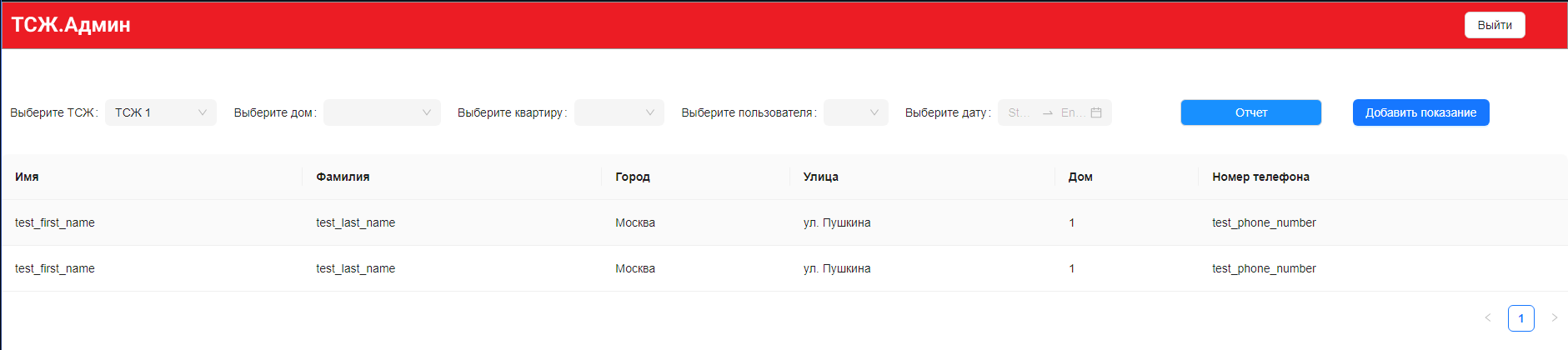


Рисунок 34. Вывод отчета с информацией о жильцах, прикрепленных к конкретному ТСЖ.

На рисунке 35 продемонстрирован вывод отчета с информацией о жильцах, живущих в определенной квартире. Для получения отчета необходимо выбрать нужный адрес дома и номер квартиры.

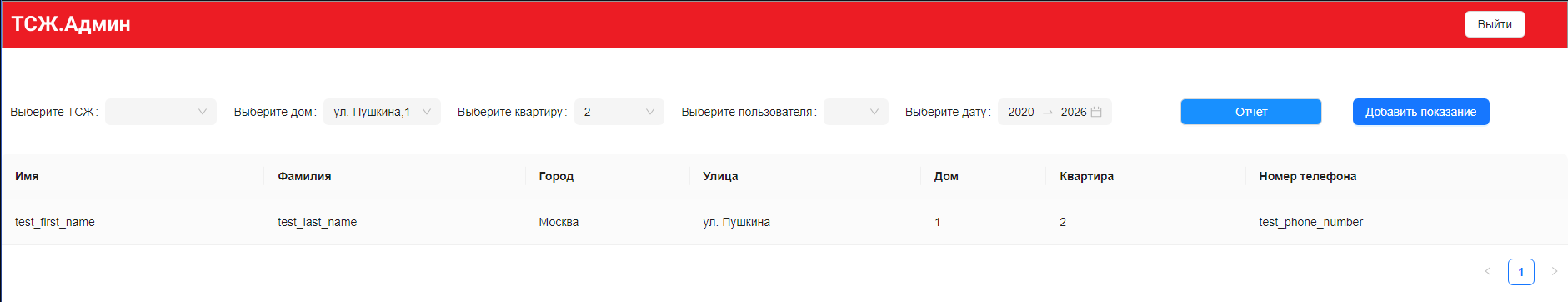


Рисунок 35. Вывод отчета с информацией о жильцах, проживающих в определенной квартире.

На рисунке 36 продемонстрирован отчет о предоставлении информации о всех арендаторах.

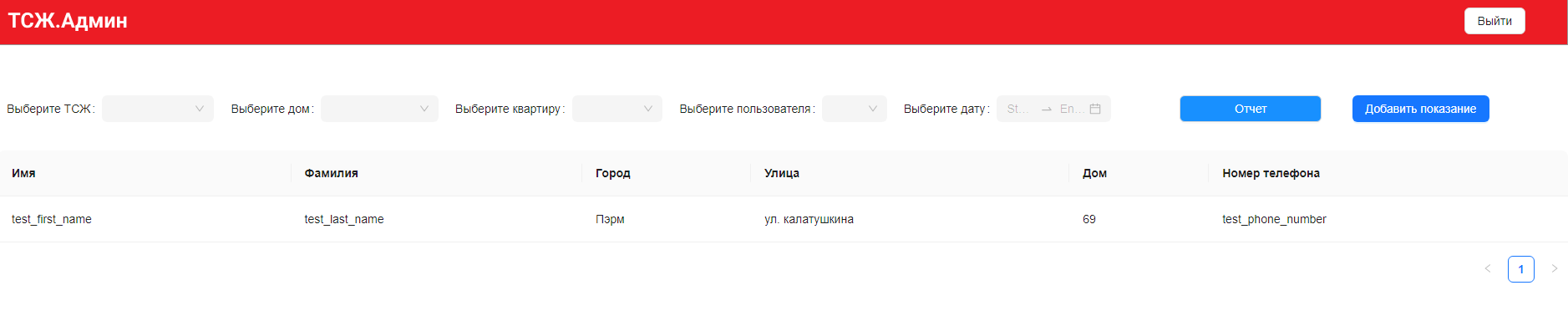


Рисунок 36. Вывод информации о всех арендаторах.

На рисунке 37 представлена реакция ИС на ошибочный ввод данных.

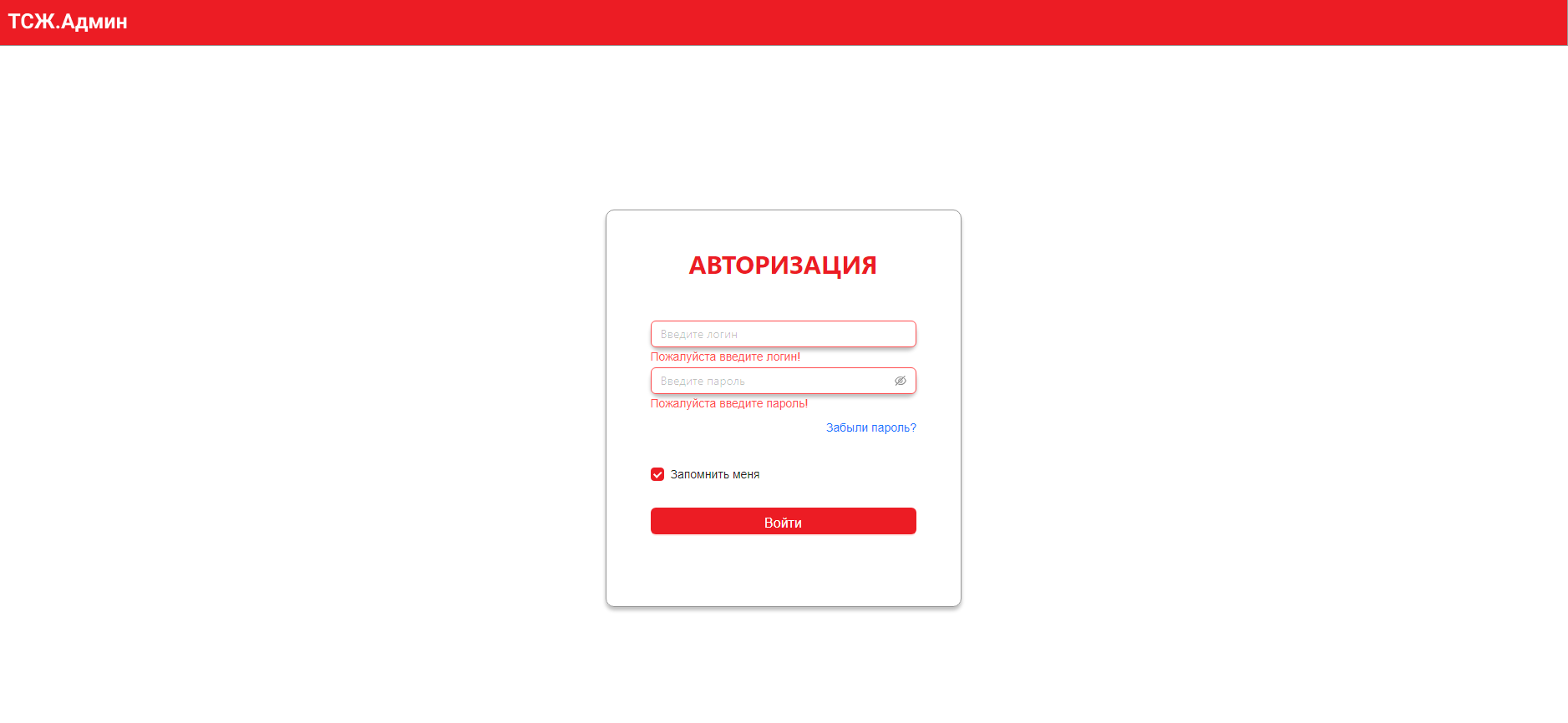


Рисунок 37. Форма авторизации с некорректными данными.

Тестирование КИС

Тестирование

Для проверки корректного функционирования проводилось ручное тестирование.

Тест авторизации

В поля формы авторизации заполняются данные: Login – test\_ADMIN, Password – test\_ADMIN.

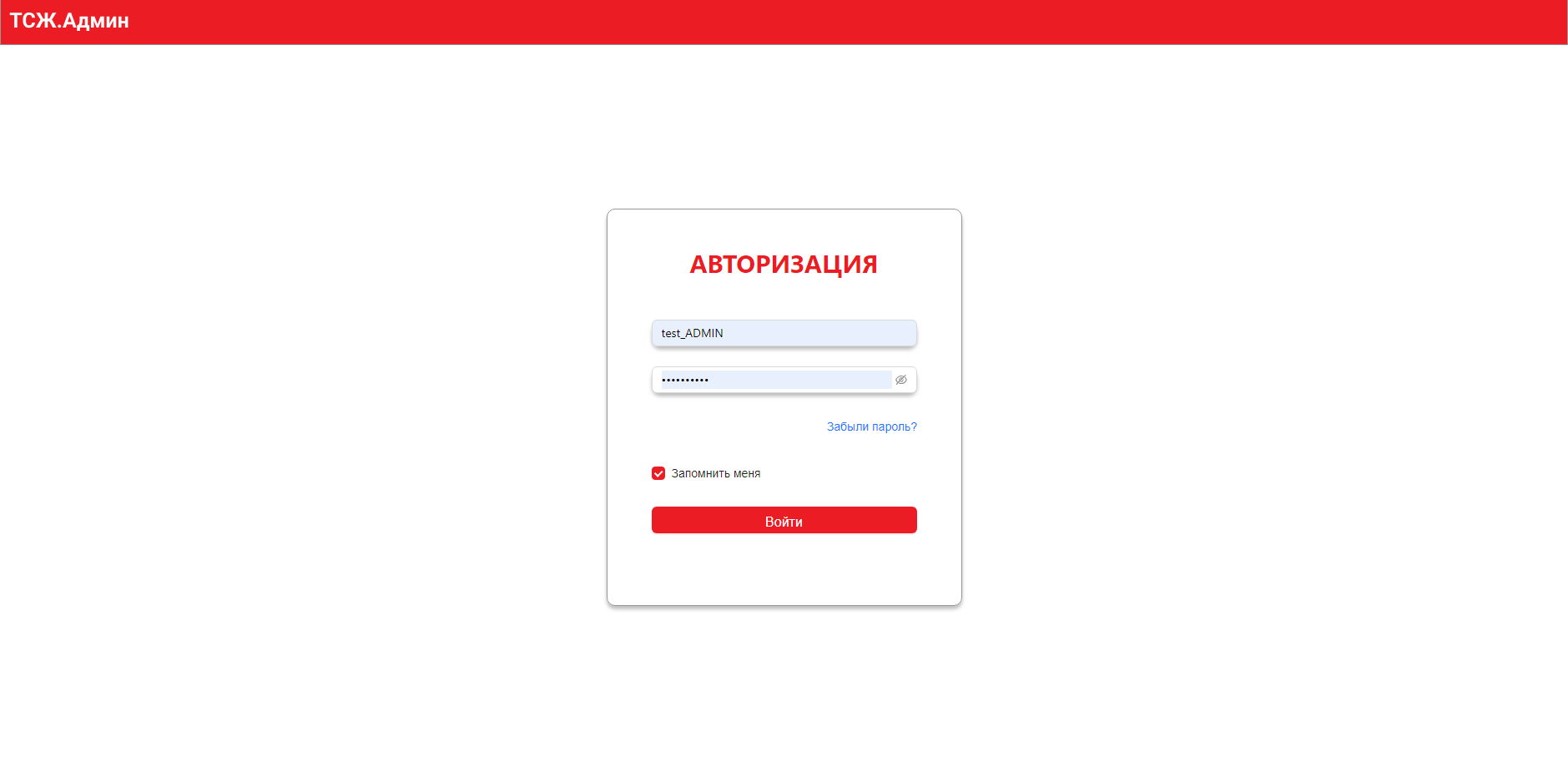


Рисунок 38. Страница авторизация пользователя.

В результате теста пользователь успешно авторизуется и попадает в следующее окно.

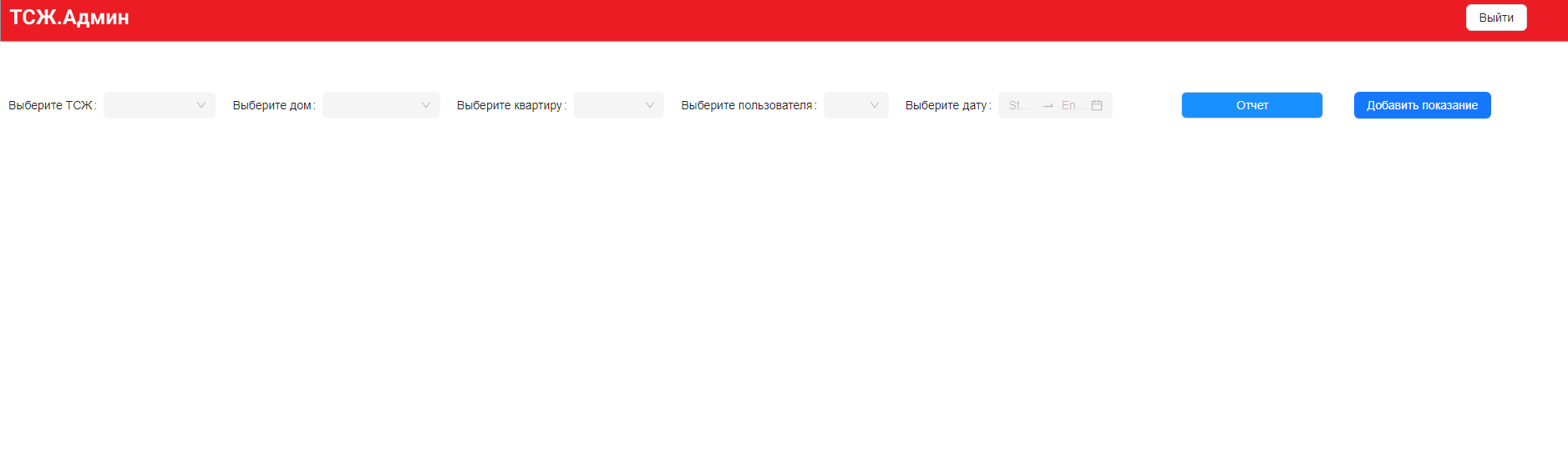


Рисунок 39. Окно после успешной авторизации.

На рисунке 40 продемонстрирована реакция ИС на отсутствие введенных данных.

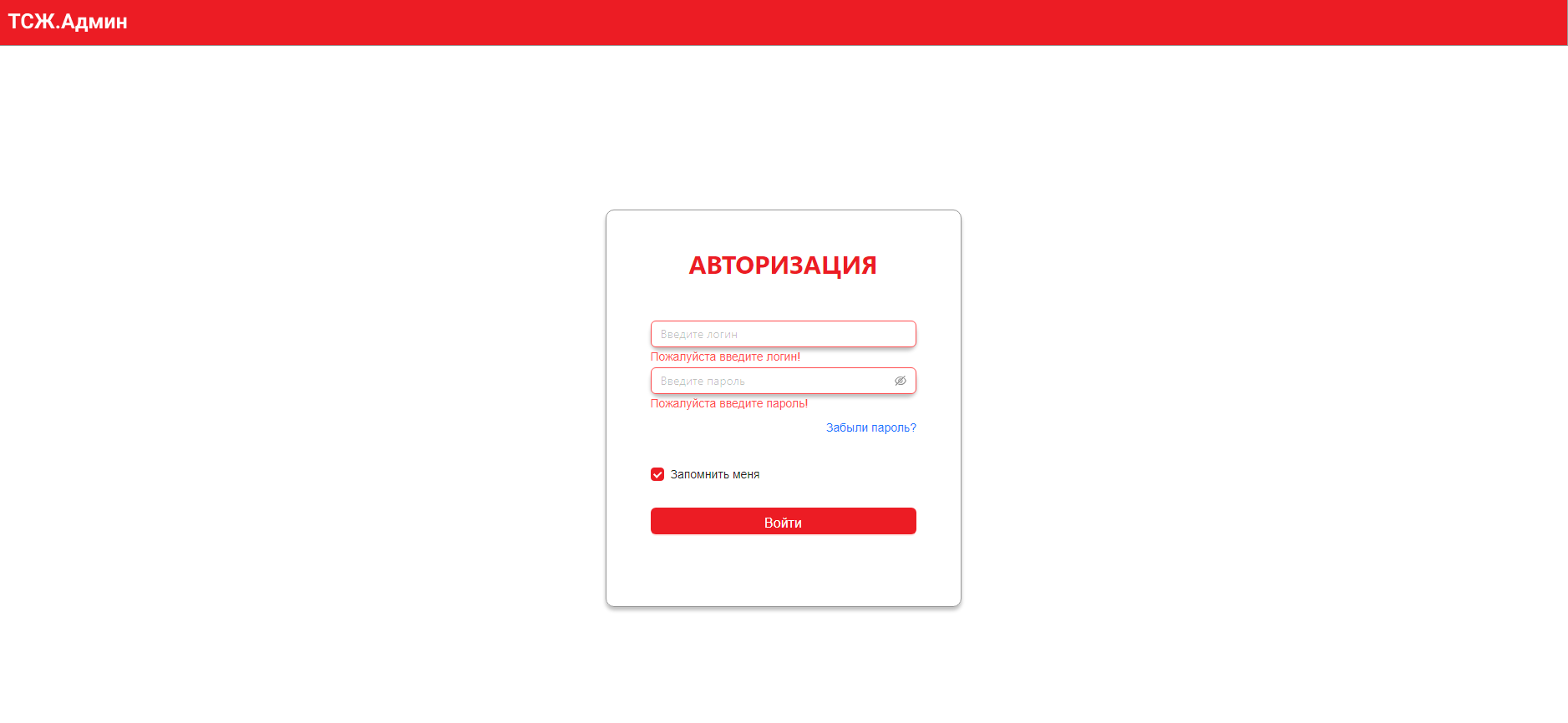


Рисунок 40. Реакция ИС на отсутствие введенных данных для авторизации.

На рисунке 41 показана неудавшаяся авторизация из-за некорректных данных.

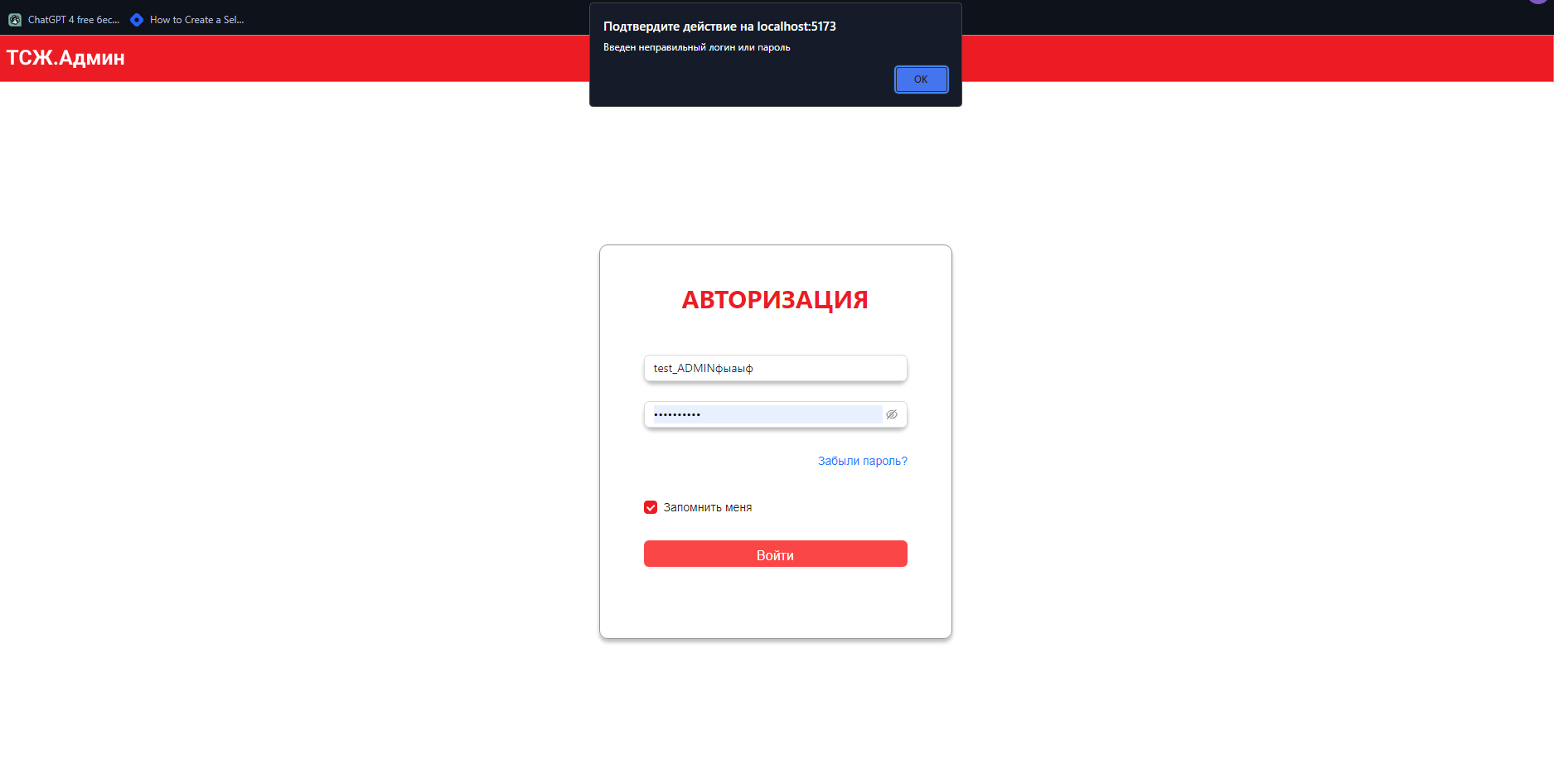


Рисунок 41. Реакция ИС при неудавшейся авторизации.

Тест генерации отчетов

При попытке получения отчета с пустыми полями, будет выведено сообщение о необходимости заполнения этих полей. На рисунке 5 показана реакция ИС на отсутствие введенных данных.

