ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ РЕСПУБЛИКИ ХАКАСИЯ

«ХАКАССКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

**ОТЧЕТ**

по производственной практике

по профессиональному модулю

**ПМ.01. РАЗРАБОТКА МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ КОМПЬЮТЕРНЫХ СИСТЕМ**

специальности 09.02.07 Информационные системы и программирование   
Квалификация Программист

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент гр. | ИС(ПРО)-41 |  |  |  | Салапонов А.А |
|  |  |  | *подпись* |  | *Фамилия И.О.* |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Руководитель практики  от предприятия |  |  |  |  |  |  | Костючек И.В |
|  | *оценка* |  | *дата* |  | *подпись* |  | *Фамилия И.О.* |
| М.П. |  |  |  |  |  |  |  |
| Руководитель практики  от ГБПОУ РХ ХПК |  |  |  |  |  |  | Черкашин Д.С |
|  | *оценка* |  | *дата* |  | *подпись* |  | *Фамилия И.О.* |

Абакан 2025 г.

**СОДЕРЖАНИЕ**

[1 Исследование информационной системы предприятия\организации 3](#_Toc193970591)

[2 Анализ предметной области 4](#_Toc193970592)

[3 Проектирование программного модуля 5](#_Toc193970593)

[3.1 Проектирование диаграмм 5](#_Toc193970594)

[3.2 Разработка макетов интерфейса программного модуля 17](#_Toc193970595)

[4 Разработка программного модуля 21](#_Toc193970596)

[4.1 Разработка интерфейсов программного модуля 21](#_Toc193970597)

[4.2 Разработка кода программного модуля 26](#_Toc193970598)

[5 Тестирование программного модуля 31](#_Toc193970599)

[5.1 Разработка тест-кейсов 31](#_Toc193970600)

[5.2 Модульное тестирование 38](#_Toc193970601)

[Приложения 42](#_Toc193970602)

[Приложение А – Спецификация требований 43](#_Toc193970603)

[Приложение Б – Репозиторий с документами в системе контроля версий 44](#_Toc193970604)

1. **Исследование информационной системы предприятия\организации**

Сабинский сельсовет, расположенный в Бейском районе Республики Хакасия, включает в себя четыре населённых пункта: село Сабинка (административный центр), деревни Новокурск, Калы и Красный Катамор. Общая численность населения составляет 2634 человека по данным на 2021 год.

Для обеспечения эффективного управления и предоставления услуг населению, администрация Сабинского сельсовета использует различные информационные системы и ресурсы.

Информационные системы, используемые в администрации:

Для ведения бухгалтерского учета в администрации Сабинского сельсовета используется программа 1С: Бухгалтерия. Эта система позволяет автоматизировать бухгалтерский и налоговый учет, формировать отчетность для налоговых органов, рассчитывать заработную плату сотрудников, вести кадровый учет, а также контролировать бюджетные расходы и доходы. Использование 1С облегчает документооборот и повышает прозрачность финансовых операций.

Для учета аренды земельных участков и сдачи жилья по социальному найму используется система БАРС. Она позволяет регистрировать договора аренды, отслеживать платежи, вести учет имущества и формировать отчеты по использованию муниципального жилого фонда. Данная система также обеспечивает своевременное обновление информации о состоянии жилого фонда, задолженностях по аренде и планируемых мерах по управлению муниципальной недвижимостью.

Дополнительно в администрации ведется Электронная похозяйственная книга, которая содержит сведения о домохозяйствах, земельных участках, имущественных правах и иных ресурсах, находящихся в собственности или аренде жителей сельсовета. Эта система упрощает учет земельных и имущественных отношений, способствует оперативному предоставлению справочной информации, а также помогает при планировании территориального развития.

Используемые информационные системы позволяют администрации сельсовета эффективно управлять финансовыми, имущественными и земельными вопросами, а также повышают прозрачность взаимодействия с гражданами и государственными структурами.

1. **Анализ предметной области**

Обращения граждан являются важной частью взаимодействия населения с органами местного самоуправления. Они представляют собой письменные запросы, жалобы, предложения или заявления, направленные в администрацию Сабинского сельсовета для решения различных вопросов.

**Основные категории обращений граждан**

* **Жилищно-коммунальные услуги** (ремонт дорог, уличное освещение, водоснабжение, вывоз мусора и др.).
* **Социальная поддержка** (назначение пособий, льгот, субсидий, материальная помощь).
* **Земельные и имущественные вопросы** (выделение земельных участков, аренда, приватизация).
* **Благоустройство и экология** (озеленение, создание зон отдыха, очистка территории).
* **Транспортное обеспечение** (работа общественного транспорта, обустройство остановок).
* **Образование и культура** (работа школ, детских садов, библиотек, домов культуры).
* **Здравоохранение** (доступность медицинских услуг, работа ФАПов).
* **Трудоустройство и экономика** (содействие в поиске работы, поддержка предпринимательства).

**Порядок обработки обращений:**

**Подача обращения**

* Граждане подают обращение в письменной форме.
* Важно указать ФИО заявителя, контактные данные и суть обращения.

**Регистрация обращения**

* Обращение фиксируется на бумажном носителе.
* Каждому обращению присваивается уникальный номер.

**Рассмотрение обращения**

* Определяется ответственный специалист, занимающийся данным вопросом.
* При необходимости запрашиваются дополнительные сведения.

**Принятие решения и ответ заявителю**

* Администрация принимает меры по решению вопроса или направляет обоснованный отказ.

1. **Проектирование программного модуля**
   1. **Проектирование диаграмм**

Для визуализации и структурирования проектных решений используются различные виды диаграмм. Они помогают лучше понять систему, выявить возможные ошибки еще на этапе проектирования и упростить дальнейшую разработку.

В данном разделе представлены основные диаграммы, используемые при проектировании системы:

ER-диаграмма (Entity-Relationship Diagram, диаграмма "сущность-связь") – позволяет определить структуру базы данных, выявить ключевые сущности, их атрибуты и взаимосвязи между ними. Данная диаграмма помогает спроектировать эффективную модель хранения данных. Диаграмма для данного программного модуля представлена на рисунке 3.1

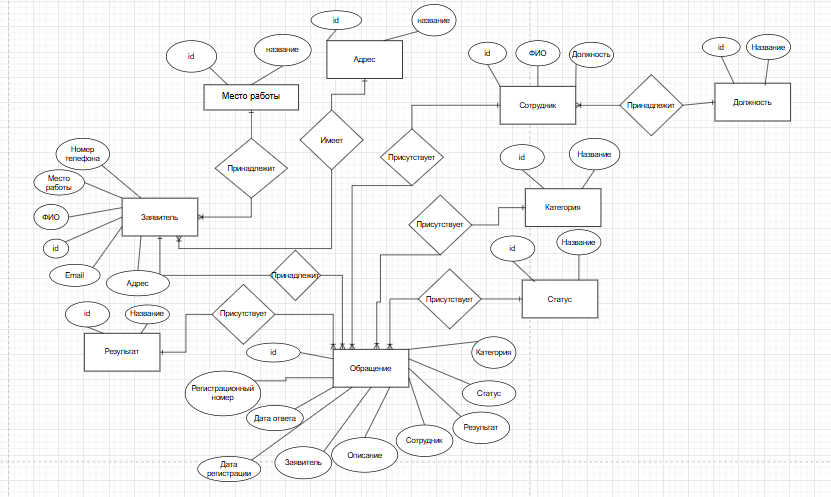


Рисунок 3.1 – Er – диаграмма.

**Классы и их атрибуты:**

**Заявитель (Applicant)**

**Атрибуты:**

* id (уникальный идентификатор)
* ФИО (полное имя)
* Номер телефона
* Email
* Место работы
* Адрес

**Связи:**

* Принадлежит **Месту работы**
* **Имеет Адрес**
* Может подавать **Обращения**

**Место работы (WorkPlace)**

**Атрибуты:**

* id
* Название

**Связи:**

* Может включать нескольких **Заявителей**

**Адрес (Address)**

**Атрибуты:**

* id
* Название

**Связи:**

* Может быть связан с несколькими **Заявителями**

**Обращение (Appeal)**

**Атрибуты:**

* id
* Регистрационный номер
* Заявитель
* Сотрудник
* Дата регистрации
* Дата ответа
* Описание
* Категория
* Статус
* Результат

**Связи:**

* Принадлежит **Заявителю**
* Может относиться к одной **Категории**
* Имеет один **Статус**
* Может быть обработано **Сотрудником**
* Связано с одним **Результатом**

**Категория (Category)**

**Атрибуты:**

* id
* Название

**Связи:**

* Может быть связана с несколькими **Обращениями**

**Статус (Status)**

**Атрибуты:**

* id
* Название

**Связи:**

* Может быть связан с несколькими **Обращениями**

**Результат (Result)**

**Атрибуты:**

* id
* Название

**Связи:**

* Может быть связан с несколькими **Обращениями**

**Сотрудник (Employee)**

**Атрибуты:**

* id
* ФИО

**Связи:**

* Может принадлежать **Должности**
* Может участвовать в обработке нескольких **Обращений**

**Должность (Position)**

**Атрибуты:**

* id
* Название

**Связи:**

* Может быть у нескольких **Сотрудников**

**Основные связи между классами:**

**Заявитель** → принадлежит → **Месту работы**

**Заявитель** → может подать → **Обращение**

**Обращение** → имеет → **Категорию, Статус, Результат**

**Обращение** → может быть обработано → **Сотрудником**

**Сотрудник** → имеет → **Должность**

IDEF0-диаграммы – используются для описания функциональных процессов системы.