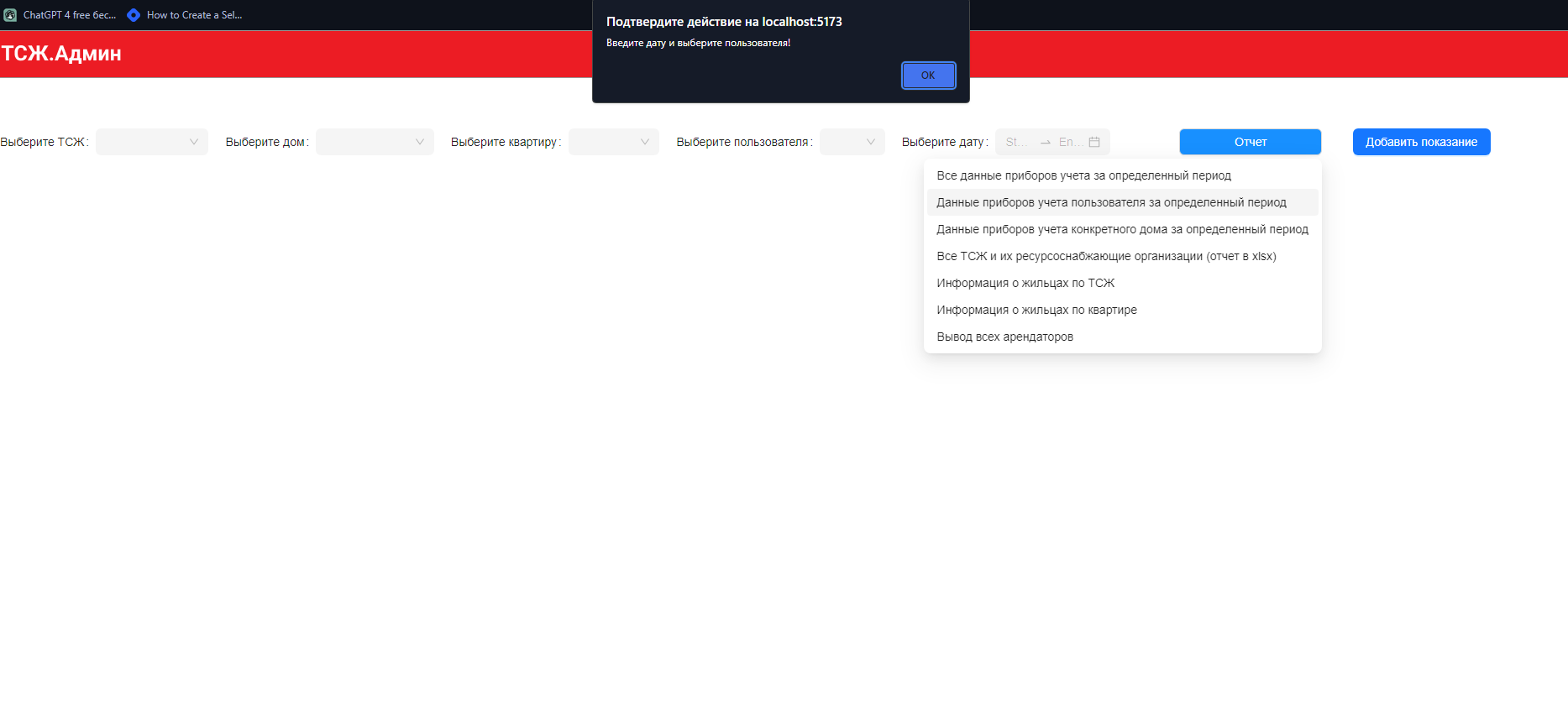


Рисунок 5. Реакция ИС на отсутствие введенных данных.

Для генерации отчётов выбираются следующие данные (пользователь, начальная дата, конечная дата), как показано на рисунках 6, 7.

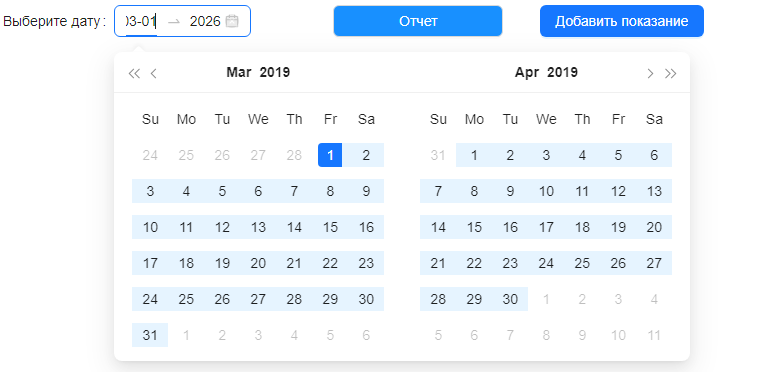


Рисунок 6. Выбор даты для генерации отчета.

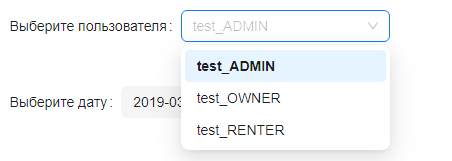


Рисунок 7. Выбор пользователя для генерации отчета.

При выборе вида отчета «Все ТСЖ и их ресурсоснабжающие организации (отчет в xlsx)» открывается окно выбора места сохранения отчета в формате xlsx. Это продемонстрировано на рисунке 8.

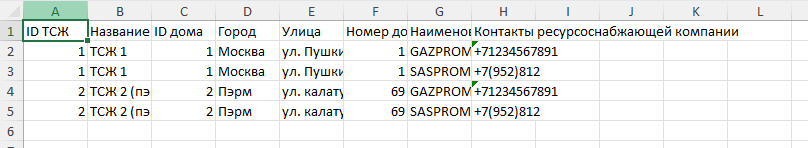


Рисунок 8. Отчет с выводом всех ТСЖ и их ресурсоснабжающих организаций в Excel.

Для получения отчета с данными приборов учета пользователя за определенный период необходимо выбрать пользователя и указать временной период, за который необходимо получить информацию.

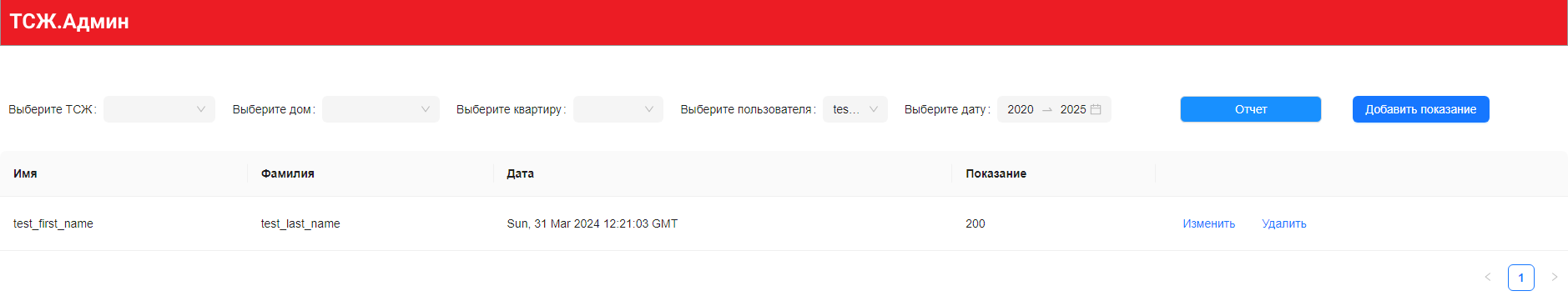


Рисунок 9. Отчет с выводом данных приборов учета пользователя за определенный период.

Для получения отчета с данными приборов учета выбранного дома за определенный период необходимо выбрать из списка необходимый дом и указать временной период, за который необходимо получить информацию.

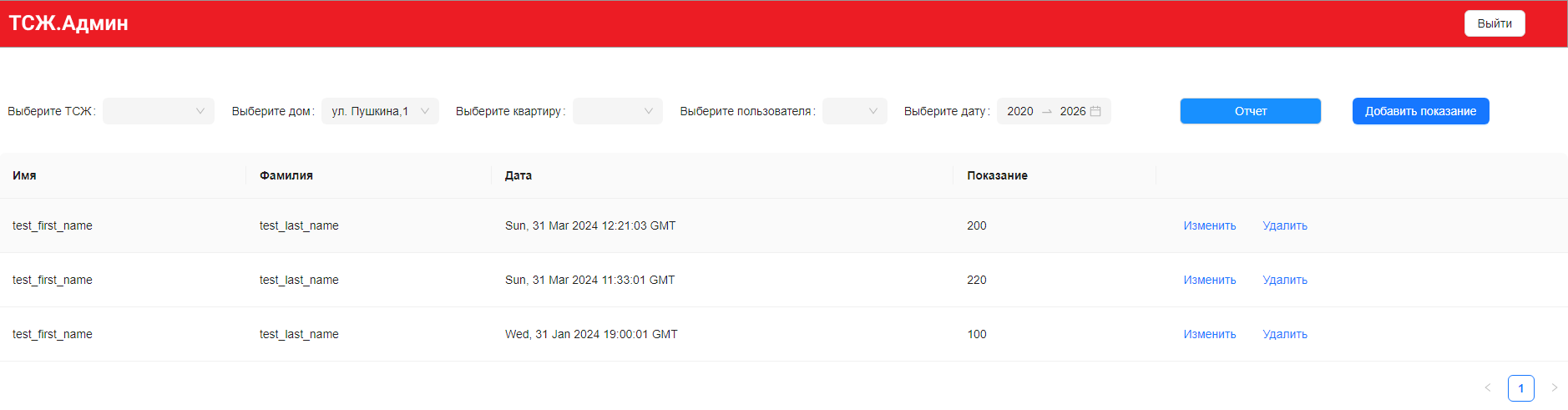


Рисунок 10. Отчет с выводом данных приборов учета выбранного дома за определенный период.

На рисунке 11 продемонстрирован вывод отчета с информацией о жильцах, прикрепленных с конкретному ТСЖ. Для получения отчета необходимо выбрать нужную ТСЖ из списка.

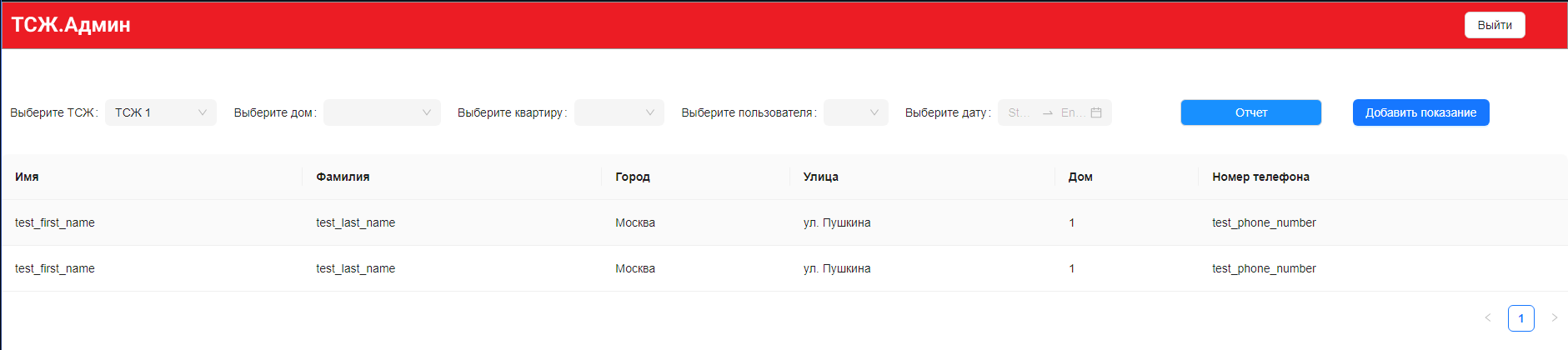


Рисунок 11. Вывод отчета с информацией о жильцах, прикрепленных к конкретному ТСЖ.

На рисунке 12 продемонстрирован вывод отчета с информацией о жильцах, живущих в определенной квартире. Для получения отчета необходимо выбрать нужный адрес дома и номер квартиры.

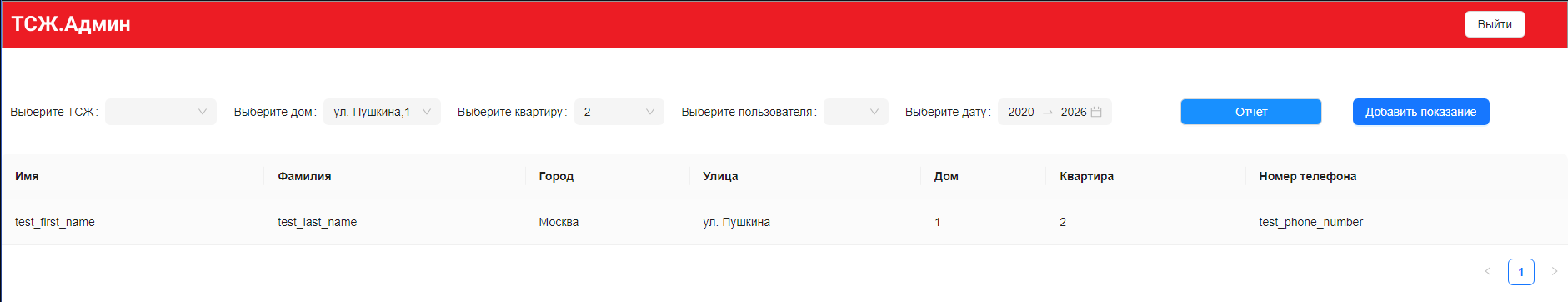


Рисунок 12. Вывод отчета с информацией о жильцах, проживающих в определенной квартире.

На рисунке 13 продемонстрирован отчет о предоставлении информации о всех арендаторах.

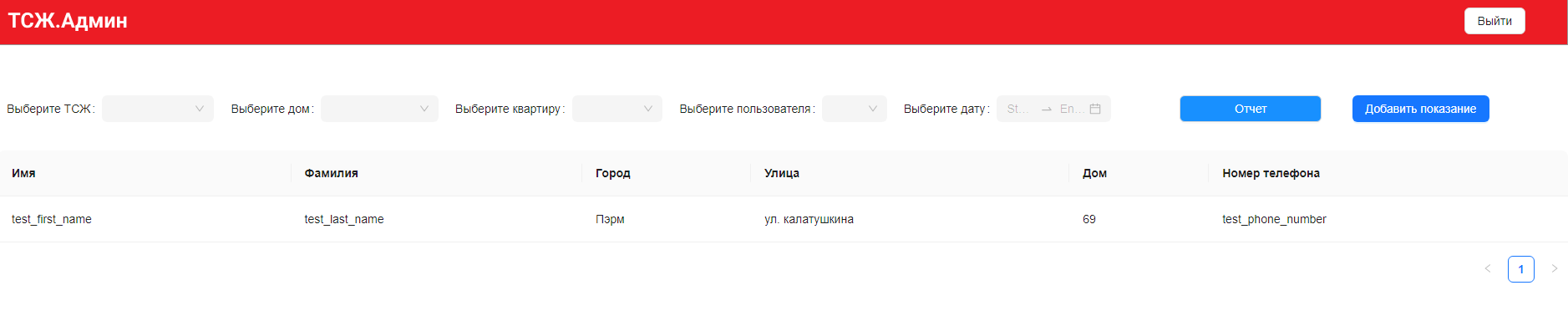


Рисунок 13. Вывод информации о всех арендаторах.

Тест добавления, удаления и изменения данных

На рисунке 42 показаны изначальные показания счетчиков.

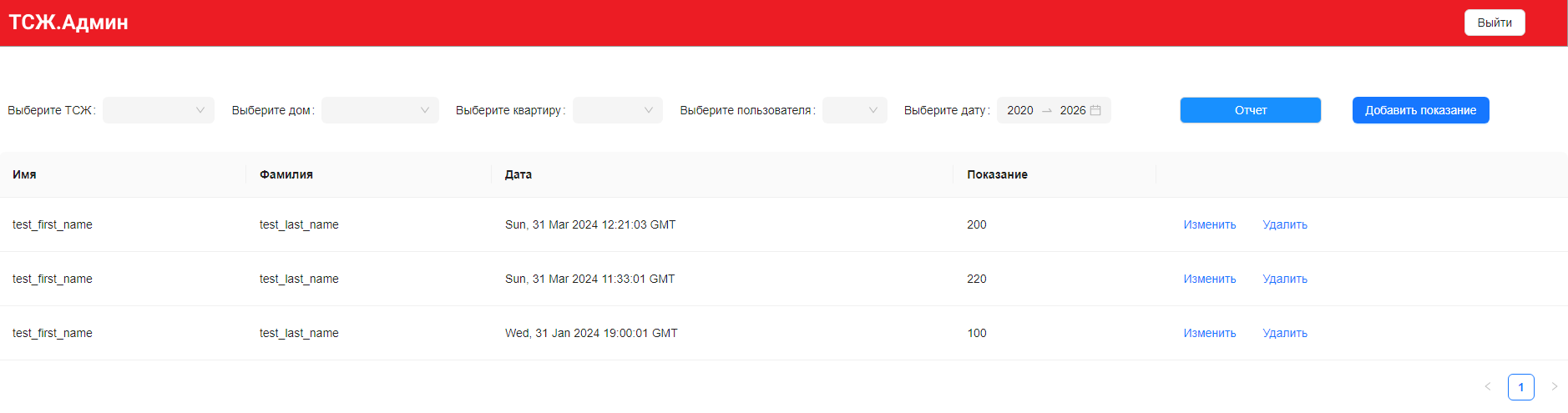


Рисунок 42. Вывод всех значений счетчиков

Для тестирования добавления новых показаний счетчика необходимо нажать кнопку «Добавить показание», после чего появится форма для заполнения данных о новом показании, как показано на рисунке 43.

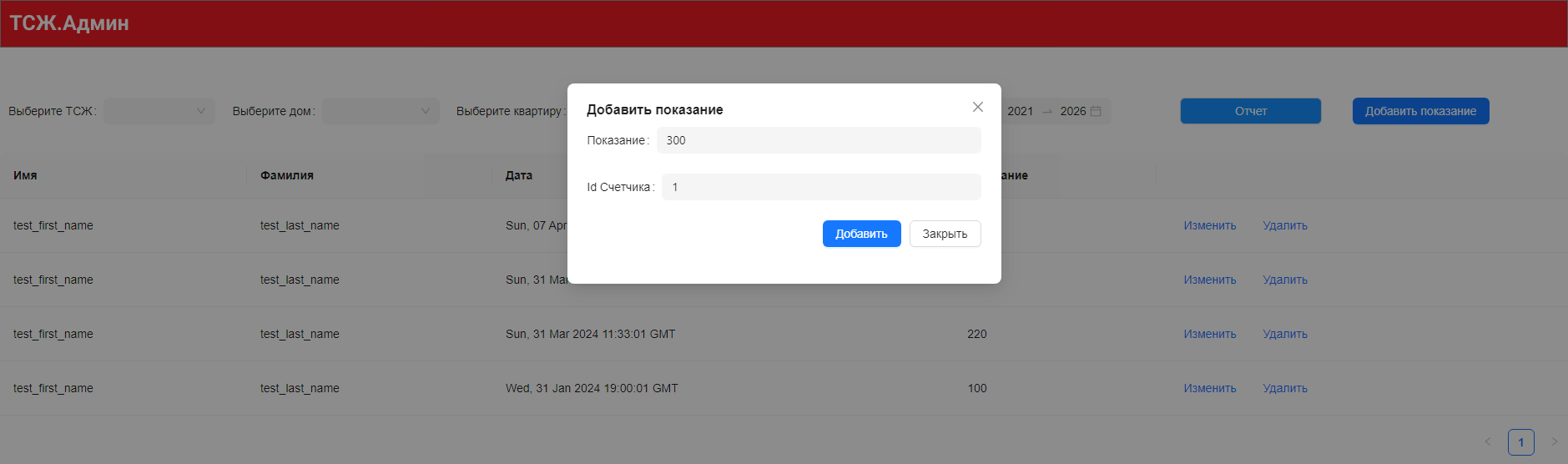


Рисунок 43. Окно добавления значений в показания счетчика.

После добавления показания, список со всеми значениями будет выглядеть так:

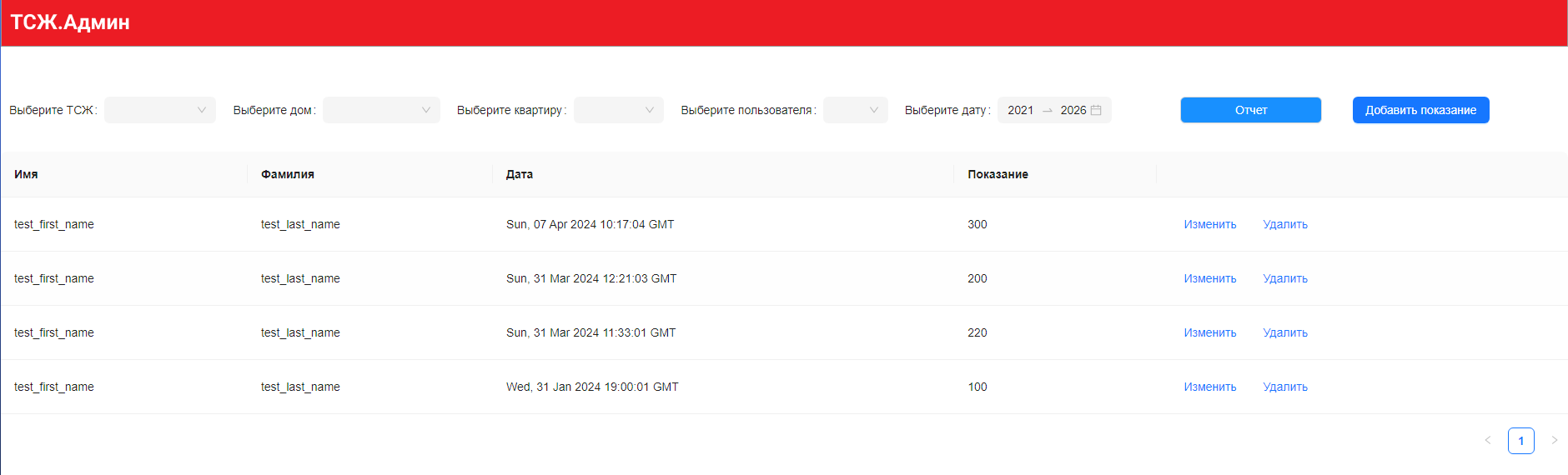


Рисунок 44. Вывод обновленных значений счетчиков

Для изменения конкретного показания, необходимо нажать кнопку «Изменить». После этого откроется окно для ввода данных:

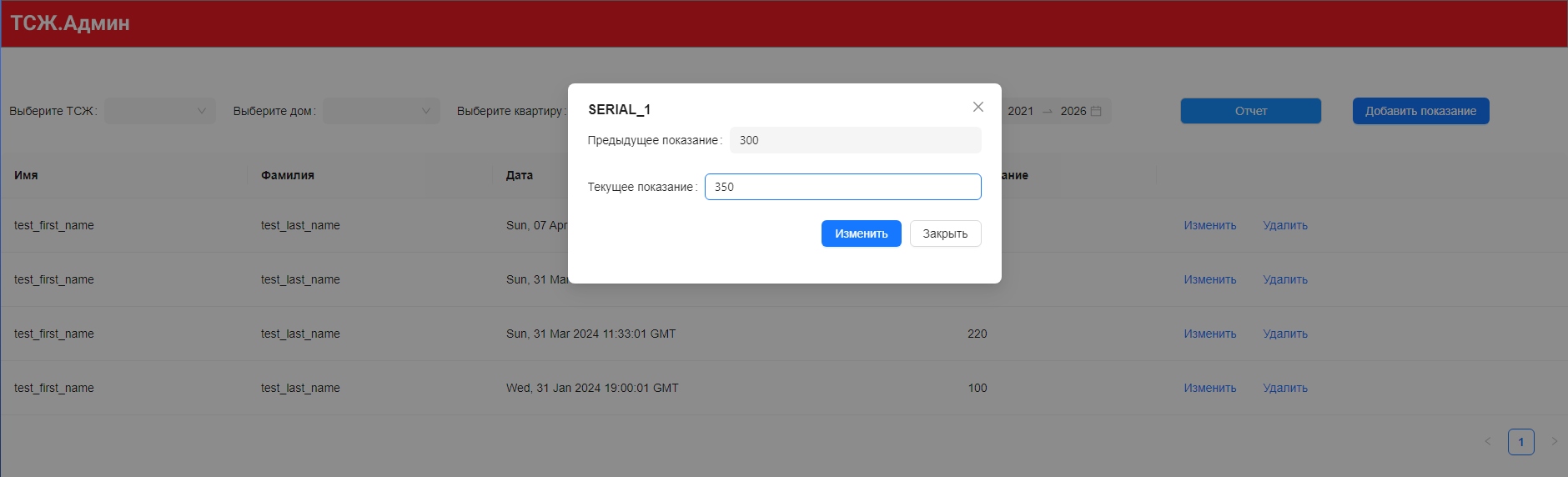


Рисунок 45. Окно для изменения показания счетчика.

После изменения показания, список со всеми значениями будет выглядеть так:

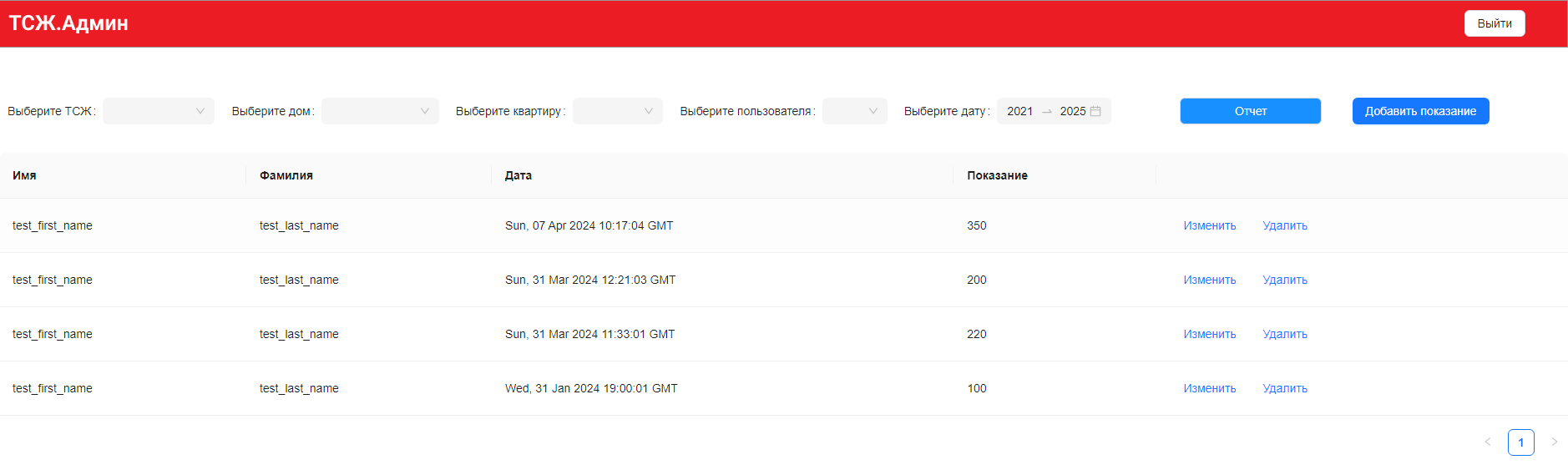


Рисунок 46. Вывод обновленных значений счетчиков

Для удаления показания необходимо нажать кнопку «Удалить» рядом с этим показанием. Список всех показаний, после удаления первого показания будет выглядеть так:

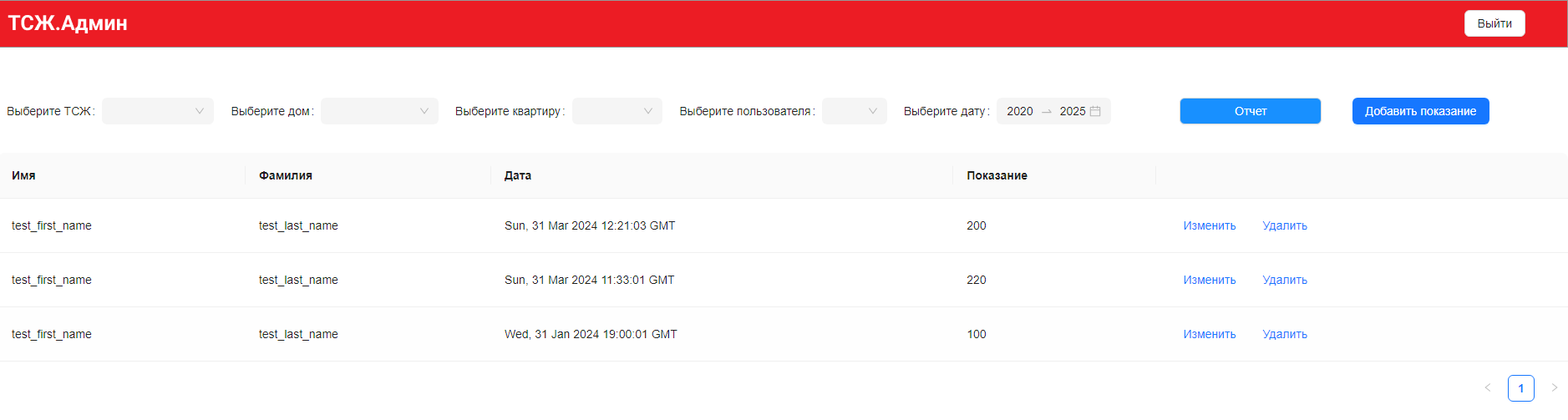


Рисунок 47. Вывод обновленных значений счетчиков.

Приложение

Код Java приложения

|  |
| --- |
| package ru.tsj;  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.beans.factory.annotation.Qualifier; import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Import; import org.springframework.context.annotation.Primary; import org.springframework.security.access.PermissionEvaluator; import org.springframework.security.access.expression.method.DefaultMethodSecurityExpressionHandler; import org.springframework.security.authentication.AuthenticationManager; import org.springframework.security.authentication.dao.DaoAuthenticationProvider; import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder; import org.springframework.security.config.annotation.web.builders.HttpSecurity; import org.springframework.security.core.userdetails.UserDetailsService; import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder; import org.springframework.security.crypto.password.PasswordEncoder; import org.springframework.security.web.access.expression.DefaultWebSecurityExpressionHandler; import ru.tsj.config.ApplicationConfig; import ru.tsj.data.repos.UserInFlatRepository; import ru.tsj.data.repos.UserRepository; import ru.tsj.web.security.SecurityUserDetailsManager;  @SpringBootApplication @Import(ApplicationConfig.class) public class Application {  public static void main(String[] args) {  SpringApplication.run(Application.class, args);  }   @Bean  public DefaultMethodSecurityExpressionHandler methodSecurityExpressionHandler(@Qualifier("defaultPermissionEvaluator") PermissionEvaluator permissionEvaluator) {  DefaultMethodSecurityExpressionHandler methodSecurityExpressionHandler = new DefaultMethodSecurityExpressionHandler();  methodSecurityExpressionHandler.setPermissionEvaluator(permissionEvaluator);  return methodSecurityExpressionHandler;  }   @Bean  public DefaultWebSecurityExpressionHandler webExpressionHandler(@Qualifier("defaultPermissionEvaluator") PermissionEvaluator permissionEvaluator) {  DefaultWebSecurityExpressionHandler webExpressionHandler = new DefaultWebSecurityExpressionHandler();  webExpressionHandler.setPermissionEvaluator(permissionEvaluator);  return webExpressionHandler;  } } |
| Содержимое файла Application.java |

|  |
| --- |
| package ru.tsj.business;  import ru.tsj.data.entity.Address;  import java.util.List;  public interface AddressService {  Address addAddress(Address address, Integer flatsCount);  boolean isExists(Address address);  boolean isExistsById(Long addressId);  List<Address> getAll();  Address getAddressById(Long id);  Address updateAddress(Address address); } |
| Содержимое файла AddressService.java |

|  |
| --- |
| package ru.tsj.business;   import ru.tsj.data.entity.Address; import ru.tsj.data.entity.Flat; import ru.tsj.data.entity.User; import ru.tsj.data.enumerate.Role;  import java.util.List;  public interface FlatService {   Flat addFlat(Flat flat);  List<Flat> addFlatsByRange(Integer startNumber, Integer endNumber, Address address);  boolean isExists(Flat flat);  boolean isExistsById(Long flatId);  List<Flat> getAll();  Flat updateFlat(Flat flat);  Flat getFlatById(Long id);  List<Flat> getFlatByAddress(Address address);  List<Flat> getFlatByAddressId(Long addressId); } |
| Содержимое файла FlatService.java |

|  |
| --- |
| package ru.tsj.business;  import ru.tsj.data.entity.\*;  import java.sql.Date;  public interface MainService {   Meter addMeter(String serialNumber, Date installationDate, Long userId, Long flatId, Long resourceId);   Provider addProvider(String name, String contact, Long userId);   ValueHistory addValue(String value, Long meterId);   Tsj addTsj(String name, Long managerId);   Address addAddress(Long tsjId, String city, String street, Long houseNumber, Long userId);   Flat addFlat(Long number, Long addressId, Long userId);   Resource addResource(String name, Double price, Long providerId, Long userId);  } |
| Содержимое файла MainService.java |

|  |
| --- |
| package ru.tsj.business;  import ru.tsj.data.entity.Meter; import ru.tsj.data.entity.ValueHistory;  import java.sql.Date; import java.sql.Time; import java.sql.Timestamp; import java.util.List;  public interface MeterService {   Meter add(String serialNumber, Timestamp installationDate, Long resourceId, Long flatId);   Meter add(Meter meter);   Meter getMeterById(Long meterId);   Meter updateMeter(Meter meter);   List<Meter> getAll();  } |
| Содержимое файла MeterService.java |

|  |
| --- |
| package ru.tsj.business;  import ru.tsj.data.entity.Provider;  import java.util.List;  public interface ProviderService {   Provider add(String name, String contact);   Provider add(Provider provider);   Provider getById(Long id);   List<Provider> getAll();   Provider updateProvider(Provider provider);  } |
| Содержимое файла ProviderService.java |

|  |
| --- |
| package ru.tsj.business;  import ru.tsj.data.entity.Resource;  import java.util.List;  public interface ResourceService {   Resource add(String name, Double price, Long providerId);   Resource getById(Long resourceId);   Resource add(Resource resource);   List<Resource> getAll();   Resource updateResource(Resource resource); } |
| Содержимое файла ResourceService.java |

|  |
| --- |
| package ru.tsj.business;  import ru.tsj.data.entity.Tsj; import ru.tsj.data.entity.User; import ru.tsj.data.entity.report.dto.TsjAndProviderDto;  import java.util.List;  public interface TsjService {   Tsj addTsj(Tsj tsj);  User getManager(Long tsjId);  Tsj setManager(Long tsjId, Long managerId);  Tsj addAddress(Long tsjId, Long addressId);  Tsj findOneById(Long tsjId);  List<Tsj> findAll();  Tsj updateTsj(Tsj tsj);  boolean isExists(Tsj tsj);  boolean isExistsById(Long tsjId);  List<TsjAndProviderDto> getTsjAndProviderDtos(); } |
| Содержимое файла TsjService.java |

|  |
| --- |
| package ru.tsj.business;   import ru.tsj.data.entity.Flat; import ru.tsj.data.entity.User; import ru.tsj.data.enumerate.Role;  public interface UserInFlatService {  boolean saveLink(Long userId, Long flatId);  boolean saveLink(Long userId, Long flatId, Role role);  boolean deleteLink(Long userId, Long flatId);  Role getRole(Long userId, Long flatId);  boolean setRole(Long userId, Long flatId, Role role);  boolean checkRole(Long userId, Long flatId, Role minRole); } |
| Содержимое файла UserInFlatService.java |

|  |
| --- |
| package ru.tsj.business;  import ru.tsj.data.entity.User; import ru.tsj.data.entity.report.dto.RentersAndFlatDto; import ru.tsj.data.entity.report.dto.UserAndAddressDto;  import javax.servlet.http.HttpServletResponse; import java.util.List;  public interface UserService {  User signup(User user, HttpServletResponse response);  List<User> findAll();  User findOne(Long userId);  String addRenter(String login, Long flatId);  List<User> findAllAdmins();  boolean isExists(User user);  boolean isExistsById(Long userId);  boolean isExistsByLogin(String login);  List<UserAndAddressDto> getAllUsersInfoByTsjId(Long tsjId);  List<UserAndAddressDto> getAllUsersInfoByFlatId(Long flatId, Long addressId);  List<RentersAndFlatDto> getAllRentersAndFlatDto();  User setIsAdmin(String login, boolean isAdmin); } |
| Содержимое файла UserService.java |

|  |
| --- |
| package ru.tsj.business;  import ru.tsj.data.entity.Meter; import ru.tsj.data.entity.ValueHistory; import ru.tsj.data.entity.report.dto.ValueHistoryReportDto;  import java.sql.Timestamp; import java.util.List;  public interface ValueHistoryService {   ValueHistory add(String value, Double price, Meter meter);   ValueHistory add(String value, Long meterId);   ValueHistory add(ValueHistory valueHistory);   void delete(Long id);   ValueHistory update(ValueHistory valueHistory);   ValueHistory getById(Long id);   List<ValueHistory> getAll();   List<ValueHistory> getAllByMeter(Meter meter);   List<ValueHistoryReportDto> getAllValueHistoryByMonth(Timestamp dateStart, Timestamp dateEnd);  List<ValueHistoryReportDto> getAllValueHistoryByMonthAndAddressId(Timestamp dateStart, Timestamp dateEnd, Long addressId);  List<ValueHistoryReportDto> getAllValueHistoryByMonthAndUserLogin(Timestamp dateStart, Timestamp dateEnd, String userLogin); } |
| Содержимое файла ValueHistoryService.java |

|  |
| --- |
| package ru.tsj.business.exceptions;  public class BaseException extends RuntimeException {   public BaseException(String message) {  super(message);  } } |
| Содержимое файла BaseException.java |

|  |
| --- |
| package ru.tsj.business.exceptions;  import org.springframework.util.Assert;  /\*\*  \* Исключение при повторном создании сущности с заданным ключом  \*/ public class EntityAlreadyExistsException extends BaseException{   public EntityAlreadyExistsException(String message) {  super(message);  }   public EntityAlreadyExistsException(String type, Object id) {  this(formatMessage(type, id));  }   private static String formatMessage(String type, Object id) {  Assert.hasText(type, "Тип не может быть пустым");  Assert.notNull(id, "Идентификатор объекта не может быть null");  Assert.hasText(id.toString(), "Идентификатор не может юыьб пустым");  return String.format("%s с ключом %s уже существует", type, id);  } } |
| Содержимое файла EntityAlreadyExistsException.java |

|  |
| --- |
| package ru.tsj.business.exceptions;  /\*\*  \* Выбрасывается при конфликтах с уже существующими данными  \*/ public class EntityConflictException extends BaseException {  public EntityConflictException(String message) {  super(message);  } } |
| Содержимое файла EntityConflictException.java |

|  |
| --- |
| package ru.tsj.business.exceptions;  import org.springframework.util.Assert;  /\*\*  \*  \*/ public class EntityHasDetailException extends BaseException {  public EntityHasDetailException(String message) {  super(message);  }   public EntityHasDetailException(String type, Object id) {  this(formatMessage(type, id));  }   private static String formatMessage(String type, Object id) {  Assert.hasText(type, "Тип не может быть пустым");  Assert.notNull(id, "Идентификатор объекта не может быть null");  Assert.hasText(id.toString(), "Идентификатор не может юыьб пустым");  return String.format("%s ссылается на удаляемый объект ключом с идентификатором %s", type, id);  } } |
| Содержимое файла EntityHasDetailException.java |

|  |
| --- |
| package ru.tsj.business.exceptions;  /\*\*  \* Исключение выбрасывается при вызове метода сервиса с некорректными параметрами  \*/  public class EntityIllegalArgumentException extends BaseException{  public EntityIllegalArgumentException(String message) {  super(message);  } } |
| Содержимое файла EntityIllegalArgumentException.java |

|  |
| --- |
| package ru.tsj.business.exceptions;  import org.springframework.util.Assert;  /\*\*  \* Сущность не найдена  \*/ public class EntityNotFoundException extends BaseException{  public EntityNotFoundException(String message) {  super(message);  }   public EntityNotFoundException(String type, Object id) {  this(formatMessage(type, id));  }   private static String formatMessage(String type, Object id) {  Assert.hasText(type, "Тип не может быть пустым");  Assert.notNull(id, "Идентификатор объекта не может быть null");  Assert.hasText(id.toString(), "Идентификатор не может быть пустым");  return String.format("%s с ключом %s не найден", type, id);  } } |
| Содержимое файла EntityNotFoundException.java |

|  |
| --- |
| package ru.tsj.business.exceptions;  import org.springframework.util.Assert;  /\*\*  \* Исключение при ошибке валидации сущности  \*/ public class EntityValidationException extends BaseException{   public EntityValidationException(String message) {  super(message);  }   public EntityValidationException(String entityType, String message) {  this(formatMessage(entityType, message));  }   private static String formatMessage(String entityType, String message) {  Assert.hasText(entityType, "Тип не может быть пустым");  Assert.notNull(message, "Сообщение исключения не может быть null");  Assert.hasText(message, "Сообщение исключения не может быть пустым");  return String.format("Ошибка валидации сущности %s: %s", entityType, message);  } } |
| Содержимое файла EntityValidationException.java |

|  |
| --- |
| package ru.tsj.business.exceptions;  import org.springframework.util.Assert;  /\*\*  \*  \*/ public class UserNotFoundException extends BaseException {  public UserNotFoundException(String message) {  super(message);  } } |
| Содержимое файла UserNotFoundException.java |

|  |
| --- |
| package ru.tsj.business.impl;  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.annotation.Lazy; import org.springframework.stereotype.Service; import ru.tsj.business.AddressService; import ru.tsj.business.FlatService; import ru.tsj.business.TsjService; import ru.tsj.business.exceptions.BaseException; import ru.tsj.business.exceptions.EntityAlreadyExistsException; import ru.tsj.business.exceptions.EntityIllegalArgumentException; import ru.tsj.business.exceptions.EntityNotFoundException; import ru.tsj.data.entity.Address; import ru.tsj.data.repos.AddressRepository; import ru.tsj.utils.EntityValidator;  import java.util.List;  @Service public class AddressServiceImpl implements AddressService {  private final AddressRepository addressRepository;  private final FlatService flatService;   @Lazy  @Autowired  private TsjService tsjService;   public AddressServiceImpl(AddressRepository addressRepository, FlatService flatService) {  this.addressRepository = addressRepository;  this.flatService = flatService;  }   @Override  public Address addAddress(Address address, Integer flatsCount) {  EntityValidator.validateAddress(address);   if (isExists(address))  throw new EntityAlreadyExistsException("Дом с таким адресом уже существует!");   address.setTsj(tsjService.findOneById(address.getTsj().getId())); // todo костыль   Address savedAddress = addressRepository.save(address);  if (savedAddress == null)  throw new BaseException("Ошибка при сохранении адреса!");   flatService.addFlatsByRange(1, flatsCount, savedAddress);   return savedAddress;  }   @Override  public boolean isExists(Address address) {  if (address.getId() == null)  return false;  EntityValidator.validateAddress(address);  return addressRepository.exists(address.getId());  }   @Override  public boolean isExistsById(Long addressId) {  return addressRepository.findOne(addressId) != null;  }   @Override  public List<Address> getAll() {  return addressRepository.findAll();  }   @Override  public Address getAddressById(Long id) {  if (id == null)  throw new EntityIllegalArgumentException("id не может быть null");   Address foundAddress = addressRepository.findOne(id);  if (foundAddress == null)  throw new EntityNotFoundException(String.format("Сущность %s с id = %d не найдена", Address.ENTITY\_TYPE, id));   return foundAddress;  }   @Override  public Address updateAddress(Address address) {  EntityValidator.validateAddress(address);  return addressRepository.save(address);  } } |
| Содержимое файла AddressServiceImpl.java |

|  |
| --- |
| package ru.tsj.business.impl;  import org.springframework.stereotype.Service; import ru.tsj.business.FlatService; import ru.tsj.business.UserInFlatService; import ru.tsj.business.UserService; import ru.tsj.business.exceptions.EntityAlreadyExistsException; import ru.tsj.business.exceptions.EntityIllegalArgumentException; import ru.tsj.business.exceptions.EntityNotFoundException; import ru.tsj.data.entity.Address; import ru.tsj.data.entity.Flat; import ru.tsj.data.entity.User; import ru.tsj.data.entity.ValueHistory; import ru.tsj.data.enumerate.Role; import ru.tsj.data.repos.FlatRepository; import ru.tsj.utils.EntityValidator;  import java.util.ArrayList; import java.util.List;  @Service public class FlatServiceImpl implements FlatService {  private final FlatRepository flatRepository;  private final UserService userService;  private final UserInFlatService userInFlatService;   public FlatServiceImpl(FlatRepository flatRepository, UserService userService, UserInFlatService userInFlatService) {  this.flatRepository = flatRepository;  this.userService = userService;  this.userInFlatService = userInFlatService;  }   @Override  public Flat addFlat(Flat flat) {  EntityValidator.validateFlat(flat);  if (isExists(flat))  throw new EntityAlreadyExistsException("Такая квартира уже существует!");   return flatRepository.save(flat);  }   @Override  public List<Flat> addFlatsByRange(Integer startNumber, Integer endNumber, Address address) {  EntityValidator.validateAddress(address);   List<Flat> savedFlats = new ArrayList<>();  for (int i = startNumber; i <= endNumber; i++) {  Flat flat = new Flat();  flat.setNumber((long) i);  flat.setAddress(address);  savedFlats.add(addFlat(flat));  }   return savedFlats;  }   @Override  public boolean isExists(Flat flat) {  if (flat.getId() == null)  return false;  EntityValidator.validateFlat(flat);  return flatRepository.exists(flat.getId());  }   @Override  public boolean isExistsById(Long flatId) {  return flatRepository.findOne(flatId) != null;  }   @Override  public List<Flat> getAll() {  return flatRepository.findAll();  }   @Override  public Flat updateFlat(Flat flat) {  EntityValidator.validateFlat(flat);  return flatRepository.save(flat);  }   @Override  public Flat getFlatById(Long id) {  if (id == null)  throw new EntityIllegalArgumentException("id не может быть null");   Flat foundFlat = flatRepository.findOne(id);  if (foundFlat == null)  throw new EntityNotFoundException(String.format("Сущность %s с id = %d не найдена", Flat.ENTITY\_TYPE, id));   return foundFlat;  }   @Override  public List<Flat> getFlatByAddress(Address address) {  if (address == null)  throw new EntityIllegalArgumentException("Адрес не может быть null");  if (address.getId() == null)  throw new EntityIllegalArgumentException("id адреса не может быть null");  return flatRepository.getFlatsByAddress(address);  }   @Override  public List<Flat> getFlatByAddressId(Long addressId) {  if (addressId == null)  throw new EntityIllegalArgumentException("id не может быть null");   return flatRepository.getFlatsByAddressId(addressId);  }  } |
| Содержимое файла FlatServiceImpl.java |