Диаграмма нулевого уровня представляет систему в целом, включая ее входные и выходные данные, механизмы управления и исполнительные элементы. Диаграмма представлена на рисунке 3.2



Рисунок 3.2 – Диаграмма IDEF0 – нулевого уровня.

Процесс: A0 - Управление обращениями граждан

**Описание**: Система предназначена для управления обращениями граждан, включая их регистрацию, обработку, назначение ответственных и отслеживание статуса.

**Входы (Inputs)**:

* Заявления граждан (данные от заявителей: ФИО, телефон, email, описание обращения)
* Информация о местах работы и адресах
* Данные сотрудников (ФИО, должности)

**Выходы (Outputs):**

* Зарегистрированные обращения (с рег. номером, датой регистрации, статусом, результатом)

**Управление (Controls):**

* Правила обработки обращений (категории, статусы, результаты)
* Требования к заполнению полей (обязательные поля, форматы дат)
* Законодательные нормы (например, сроки ответа)

**Механизмы (Mechanisms):**

* Пользователь (оператор системы)
* База данных (PostgreSQL через SQLAlchemy)
* Интерфейс приложения (PySide6)

Диаграмма первого уровня детализирует отдельные функциональные блоки, обеспечивая более глубокое понимание выполняемых процессов. Диаграмма представлена на рисунке 3.3.

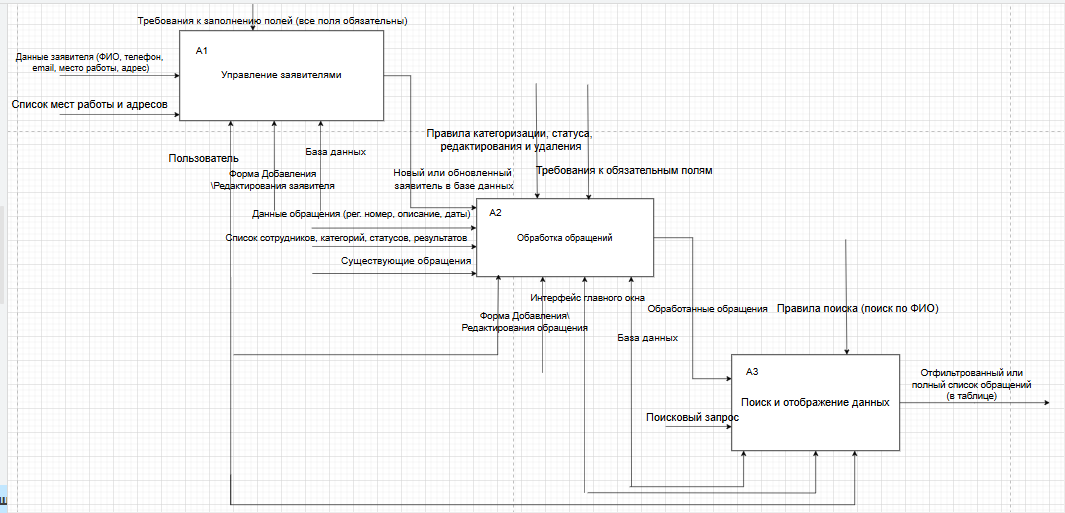


Рисунок 3.3 – Диаграмма IDEF0 – первого уровня.

A1 - Управление заявителями

Описание: Добавление и обновление данных заявителей.

Входы:

* Данные заявителя (ФИО, телефон, email, место работы, адрес) — от пользователя
* Список мест работы и адресов — из базы данных

Выходы:

* Новый/обновленный заявитель в базе данных

Управление:

* Требования к заполнению полей (все поля обязательны)

Механизмы:

* Пользователь
* Форма Добавления/Редактирования Заявителя
* База данных

Комментарий: Выход "Новый/обновленный заявитель" становится входным данным для A2, так как заявитель нужен для создания обращения.

A2 - Обработка обращений

Описание: Регистрация, редактирование и удаление обращений.

Входы:

* Новый/обновленный заявитель — из A1
* Данные обращения (рег. номер, описание, даты) — от пользователя
* Список сотрудников, категорий, статусов, результатов — из базы данных
* Существующие обращения — из базы данных (для редактирования/удаления)

Выходы:

* + Обработанные обращения (новые, обновленные или удаленные) в базе данных

Управление:

* Правила категоризации, статуса, редактирования и удаления
* Требования к обязательным полям

Механизмы:

* Пользователь
* Форма Добавления\Редактирования обращения
* Интерфейс главного окна
* База данных

Комментарий: Выход "Обработанные обращения" становится входным данным для A3, так как они нужны для отображения и поиска.

A3 - Поиск и отображение данных

Описание: Фильтрация и отображение списка обращений.

Входы:

* Обработанные обращения — из A2
* Поисковый запрос — от пользователя

Выходы:

* Отфильтрованный или полный список обращений (в таблице)

Управление:

* Правила поиска (поиск по ФИО)

Механизмы:

* Пользователь
* Интерфейс главного окна
* База данных

Комментарий: Этот выход является конечным результатом системы.

Диаграмма классов (Class Diagram) – ключевая диаграмма объектно-ориентированного проектирования, отображающая классы системы, их атрибуты, методы и взаимосвязи между ними. Она помогает формализовать структуру программы и определить основные компоненты системы. Диаграмма представлена на рисунках 3.4 – 3.5.

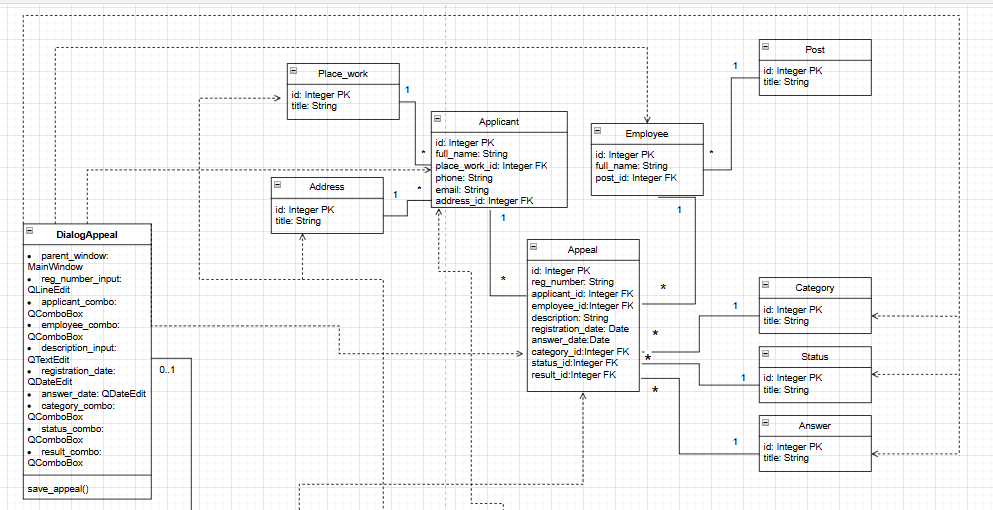


Рисунок 3.4 – Диаграмма классов.

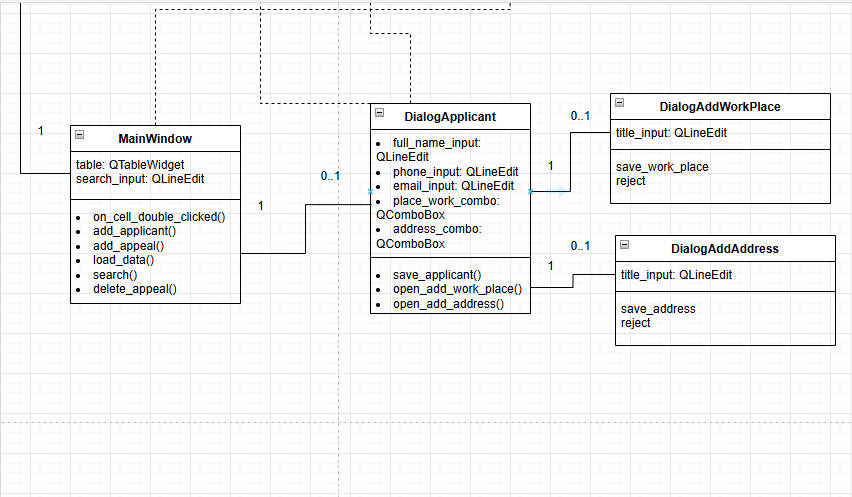


Рисунок 3.5 – Диаграмма классов

MainWindow

Атрибуты:

* table: QTableWidget — таблица для отображения данных.
* search\_input: QLineEdit — поле ввода для поиска.

Методы:

* on\_cell\_double\_clicked() — обработчик двойного клика по ячейке таблицы.
* add\_applicant() — добавление нового заявителя.
* add\_appeal() — добавление нового обращения.
* load\_data() — загрузка данных в таблицу.
* search() — поиск обращений.
* delete\_appeal() — удаление обращения.