|  |
| --- |
| package ru.tsj.business.impl;  import org.springframework.stereotype.Service; import ru.tsj.business.MeterService; import ru.tsj.business.exceptions.EntityAlreadyExistsException; import ru.tsj.business.exceptions.EntityIllegalArgumentException; import ru.tsj.data.entity.Flat; import ru.tsj.data.entity.Meter; import ru.tsj.data.entity.Resource; import ru.tsj.data.entity.ValueHistory; import ru.tsj.data.repos.FlatRepository; import ru.tsj.data.repos.MeterRepository; import ru.tsj.data.repos.ResourceRepository; import ru.tsj.utils.EntityValidator;  import java.sql.Date; import java.sql.Timestamp; import java.util.List;  @Service public class MeterServiceImpl implements MeterService {   private final MeterRepository meterRepository;  private final FlatRepository flatRepository;  private final ResourceRepository resourceRepository;    public MeterServiceImpl(MeterRepository meterRepository, FlatRepository flatRepository, ResourceRepository resourceRepository) {  this.meterRepository = meterRepository;  this.flatRepository = flatRepository;  this.resourceRepository = resourceRepository;  }    @Override  public Meter add(String serialNumber, Timestamp installationDate, Long resourceId, Long flatId) {  Meter meter = new Meter();  meter.setSerialNumber(serialNumber);  meter.setInstallationDate(installationDate);   Resource resource = resourceRepository.findOne(resourceId);  if (resource == null)  throw new EntityIllegalArgumentException(String.format("%s с id = %d не найден", Resource.ENTITY\_TYPE, resourceId));  meter.setResource(resource);   Flat flat = flatRepository.findOne(flatId);  if (flat == null)  throw new EntityIllegalArgumentException(String.format("%s с id = %d не найден", Flat.ENTITY\_TYPE, flatId));  meter.setFlat(flat);   return meterRepository.save(meter);  }   @Override  public Meter add(Meter meter) {  EntityValidator.validateMeter(meter);   if (isExists(meter))  throw new EntityAlreadyExistsException(String.format("Сущность %s с id = %d уже существует",  Meter.ENTITY\_TYPE, meter.getId()));   return meterRepository.save(meter);  }   @Override  public Meter getMeterById(Long meterId) {  return meterRepository.findOne(meterId);  }   @Override  public Meter updateMeter(Meter meter) {  EntityValidator.validateMeter(meter);  return meterRepository.save(meter);  }   @Override  public List<Meter> getAll() {  return meterRepository.findAll();  }   private boolean isExists(Meter meter) {  if (meter.getId() == null)  return false;  EntityValidator.validateMeter(meter);  return meterRepository.exists(meter.getId());  } } |
| Содержимое файла MeterServiceImpl.java |

|  |
| --- |
| package ru.tsj.business.impl;   import org.springframework.stereotype.Service; import ru.tsj.business.ProviderService; import ru.tsj.business.exceptions.EntityAlreadyExistsException; import ru.tsj.business.exceptions.EntityIllegalArgumentException; import ru.tsj.business.exceptions.EntityNotFoundException; import ru.tsj.data.entity.Provider; import ru.tsj.data.entity.Tsj; import ru.tsj.data.repos.ProviderRepository; import ru.tsj.utils.EntityValidator;  import java.util.List;  @Service public class ProviderServiceImpl implements ProviderService {   private final ProviderRepository providerRepository;   public ProviderServiceImpl(ProviderRepository providerRepository) {  this.providerRepository = providerRepository;  }    @Override  public Provider add(String name, String contact) {  Provider provider = new Provider();  provider.setContact(contact);  provider.setName(name);   return providerRepository.save(provider);  }   @Override  public Provider add(Provider provider) {  EntityValidator.validateProvider(provider);  if (isExists(provider))  throw new EntityAlreadyExistsException(String.format("Сущность %s с id = %d уже существует",  Provider.ENTITY\_TYPE, provider.getId()));  return providerRepository.save(provider);  }   @Override  public Provider getById(Long id) {  if (id == null)  throw new EntityIllegalArgumentException("id не может быть null");   Provider foundProvider = providerRepository.findOne(id);  if (foundProvider == null)  throw new EntityNotFoundException(String.format("Сущность %s с id = %d не найдена", Provider.ENTITY\_TYPE, id));   return foundProvider;  }   @Override  public List<Provider> getAll() {  return providerRepository.findAll();  }   @Override  public Provider updateProvider(Provider provider) {  EntityValidator.validateProvider(provider);  return providerRepository.save(provider);  }   private boolean isExists(Provider provider) {  if (provider.getId() == null)  return false;  EntityValidator.validateProvider(provider);  return providerRepository.exists(provider.getId());  } } |
| Содержимое файла ProviderServiceImpl.java |

|  |
| --- |
| package ru.tsj.business.impl;  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Service; import ru.tsj.business.ResourceService; import ru.tsj.business.exceptions.EntityAlreadyExistsException; import ru.tsj.business.exceptions.EntityIllegalArgumentException; import ru.tsj.business.exceptions.EntityNotFoundException; import ru.tsj.data.entity.Provider; import ru.tsj.data.entity.Resource; import ru.tsj.data.entity.Tsj; import ru.tsj.data.repos.ProviderRepository; import ru.tsj.data.repos.ResourceRepository; import ru.tsj.utils.EntityValidator;  import java.util.List;  @Service public class ResourceServiceImpl implements ResourceService {   private final ResourceRepository resourceRepository;  private final ProviderRepository providerRepository;   public ResourceServiceImpl(ProviderRepository providerRepository, ResourceRepository resourceRepository) {  this.providerRepository = providerRepository;  this.resourceRepository = resourceRepository;  }    @Override  public Resource add(String name, Double price, Long providerId) {  Provider provider = providerRepository.findOne(providerId);  if (provider == null)  throw new EntityIllegalArgumentException(String.format("%s с id = %d не найден", Provider.ENTITY\_TYPE, providerId));   Resource resource = new Resource();  resource.setProvider(provider);  resource.setName(name);  resource.setPrice(price);   return resourceRepository.save(resource);  }   @Override  public Resource getById(Long resourceId) {  if (resourceId == null)  throw new EntityIllegalArgumentException("id не может быть null");   Resource foundResource = resourceRepository.findOne(resourceId);  if (foundResource == null)  throw new EntityNotFoundException(String.format("Сущность %s с id = %d не найдена", Resource.ENTITY\_TYPE, resourceId));  return resourceRepository.findOne(resourceId);  }   @Override  public Resource add(Resource resource) {  EntityValidator.validateResource(resource);  if (isExists(resource))  throw new EntityAlreadyExistsException("Такой ресурс уже существует!");   return resourceRepository.save(resource);  }   @Override  public List<Resource> getAll() {  return resourceRepository.findAll();  }   @Override  public Resource updateResource(Resource resource) {  EntityValidator.validateResource(resource);  return resourceRepository.save(resource);  }   private boolean isExists(Resource resource) {  if (resource.getId() == null)  return false;  EntityValidator.validateResource(resource);  return resourceRepository.exists(resource.getId());  } } |
| Содержимое файла ResourceServiceImpl.java |

|  |
| --- |
| package ru.tsj.business.impl;  import org.springframework.stereotype.Service; import ru.tsj.business.AddressService; import ru.tsj.business.TsjService; import ru.tsj.business.UserService; import ru.tsj.business.exceptions.EntityAlreadyExistsException; import ru.tsj.business.exceptions.EntityIllegalArgumentException; import ru.tsj.business.exceptions.EntityNotFoundException; import ru.tsj.data.entity.Address; import ru.tsj.data.entity.Flat; import ru.tsj.data.entity.Tsj; import ru.tsj.data.entity.User; import ru.tsj.data.entity.report.DtoMapper; import ru.tsj.data.entity.report.dto.TsjAndProviderDto; import ru.tsj.data.repos.AddressRepository; import ru.tsj.data.repos.TsjRepository; import ru.tsj.utils.EntityValidator;  import java.util.List;  @Service public class TsjServiceImpl implements TsjService {   private final TsjRepository tsjRepository;  private final UserService userService;  private final AddressService addressService;  private final AddressRepository addressRepository;   public TsjServiceImpl(TsjRepository tsjRepository, UserService userService, AddressService addressService, AddressRepository addressRepository) {  this.tsjRepository = tsjRepository;  this.userService = userService;  this.addressService = addressService;  this.addressRepository = addressRepository;  }   public Tsj addTsj(Tsj tsj) {  EntityValidator.validateTsj(tsj);  if (isExists(tsj))  throw new EntityAlreadyExistsException("Такое ТСЖ уже существует!");   return tsjRepository.save(tsj);  }   @Override  public User getManager(Long tsjId) {  if (!isExistsById(tsjId))  throw new EntityIllegalArgumentException("Такого ТСЖ не существует!");   return tsjRepository.findOne(tsjId).getManager();  }   @Override  public Tsj setManager(Long tsjId, Long managerId) {  if (!isExistsById(tsjId))  throw new EntityIllegalArgumentException("Такого ТСЖ не существует!");  if (!userService.isExistsById(managerId))  throw new EntityIllegalArgumentException("Такого пользователя не существует!");   Tsj foundedTsj = tsjRepository.findOne(tsjId);  User foundedUser = userService.findOne(managerId);   foundedTsj.setManager(foundedUser);  return tsjRepository.save(foundedTsj);  }   @Override  public Tsj addAddress(Long tsjId, Long addressId) {  if (!isExistsById(tsjId))  throw new EntityIllegalArgumentException("Такого ТСЖ не существует!");  if (!addressService.isExistsById(addressId))  throw new EntityIllegalArgumentException("Такого адреса не существует!");   Tsj foundedTsj = tsjRepository.findOne(tsjId);  Address foundedAddress = addressRepository.findOne(addressId);   foundedAddress.setTsj(foundedTsj);   addressRepository.save(foundedAddress);   return tsjRepository.findOne(tsjId);  }   @Override  public Tsj findOneById(Long tsjId) {  if (tsjId == null)  throw new EntityIllegalArgumentException("id не может быть null");   Tsj foundTsj = tsjRepository.findOne(tsjId);  if (foundTsj == null)  throw new EntityNotFoundException(String.format("Сущность %s с id = %d не найдена", Tsj.ENTITY\_TYPE, tsjId));   return foundTsj;  }   @Override  public List<Tsj> findAll() {  return tsjRepository.findAll();  }   @Override  public Tsj updateTsj(Tsj tsj) {  EntityValidator.validateTsj(tsj);  return tsjRepository.save(tsj);  }   @Override  public boolean isExists(Tsj tsj) {  if (tsj.getId() == null)  return false;  EntityValidator.validateTsj(tsj);  return tsjRepository.exists(tsj.getId());  }   @Override  public boolean isExistsById(Long tsjId) {  return tsjRepository.findOne(tsjId) != null;  }   @Override  public List<TsjAndProviderDto> getTsjAndProviderDtos() {  return DtoMapper.mapArrayToList(tsjRepository.getReport(), DtoMapper::mapToTsjAndProviderDto);  } } |
| Содержимое файла TsjServiceImpl.java |

|  |
| --- |
| package ru.tsj.business.impl;  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Service; import ru.tsj.business.UserInFlatService; import ru.tsj.business.UserService; import ru.tsj.data.entity.Flat; import ru.tsj.data.entity.User; import ru.tsj.data.entity.UserInFlat; import ru.tsj.data.entity.UserInFlatId; import ru.tsj.data.enumerate.Role; import ru.tsj.data.repos.UserInFlatRepository; import ru.tsj.data.repos.UserRepository; import ru.tsj.utils.EntityValidator; import ru.tsj.utils.PasswordUtils;  @Service public class UserInFlatServiceImpl implements UserInFlatService {   private final UserInFlatRepository userInFlatRepository;   public UserInFlatServiceImpl(UserInFlatRepository userInFlatRepository) {  this.userInFlatRepository = userInFlatRepository;  }   @Override  public boolean saveLink(Long userId, Long flatId) {  return saveLink(userId, flatId, Role.BLANK);  }   @Override  public boolean saveLink(Long userId, Long flatId, Role role) {  if (!deleteLink(userId, flatId))  return false;   UserInFlat userInFlat = userInFlatRepository.save(new UserInFlat(userId, flatId, role));  return userInFlat != null;  }   @Override  public boolean deleteLink(Long userId, Long flatId) { // todo isExists  UserInFlat userInFlat = userInFlatRepository.findOne(new UserInFlatId(userId, flatId));  if (userInFlat == null)  return false;   userInFlatRepository.delete(new UserInFlat(userId, flatId, Role.BLANK));   userInFlat = userInFlatRepository.findOne(new UserInFlatId(userId, flatId));  return userInFlat == null;  }   @Override  public Role getRole(Long userId, Long flatId) {  UserInFlat userInFlat = userInFlatRepository.findOne(new UserInFlatId(userId, flatId));  if (userInFlat == null)  return null;   return userInFlat.getRole();  }   @Override  public boolean setRole(Long userId, Long flatId, Role role) {  UserInFlat userInFlat = userInFlatRepository.findOne(new UserInFlatId(userId, flatId));  if (userInFlat == null)  return false;   userInFlat.setRole(role);   return userInFlatRepository.save(userInFlat) != null;  }   @Override  public boolean checkRole(Long userId, Long flatId, Role minRole) {  Role role = getRole(userId, flatId);  if (role == null)  return false;  return role.priority >= minRole.priority;  } } |
| Содержимое файла UserInFlatServiceImpl.java |

|  |
| --- |
| package ru.tsj.business.impl;  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.annotation.Lazy; import org.springframework.stereotype.Service; import ru.tsj.business.FlatService; import ru.tsj.business.UserInFlatService; import ru.tsj.business.UserService; import ru.tsj.business.exceptions.EntityAlreadyExistsException; import ru.tsj.data.entity.User; import ru.tsj.data.entity.report.DtoMapper; import ru.tsj.data.entity.report.dto.RentersAndFlatDto; import ru.tsj.data.entity.report.dto.UserAndAddressDto; import ru.tsj.data.enumerate.Role; import ru.tsj.data.repos.UserRepository; import ru.tsj.utils.EntityValidator; import ru.tsj.utils.PasswordUtils; import ru.tsj.web.security.TokenAuthenticationService;  import javax.servlet.http.HttpServletResponse; import java.util.List;  @Service public class UserServiceImpl implements UserService {   private final UserRepository userRepository;   @Lazy  @Autowired  private FlatService flatService;  private final UserInFlatService userInFlatService;   public UserServiceImpl(UserRepository userRepository, UserInFlatService userInFlatService) {  this.userRepository = userRepository;  this.userInFlatService = userInFlatService;  }   @Override  public User signup(User user, HttpServletResponse response) {  EntityValidator.validateUser(user);  if (isExists(user))  throw new EntityAlreadyExistsException("Такой пользователь уже существует!");   user.setPassword(PasswordUtils.hashPassword(user.getPassword()));   userRepository.save(user);   TokenAuthenticationService.addAuthentication(response, user.getLogin());   return user;  }   @Override  public List<User> findAll() {  return userRepository.findAll();  }   @Override  public User findOne(Long userId) {  return userRepository.findOne(userId);  }   @Override  public String addRenter(String login, Long flatId) {  if (isExistsByLogin(login))  throw new EntityAlreadyExistsException("Такого пользователя не существует!");  if (flatService.isExistsById(flatId))  throw new EntityAlreadyExistsException("Такой квартиры не существует!");   User renter = userRepository.findOneByLogin(login);   userInFlatService.saveLink(renter.getId(), flatId, Role.RENTER);   return login;  }   @Override  public List<User> findAllAdmins() {  return userRepository.findAllAdmins();  }   @Override  public boolean isExists(User user) { // if (user.getId() == null) // return false; // boolean existsByLogin = userRepository.findByLogin(user.getLogin()).isEmpty(); // boolean existsById = userRepository.exists(user.getId()); // return !existsByLogin && !existsById;  return !userRepository.findByLogin(user.getLogin()).isEmpty();  }   @Override  public boolean isExistsById(Long userId) {  return userRepository.findOne(userId) != null;  }   @Override  public boolean isExistsByLogin(String login) {  return userRepository.findOneByLogin(login) != null;  }   @Override  public List<UserAndAddressDto> getAllUsersInfoByTsjId(Long tsjId) {  return DtoMapper.mapArrayToList(userRepository.getAllUsersInfoByTsjId(tsjId), DtoMapper::mapToUserAndAddressDto);  }   @Override  public List<UserAndAddressDto> getAllUsersInfoByFlatId(Long flatId, Long addressId) {  return DtoMapper.mapArrayToList(userRepository.getAllUsersInfoByFlat(flatId, addressId), DtoMapper::mapToUserAndAddressDto);  }   @Override  public List<RentersAndFlatDto> getAllRentersAndFlatDto() {  return DtoMapper.mapArrayToList(userRepository.getAllRentersInfo(), DtoMapper::mapToRentersAndFlatDto);  }   @Override  public User setIsAdmin(String login, boolean isAdmin) {  User user = userRepository.findOneByLogin(login);  user.setAdmin(isAdmin);  return userRepository.save(user);  } } |
| Содержимое файла UserServiceImpl.java |

|  |
| --- |
| package ru.tsj.business.impl;  import org.springframework.stereotype.Service; import ru.tsj.business.MeterService; import ru.tsj.business.ValueHistoryService; import ru.tsj.business.exceptions.EntityAlreadyExistsException; import ru.tsj.business.exceptions.EntityIllegalArgumentException; import ru.tsj.business.exceptions.EntityNotFoundException; import ru.tsj.data.entity.Meter; import ru.tsj.data.entity.ValueHistory; import ru.tsj.data.entity.report.DtoMapper; import ru.tsj.data.entity.report.dto.ValueHistoryReportDto; import ru.tsj.data.repos.ValueHistoryRepository; import ru.tsj.utils.EntityValidator;  import java.sql.Timestamp; import java.util.List;  @Service public class ValueHistoryServiceImpl implements ValueHistoryService {   private final ValueHistoryRepository valueHistoryRepository;  private MeterService meterService;   public ValueHistoryServiceImpl(ValueHistoryRepository valueHistoryRepository, MeterService meterService) {  this.valueHistoryRepository = valueHistoryRepository;  this.meterService = meterService;  }    @Override  public ValueHistory add(String value, Double price, Meter meter) {  if (value == null)  throw new EntityIllegalArgumentException("Значение value не может быть null");   if (price == null)  throw new EntityIllegalArgumentException("Значение price не может быть null");   if (meter == null)  throw new EntityIllegalArgumentException("Значение meter не может быть null");   double cost;  ValueHistory lastValue = getLastValue(meter);  if (lastValue == null || lastValue.getValue() == null)  cost = parseString(value) \* price;  else {  cost = (parseString(value) - parseString(lastValue.getValue())) \* price;  if (cost < 0)  throw new EntityIllegalArgumentException("Стоимость не может быть меньше 0");  }   ValueHistory valueHistory = new ValueHistory();  valueHistory.setValue(value);  valueHistory.setMeter(meter);  valueHistory.setCost(cost);   return valueHistoryRepository.save(valueHistory);  }   @Override  public ValueHistory add(String value, Long meterId) {  return null;  }   private double calculateCost(ValueHistory valueHistory) {  double cost;  String value = valueHistory.getValue();   Long meterId = valueHistory.getMeter().getId();  valueHistory.setMeter(getMeter(meterId));//todo костыли валидация   double price = valueHistory.getMeter().getResource().getPrice();  ValueHistory lastValue = getLastValue(valueHistory.getMeter());   if (lastValue == null || lastValue.getValue() == null)  cost = parseString(value) \* price;  else {  cost = (parseString(value) - parseString(lastValue.getValue())) \* price;  if (cost < 0)  throw new EntityIllegalArgumentException("Стоимость не может быть меньше 0");  }   return cost;  }   private Meter getMeter(Long id) {  return meterService.getMeterById(id);  }   private boolean isExists(ValueHistory valueHistory) {  if (valueHistory.getId() == null)  return false;  EntityValidator.validateValueHistory(valueHistory);  return valueHistoryRepository.exists(valueHistory.getId());  }   @Override  public ValueHistory add(ValueHistory valueHistory) {  EntityValidator.validateValueHistory(valueHistory);   if (isExists(valueHistory))  throw new EntityAlreadyExistsException(String.format("Сущность %s с id = %d уже существует",  ValueHistory.ENTITY\_TYPE, valueHistory.getId()));   valueHistory.setCost(calculateCost(valueHistory));  return valueHistoryRepository.save(valueHistory);  }   @Override  public void delete(Long id) {  if (id == null)  throw new EntityIllegalArgumentException("id не может быть null");   if (!valueHistoryRepository.exists(id))  throw new EntityNotFoundException(ValueHistory.ENTITY\_TYPE, id);   valueHistoryRepository.delete(id);  }   @Override  public ValueHistory update(ValueHistory valueHistory) {  EntityValidator.validateValueHistory(valueHistory);  valueHistory.setCost(calculateCost(valueHistory));  return valueHistoryRepository.save(valueHistory);  }   @Override  public ValueHistory getById(Long id) {  if (id == null)  throw new EntityIllegalArgumentException("id не может быть null");   ValueHistory foundValueHistory = valueHistoryRepository.findOne(id);  if (foundValueHistory == null)  throw new EntityNotFoundException(String.format("Сущность %s с id = %d не найдена", ValueHistory.ENTITY\_TYPE, id));   return foundValueHistory;  }   @Override  public List<ValueHistory> getAll() {  return valueHistoryRepository.findAll();  }   @Override  public List<ValueHistory> getAllByMeter(Meter meter) {  EntityValidator.validateMeter(meter);  return valueHistoryRepository.getAllByMeter(meter);  }   @Override  public List<ValueHistoryReportDto> getAllValueHistoryByMonth(Timestamp dateStart, Timestamp dateEnd) {  return DtoMapper.mapArrayToList(valueHistoryRepository.getAllValueHistoryByMonth(dateStart, dateEnd), DtoMapper::mapToValueHistoryReportDto);  }   @Override  public List<ValueHistoryReportDto> getAllValueHistoryByMonthAndAddressId(Timestamp dateStart, Timestamp dateEnd, Long addressId) {  return DtoMapper.mapArrayToList(valueHistoryRepository.getAllValueHistoryByMonthAndAddressId(dateStart, dateEnd, addressId), DtoMapper::mapToValueHistoryReportDto);  }   @Override  public List<ValueHistoryReportDto> getAllValueHistoryByMonthAndUserLogin(Timestamp dateStart, Timestamp dateEnd, String userLogin) {  return DtoMapper.mapArrayToList(valueHistoryRepository.getAllValueHistoryByMonthAndUserLogin(dateStart, dateEnd, userLogin), DtoMapper::mapToValueHistoryReportDto);  }   private ValueHistory getLastValue(Meter meter) {  List<ValueHistory> valueHistoryList = valueHistoryRepository.getAllByMeter(meter);  if (valueHistoryList == null || valueHistoryList.isEmpty())  return null;  else  return valueHistoryList.get(valueHistoryList.size() - 1);  }   private Double parseString(String s) {  return Double.valueOf(s);  } } |
| Содержимое файла ValueHistoryServiceImpl.java |