Описание: Основное окно приложения, отображающее таблицу с обращениями и предоставляющее интерфейс для управления данными.

DialogApplicant

Атрибуты:

* full\_name\_input: QLineEdit — поле ввода ФИО.
* phone\_input: QLineEdit — поле ввода телефона.
* email\_input: QLineEdit — поле ввода email.
* place\_work\_combo: QComboBox — выпадающий список для выбора места работы.
* address\_combo: QComboBox — выпадающий список для выбора адреса.

Методы:

* save\_applicant() — сохранение данных заявителя.
* open\_add\_work\_place() — открытие диалога для добавления места работы.
* open\_add\_address() — открытие диалога для добавления адреса.

Описание: Диалоговое окно для добавления или редактирования заявителя.

DialogAppeal

Атрибуты:

* reg\_number\_input: QLineEdit — поле ввода регистрационного номера.
* applicant\_combo: QComboBox — выпадающий список для выбора заявителя.
* employee\_combo: QComboBox — выпадающий список для выбора сотрудника.
* description\_input: QTextEdit — поле ввода описания.
* registration\_date: QDateEdit — поле выбора даты регистрации.
* answer\_date: QDateEdit — поле выбора даты ответа.
* category\_combo: QComboBox — выпадающий список для выбора категории.
* status\_combo: QComboBox — выпадающий список для выбора статуса.
* result\_combo: QComboBox — выпадающий список для выбора результата.
* parent\_window: QMainWindow — ссылка на родительское окно.

Методы:

* save\_appeal() — сохранение данных обращения.

Описание: Диалоговое окно для добавления или редактирования обращения.

DialogAddWorkPlace

Атрибуты:

* title\_input: QLineEdit — поле ввода названия места работы.

Методы:

* save\_work\_place() — сохранение нового места работы.
* reject() — отмена действия и закрытие диалога.

Описание: Диалоговое окно для добавления нового места работы.

DialogAddAddress

Атрибуты:

* title\_input: QLineEdit — поле ввода адреса.

Методы:

* save\_address() — сохранение нового адреса.
* reject() — отмена действия и закрытие диалога.

Описание: Диалоговое окно для добавления нового адреса.

Place\_work

Атрибуты:

* id: Integer PK — первичный ключ.
* title: String — название места работы.

Описание: Класс модели для хранения данных о местах работы.

Address

Атрибуты:

* id: Integer PK — первичный ключ.
* address: String — адрес.

Описание: Класс модели для хранения данных об адресах.

Applicant

Атрибуты:

* id: Integer PK — первичный ключ.
* full\_name: String — ФИО заявителя.
* place\_work\_id: Integer FK — внешний ключ на Place\_work.
* phone: String — телефон.
* email: String — email.
* address\_id: Integer FK — внешний ключ на Address.

Описание: Класс модели для хранения данных о заявителях.

Employee

Атрибуты:

* id: Integer PK — первичный ключ.
* full\_name: String — ФИО сотрудника.
* post\_id: Integer FK — внешний ключ на Post.

Описание: Класс модели для хранения данных о сотрудниках.

Post

Атрибуты:

* id: Integer PK — первичный ключ.
* title: String — название должности.

Описание: Класс модели для хранения данных о должностях.

Appeal

Атрибуты:

* id: Integer PK — первичный ключ.
* reg\_number: String — регистрационный номер.
* applicant\_id: Integer FK — внешний ключ на Applicant.
* employee\_id: Integer FK — внешний ключ на Employee.
* description: String — описание обращения.
* registration\_date: Date — дата регистрации.
* answer\_date: Date — дата ответа.
* category\_id: Integer FK — внешний ключ на Category.
* status\_id: Integer FK — внешний ключ на Status.
* result\_id: Integer FK — внешний ключ на Answer.

Описание: Класс модели для хранения данных об обращениях.

Category

Атрибуты:

* id: Integer PK — первичный ключ.
* title: String — название категории.

Описание: Класс модели для хранения данных о категориях обращений.

Status

Атрибуты:

* id: Integer PK — первичный ключ.
* title: String — название статуса.

Описание: Класс модели для хранения данных о статусах обращений.

Answer

Атрибуты:

* id: Integer PK — первичный ключ.
* title: String — название результата.

Описание: Класс модели для хранения данных о результатах обращений.

Ассоциации:

MainWindow:

* 0..1 → 1 DialogApplicant: MainWindow может открывать одно окно DialogApplicant (через add\_applicant или on\_cell\_double\_clicked).
* 0..1 → 1 DialogAppeal: MainWindow может открывать одно окно DialogAppeal (через add\_appeal или on\_cell\_double\_clicked).