|  |
| --- |
| package ru.tsj.business.report;  import org.apache.poi.xssf.usermodel.XSSFRow; import org.apache.poi.xssf.usermodel.XSSFSheet; import org.apache.poi.xssf.usermodel.XSSFWorkbook; import org.springframework.stereotype.Service; import ru.tsj.business.TsjService; import ru.tsj.business.exceptions.EntityIllegalArgumentException; import ru.tsj.data.entity.report.dto.TsjAndProviderDto;  import javax.servlet.http.HttpServletResponse; import java.util.List;  @Service public class TsjReporter {   private final TsjService tsjService;   public TsjReporter(TsjService tsjService) {  this.tsjService = tsjService;  }    public void getTsjReport(HttpServletResponse response) {  XSSFWorkbook workbook = new XSSFWorkbook();  XSSFSheet sheet = workbook.createSheet();   createHeaderRow(sheet, 0);   List<TsjAndProviderDto> reportData = tsjService.getTsjAndProviderDtos();  for (int i = 0; i < reportData.size(); i++)  addFilledRow(sheet, i + 1, reportData.get(i));   try {  response.addHeader("Content-Disposition", "attachment; filename=report.xlsx");  workbook.write(response.getOutputStream());  }  catch (Exception e) {  throw new EntityIllegalArgumentException("Не удалось произвести эскпорт отчета!");  }  }   private void createHeaderRow(XSSFSheet sheet, int index) {  XSSFRow row = sheet.createRow(index);   row.createCell(0).setCellValue("ID ТСЖ");  row.createCell(1).setCellValue("Название ТСЖ");  row.createCell(2).setCellValue("ID дома");  row.createCell(3).setCellValue("Город");  row.createCell(4).setCellValue("Улица");  row.createCell(5).setCellValue("Номер дома");  row.createCell(6).setCellValue("Наименование ресурсоснабжающей компании");  row.createCell(7).setCellValue("Контакты ресурсоснабжающей компании");  }   private void addFilledRow(XSSFSheet sheet, int index, TsjAndProviderDto dto) {  XSSFRow row = sheet.createRow(index);   row.createCell(0).setCellValue(dto.getTsjId());  row.createCell(1).setCellValue(dto.getTsjName());  row.createCell(2).setCellValue(dto.getAddressId());  row.createCell(3).setCellValue(dto.getCity());  row.createCell(4).setCellValue(dto.getStreet());  row.createCell(5).setCellValue(dto.getHouseNumber());  row.createCell(6).setCellValue(dto.getProviderName());  row.createCell(7).setCellValue(dto.getProviderContact());  } } |
| Содержимое файла TsjReporter.java |

|  |
| --- |
| package ru.tsj.config;  import org.springframework.context.annotation.Import;  @Import({SecurityConfig.class, SwaggerConfig.class}) public class ApplicationConfig { } |
| Содержимое файла ApplicationConfig.java |

|  |
| --- |
| package ru.tsj.config;  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.annotation.\*; import org.springframework.http.HttpMethod; import org.springframework.security.authentication.dao.DaoAuthenticationProvider; import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder; import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity; import org.springframework.security.config.annotation.web.builders.HttpSecurity; import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity; import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter; import org.springframework.security.config.http.SessionCreationPolicy; import org.springframework.security.core.userdetails.UserDetailsService; import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder; import org.springframework.security.crypto.password.PasswordEncoder; import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter; import ru.tsj.web.jwt.JWTLoginFilter; import ru.tsj.web.jwt.JwtAuthenticationFilter; import ru.tsj.web.security.SecurityUserDetailsManager;  @EnableWebSecurity @EnableGlobalMethodSecurity(prePostEnabled = true) public class SecurityConfig extends WebSecurityConfigurerAdapter {   private final SecurityUserDetailsManager userDetailsManager;  private final UserDetailsService userDetailsService;   @Autowired  public SecurityConfig(SecurityUserDetailsManager userDetailsManager, UserDetailsService userDetailsService) {  this.userDetailsManager = userDetailsManager;  this.userDetailsService = userDetailsService;  }   @Override  protected void configure(HttpSecurity http) throws Exception {  http.csrf().disable().cors().disable()  .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)  .and()  .authorizeRequests()  .antMatchers(getSwaggerPatterns()).permitAll()  .antMatchers(HttpMethod.POST, "/user/signup").permitAll()  .antMatchers(HttpMethod.POST, "/auth/\*\*").permitAll()  .anyRequest().authenticated()  .and()  .addFilterBefore(new JWTLoginFilter("/auth/authorize", authenticationManager()), UsernamePasswordAuthenticationFilter.class)  .addFilterBefore(new JwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);  }   @Override  protected void configure(AuthenticationManagerBuilder auth) throws Exception {  DaoAuthenticationProvider daoAuthenticationProvider = new DaoAuthenticationProvider();  daoAuthenticationProvider.setUserDetailsService(userDetailsService);  daoAuthenticationProvider.setPasswordEncoder(passwordEncoder());   auth.authenticationProvider(daoAuthenticationProvider);   auth.userDetailsService(userDetailsManager);  }   @Bean  public PasswordEncoder passwordEncoder() {  return new BCryptPasswordEncoder(10);  }    private static String[] getSwaggerPatterns() {  return new String[] {  "/swagger-ui.html",  "/swagger-ui.html/\*\*",  "/swagger-ui",  "/swagger-ui/\*\*",  "/v2/api-docs",  "/v2/api-docs/\*\*",  "/v3/api-docs",  "/v3/api-docs/\*\*",  "/swagger-resources/\*\*",  "/webjars/\*\*"  };  } } |
| Содержимое файла SecurityConfig.java |

|  |
| --- |
| package ru.tsj.config;  import io.swagger.annotations.Api; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration; import springfox.documentation.builders.PathSelectors; import springfox.documentation.builders.RequestHandlerSelectors; import springfox.documentation.service.ApiInfo; import springfox.documentation.service.ApiKey; import springfox.documentation.spi.DocumentationType; import springfox.documentation.spring.web.plugins.Docket; import springfox.documentation.swagger2.annotations.EnableSwagger2;  import java.util.Collections;  @EnableSwagger2 public class SwaggerConfig {   public static final String AUTHORIZATION\_HEADER = "Authorization";   private ApiKey apiKey(){  return new ApiKey("JWT", AUTHORIZATION\_HEADER, "header");  }   @Bean  public Docket api() {  return new Docket(DocumentationType.SWAGGER\_2)  .apiInfo(apiInfo())  .securitySchemes(Collections.singletonList(apiKey()))  .select()  .apis(RequestHandlerSelectors.withClassAnnotation(Api.class))  .paths(PathSelectors.any())  .build();  }   private ApiInfo apiInfo(){  return new ApiInfo(  "TSJ Service",  "TSJ REST API Documentation \n",  "0.1",  "",  null,  "",  "",  Collections.emptyList()  );  } } |
| Содержимое файла SwaggerConfig.java |

|  |
| --- |
| package ru.tsj.data.entity;  import lombok.\*;  import javax.persistence.\*;  @Entity @Table(name = "address") @Getter @Setter @NoArgsConstructor @AllArgsConstructor public class Address {   public static final String ENTITY\_TYPE = "Address";   @Id  @GeneratedValue(strategy = GenerationType.IDENTITY, generator = "address\_id\_seq")  @SequenceGenerator(name = "address\_id\_seq", sequenceName = "address\_id\_seq", allocationSize = 1)  private Long id;   private String city;   private String street;   @Column(name = "house\_number")  private Long houseNumber;   @ManyToOne(fetch = FetchType.EAGER, cascade = CascadeType.MERGE)  @JoinColumn(name = "tsj\_id", referencedColumnName = "id", nullable = false)  private Tsj tsj; } |
| Содержимое файла Address.java |

|  |
| --- |
| package ru.tsj.data.entity;  import lombok.\*;  import javax.persistence.\*; import java.util.List; import java.util.Set;  @Entity @Table(name = "flat") @Getter @Setter @NoArgsConstructor @AllArgsConstructor public class Flat {   public static final String ENTITY\_TYPE = "Flat";   @Id  @GeneratedValue(strategy = GenerationType.IDENTITY, generator = "flat\_id\_seq")  @SequenceGenerator(name = "flat\_id\_seq", sequenceName = "flat\_id\_seq", allocationSize = 1)  private Long id;   private Long number;   @ManyToOne(fetch = FetchType.EAGER, cascade=CascadeType.MERGE)  @JoinColumn(name = "address\_id")  private Address address;  // @OneToMany(mappedBy = "flat", fetch = FetchType.EAGER, cascade=CascadeType.MERGE) // private List<Meter> meters; } |
| Содержимое файла Flat.java |

|  |
| --- |
| package ru.tsj.data.entity;  import lombok.\*;  import javax.persistence.\*; import java.sql.Timestamp;  @Getter @Setter @Entity @Table(name = "meter") @NoArgsConstructor @AllArgsConstructor public class Meter {   public static final String ENTITY\_TYPE = "Meter";   @Id  @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "meter\_id\_seq")  @SequenceGenerator(name = "meter\_id\_seq", sequenceName = "meter\_id\_seq", allocationSize = 1)  private Long id;   @Column(name = "serial\_number")  private String serialNumber;   @ManyToOne(fetch = FetchType.EAGER)  @JoinColumn(name = "resource\_id")  private Resource resource;   @Column(name = "installation\_date")  private Timestamp installationDate;   @ManyToOne(fetch = FetchType.EAGER, cascade=CascadeType.MERGE)  @JoinColumn(name = "flat\_id", referencedColumnName = "id", nullable = false)  private Flat flat; } |
| Содержимое файла Meter.java |

|  |
| --- |
| package ru.tsj.data.entity;  import lombok.\*;  import javax.persistence.\*;  @Entity @Table(name = "provider") @Getter @Setter @NoArgsConstructor @AllArgsConstructor public class Provider {   public static final String ENTITY\_TYPE = "Provider";   @Id  @GeneratedValue(strategy = GenerationType.IDENTITY, generator = "provider\_id\_seq")  @SequenceGenerator(name = "provider\_id\_seq", sequenceName = "provider\_id\_seq", allocationSize = 1)  private Long id;   private String name;   private String contact; } |
| Содержимое файла Provider.java |

|  |
| --- |
| package ru.tsj.data.entity;  import lombok.\*;  import javax.persistence.\*;  @Entity @Table(name = "resource") @Getter @Setter @NoArgsConstructor @AllArgsConstructor public class Resource {   public static final String ENTITY\_TYPE = "Resource";   @Id  @GeneratedValue(strategy = GenerationType.IDENTITY, generator = "resource\_id\_seq")  @SequenceGenerator(name = "resource\_id\_seq", sequenceName = "resource\_id\_seq", allocationSize = 1)  private Long id;   private String name;   private Double price;   @ManyToOne(fetch = FetchType.EAGER, cascade=CascadeType.MERGE)  @JoinColumn(name="provider\_id")  private Provider provider; } |
| Содержимое файла Resource.java |

|  |
| --- |
| package ru.tsj.data.entity;  import lombok.\*;  import javax.persistence.\*;  @Getter @Setter @Entity @Table(name = "tsj") @NoArgsConstructor @AllArgsConstructor public class Tsj {   public static final String ENTITY\_TYPE = "Tsj";   @Id  @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "tsj\_id\_seq")  @SequenceGenerator(name = "tsj\_id\_seq", sequenceName = "tsj\_id\_seq", allocationSize = 1)  private Long id;   private String name;   @ManyToOne(fetch = FetchType.EAGER, cascade=CascadeType.MERGE)  @JoinColumn(name = "manager\_id", referencedColumnName = "id", nullable = false)  private User manager; } |
| Содержимое файла Tsj.java |

|  |
| --- |
| package ru.tsj.data.entity;  import com.fasterxml.jackson.annotation.JsonIgnore; import lombok.\*;  import javax.persistence.\*; import java.util.List;  @Entity @Table(name = "\"user\"") @Getter @Setter @ToString @NoArgsConstructor @AllArgsConstructor public class User {   public static final String ENTITY\_TYPE = "User";   @Id  @GeneratedValue(strategy = GenerationType.IDENTITY, generator = "user\_id\_seq")  @SequenceGenerator(name = "user\_id\_seq", sequenceName = "user\_id\_seq", allocationSize = 1)  private Long id;   private String login;  // @JsonIgnore // todo чота придумать  private String password;   @Column(name = "first\_name")  private String firstName;   @Column(name = "last\_name")  private String lastName;   private String surname;   @Column(name = "phone\_number")  private String phoneNumber;   @JsonIgnore  @Column(name = "is\_admin")  private boolean isAdmin;   @JsonIgnore  @ManyToMany(fetch = FetchType.EAGER, cascade=CascadeType.MERGE)  @JoinTable(  name = "user\_in\_flat",  joinColumns = @JoinColumn(name = "user\_id"),  inverseJoinColumns = @JoinColumn(name = "flat\_id")  )  private List<Flat> flats;  // // @OneToMany(mappedBy = "manager", fetch = FetchType.EAGER, cascade=CascadeType.MERGE) // private List<Tsj> tsjs; } |
| Содержимое файла User.java |

|  |
| --- |
| package ru.tsj.data.entity;  import lombok.\*; import ru.tsj.data.enumerate.Role;  import javax.persistence.\*;  @Entity @Getter @Setter @Table(name = "user\_in\_flat") @NoArgsConstructor @AllArgsConstructor @IdClass(UserInFlatId.class) public class UserInFlat {   public static final String ENTITY\_TYPE = "UserInFlat";   @Id  @Column(name = "user\_id")  private Long userId;   @Id  @Column(name = "flat\_id")  private Long flatId;   @Enumerated(EnumType.STRING)  private Role role; } |
| Содержимое файла UserInFlat.java |

|  |
| --- |
| package ru.tsj.data.entity;  import lombok.\*;  import javax.persistence.\*; import java.io.Serializable;  @Getter @Setter @NoArgsConstructor @AllArgsConstructor @EqualsAndHashCode public class UserInFlatId implements Serializable {  private Long userId;   private Long flatId; } |
| Содержимое файла UserInFlatId.java |

|  |
| --- |
| package ru.tsj.data.entity;  import lombok.\*;  import javax.persistence.\*; import java.sql.Timestamp;  @Entity @Table(name = "value\_history") @Getter @Setter @NoArgsConstructor @AllArgsConstructor public class ValueHistory {   public static final String ENTITY\_TYPE = "ValueHistory";   @Id  @GeneratedValue(strategy = GenerationType.IDENTITY, generator = "value\_history\_id\_seq")  @SequenceGenerator(name = "value\_history\_id\_seq", sequenceName = "value\_history\_id\_seq", allocationSize = 1)  private Long id;   @Column(name = "value")  private String value;   @Column(name = "date")  private Timestamp date;   @ManyToOne(fetch = FetchType.EAGER, cascade=CascadeType.MERGE)  @JoinColumn(name = "meter\_id", referencedColumnName = "id", nullable = false)  private Meter meter;   @Column(name = "cost")  private Double cost; } |
| Содержимое файла ValueHistory.java |

|  |
| --- |
| package ru.tsj.data.entity.report;  import ru.tsj.data.entity.Meter; import ru.tsj.data.entity.ValueHistory; import ru.tsj.data.entity.report.dto.RentersAndFlatDto; import ru.tsj.data.entity.report.dto.TsjAndProviderDto; import ru.tsj.data.entity.report.dto.UserAndAddressDto; import ru.tsj.data.entity.report.dto.ValueHistoryReportDto;  import java.sql.Timestamp; import java.util.Arrays; import java.util.List; import java.util.function.Function; import java.util.stream.Collectors;  public class DtoMapper { // valueHistoryRepository.getAllValueHistoryByMonth(dateStart, dateEnd).stream().map(DtoMapper::mapToValueHistoryReportDto).collect(Collectors.toList());   public static <T> List<T> mapArrayToList(List<Object[]> data, Function<Object[], T> mapperFunction) {  return data.stream().map(mapperFunction).collect(Collectors.toList());  } // 1970-01-01 00:00:01  public static ValueHistoryReportDto mapToValueHistoryReportDto(Object[] rawData) {  List<String> values = mapToStringList(rawData);   Meter meter = new Meter();  meter.setId(Long.parseLong(values.get(9)));  ValueHistory valueHistory = new ValueHistory(Long.parseLong(values.get(6)), values.get(7), Timestamp.valueOf(values.get(8)), meter, Double.valueOf(values.get(10)));  return new ValueHistoryReportDto(values.get(0), values.get(1), values.get(2), values.get(3), values.get(4), values.get(5), valueHistory);  }   public static TsjAndProviderDto mapToTsjAndProviderDto(Object[] rawData) {  List<String> values = mapToStringList(rawData);   return new TsjAndProviderDto(Long.parseLong(values.get(0)), values.get(1), Long.parseLong(values.get(2)), values.get(3), values.get(4), Long.parseLong(values.get(5)), values.get(6), values.get(7));  }   public static UserAndAddressDto mapToUserAndAddressDto(Object[] rawData) {  List<String> values = mapToStringList(rawData);   return new UserAndAddressDto(values.get(0), values.get(1), values.get(2), values.get(3), values.get(4), Long.parseLong(values.get(5)), values.get(6), values.get(7), Long.parseLong(values.get(8)), values.get(9));  }   public static RentersAndFlatDto mapToRentersAndFlatDto(Object[] rawData) {  List<String> values = mapToStringList(rawData);   return new RentersAndFlatDto(values.get(0), values.get(1), values.get(2), values.get(3), values.get(4), Long.parseLong(values.get(5)), values.get(6), values.get(7), Long.parseLong(values.get(8)));  }   public static List<String> mapToStringList(Object[] rawData) {  return Arrays.stream(rawData).map(Object::toString).collect(Collectors.toList()); // return Arrays.stream(rawData).map(rd -> rd == null ? "-1" : rd.toString()).collect(Collectors.toList());  } } |
| Содержимое файла DtoMapper.java |

|  |
| --- |
| package ru.tsj.data.entity.report.dto;  import lombok.AllArgsConstructor; import lombok.Data;  @Data @AllArgsConstructor public class RentersAndFlatDto {  // select u.login, u.first\_name, u.last\_name, u.surname, u.phone\_number, f.number, a.city, a.street, a.house\_number   private String login;  private String firstName;  private String lastName;  private String surName;  private String phoneNumber;  private Long flatNumber;  private String city;  private String street;  private Long houseNumber;  } |
| Содержимое файла RentersAndFlatDto.java |

|  |
| --- |
| package ru.tsj.data.entity.report.dto;  import lombok.AllArgsConstructor; import lombok.Data;  @Data @AllArgsConstructor public class TsjAndProviderDto {  private Long tsjId;  private String tsjName;  private Long addressId;  private String city;  private String street;  private Long houseNumber;  private String providerName;  private String providerContact; } |
| Содержимое файла TsjAndProviderDto.java |

|  |
| --- |
| package ru.tsj.data.entity.report.dto;  import lombok.AllArgsConstructor; import lombok.Data;  @Data @AllArgsConstructor public class UserAndAddressDto {  private String login;  private String firstName;  private String lastName;  private String surName;  private String phoneNumber;  private Long flatNumber;  private String city;  private String street;  private Long houseNumber;  private String tsjName; } |
| Содержимое файла UserAndAddressDto.java |

|  |
| --- |
| package ru.tsj.data.entity.report.dto;  import lombok.AllArgsConstructor; import lombok.Data; import ru.tsj.data.entity.ValueHistory;  @Data @AllArgsConstructor public class ValueHistoryReportDto {  String login;  String firstName;  String lastName;  String surName;  String phoneNumber;  String serialNumber;  ValueHistory valueHistory; } |
| Содержимое файла ValueHistoryReportDto.java |

|  |
| --- |
| package ru.tsj.data.enumerate; public enum Role {  OWNER(3),  RENTER(2),  BLANK(1);  public final int priority;   Role(int priority) {  this.priority = priority;  }  // @Override // public String toString() { // return this.name().toString(); // } } |
| Содержимое файла Role.java |

|  |
| --- |
| package ru.tsj.data.repos;  import org.springframework.data.jpa.repository.JpaRepository; import org.springframework.stereotype.Repository; import ru.tsj.data.entity.Address;  public interface AddressRepository extends JpaRepository<Address, Long> {   Address findById(Long id);  void deleteById(Long id); } |
| Содержимое файла AddressRepository.java |

|  |
| --- |
| package ru.tsj.data.repos;  import org.springframework.data.jpa.repository.JpaRepository; import ru.tsj.data.entity.Address; import ru.tsj.data.entity.Flat;  import java.util.List;  public interface FlatRepository extends JpaRepository<Flat, Long> {   List<Flat> getFlatsByAddress(Address address);   List<Flat> getFlatsByAddressId(Long addressId);  } |
| Содержимое файла FlatRepository.java |

|  |
| --- |
| package ru.tsj.data.repos;  import org.springframework.data.jpa.repository.JpaRepository; import ru.tsj.data.entity.Meter;  public interface MeterRepository extends JpaRepository<Meter, Long> { } |
| Содержимое файла MeterRepository.java |

|  |
| --- |
| package ru.tsj.data.repos;  import org.springframework.data.jpa.repository.JpaRepository; import ru.tsj.data.entity.Provider;  public interface ProviderRepository extends JpaRepository<Provider, Long> { } |
| Содержимое файла ProviderRepository.java |

|  |
| --- |
| package ru.tsj.data.repos;  import org.springframework.data.jpa.repository.JpaRepository; import ru.tsj.data.entity.Resource;  public interface ResourceRepository extends JpaRepository<Resource, Long> { } |
| Содержимое файла ResourceRepository.java |

|  |
| --- |
| package ru.tsj.data.repos;  import org.springframework.data.jpa.repository.JpaRepository; import ru.tsj.data.entity.UserInFlat;  public interface RoleRepository extends JpaRepository<UserInFlat, Long> { } |
| Содержимое файла RoleRepository.java |

|  |
| --- |
| package ru.tsj.data.repos;  import lombok.Value; import org.springframework.data.jpa.repository.JpaRepository; import org.springframework.data.jpa.repository.Query; import ru.tsj.data.entity.Tsj;  import java.util.List;  public interface TsjRepository extends JpaRepository<Tsj, Long> {   Tsj findById(Long id);  void deleteById(Long id);   @Query(value = "select distinct\n" +  "tsj.id as tsjId, tsj.name as tsjName, a.id as addressId, a.city as city, a.street as street,\n" +  "a.house\_number as housenumber, p.name as providername, p.contact as providercontact\n" +  "from tsj\n" +  "left join address a on tsj.id = a.tsj\_id\n" +  "left join flat f on a.id = f.address\_id\n" +  "left join meter m on m.flat\_id = f.id\n" +  "left join resource r on r.id = m.resource\_id\n" +  "inner join provider p on p.id = r.provider\_id\n" +  "order by tsj.id asc", nativeQuery = true)  List<Object[]> getReport(); } |
| Содержимое файла TsjRepository.java |

|  |
| --- |
| package ru.tsj.data.repos;  import org.springframework.data.jpa.repository.JpaRepository; import ru.tsj.data.entity.Flat; import ru.tsj.data.entity.UserInFlat; import ru.tsj.data.entity.UserInFlatId;  import java.util.List;  public interface UserInFlatRepository extends JpaRepository<UserInFlat, UserInFlatId> {   List<UserInFlat> getUserInFlatsByUserId(Long userId);  } |
| Содержимое файла UserInFlatRepository.java |

|  |
| --- |
| package ru.tsj.data.repos;  import org.springframework.data.jpa.repository.JpaRepository; import org.springframework.data.jpa.repository.Query; import org.springframework.data.repository.query.Param; import ru.tsj.data.entity.User;  import java.util.List;  public interface UserRepository extends JpaRepository<User, Long> {   User findById(Long id);  void deleteById(Long id);   User findOneByLogin(String login);   List<User> findByLogin(String login);   @Query(value = "select \* from \"user\" where id in (select user\_id from user\_in\_flat where role = 'ADMIN');", nativeQuery = true)  List<User> findAllAdmins();   @Query(value = "select u.login, u.first\_name, u.last\_name, u.surname, u.phone\_number, f.number, a.city, a.street, a.house\_number, tsj.name from \"user\" u\n" +  "inner join user\_in\_flat uif on u.id = uif.user\_id\n" +  "inner join flat f on uif.flat\_id = f.id\n" +  "inner join address a on f.address\_id = a.id\n" +  "inner join tsj tsj on a.tsj\_id = tsj.id\n" +  "where tsj.id = :tsjId", nativeQuery = true)  List<Object[]> getAllUsersInfoByTsjId(@Param("tsjId") Long tsjId);   @Query(value = "select u.login, u.first\_name, u.last\_name, u.surname, u.phone\_number, f.number, a.city, a.street, a.house\_number, '' from \"user\" u\n" +  "inner join user\_in\_flat uif on u.id = uif.user\_id\n" +  "inner join flat f on uif.flat\_id = f.id\n" +  "left join address a on f.address\_id = a.id\n" +  "where f.id = :flatId and a.id = :addressId", nativeQuery = true)  List<Object[]> getAllUsersInfoByFlat(@Param("flatId") Long flatId, @Param("addressId") Long addressId);   @Query(value = "select u.login, u.first\_name, u.last\_name, u.surname, u.phone\_number, f.number, a.city, a.street, a.house\_number from \"user\" u\n" +  "inner join user\_in\_flat uif on u.id = uif.user\_id\n" +  "inner join flat f on uif.flat\_id = f.id\n" +  "left join address a on f.address\_id = a.id\n" +  "where uif.role = 'RENTER'", nativeQuery = true)  List<Object[]> getAllRentersInfo();   } |
| Содержимое файла UserRepository.java |