DialogApplicant:

* 0..1 → 1 DialogAddWorkPlace: DialogApplicant может открывать одно окно DialogAddWorkPlace (через open\_add\_work\_place).
* 0..1 → 1 DialogAddAddress: DialogApplicant может открывать одно окно DialogAddAddress (через open\_add\_address).

Applicant:

* \* → 1 Place\_work: Многие заявители могут быть связаны с одним местом работы.
* \* → 1 Address: Многие заявители могут быть связаны с одним адресом.
* 1 → \* Appeal: Один заявитель может иметь несколько обращений.

Employee:

* **\* → 1 Post**: Многие сотрудники могут иметь одну и ту же должность.
* 1 → \* Appeal: Один сотрудник может быть ответственным за несколько обращений.

Appeal:

* \* → 1 Applicant: Многие обращения могут быть связаны с одним заявителем (через applicant\_id).
* \* → 1 Employee: Многие обращения могут быть связаны с одним сотрудником (через employee\_id).
* \* → 1 Category: Многие обращения могут быть отнесены к одной категории (через category\_id).
* \* → 1 Status: Многие обращения могут иметь один и тот же статус (через status\_id).
* \* → 1 Answer: Многие обращения могут иметь одинаковый результат (через result\_id).

Зависимости (пунктирные стрелки ---->):

* DialogApplicant → Applicant, Place\_work, Address (для работы с данными).
* DialogAppeal → Appeal, Applicant, Employee, Category, Status, Answer (для работы с данными).
* MainWindow → Appeal (для отображения в таблице через load\_data).

Диаграмма вариантов использования (Use Case Diagram) – графически отображает взаимодействие пользователей (актеров) с системой, определяя основные функциональные сценарии работы. Позволяет определить, какие действия доступны различным категориям пользователей и какие процессы выполняются в системе. Диаграмма представлена на рисунке 3.5



Рисунок 3.5 – Диаграмма вариантов использования.

В данной диаграмме вариантов использования представлен 1 актёр и 9 вариантов использования.

Присутствуют 2 отношения включения между «Добавить заявителя» и «Добавить адрес», «Добавить заявителя» и «Добавить место работы

Также присутствуют 8 отношений расширения между «Просмотр списка обращений» и «Добавить заявителя», «Просмотр списка обращений» и «Добавить обращение», «Просмотр списка обращений» и «Редактировать заявителя», «Просмотр списка обращений» и «Редактировать обращение», «Просмотр списка обращений» и «Удалить обращение», «Просмотр списка обращений» и «Поиск обращений», «Редактировать заявителя» и «Добавить адрес», «Редактировать заявителя» и «Добавить место работы».

И 7 отношений ассоциации между «Пользователь» и «Добавить заявителя», «Редактировать заявителя», «Просмотр списка обращений», «Добавить обращение», «Поиск обращений», «Редактировать обращение», «Удалить обращение».

* 1. **Разработка макетов интерфейса программного модуля**

Проектирование пользовательского интерфейса (UI) является важным этапом разработки программного модуля, так как удобство взаимодействия пользователя с системой напрямую влияет на эффективность работы с приложением. Грамотно спроектированный интерфейс повышает понятность, скорость выполнения операций и общий пользовательский опыт.

На данном этапе разрабатываются макеты интерфейса, которые представляют собой графические прототипы окон приложения. Они позволяют заранее продумать расположение элементов управления, логику взаимодействия пользователя с программой и обеспечить удобство работы с системой.

Макет главного окна:

Основное окно приложения, отображающее список обращений граждан. Пользователь может просматривать, искать, добавлять, редактировать и удалять обращения. Также через это окно осуществляется переход к редактированию заявителей. Макет представлен на рисунке 3.6.

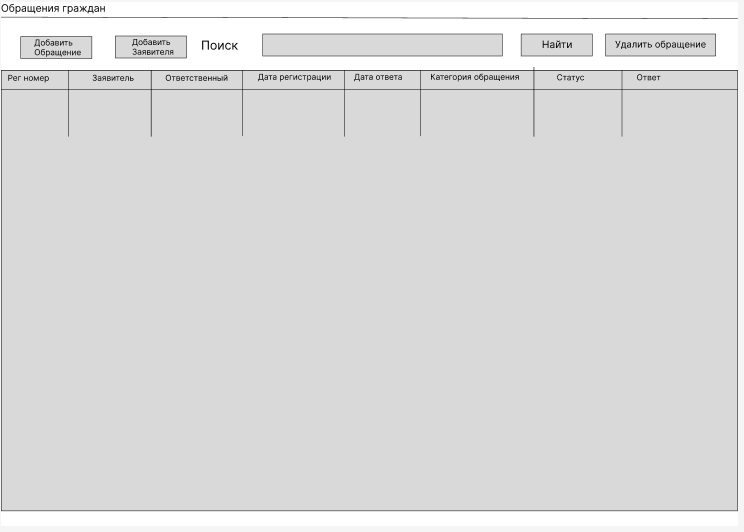


Рисунок 3.6 – Макет главного окна.

Форма добавления/редактирования заявителя:

Используется для ввода и редактирования информации о заявителе. Также предоставляет возможность добавлять новые адреса и места работы через дополнительные диалоги. Макет представлен на рисунке 3.7.

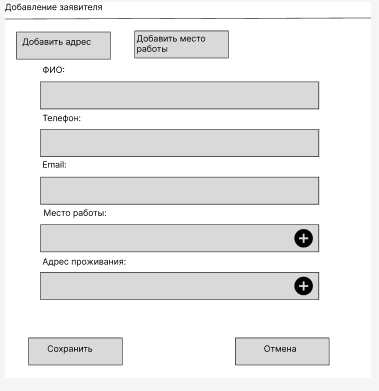


Рисунок 3.7 – Макет формы добавления/редактирования заявителя

Форма добавления/редактирования обращения:

Позволяет ввести данные об обращении гражданина: номер, заявителя, ответственного, описание, даты, категорию, статус и результат. Используется как при добавлении нового обращения, так и при редактировании существующего. Макет представлен на рисунке 3.8.

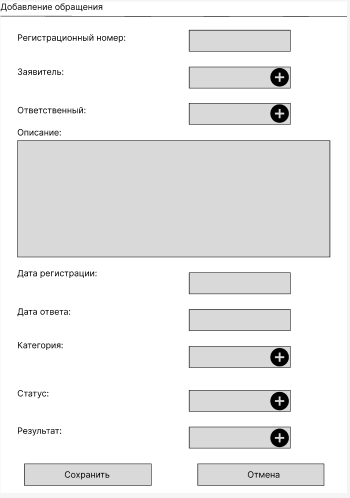


Рисунок 3.8 – Макет формы добавления\редактирования обращения.

Форма добавления места работы:

Простое окно для ввода названия нового места работы, которое затем можно выбрать в форме заявителя. Макет представлен на рисунке 3.9

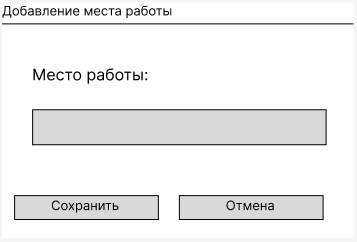


Рисунок 3.9 – Макет формы добавления места работы.

Форма добавления адреса проживания:

Простое окно для ввода нового адреса, который добавляется в список доступных адресов для заявителей. Макет представлен на рисунке 3.10.

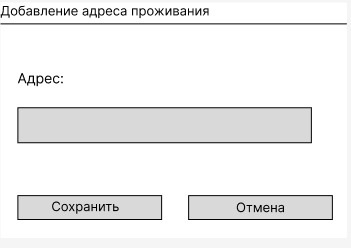


Рисунок 3.10 – Макет формы добавления адреса проживания

1. **Разработка программного модуля**
   1. **Разработка интерфейсов программного модуля**

Главного окно:

Элементы интерфейса:

* QLabel("Поиск"): Надпись перед полем поиска.
* QLineEdit (self.search\_input): Поле для ввода текста поиска. Используется для фильтрации обращений по имени заявителя или сотрудника.
* QPushButton("Найти"): Кнопка запуска поиска. Вызывает метод search.
* QPushButton("Добавить Заявителя"): Открывает DialogApplicant для создания нового заявителя.
* QPushButton("Добавить обращение"): Открывает DialogAppeal для создания нового обращения.
* QPushButton("Удалить обращение"): Удаляет выбранное обращение из таблицы после подтверждения.
* QTableWidget (self.table): Таблица с 8 столбцами: "Рег номер", "Заявитель", "Ответственный", "Дата регистрации", "Дата ответа", "Категория обращения", "Статус", "Результат". Двойной клик по ячейке открывает редактирование (заявителя или обращения).

Сообщения:

* QMessageBox.question: "Вы уверены, что хотите удалить обращение с номером {reg\_number}?" — при удалении обращения (с кнопками "Да" и "Нет").

PyQt элементы: QMainWindow, QPushButton, QLineEdit, QTableWidget, QLabel, QVBoxLayout, QHBoxLayout, QMessageBox.

Интерфейс главного окна представлен на рисунке 4.1.