|  |
| --- |
| package ru.tsj.data.repos;  import org.springframework.data.jpa.repository.JpaRepository; import org.springframework.data.jpa.repository.Query; import org.springframework.data.repository.query.Param; import ru.tsj.data.entity.Meter; import ru.tsj.data.entity.ValueHistory;  import java.sql.Timestamp; import java.util.List;  public interface ValueHistoryRepository extends JpaRepository<ValueHistory, Long> {   List<ValueHistory> getAllByMeter(Meter meter);   @Query(value = "select u.login, u.first\_name, u.last\_name, u.surname, u.phone\_number, m.serial\_number, vh.\* from value\_history vh \n" +  "left join meter m on vh.meter\_id = m.id\n" +  "left join flat f on m.flat\_id = f.id\n" +  "left join user\_in\_flat uif on f.id = uif.flat\_id\n" +  "left join \"user\" u on uif.user\_id = u.id\n" +  "where uif.role = 'OWNER'\n" +  "and vh.date between :dateStart and :dateEnd", nativeQuery = true)  List<Object[]> getAllValueHistoryByMonth(@Param("dateStart") Timestamp dateStart, @Param("dateEnd") Timestamp dateEnd);   @Query(value = "select u.login, u.first\_name, u.last\_name, u.surname, u.phone\_number, m.serial\_number, vh.\* from value\_history vh \n" +  "left join meter m on vh.meter\_id = m.id\n" +  "left join flat f on m.flat\_id = f.id\n" +  "left join user\_in\_flat uif on f.id = uif.flat\_id\n" +  "left join \"user\" u on uif.user\_id = u.id\n" +  "where u.login = :userLogin\n" +  "and vh.date between :dateStart and :dateEnd", nativeQuery = true)  List<Object[]> getAllValueHistoryByMonthAndUserLogin(@Param("dateStart") Timestamp dateStart, @Param("dateEnd") Timestamp dateEnd, @Param("userLogin") String userLogin);   @Query(value = "select u.login, u.first\_name, u.last\_name, u.surname, u.phone\_number, m.serial\_number, vh.\* from value\_history vh \n" +  "left join meter m on vh.meter\_id = m.id\n" +  "left join flat f on m.flat\_id = f.id\n" +  "left join user\_in\_flat uif on f.id = uif.flat\_id\n" +  "left join \"user\" u on uif.user\_id = u.id\n" +  "where uif.role in ('OWNER', 'RENTER') and f.address\_id = :addressId\n" +  "and vh.date between :dateStart and :dateEnd", nativeQuery = true)  List<Object[]> getAllValueHistoryByMonthAndAddressId(@Param("dateStart") Timestamp dateStart, @Param("dateEnd") Timestamp dateEnd, @Param("addressId") Long addressId);  } |
| Содержимое файла ValueHistoryRepository.java |

|  |
| --- |
| package ru.tsj.utils;  import ru.tsj.business.exceptions.EntityValidationException; import ru.tsj.data.entity.\*;  public class EntityValidator {   public static void validateResource(Resource resource) {  if (resource == null)  throw new EntityValidationException(ValueHistory.ENTITY\_TYPE, "Ресурс не может быть null");  if (StringUtils.isNullOrEmpty(resource.getName()))  throw new EntityValidationException(ValueHistory.ENTITY\_TYPE, "Имя ресурса не может быть null");  if (resource.getPrice() == null)  throw new EntityValidationException(ValueHistory.ENTITY\_TYPE, "Цена ресурса не может быть null");  if (resource.getProvider() == null)  throw new EntityValidationException(ValueHistory.ENTITY\_TYPE, "Провайдер ресурса не может быть null");   }   public static void validateValueHistory(ValueHistory valueHistory) {  if (valueHistory == null)  throw new EntityValidationException(ValueHistory.ENTITY\_TYPE, "История значений не может быть null"); // if (valueHistory.getMeter() == null) // throw new EntityValidationException(ValueHistory.ENTITY\_TYPE, "Счетчик не может быть null"); // validateMeter(valueHistory.getMeter());  if (StringUtils.isNullOrEmpty(valueHistory.getValue()))  throw new EntityValidationException(ValueHistory.ENTITY\_TYPE, "Значение счетчика не может быть null");  if (valueHistory.getDate() == null)  throw new EntityValidationException(ValueHistory.ENTITY\_TYPE, "Дата значения не может быть null"); // if (valueHistory.getCost() == null) // throw new EntityValidationException(ValueHistory.ENTITY\_TYPE, "Расчетная стоимость по показаниям счетчика не может быть null");  }   public static void validateProvider(Provider provider) {  if (provider == null)  throw new EntityValidationException(Provider.ENTITY\_TYPE, "Провайдер не может быть null");  if (StringUtils.isNullOrEmpty(provider.getName()))  throw new EntityValidationException(Provider.ENTITY\_TYPE, "Имя провайдера не может быть null или пустым");  if (StringUtils.isNullOrEmpty(provider.getContact()))  throw new EntityValidationException(Provider.ENTITY\_TYPE, "Контактные данные провайдера не могут быть null или пустым");  }   public static void validateTsj(Tsj tsj) {  if (tsj == null)  throw new EntityValidationException(Tsj.ENTITY\_TYPE, "ТСЖ не может быть null");  if (StringUtils.isNullOrEmpty(tsj.getName()))  throw new EntityValidationException(Tsj.ENTITY\_TYPE, "Имя ТСЖ не может быть null или пустым");  }   public static void validateUser(User user) {  if (user == null)  throw new EntityValidationException(User.ENTITY\_TYPE, "Пользователь не может быть null");  if (StringUtils.isNullOrEmpty(user.getLogin()))  throw new EntityValidationException(User.ENTITY\_TYPE, "Логин пользователя не может быть null или пустым");  if (StringUtils.isNullOrEmpty(user.getPassword()))  throw new EntityValidationException(User.ENTITY\_TYPE, "Пароль пользователя не может быть null или пустым");  if (StringUtils.isNullOrEmpty(user.getFirstName()))  throw new EntityValidationException(User.ENTITY\_TYPE, "Имя пользователя не может быть null или пустым");  if (StringUtils.isNullOrEmpty(user.getLastName()))  throw new EntityValidationException(User.ENTITY\_TYPE, "Фамилия пользователя не может быть null или пустым");  if (StringUtils.isNullOrEmpty(user.getPhoneNumber()))  throw new EntityValidationException(User.ENTITY\_TYPE, "Номер телефона пользователя не может быть null или пустым");  }   public static void validateFlat(Flat flat) {  if (flat == null)  throw new EntityValidationException(Flat.ENTITY\_TYPE, "Квартира не может быть null");  if (flat.getNumber() == null || flat.getNumber() < 1)  throw new EntityValidationException(Flat.ENTITY\_TYPE, "Номер квартиры не может быть null или меньше 1");  if (flat.getAddress() == null)  throw new EntityValidationException(Flat.ENTITY\_TYPE, "Адрес квартиры не может быть null");  }   public static void validateAddress(Address address) {  if (address == null)  throw new EntityValidationException(Address.ENTITY\_TYPE, "Адрес не может быть null");  }   public static void validateMeter(Meter meter) {  if (meter == null)  throw new EntityValidationException(Meter.ENTITY\_TYPE, "Счетчик не может быть null");  if (meter.getFlat() == null)  throw new EntityValidationException(Meter.ENTITY\_TYPE, "Квартира не может быть null");  if (meter.getResource() == null)  throw new EntityValidationException(Meter.ENTITY\_TYPE, "Ресурс не может быть null");  if (StringUtils.isNullOrEmpty(meter.getSerialNumber()))  throw new EntityValidationException(Meter.ENTITY\_TYPE, "Серийный номер не может быть null или пустым");  if (meter.getInstallationDate() == null)  throw new EntityValidationException(Meter.ENTITY\_TYPE, "Дата установки не может быть null");  } } |
| Содержимое файла EntityValidator.java |

|  |
| --- |
| package ru.tsj.utils;   import org.springframework.security.crypto.password.PasswordEncoder; import org.springframework.stereotype.Component;  @Component public class PasswordUtils {   private static PasswordEncoder passwordEncoder;   public PasswordUtils(PasswordEncoder passwordEncoder) {  PasswordUtils.passwordEncoder = passwordEncoder;  }   public static String hashPassword(String password) {  return passwordEncoder.encode(password);  }  // public static String hashPassword(String password) { // return password; // } } |
| Содержимое файла PasswordUtils.java |

|  |
| --- |
| package ru.tsj.utils;  public class StringUtils {   public static boolean isNullOrEmpty(String str) {  return str == null || str.isEmpty();  } } |
| Содержимое файла StringUtils.java |

|  |
| --- |
| package ru.tsj.web.controller;  import io.swagger.annotations.Api; import org.springframework.web.bind.annotation.\*; import ru.tsj.business.AddressService; import ru.tsj.data.entity.Address;  import java.util.List;  @Api @RestController @RequestMapping("/address") public class AddressController {   private final AddressService addressService;   public AddressController(AddressService addressService) {  this.addressService = addressService;  }   @PostMapping("/add")  public Address addAddress(@RequestBody Address address, @RequestParam Integer flatsCount) {  return addressService.addAddress(address, flatsCount);  }   @GetMapping("/getById")  public Address getById(@RequestParam Long id) {  return addressService.getAddressById(id);  }   @GetMapping("/getAll")  public List<Address> getAll() {  return addressService.getAll();  }   @PutMapping("/update")  public Address updateAddress(@RequestBody Address address) {  return addressService.updateAddress(address);  }  } |
| Содержимое файла AddressController.java |

|  |
| --- |
| package ru.tsj.web.controller;  import io.swagger.annotations.Api; import org.springframework.web.bind.annotation.\*; import ru.tsj.business.FlatService; import ru.tsj.business.UserInFlatService; import ru.tsj.data.entity.Address; import ru.tsj.data.entity.Flat; import ru.tsj.data.enumerate.Role;  import java.util.List;  @Api @RestController @RequestMapping("/flat") public class FlatController {   private final FlatService flatService;  private final UserInFlatService userInFlatService;   public FlatController(FlatService flatService, UserInFlatService userInFlatService) {  this.flatService = flatService;  this.userInFlatService = userInFlatService;  }   @PostMapping("/addFlat")  public Flat addFlat(@RequestBody Flat flat) {  return flatService.addFlat(flat);  }   @PostMapping("/addFlatsByRange")  public List<Flat> addFlatsByRange(@RequestParam Integer startNumber, @RequestParam Integer endNumber, @RequestBody Address address) {  return flatService.addFlatsByRange(startNumber, endNumber, address);  }   @PostMapping("/save")  public boolean saveLink(@RequestParam Long userId, @RequestParam Long flatId) {  return userInFlatService.saveLink(userId, flatId);  }   @DeleteMapping("/delete")  public boolean deleteLink(@RequestParam Long userId, @RequestParam Long flatId) {  return userInFlatService.deleteLink(userId, flatId);  }   @PutMapping("/setRole")  public boolean setRole(@RequestParam Long userId, @RequestParam Long flatId, @RequestParam Role role) {  return userInFlatService.setRole(userId, flatId, role);  }   @GetMapping("/getAll")  public List<Flat> getAll() {  return flatService.getAll();  }   @GetMapping("/getById")  public Flat getById(@RequestParam Long id) {  return flatService.getFlatById(id);  }   @PutMapping("/update")  public Flat updateFlat(@RequestBody Flat flat) {  return flatService.updateFlat(flat);  }   @GetMapping("/getByAddress")  public List<Flat> getByAddress(@RequestBody Address address) {  return flatService.getFlatByAddress(address);  }   @GetMapping("/getByAddressId")  public List<Flat> getByAddress(@RequestParam Long addressId) {  return flatService.getFlatByAddressId(addressId);  } } |
| Содержимое файла FlatController.java |

|  |
| --- |
| package ru.tsj.web.controller;  import io.swagger.annotations.Api; import org.springframework.web.bind.annotation.\*; import ru.tsj.business.MeterService; import ru.tsj.data.entity.Meter; import java.sql.Timestamp; import java.util.List;  @Api @RestController @RequestMapping("/meter") public class MeterController {   private final MeterService meterService;   public MeterController(MeterService meterService) {  this.meterService = meterService;  }   // @PostMapping("/add") // public Meter add(@RequestParam String serialNumber, // @RequestParam Timestamp installationDate, // @RequestParam Long resourceId, // @RequestParam Long flatId) { // return meterService.add(serialNumber, installationDate, resourceId, flatId); // }   @GetMapping("/getById")  public Meter getById(@RequestParam Long id) {  return meterService.getMeterById(id);  }   @GetMapping("/getAll")  public List<Meter> getAll() {  return meterService.getAll();  }   @PutMapping("/update")  public Meter updateMeter(@RequestBody Meter meter) {  return meterService.updateMeter(meter);  }   @PostMapping("/add")  public Meter add(@RequestBody Meter meter) {  return meterService.add(meter);  }  } |
| Содержимое файла MeterController.java |

|  |
| --- |
| package ru.tsj.web.controller;  import io.swagger.annotations.Api; import org.springframework.web.bind.annotation.\*; import ru.tsj.business.ProviderService; import ru.tsj.data.entity.Meter; import ru.tsj.data.entity.Provider;  import java.util.List;  @Api @RestController @RequestMapping("/provider") public class ProviderController {   private final ProviderService providerService;   public ProviderController(ProviderService providerService) {  this.providerService = providerService;  }    @PostMapping("/add")  public Provider add(@RequestBody Provider provider) {  return providerService.add(provider);  }   @GetMapping("/getById")  public Provider getById(@RequestParam Long id) {  return providerService.getById(id);  }   @GetMapping("/getAll")  public List<Provider> getAll() {  return providerService.getAll();  }   @PutMapping("/update")  public Provider updateMeter(@RequestBody Provider provider) {  return providerService.updateProvider(provider);  } } |
| Содержимое файла ProviderController.java |

|  |
| --- |
| package ru.tsj.web.controller;  import io.swagger.annotations.Api; import org.springframework.web.bind.annotation.\*; import ru.tsj.business.ResourceService; import ru.tsj.data.entity.Resource;  import java.util.List;  @Api @RestController @RequestMapping("/resource") public class ResourceController {   private final ResourceService resourceService;   public ResourceController(ResourceService resourceService) {  this.resourceService = resourceService;  }   @PostMapping("/add")  public Resource add(@RequestBody Resource resource) {  return resourceService.add(resource);  }   @GetMapping("/getById")  public Resource getById(@RequestParam Long id) {  return resourceService.getById(id);  }   @GetMapping("/getAll")  public List<Resource> getAll() {  return resourceService.getAll();  }   @PutMapping("/update")  public Resource update(@RequestBody Resource resource) {  return resourceService.updateResource(resource);  }  } |
| Содержимое файла ResourceController.java |

|  |
| --- |
| package ru.tsj.web.controller;  import io.swagger.annotations.Api; import org.springframework.security.access.prepost.PreAuthorize; import org.springframework.web.bind.annotation.\*; import ru.tsj.business.TsjService; import ru.tsj.business.report.TsjReporter; import ru.tsj.data.entity.Tsj; import ru.tsj.data.entity.report.dto.TsjAndProviderDto;  import javax.servlet.http.HttpServletResponse;  import java.util.List;  @Api @RestController @RequestMapping("/tsj") public class TsjController {   private final TsjService tsjService;  private final TsjReporter tsjReporter;   public TsjController(TsjService tsjService, TsjReporter tsjReporter) {  this.tsjService = tsjService;  this.tsjReporter = tsjReporter;  }   @PostMapping("/add")  public Tsj addTsj(@RequestBody Tsj tsj) {  return tsjService.addTsj(tsj);  }   @PostMapping("/setManager")  @PreAuthorize("hasPermission('tsj', 'MANAGER')")  public Tsj setManager(@RequestParam Long tsjId, @RequestParam Long managerId) {  return tsjService.setManager(tsjId, managerId);  }   @GetMapping("/getById")  public Tsj getById(@RequestParam Long id) {  return tsjService.findOneById(id);  }   @GetMapping("/getAll")  public List<Tsj> getById() {  return tsjService.findAll();  }   @PutMapping("/update")  public Tsj updateTsj(@RequestBody Tsj tsj) {  return tsjService.updateTsj(tsj);  }   @PostMapping("/test")  @PreAuthorize("hasPermission('flat', 'BLANK')")  public String test(@RequestParam Long flatId) {  return "test";  }   @GetMapping(value = "/report")  public void getReport(HttpServletResponse response) {  tsjReporter.getTsjReport(response);  }   @GetMapping(value = "/getTsjAndProviderDto")  public List<TsjAndProviderDto> getTsjAndProviderDto() {  return tsjService.getTsjAndProviderDtos();  } } |
| Содержимое файла TsjController.java |

|  |
| --- |
| package ru.tsj.web.controller;  import io.swagger.annotations.Api; import org.springframework.web.bind.annotation.\*; import ru.tsj.business.UserService; import ru.tsj.data.entity.User; import ru.tsj.data.entity.report.dto.RentersAndFlatDto; import ru.tsj.data.entity.report.dto.UserAndAddressDto;  import javax.servlet.http.HttpServletResponse; import java.util.List;  @Api @RestController @RequestMapping("/user") public class UserController {   private final UserService userService;   public UserController(UserService userService) {  this.userService = userService;  }   @PostMapping("/signup")  public User signup(@RequestBody User user, HttpServletResponse response) {  user.setAdmin(false);   return userService.signup(user, response);  }   @PostMapping("/setIsAdmin")  public User setIsAdmin(@RequestParam String login, @RequestParam boolean isAdmin) {  return userService.setIsAdmin(login, isAdmin);  }   @PostMapping("/addRenter")  public String addRenter(@RequestParam String login, @RequestParam Long flatId) {  return userService.addRenter(login, flatId);  }   @GetMapping("/getAllUsersInfoByTsjId")  public List<UserAndAddressDto> getAllUsersInfoByTsjId(@RequestParam Long tsjId) {  return userService.getAllUsersInfoByTsjId(tsjId);  }   @GetMapping("/getAllUsersInfoByFlatId")  public List<UserAndAddressDto> getAllUsersInfoByFlatId(@RequestParam Long flatId, @RequestParam Long addressId) {  return userService.getAllUsersInfoByFlatId(flatId, addressId);  }   @GetMapping("/getAllRentersAndFlat")  public List<RentersAndFlatDto> getAllRentersAndFlat() {  return userService.getAllRentersAndFlatDto();  }    @GetMapping("/getAll")  public List<User> getAll() {  return userService.findAll();  } } |
| Содержимое файла UserController.java |

|  |
| --- |
| package ru.tsj.web.controller;  import io.swagger.annotations.Api; import org.springframework.web.bind.annotation.\*; import ru.tsj.business.ValueHistoryService; import ru.tsj.data.entity.Meter; import ru.tsj.data.entity.ValueHistory; import ru.tsj.data.entity.report.dto.ValueHistoryReportDto;  import java.sql.Timestamp; import java.util.List;  @Api @RestController @RequestMapping("/valueHistory") public class ValueHistoryController {   private final ValueHistoryService valueHistoryService;   public ValueHistoryController(ValueHistoryService valueHistoryService) {  this.valueHistoryService = valueHistoryService;  }   @PostMapping("/add")  public ValueHistory add(@RequestBody ValueHistory valueHistory) {  return valueHistoryService.add(valueHistory);  }   @GetMapping("/getById")  public ValueHistory getById(@RequestParam Long id) {  return valueHistoryService.getById(id);  }   @GetMapping("/getAll")  public List<ValueHistory> getAll() {  return valueHistoryService.getAll();  }   @PutMapping("/update")  public ValueHistory update(@RequestBody ValueHistory valueHistory) {  return valueHistoryService.update(valueHistory);  }   @DeleteMapping("/delete")  public void delete(@RequestParam Long id) {  valueHistoryService.delete(id);  }   @GetMapping("/getByMeter")  public List<ValueHistory> getByMeter(@RequestBody Meter meter) {  return valueHistoryService.getAllByMeter(meter);  }   @GetMapping("/getAllValueHistoryByMonth")  public List<ValueHistoryReportDto> getAllValueHistoryByMonth(@RequestParam Timestamp dateStart, @RequestParam Timestamp dateEnd) {  return valueHistoryService.getAllValueHistoryByMonth(dateStart, dateEnd);  }   @GetMapping("/getAllValueHistoryByMonthAndAddressId")  public List<ValueHistoryReportDto> getAllValueHistoryByMonthAndAddressId(@RequestParam Timestamp dateStart, @RequestParam Timestamp dateEnd, @RequestParam Long addressId) {  return valueHistoryService.getAllValueHistoryByMonthAndAddressId(dateStart, dateEnd, addressId);  }   @GetMapping("/getAllValueHistoryByMonthAndUserLogin")  public List<ValueHistoryReportDto> getAllValueHistoryByMonthAndUserLogin(@RequestParam Timestamp dateStart, @RequestParam Timestamp dateEnd, @RequestParam String userLogin) {  return valueHistoryService.getAllValueHistoryByMonthAndUserLogin(dateStart, dateEnd, userLogin);  } } |
| Содержимое файла ValueHistoryController.java |

|  |
| --- |
| package ru.tsj.web.controller.exceptions;  import lombok.Getter; import lombok.Setter;  @Getter @Setter public class ErrorResponseEntity {   private String message;  private String error;  private int status;   public ErrorResponseEntity(String message, String error, int status) {  this.message = message;  this.error = error;  this.status = status;  } } |
| Содержимое файла ErrorResponseEntity.java |

|  |
| --- |
| package ru.tsj.web.controller.exceptions;  import org.springframework.http.HttpStatus; import org.springframework.web.bind.annotation.ControllerAdvice; import org.springframework.web.bind.annotation.ExceptionHandler; import org.springframework.web.bind.annotation.ResponseBody; import org.springframework.web.bind.annotation.ResponseStatus; import ru.tsj.business.exceptions.\*;  @ControllerAdvice public class ExceptionController {   @ExceptionHandler(BaseException.class)  @ResponseStatus(HttpStatus.INTERNAL\_SERVER\_ERROR)  @ResponseBody  public ErrorResponseEntity handleBaseException(BaseException e) {  return createErrorResponseEntity(e, HttpStatus.INTERNAL\_SERVER\_ERROR);  }   @ExceptionHandler(EntityAlreadyExistsException.class)  @ResponseStatus(HttpStatus.CONFLICT)  @ResponseBody  public ErrorResponseEntity handleEntityAlreadyExistsException(EntityAlreadyExistsException e) {  return createErrorResponseEntity(e, HttpStatus.CONFLICT);  }   @ExceptionHandler(EntityConflictException.class)  @ResponseStatus(HttpStatus.CONFLICT)  @ResponseBody  public ErrorResponseEntity handleEntityConflictException(EntityConflictException e) {  return createErrorResponseEntity(e, HttpStatus.CONFLICT);  }   @ExceptionHandler(EntityHasDetailException.class)  @ResponseStatus(HttpStatus.CONFLICT)  @ResponseBody  public ErrorResponseEntity handleEntityHasDetailException(EntityHasDetailException e) {  return createErrorResponseEntity(e, HttpStatus.CONFLICT);  }   @ExceptionHandler(EntityIllegalArgumentException.class)  @ResponseStatus(HttpStatus.BAD\_REQUEST)  @ResponseBody  public ErrorResponseEntity handleEntityIllegalArgumentException(EntityIllegalArgumentException e) {  return createErrorResponseEntity(e, HttpStatus.BAD\_REQUEST);  }   @ExceptionHandler(EntityNotFoundException.class)  @ResponseStatus(HttpStatus.NOT\_FOUND)  @ResponseBody  public ErrorResponseEntity handleEntityNotFoundException(EntityNotFoundException e) {  return createErrorResponseEntity(e, HttpStatus.NOT\_FOUND);  }   @ExceptionHandler(UserNotFoundException.class)  @ResponseStatus(HttpStatus.UNAUTHORIZED)  @ResponseBody  public ErrorResponseEntity handleUserNotFoundException(UserNotFoundException e) {  return createErrorResponseEntity(e, HttpStatus.UNAUTHORIZED);  }   @ExceptionHandler(EntityValidationException.class)  @ResponseStatus(HttpStatus.BAD\_REQUEST)  @ResponseBody  public ErrorResponseEntity handleEntityValidationException(EntityValidationException e) {  return createErrorResponseEntity(e, HttpStatus.BAD\_REQUEST);  }   private static ErrorResponseEntity createErrorResponseEntity (BaseException e, HttpStatus httpStatus) {  return new ErrorResponseEntity(e.getMessage(), httpStatus.getReasonPhrase(), httpStatus.value());  } } |
| Содержимое файла ExceptionController.java |

|  |
| --- |
| package ru.tsj.web.jwt;  public class AccountCredentials {  private String login;  private String password;   public String getLogin() {  return login;  }   public String getPassword() {  return password;  } } |
| Содержимое файла AccountCredentials.java |

|  |
| --- |
| package ru.tsj.web.jwt;  import org.springframework.security.core.Authentication; import org.springframework.security.core.context.SecurityContextHolder; import org.springframework.web.filter.GenericFilterBean; import ru.tsj.web.security.TokenAuthenticationService;  import javax.servlet.FilterChain; import javax.servlet.ServletException; import javax.servlet.ServletRequest; import javax.servlet.ServletResponse; import javax.servlet.http.HttpServletRequest; import java.io.IOException;  public class JwtAuthenticationFilter extends GenericFilterBean {  @Override  public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain) throws IOException, ServletException {  HttpServletRequest servletRequest = (HttpServletRequest) request;  Authentication authentication = TokenAuthenticationService.getAuthentication(servletRequest);  SecurityContextHolder.getContext().setAuthentication(authentication);  filterChain.doFilter(request, response);  } } |
| Содержимое файла JwtAuthenticationFilter.java |

|  |
| --- |
| package ru.tsj.web.jwt;  import com.fasterxml.jackson.databind.ObjectMapper; import org.springframework.security.authentication.AuthenticationManager; import org.springframework.security.authentication.UsernamePasswordAuthenticationToken; import org.springframework.security.core.Authentication; import org.springframework.security.core.AuthenticationException; import org.springframework.security.core.GrantedAuthority; import org.springframework.security.web.authentication.AbstractAuthenticationProcessingFilter; import org.springframework.security.web.util.matcher.AntPathRequestMatcher; import ru.tsj.web.security.TokenAuthenticationService;  import javax.servlet.FilterChain; import javax.servlet.ServletException; import javax.servlet.http.HttpServletRequest; import javax.servlet.http.HttpServletResponse; import java.io.IOException; import java.util.Collections;  public class JWTLoginFilter extends AbstractAuthenticationProcessingFilter {  public JWTLoginFilter(String url, AuthenticationManager authManager) {  super(new AntPathRequestMatcher(url));  setAuthenticationManager(authManager);  }   @Override  public Authentication attemptAuthentication(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse)  throws AuthenticationException, IOException, ServletException {  AccountCredentials credentials = new ObjectMapper().readValue(httpServletRequest.getInputStream(), AccountCredentials.class);  return getAuthenticationManager().authenticate(  new UsernamePasswordAuthenticationToken(  credentials.getLogin(),  credentials.getPassword(),  Collections.<GrantedAuthority>emptyList()  )  );  }   @Override  public void successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain chain, Authentication auth) {  TokenAuthenticationService.addAuthentication(response, auth.getName());  }  } |
| Содержимое файла JWTLoginFilter.java |

|  |
| --- |
| package ru.tsj.web.security;  import org.springframework.security.access.PermissionEvaluator; import org.springframework.security.core.Authentication; import org.springframework.security.core.userdetails.User; import org.springframework.stereotype.Component; import org.springframework.web.context.request.RequestAttributes; import org.springframework.web.context.request.RequestContextHolder; import org.springframework.web.context.request.ServletRequestAttributes; import ru.tsj.business.TsjService; import ru.tsj.business.UserInFlatService; import ru.tsj.business.exceptions.EntityIllegalArgumentException; import ru.tsj.business.exceptions.UserNotFoundException; import ru.tsj.data.enumerate.Role;  import java.io.Serializable;  @Component public class DefaultPermissionEvaluator implements PermissionEvaluator {   private final static String FLAT\_ID = "flatId";  private final static String TSJ\_ID = "tsjId";   private final TsjService tsjService;  private final UserInFlatService userInFlatService;   public DefaultPermissionEvaluator(TsjService tsjService, UserInFlatService userInFlatService) {  this.tsjService = tsjService;  this.userInFlatService = userInFlatService;  }   @Override  public boolean hasPermission(Authentication authentication, Object resource, Object minRole) { // resource - объект, например квартира, тсж, счетчик... // minRole - минимальная роль для чела    SecurityUser user;  if (authentication.getPrincipal() instanceof User)  user = (SecurityUser) authentication.getPrincipal();  else  throw new UserNotFoundException("Ошибка получения пользователя");  if (user.getUser().isAdmin())  return true;   RequestAttributes requestAttributes = RequestContextHolder.getRequestAttributes();   if (resource.equals("tsj")) {  Long tsjId = null;  try {  tsjId = Long.parseLong(((ServletRequestAttributes)requestAttributes).getRequest().getParameter(TSJ\_ID));  } catch (Exception e) {  throw new IllegalArgumentException(e.getMessage());  }   return user.getUser().getId().equals(tsjService.getManager(tsjId).getId());  }  else if (resource.equals("flat")) {  Long flatId = null;  try {  flatId = Long.parseLong(((ServletRequestAttributes)requestAttributes).getRequest().getParameter(FLAT\_ID));  } catch (Exception e) {  throw new IllegalArgumentException(e.getMessage());  }   return userInFlatService.checkRole(user.getUser().getId(), flatId, Role.valueOf(minRole.toString()));  }  throw new EntityIllegalArgumentException("Данный объект не поддерживается");   }   @Override  public boolean hasPermission(Authentication authentication, Serializable serializable, String s, Object o) {  return false;  } } |
| Содержимое файла DefaultPermissionEvaluator.java |

|  |
| --- |
| package ru.tsj.web.security;  import lombok.NoArgsConstructor; import org.springframework.security.core.GrantedAuthority;  @NoArgsConstructor public class SecurityPermission implements GrantedAuthority {   public SecurityPermission(String name) {  this.name = name;  }   private String name;   public void setAuthority(String name) {  this.name = name;  }   @Override  public String getAuthority() {  return name;  } } |
| Содержимое файла SecurityPermission.java |

|  |
| --- |
| package ru.tsj.web.security;  import lombok.Getter; import lombok.Setter; import org.springframework.security.core.GrantedAuthority; import org.springframework.security.core.userdetails.User;  import java.util.Collection;  @Getter @Setter public class SecurityUser extends User {   private ru.tsj.data.entity.User user;  public SecurityUser(ru.tsj.data.entity.User user, String username, String password, Collection<? extends GrantedAuthority> authorities) {  super(username, password, authorities);  this.user = user;  } } |
| Содержимое файла SecurityUser.java |

|  |
| --- |
| package ru.tsj.web.security;  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.security.core.userdetails.UserDetails; import org.springframework.security.core.userdetails.UsernameNotFoundException; import org.springframework.security.provisioning.UserDetailsManager; import org.springframework.stereotype.Component; import ru.tsj.business.exceptions.EntityAlreadyExistsException; import ru.tsj.data.entity.User; import ru.tsj.data.entity.UserInFlat; import ru.tsj.data.repos.UserInFlatRepository; import ru.tsj.data.repos.UserRepository;  import java.util.ArrayList; import java.util.List; import java.util.stream.Collectors;  @Component public class SecurityUserDetailsManager implements UserDetailsManager {   private final UserRepository userRepository;  private final UserInFlatRepository userInFlatRepository;   @Autowired  public SecurityUserDetailsManager(UserRepository userRepository, UserInFlatRepository userInFlatRepository) {  this.userRepository = userRepository;  this.userInFlatRepository = userInFlatRepository;  }    @Override  public void createUser(UserDetails userDetails) {   }   @Override  public void updateUser(UserDetails userDetails) {   }   @Override  public void deleteUser(String s) {   }   @Override  public void changePassword(String s, String s1) {   }   @Override  public boolean userExists(String s) {  return false;  }   @Override  public UserDetails loadUserByUsername(String login) throws UsernameNotFoundException {  User foundedUser = userRepository.findOneByLogin(login);  if (foundedUser == null)  throw new EntityAlreadyExistsException("Пользователь не найден");   List<SecurityPermission> permissions = userInFlatRepository.getUserInFlatsByUserId(foundedUser.getId()).stream().map(uif -> new SecurityPermission(String.format("%d.%d", uif.getFlatId(), foundedUser.getId()))).collect(Collectors.toList());   return new SecurityUser(foundedUser, foundedUser.getLogin(), foundedUser.getPassword(), permissions);  } } |
| Содержимое файла SecurityUserDetailsManager.java |

|  |
| --- |
| package ru.tsj.web.security;  import io.jsonwebtoken.JwtException; import io.jsonwebtoken.Jwts; import io.jsonwebtoken.SignatureAlgorithm; import org.slf4j.Logger; import org.slf4j.LoggerFactory; import org.springframework.security.authentication.UsernamePasswordAuthenticationToken; import org.springframework.security.core.Authentication; import org.springframework.security.core.userdetails.UserDetails; import org.springframework.stereotype.Service; import ru.tsj.data.entity.User;  import javax.servlet.http.HttpServletRequest; import javax.servlet.http.HttpServletResponse; import java.util.Date;  import static org.springframework.util.StringUtils.hasText;   @Service public class TokenAuthenticationService {  private static final String SECRET = "Secret"; // private static final long EXPIRATION\_TIME = 864\_000\_000;  private static final long EXPIRATION\_TIME = 864000000;  private static final String TOKEN\_PREFIX = "Bearer ";  private static final String HEADER\_STRING = "Authorization";   private static Logger LOG = LoggerFactory.getLogger(TokenAuthenticationService.class);   private static SecurityUserDetailsManager securityUserDetailsManager;   public TokenAuthenticationService(SecurityUserDetailsManager securityUserDetailsManager) {  this.securityUserDetailsManager = securityUserDetailsManager;  }   public static void addAuthentication(HttpServletResponse response, String username) {  response.addHeader(HEADER\_STRING, TOKEN\_PREFIX + generateToken(username));  }   public static Authentication getAuthentication(HttpServletRequest request) {  String token = getToken(request);   if (!hasText(token)) {  return null;  }   String userName = getUsername(token);   SecurityUser user = (SecurityUser) securityUserDetailsManager.loadUserByUsername(userName);  return new UsernamePasswordAuthenticationToken(user, null, user.getAuthorities());  }   private static String generateToken(String username) {  return Jwts.builder()  .setSubject(username)  .setExpiration(new Date(System.currentTimeMillis() + EXPIRATION\_TIME))  .signWith(SignatureAlgorithm.HS512, SECRET)  .compact();  }   private static String getToken(HttpServletRequest request) {  if (request.getHeader(HEADER\_STRING) != null)  return request.getHeader(HEADER\_STRING).replace(TOKEN\_PREFIX, "");  else  return null;  }   public static String getUsername(String token) {  try {  return token != null ? Jwts.parser()  .setSigningKey(SECRET)  .parseClaimsJws(token)  .getBody()  .getSubject() : null;  } catch (JwtException e) {  LOG.info("Ошибка обработки токена: {}", token);  return null;  }  } } |
| Содержимое файла TokenAuthenticationService.java |

Код Frontend части приложения

|  |
| --- |
| import axios from 'axios';  // Создаем экземпляр axios const $api = axios.create({  withCredentials:true, });  $api.interceptors.request.use((config) => {  // Добавляем токен в заголовки  config.headers.Authorization = `Bearer ${localStorage.getItem('jwtToken')}`;  return config;  }, (error) => {  return Promise.reject(error); });  export default $api; |
| Содержимое файла api.ts |

|  |
| --- |
| import { BrowserRouter as Router, } from "react-router-dom"; import { ProjectRoutes } from "./shared/ProjectRoutes/Routes"; import './styles/base.css';  const App = () => (  <Router>  <ProjectRoutes />  </Router> )  export default App |
| Содержимое файла App.tsx |

|  |
| --- |
| import ReactDOM from 'react-dom/client' import App from './App.tsx' import 'normalize.css';  ReactDOM.createRoot(document.getElementById('root') as HTMLElement).render(  <App /> ) |
| Содержимое файла main.tsx |

|  |
| --- |
| import { Button } from 'antd'; import $api from '../../api.ts'; import { API\_DELETE\_VALUE} from '../../constants/routes.ts';   const DeleteButton = ({props}) => {  const deleteSubmit = () => {  $api.delete(API\_DELETE\_VALUE, { params: {  id: props.valueHistory.id  }})  .then(async result => console.log(result.data))  .catch(error => {  console.error(error);  });  }   return (  <Button type='link' htmlType='submit' onClick={deleteSubmit}>Удалить</Button>  ) }  export default DeleteButton |
| Содержимое файла DeleteButton.tsx |

|  |
| --- |
| import React, { useState } from 'react'; import { Button, Form, Input, Modal } from 'antd'; import $api from '../../api.ts'; import { API\_POST\_VALUE } from '../../constants/routes.ts';   const CreateAddModal: React.FC = () => {  const [isModalOpen, setIsModalOpen] = useState(false);  const [value, setValue] = useState('');  const [meterId, setMeterId] = useState('');   const json = JSON.stringify({  value: value,  date: new Date().getTime(),  meter:{  id: meterId,  }});  const handleFormSubmit = () => {  $api.post(API\_POST\_VALUE, json, {  headers: {  'Content-Type': 'application/json'  }  }  )  .then(async result => console.log(result.data))  .catch(error => {  console.error(error);  });  setIsModalOpen(false);  }   const showModal = () => {  setIsModalOpen(true);  };    const handleCancel = () => {  setIsModalOpen(false);  };   return (  <>  <Button type="primary" onClick={showModal}>  Добавить показание  </Button>   <Modal title='Добавить показание' open={isModalOpen} footer='' onCancel={handleCancel}>  <Form variant="filled" onFinish={handleFormSubmit} style={{ maxWidth: 600 }}>  <Form.Item label="Показание" name="Input" >  <Input onChange={e => setValue(e.target.value)}/>  </Form.Item>   <Form.Item  label="Id Счетчика"  name="InputNumber"  >  <Input onChange={e => setMeterId(e.target.value)}>  </Input>  </Form.Item>  <Form.Item>  <div style={{display:'flex',flexDirection:'row',justifyContent:'flex-end'}}>  <Button style={{marginRight:'10px'}} type='primary' htmlType='submit'>Добавить</Button>  <Button onClick={handleCancel}>Закрыть</Button>  </div>  </Form.Item>  </Form>  </Modal>  </>  ); };  export default CreateAddModal; |
| Содержимое файла CreateAddModal.tsx |

|  |
| --- |
| import React, { useState } from 'react'; import { Button, Form, Input, Modal } from 'antd'; import $api from '../../api.ts'; import { API\_PUT\_VALUE } from '../../constants/routes.ts';   interface CreateChangeInfoModalProps {  props: any; }  const CreateChangeInfoModal: React.FC<CreateChangeInfoModalProps> = ({props}) => {  const [isModalOpen, setIsModalOpen] = useState(false);  const [inputValue, setInputValue] = useState(props.valueHistory.value);  const values = { ...props }  const handleFormSubmit = () => {  $api.put(API\_PUT\_VALUE, { ...values.valueHistory, value: inputValue })  .then(async result => console.log(result.data))  .catch(error => {  console.error(error);  });  setIsModalOpen(false);  }   const showModal = () => {  setIsModalOpen(true);  console.log(props)  };    const handleCancel = () => {  setIsModalOpen(false);  };   return (  <>  <Button type="link" onClick={showModal}>  Изменить  </Button>   <Modal title={props.serialNumber} open={isModalOpen} footer='' onCancel={handleCancel}>  <Form variant="filled" onFinish={handleFormSubmit} style={{ maxWidth: 600 }}>  <Form.Item label="Предыдущее показание" name="Input" initialValue={props.valueHistory.value}>  <Input/>  </Form.Item>   <Form.Item  label="Текущее показание"  name="InputNumber"   >  <Input value={inputValue} onChange={(e) => setInputValue(e.target.value)}>  </Input>  </Form.Item>  <Form.Item  name="InputNumber"  >  <div style={{display:'flex',flexDirection:'row',justifyContent:'flex-end'}}>  <Button style={{marginRight:'10px'}} type='primary' htmlType='submit'>Изменить</Button>  <Button onClick={handleCancel}>Закрыть</Button>  </div>  </Form.Item>  </Form>  </Modal>  </>  ); };  export default CreateChangeInfoModal; |
| Содержимое файла CreateChangeInfoModal.tsx |

|  |
| --- |
| import { Select } from 'antd'; import { useEffect, useState } from 'react'; import { API\_FLAT\_GET\_ALL } from '../../constants/routes.ts'; import $api from '../../api.ts';   const SelectFlat = ({props}) =>{   const [flatData, setFlatData] = useState([])   useEffect(() => {  $api.get(API\_FLAT\_GET\_ALL,{}  )  .then(async result => {  return setFlatData(await result.data);  });  },[])   return(  <Select onChange={(props)}>  {flatData.map((item) => (  <Select.Option key = {item.id} value={item.name}>  {item.number}  </Select.Option>  ))}  </Select>  ) } export default SelectFlat; |
| Содержимое файла SelectFlat.tsx |

|  |
| --- |
| import { Select } from 'antd'; import { useEffect, useState } from 'react'; import { API\_STREET\_GET\_ALL } from '../../constants/routes.ts'; import $api from '../../api.ts';   const SelectStreet = ({props}) =>{  const [streetData, setStreetData] = useState([])  useEffect(() => {  $api.get(  API\_STREET\_GET\_ALL, {  }  )  .then(async result => {  setStreetData(await result.data);    });  },[])  return(<Select onChange={(props)}>  {streetData.map((item) => (  <Select.Option key = {item.id} value={item.name}>  {item.street + ',' + item.houseNumber}  </Select.Option>  ))}  </Select>)  }  export default SelectStreet; |
| Содержимое файла SelectStreet.tsx |

|  |
| --- |
| import { Select } from 'antd'; import { useEffect, useState } from 'react'; import { API\_TSJ\_GET\_ALL } from '../../constants/routes.ts'; import $api from '../../api.ts';   const SelectTsj = ({ props }) =>{  const [tsjData, setTsjData] = useState([])  useEffect(() => {  $api.get(  API\_TSJ\_GET\_ALL, {}  ).then(async result => {  return setTsjData(await result.data)  });  },[])  return(<Select onChange={( props )}>  {tsjData.map((item) => (  <Select.Option key = {item.id} value={item.name}>  {item.name}  </Select.Option>  ))}  </Select>)  }  export default SelectTsj; |
| Содержимое файла SelectTsj.tsx |

|  |
| --- |
| import { Select } from 'antd'; import { useEffect, useState } from 'react'; import { API\_USER\_GET\_ALL } from '../../constants/routes.ts'; import $api from '../../api.ts';   const SelectUsers = ({props}) =>{  const [userData, setUserData] = useState([])  useEffect(() => {  $api.get(  API\_USER\_GET\_ALL, {  }  ).then(async result => {  return setUserData(await result.data)  });  },[])  return(<Select onChange={( props )}>  {userData.map((item) => (  <Select.Option key = {item.id} value={item.login}>  {item.login}  </Select.Option>  ))}  </Select>)  }  export default SelectUsers; |
| Содержимое файла SelectUsers.tsx |

|  |
| --- |
| import { useEffect, useState } from 'react'; import {  API\_GET\_ALL\_VALUE\_HISTORY\_BY\_DATE\_AND\_STREET, } from '../../constants/routes.ts'; import { Table } from 'antd'; import $api from '../../api.ts'; import CreateChangeInfoModal from '../modals/CreateChangeInfoModal.tsx'; import DeleteButton from '../buttons/DeleteButton.tsx';   const TableValuesByDateAndStreet = ({ params }) => {  const [tableData, setTableData] = useState([]);  const [columns, setColumns] = useState([]);   useEffect(() => {  $api.get(API\_GET\_ALL\_VALUE\_HISTORY\_BY\_DATE\_AND\_STREET, {  params: {  addressId: params.addressId,  dateEnd: params.dateEnd ? new Date(params.dateEnd).toISOString().replace('T', ' ').slice(0, -5) : null,  dateStart: params.dateStart ? new Date(params.dateStart).toISOString().replace('T', ' ').slice(0, -5) : null,   }  })  .then(async result => {  const data = result.data || [];  const columns = [  { title: 'Имя', dataIndex: 'firstName' },  { title: 'Фамилия', dataIndex: 'lastName' },  { title: 'Дата', dataIndex: ['valueHistory', 'date'], render: (text) => new Date(text).toUTCString()},  { title: 'Показание', dataIndex: ['valueHistory', 'value'] },  {  render: (\_, record) => (  <>  <CreateChangeInfoModal props = {record}></CreateChangeInfoModal>  <DeleteButton props={record}></DeleteButton>  </>  ),  },  ];  setColumns(columns);  setTableData(data);  })  .catch((error) => {  console.error(error);  });  }  , [params.dateEnd, params.dateStart])    return (  <Table columns={columns} dataSource={tableData}></Table>  ) }  export default TableValuesByDateAndStreet; |
| Содержимое файла TabeValuesByDateAndStreet.tsx |

|  |
| --- |
| import { useEffect, useState } from 'react'; import { API\_GET\_ALL\_RENTERS } from '../../constants/routes.ts'; import { Table } from 'antd'; import $api from '../../api.ts';   const TableAllRentersAndFlats= () => {  const [tableData, setTableData] = useState([]);  const [columns, setColumns] = useState([]);   useEffect(() => {  $api.get(API\_GET\_ALL\_RENTERS)  .then(async result => {  const data = result.data || [];  const columns = [  { title: 'Имя', dataIndex: 'firstName' },  { title: 'Фамилия', dataIndex: 'lastName' },  { title: 'Город', dataIndex: 'city' },  { title: 'Улица', dataIndex: 'street' },  { title: 'Дом', dataIndex: 'houseNumber' },  { title: 'Номер телефона', dataIndex: 'phoneNumber' },  ];  setColumns(columns);  setTableData(data);  })  .catch((error) => {  console.error(error);  });  }  , [])    return (  <Table columns={columns} dataSource={tableData}> </Table>  ) }  export default TableAllRentersAndFlats; |
| Содержимое файла TableAllRenterAndFlats.tsx |

|  |
| --- |
| import { useEffect } from 'react'; import {  API\_GET\_TSJ\_REPORT, } from '../../constants/routes.ts'; import \* as FileSaver from 'file-saver'; import $api from '../../api.ts';  const handleDownload = (blob, filename) => {  FileSaver.saveAs(blob, filename); }; const TableTsjXlsx = () => {   useEffect(() => {  const getTableData = async () => {  try {  const response = await $api.get(API\_GET\_TSJ\_REPORT, { responseType: 'blob' });  handleDownload(response.data, 'report.xlsx');  } catch (error) {  console.error(error);  }  };  getTableData();  }  , )   }  export default TableTsjXlsx; |
| Содержимое файла TableTsjXlsx.tsx |

|  |
| --- |
| import { useEffect, useState } from 'react'; import { API\_GET\_ALL\_USER\_INFO\_BY\_FLAT\_ID} from '../../constants/routes.ts'; import axios from "axios"; import { Table } from 'antd'; import $api from '../../api.ts';   const TableUserInfoByFlatId= ({ params }) => {  const [tableData, setTableData] = useState([]);  const [columns, setColumns] = useState([]);   useEffect(() => {  $api.get(API\_GET\_ALL\_USER\_INFO\_BY\_FLAT\_ID, {  params: {  addressId: params.addressId,  flatId: params.flatId,  }  })  .then(async result => {  const data = result.data || [];  const columns = [  { title: 'Имя', dataIndex: 'firstName' },  { title: 'Фамилия', dataIndex: 'lastName' },  { title: 'Город', dataIndex: 'city' },  { title: 'Улица', dataIndex: 'street' },  { title: 'Дом', dataIndex: 'houseNumber' },  { title: 'Квартира', dataIndex: 'flatNumber' },  { title: 'Номер телефона', dataIndex: 'phoneNumber' },  ];  setColumns(columns);  setTableData(data);  })  .catch((error) => {  console.error(error);  });  }  , [params.addressId, params.flatId])    return (  <Table columns={columns} dataSource={tableData}> </Table>  ) }  export default TableUserInfoByFlatId; |
| Содержимое файла TableUserInfoByFlatId.tsx |

|  |
| --- |
| import { useEffect, useState } from 'react'; import { API\_GET\_ALL\_USER\_INFO\_BY\_TSJ\_ID } from '../../constants/routes.ts'; import { Table } from 'antd'; import $api from '../../api.ts';   const TableUserInfoByTsj= ({ params }) => {  const [tableData, setTableData] = useState([]);  const [columns, setColumns] = useState([]);   useEffect(() => {  $api.get(API\_GET\_ALL\_USER\_INFO\_BY\_TSJ\_ID, {  params: {  tsjId: params.tsjId,  }  })  .then(async result => {  const data = result.data || [];  const columns = [  { title: 'Имя', dataIndex: 'firstName' },  { title: 'Фамилия', dataIndex: 'lastName' },  { title: 'Город', dataIndex: 'city' },  { title: 'Улица', dataIndex: 'street' },  { title: 'Дом', dataIndex: 'houseNumber' },  { title: 'Номер телефона', dataIndex: 'phoneNumber' },  ];  setColumns(columns);  setTableData(data);  })  .catch((error) => {  console.error(error);  });  }  , [params.tsjId])    return (  <Table columns={columns} dataSource={tableData}> </Table>  ) }  export default TableUserInfoByTsj; |
| Содержимое файла TableUsersInfoByTsj.tsx |

|  |
| --- |
| import { useEffect, useState } from 'react'; import {  API\_GET\_ALL\_VALUE\_HISTORY\_BY\_DATE } from '../../constants/routes.ts'; import { Button, Table } from 'antd'; import $api from '../../api.ts'; import CreateChangeInfoModal from '../modals/CreateChangeInfoModal.tsx'; import DeleteButton from '../buttons/DeleteButton.tsx';   const TableValuesByDate = ({ params }) => {  const [tableData, setTableData] = useState([]);  const [columns, setColumns] = useState([]);   useEffect(() => {  $api.get(API\_GET\_ALL\_VALUE\_HISTORY\_BY\_DATE, {  params: {  dateEnd: params.dateEnd ? new Date(params.dateEnd).toISOString().replace('T', ' ').slice(0, -5) : null,  dateStart: params.dateStart ? new Date(params.dateStart).toISOString().replace('T', ' ').slice(0, -5) : null,  }  })  .then(async result => {  const data = result.data || [];  const columns = [  { title: 'Имя', dataIndex: 'firstName' },  { title: 'Фамилия', dataIndex: 'lastName' },  { title: 'Дата', dataIndex: ['valueHistory', 'date'], render: (text) => new Date(text).toUTCString()},  { title: 'Показание', dataIndex: ['valueHistory', 'value']},  {  render: (\_, record) => (  <>  <CreateChangeInfoModal props = {record}></CreateChangeInfoModal>  <DeleteButton props={record}></DeleteButton>  </>  ),  },  ];  setColumns(columns);  setTableData(data);  })  .catch((error) => {  console.error(error);  });  }  , [params.dateEnd, params.dateStart])    const handleDelete = (record) => {  // Handle delete action here  console.log('Delete record', record.valueHistory);  };    return (  <Table columns={columns} dataSource={tableData}> </Table>  ) }  export default TableValuesByDate; |
| Содержимое файла TableValuesByDate.tsx |

|  |
| --- |
| import { useEffect, useState } from 'react'; import {  API\_GET\_ALL\_VALUE\_HISTORY\_BY\_DATE\_AND\_LOGIN, } from '../../constants/routes.ts'; import { Button, Table } from 'antd'; import $api from '../../api.ts'; import CreateChangeInfoModal from '../modals/CreateChangeInfoModal.tsx'; import DeleteButton from '../buttons/DeleteButton.tsx';   const TableValuesByDateAndLogin = ({ params }) => {  const [tableData, setTableData] = useState([]);  const [columns, setColumns] = useState([]);   useEffect(() => {  $api.get(API\_GET\_ALL\_VALUE\_HISTORY\_BY\_DATE\_AND\_LOGIN, {  params: {  dateEnd: params.dateEnd ? new Date(params.dateEnd).toISOString().replace('T', ' ').slice(0, -5) : null,  dateStart: params.dateStart ? new Date(params.dateStart).toISOString().replace('T', ' ').slice(0, -5) : null,  userLogin: params.userLogin  }  })  .then(async result => {  const data = result.data || [];  const columns = [  { title: 'Имя', dataIndex: 'firstName' },  { title: 'Фамилия', dataIndex: 'lastName' },  { title: 'Дата', dataIndex: ['valueHistory', 'date'], render: (text) => new Date(text).toUTCString()},  { title: 'Показание', dataIndex: ['valueHistory', 'value'] },  {  render: (\_, record) => (  <>  <CreateChangeInfoModal props = {record}></CreateChangeInfoModal>  <DeleteButton props={record}></DeleteButton>  </>  ),  },  ];  setColumns(columns);  setTableData(data);  })  .catch((error) => {  console.error(error);  });  }  , [params.dateEnd, params.dateStart])    return (  <Table columns={columns} dataSource={tableData}> </Table>  ) }  export default TableValuesByDateAndLogin; |
| Содержимое файла TableValuesByDateAndLogin.tsx |

|  |
| --- |
| export const API\_AUTH = "/auth/authorize" export const API\_TSJ\_GET\_ALL = "/tsj/getAll";  export const API\_STREET\_GET\_ALL = "/address/getAll";  export const API\_USER\_GET\_ALL = "/user/getAll";  export const API\_FLAT\_GET\_ALL = "/flat/getAll";  export const API\_GET\_ALL\_VALUE\_HISTORY\_BY\_DATE = "/valueHistory/getAllValueHistoryByMonth";  export const API\_GET\_ALL\_VALUE\_HISTORY\_BY\_DATE\_AND\_LOGIN = "/valueHistory/getAllValueHistoryByMonthAndUserLogin";  export const API\_GET\_ALL\_VALUE\_HISTORY\_BY\_DATE\_AND\_STREET = "/valueHistory/getAllValueHistoryByMonthAndAddressId";  export const API\_GET\_TSJ\_REPORT = "/tsj/report";  export const API\_GET\_ALL\_USER\_INFO\_BY\_TSJ\_ID = "/user/getAllUsersInfoByTsjId";  export const API\_GET\_ALL\_USER\_INFO\_BY\_FLAT\_ID = "/user/getAllUsersInfoByFlatId";  export const API\_GET\_ALL\_RENTERS = "/user/getAllRentersAndFlat";  export const API\_PUT\_VALUE = "/valueHistory/update"; export const API\_DELETE\_VALUE = "/valueHistory/delete"; export const API\_POST\_VALUE = "/valueHistory/add"; |
| Содержимое файла routes.ts |

|  |
| --- |
| import { Space, Button, Form, DatePicker, MenuProps, Dropdown } from 'antd'; import './index.css'; import React, { useState } from 'react'; import SelectTsj from '../../components/selects/SelectTsj.tsx'; import SelectStreet from '../../components/selects/SelectStreet.tsx'; import { Dayjs } from 'dayjs'; import SelectUsers from '../../components/selects/SelectUsers.tsx'; import SelectFlat from '../../components/selects/SelectFlat.tsx'; import TableValuesByDate from '../../components/tables/TableValuesByDate.tsx'; import TableValuesByDateAndLogin from '../../components/tables/TableValuesByDateAndLogin.tsx'; import TableValuesByDateAndStreet from '../../components/tables/TabeValuesByDateAndStreet.tsx'; import TableTsjXlsx from '../../components/tables/TableTsjXlsx.tsx'; import TableUserInfoByTsj from '../../components/tables/TableUsersInfoByTsj.tsx'; import TableUserInfoByFlatId from '../../components/tables/TableUserInfoByFlatId.tsx'; import TableAllRenterAndFlats from '../../components/tables/TableAllRenterAndFlats.tsx'; import CreateAddModal from '../../components/modals/CreateAddModal.tsx';  const { RangePicker } = DatePicker;   const items: MenuProps['items'] = [  {  label: 'Все данные приборов учета за определенный период',  key: '1',  },  {  label: 'Данные приборов учета пользователя за определенный период',  key: '2',  },  {  label: 'Данные приборов учета конкретного дома за определенный период',  key: '3',  },  {  label: 'Все ТСЖ и их ресурсоснабжающие организации (отчет в xlsx)',  key: '4',  },  {  label: 'Информация о жильцах по ТСЖ',  key: '5',  },  {  label: 'Информация о жильцах по квартире',  key: '6',  },  {  label: 'Вывод всех арендаторов',  key: '7',  }, ];   const Admin: React.FC = () =>{   const [tsj, setTsj] = useState('');  const [user, setUser] = useState('');  const [selectedKey, setSelectedKey] = useState('');  const [selectedFlatKey, setSelectedFlatKey] = useState('');  const [selectedKeyAddress, setSelectedAddressKey] = useState('');  const [date, setDate] = useState([Dayjs]);  const [selectedTable, setSelectedTable] = useState(null);  const onChange = (value: any, option: any) => {  setTsj(value);  setSelectedKey(option.key);   };  const onChangeDate = (value: any) => {  setDate(value);  };  const onChangeUser = (value: any) => {  setUser(value)  };  const onChangeStreetKey = (value, option) => {  const selectedOptionKey = option.key;  setSelectedAddressKey(selectedOptionKey);  };   const onChangeFlatKey = (value, option) => {  const selectedOptionKey = option.key;  setSelectedFlatKey(selectedOptionKey);  };  const handleMenuClick: MenuProps['onClick'] = (e) => {   switch (e.key){  case '1':  if (!date || date.length !== 2) {  alert('Введите дату!');  }  else{  setSelectedTable('table1');  }  break;  case '2':  if (!date || date.length !== 2 || !user) {  alert('Введите дату и выберите пользователя!');  }  else {  setSelectedTable('table2');  }  break;  case '3':  if (!date || date.length !== 2 || !selectedKeyAddress) {  alert('Введите дату и выберите дом!');  }  else {  setSelectedTable('table3');  }  break;  case '4':  setSelectedTable('table4');  break;  case '5':  if (!tsj) {  alert('Выберите Тсж!');  }  else {  setSelectedTable('table5');  }  break;  case '6':  if (!selectedKeyAddress || !selectedFlatKey) {  alert('Выберите адрес и квартиру!');  }  else {  setSelectedTable('table6');  }  break;  case '7':  setSelectedTable('table7');  break;  }  };   const menuProps = {  items,  onClick: handleMenuClick,  };   return(  <>  <div className="form-filter-wrapper">  <Form variant="filled" >   <Form.Item label="Выберите ТСЖ" name="Select" >  <SelectTsj props={onChange}/>  </Form.Item>   <Form.Item label="Выберите дом" name="Select2">  <SelectStreet props={onChangeStreetKey}>  </SelectStreet>  </Form.Item>  <Form.Item label="Выберите квартиру" name="Select5" >  <SelectFlat props ={onChangeFlatKey}/>  </Form.Item>  <Form.Item label="Выберите пользователя" name="Select3" >  <SelectUsers props={onChangeUser}/>  </Form.Item>   <Form.Item  label="Выберите дату"  name="RangePicker"  >  <RangePicker onChange={onChangeDate} />  </Form.Item>   <Form.Item className='form-item-btn' wrapperCol={{ offset: 6, span: 14 }}>  <Dropdown menu={menuProps}>  <Button style={{backgroundColor: '#1890FF', color: 'white', width: '170px'}}>  <Space>  Отчет  </Space>  </Button>  </Dropdown>  </Form.Item>   <Form.Item>  <CreateAddModal></CreateAddModal>  </Form.Item>  </Form>  </div>  {selectedTable === 'table1' && <TableValuesByDate params={{ dateEnd: date[1], dateStart: date[0] }} />}  {selectedTable === 'table2' && <TableValuesByDateAndLogin params={{ dateEnd: date[1], dateStart: date[0], userLogin: user }} />}  {selectedTable === 'table3' && <TableValuesByDateAndStreet params={{ addressId: selectedKeyAddress, dateEnd: date[1], dateStart: date[0] }} />}  {selectedTable === 'table4' && <TableTsjXlsx/>}  {selectedTable === 'table5' && <TableUserInfoByTsj params={{tsjId: selectedKey}}/>}  {selectedTable === 'table6' && <TableUserInfoByFlatId params={{addressId: selectedKeyAddress, flatId: selectedFlatKey }}/>}  {selectedTable === 'table7' && <TableAllRenterAndFlats/>}  </>  )}    export default Admin; |
| Содержимое файла Admin.tsx |

|  |
| --- |
| import { API\_AUTH } from "../../constants/routes"; import { ICredentials } from "../../types/credentials.types";  export const submitCredentials = (credentials: ICredentials) => fetch(  API\_AUTH, {  method: 'POST',  headers: {  'Content-Type': 'application/json;charset=utf-8'  },  body: JSON.stringify(credentials)  } ).then(response => {  if (response.status === 401) {  alert("Введен неправильный логин или пароль");  throw new Error('Unauthorized');  }  const token = response.headers.get('Authorization');  if (token!= null){  localStorage.setItem('jwtToken', token);  return token;  } })  .catch((error) => {  console.error(error);  }); |
| Содержимое файла api.ts |

|  |
| --- |
| import React from 'react'; import { Button, Checkbox, Form, Input, Typography, ConfigProvider } from 'antd'; import './index.css'; import { useLoginStore } from './store.ts'; import { useNavigate } from 'react-router-dom';   const Login: React.FC = () => {  const { Title } = Typography;  const { sendCredentials } = useLoginStore();  const navigate = useNavigate();  const onFinish = (values: any) => {  const {login, password} = values;  sendCredentials({ login,password}, navigate);  };   const onFinishFailed = (errorInfo: any) => {  console.log('Failed:', errorInfo);  alert("Введен неправильный логин или пароль")  };    return (    <div className="center-form">  <ConfigProvider theme={{  token: {  colorPrimary: '#ec1c24',  }}}  >   <div className="shadow">  <Form   layout='vertical'  name="basic"  labelCol={{ span: 8 }}   wrapperCol={{ span: 18 }}  initialValues={{ remember: true }}   onFinish={onFinish}  onFinishFailed={onFinishFailed}   autoComplete="off"  >  <Title className='form-title'>  Авторизация  </Title>  <div className="form-wrapper">  <Form.Item   className='login-menu login-form-item'  name="login"  rules={[{ required: true, message: 'Пожалуйста введите логин!' }]}  >  <div className="shadow">  <Input placeholder='Введите логин' />  </div>  </Form.Item>  <Form.Item   className='login-form-item'  name="password"   rules={[{ required: true, message: 'Пожалуйста введите пароль!' }]}  >  <div className="shadow">  <Input.Password placeholder='Введите пароль' />  </div>  </Form.Item>  <Form.Item  className='login-form-item'  name='forgot'   wrapperCol={{span:4, offset:8}}  >  <Button type="link" htmlType='button'>  Забыли пароль?  </Button>  </Form.Item>  <Form.Item  className='checkbox login-form-item-checkbox'   name="remember"   valuePropName="checked"   wrapperCol={{ span: 8, offset: 3 }}>  <Checkbox>Запомнить меня</Checkbox>  </Form.Item>  <Form.Item  className='login-form-item login-form-item-btn'  wrapperCol={{ span: 18 }}>  <Button block type="primary" htmlType="submit">  Войти  </Button>  </Form.Item>  </div>  </Form>  </div>  </ConfigProvider>  </div>  ); }    export default Login; |
| Содержимое файла Login.tsx |

|  |
| --- |
| import { create } from 'zustand'; import { devtools } from 'zustand/middleware' import { submitCredentials } from './api'; import { ICredentials, ILoginStore } from '../../types/credentials.types'; import { NavigateFunction } from 'react-router-dom';  export const useLoginStore = create<ILoginStore>()(devtools((set) => ({  token: localStorage.getItem('jwtToken'),  isAuthenticated: false,  loading: false,  error: null,  sendCredentials: async (credentials: ICredentials,navigate: NavigateFunction) => {  try {  await submitCredentials(credentials);  const token = localStorage.getItem('jwtToken');  if(token != null ){  set({ token, isAuthenticated: true });  navigate('/admin')}  else {  set({ isAuthenticated: false })  }  } catch (error) {  if (error.response && error.response.status === 401) {  alert('Введен неправильный логин или пароль');  } else {  console.log(1)  }  }  },  logout: (navigate: NavigateFunction) => {  localStorage.removeItem('jwtToken');  set({ token: null, isAuthenticated: false });  navigate('/')  }, }))); |
| Содержимое файла store.ts |

|  |
| --- |
| export const NoMatch = () => <p>It is 404 page</p>; |
| Содержимое файла index.tsx |

|  |
| --- |
| import PasswordRecovery from "./PasswordRecovery";  export default PasswordRecovery; |
| Содержимое файла index.ts |

|  |
| --- |
| import React from 'react'; import { Button, Form, Input, Typography, ConfigProvider, message } from 'antd'; import './index.css';  const PasswordRecovery: React.FC = () => {  const { Title } = Typography;  const [messageApi] = message.useMessage();  const onFinish = (values:any) => {  console.log('Success:', values);    }  const success = () => {  messageApi.open({  type: 'success',  content: 'Запрос на восстановление отправлен',  });  };    return(  <div className="center-form">  <ConfigProvider theme={{  token: {  colorPrimary: '#ec1c24',  },  }}>   <div className="shadow">    <Form className ="ant-form-pass"layout='vertical' name="basic" labelCol={{ span: 8 }} wrapperCol={{ span: 18 }}  initialValues={{ remember: true }} onFinish={onFinish} autoComplete="off">  <Title>  Восстановление <br/>  пароля  </Title>  <div className="form-wrapper">  <Form.Item className='login-recovery-menu' name="username" rules={[{ required: true, message: 'Пожалуйста введите логин!' }]}>  <div className="shadow">  <Input placeholder='Введите логин' />  </div>  </Form.Item>  <Form.Item wrapperCol={{ span: 18 }}>  <Button block onClick={success} type="primary" htmlType="submit">  Отправить заявку  </Button>  </Form.Item>  </div>  </Form>  </div>  </ConfigProvider>  </div>  ); }  export default PasswordRecovery; |
| Содержимое файла PasswordRecovery.tsx |

|  |
| --- |
| import React from 'react'; import { Button, Flex, Layout } from 'antd'; import './index.css' import { useLoginStore } from '../../pages/Login/store.ts'; import { useNavigate } from 'react-router-dom';  interface IHeaderComponent {  header: React.ReactNode; }   const HeaderComponent: React.FC<IHeaderComponent> = ({ header }) => {  const { Header } = Layout;  const isAuthenticated = useLoginStore(state => state.isAuthenticated);  const logout = useLoginStore(state => state.logout)  const navigate = useNavigate();   const onChangeButton = () => {  logout(navigate);  }    return (  <Header className="header-container">  <div>{header}</div>  {isAuthenticated ? <div className='exit-button'>  <Flex justify='flex-end' align='center'><Button onClick={onChangeButton}>Выйти</Button></Flex>  </div> : <div></div>}  </Header>  ) };  export default HeaderComponent; |
| Содержимое файла Header.tsx |

|  |
| --- |
| import SiteLayout from './SiteLayout';  export default SiteLayout; |
| Содержимое файла index.ts |

|  |
| --- |
| import React from 'react'; import { Outlet } from "react-router-dom"; import { Layout, Space } from 'antd'; import HeaderComponent from '../Header/Header';  const { Content } = Layout;  const contentStyle: React.CSSProperties = {  height: 'calc(100vh - 64px)',  backgroundColor: '#fff', };  interface ISiteLayout {  header: React.ReactNode; }  const SiteLayout: React.FC<ISiteLayout> = ({ header }) => (  <Space direction="vertical" style={{ width: '100%' }} size={[0, 48]}>  <Layout>  <HeaderComponent header={header} />  <Content style={contentStyle}>  <Outlet />  </Content>  </Layout>  </Space> );  export default SiteLayout; |
| Содержимое файла SiteLayout.tsx |

|  |
| --- |
| import { Routes, Route } from "react-router-dom"; import { NoMatch } from "../../pages/NoMatch"; import PasswordRecovery from "../../pages/PasswordRecovery" import SiteLayout from "../Layout"; import { PROJECT\_NAME } from "../../constants/main"; import Login from "../../pages/Login"; // import { PrivateRouteGuestPage } from "../../hoc/PrivateRoute"; import Admin from "../../pages/Admin";    export const ProjectRoutes = () => {  return (  <Routes>  <Route path="/" element={<SiteLayout header={<span className="project-place">{PROJECT\_NAME}</span>} />}>  <Route index element={<Login />} />  {/\*<Route element={<PrivateRouteGuestPage />}>\*/}  {/\* <Route path="user">\*/}  <Route path='admin' element={<Admin />}> </Route>  {/\*</Route>\*/}  <Route path="\*" element={<NoMatch />} />  <Route path="passrecovery" element={<PasswordRecovery />} />  </Route>  {/\*</Route>\*/}  </Routes >  ) } |
| Содержимое файла Routes.tsx |

|  |
| --- |
| import { NavigateFunction } from 'react-router-dom';  export interface ICredentials {  login: string;  password: string; }  export interface ILoginStore {  token: string | null;  isAuthenticated: boolean;  loading: boolean;  error: unknown;  sendCredentials: (credentials: ICredentials,navigate: NavigateFunction) => void;  logout: (navigate: NavigateFunction) => void; } |
| Содержимое файла credentials.types.ts |

|  |
| --- |
| export interface TableDataType {  key: string;  name: string;  resident: "Жилец" | "Арендатор";  address: string;  flat: string;  hot\_meters: string;  cold\_meters: string; }  export interface ITsj {  key: string;  name: string; }  export interface IStreet extends ITsj{ } export interface UserResponse {  login: string;  firstName: string;  lastName: string;  surName: string;  phoneNumber: string;  serialNumber: string;  valueHistory: {  id: number;  value: string;  date: number;  meter: {  id: number;  serialNumber: string | null;  resource: string | null;  installationDate: string | null;  flat: string | null;  };  cost: number;  }; } |
| Содержимое файла data.types.ts |