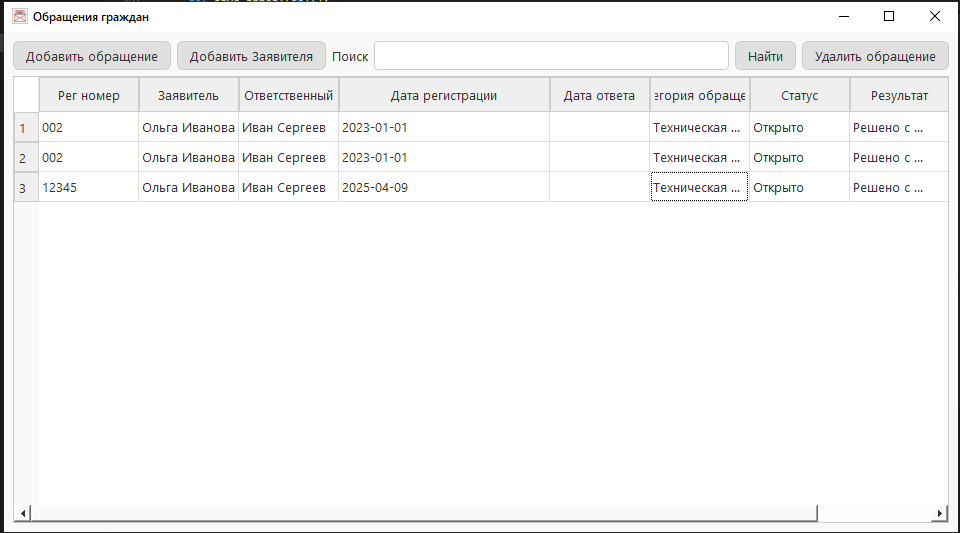


Рисунок 4.1 – Интерфейс главного окна

Форма добавления/редактирования заявителя:

Элементы интерфейса:

* QLineEdit (self.full\_name\_input): Поле для ввода ФИО. Проверка: не пустое.
* QLineEdit (self.phone\_input): Поле для телефона. Проверка: не пустое.
* QLineEdit (self.email\_input): Поле для email. Проверка: не пустое.
* QComboBox (self.place\_work\_combo): Выпадающий список мест работы. Данные из Place\_work. Проверка: выбор обязателен.
* QComboBox (self.address\_combo): Выпадающий список адресов. Данные из Address. Проверка: выбор обязателен.
* QToolBar: Панель с действиями:
* QAction("Добавить место работы"): Открывает DialogAddWorkPlace.
* QAction("Добавить адрес"): Открывает DialogAddAddress.
* QPushButton("Сохранить"): Сохраняет данные заявителя.
* QPushButton("Отмена"): Закрывает окно без сохранения.

Сообщения:

* QMessageBox.warning: "Заполните все поля!" — если хотя бы одно поле пустое.
* QMessageBox.information: "Заявитель добавлен!" или "Заявитель обновлен!" — при успешном сохранении.
* QMessageBox.critical: "Не удалось сохранить данные: ..." — при ошибке БД.

PyQt элементы: QDialog, QLineEdit, QComboBox, QToolBar, QAction, QPushButton, QFormLayout, QVBoxLayout, QMessageBox.

Интерфейс формы добавления/редактирования заявителя представлен на рисунке 4.2.

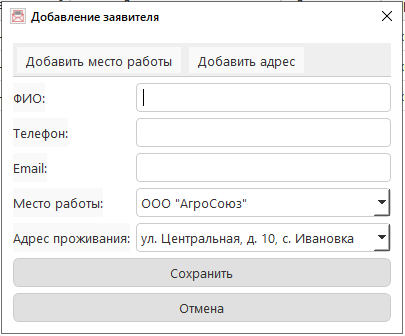


Рисунок 4.2 – Интерфейс формы добавления/редактирования заявителя.

Форма добавления/редактирования обращения:

Элементы интерфейса:

* QLineEdit (self.reg\_number\_input): Поле для рег. номера. Проверка: не пустое.
* QComboBox (self.applicant\_combo): Выбор заявителя из списка. Данные из Applicant.
* QComboBox (self.employee\_combo): Выбор ответственного сотрудника. Данные из Employee.
* QTextEdit (self.description\_input): Многострочное поле для описания обращения. Проверка: не пустое.
* QDateEdit (self.registration\_date): Поле выбора даты регистрации с календарем. По умолчанию текущая дата.
* QDateEdit (self.answer\_date): Поле выбора даты ответа. Может быть "Не указано" (1900-01-01).
* QComboBox (self.category\_combo): Выбор категории обращения. Данные из Category.
* QComboBox (self.status\_combo): Выбор статуса. Данные из Status.
* QComboBox (self.result\_combo): Выбор результата. Данные из Answer.
* QPushButton("Сохранить"): Сохраняет обращение.
* QPushButton("Отмена"): Закрывает окно без сохранения.

Сообщения:

* QMessageBox.warning: "Регистрационный номер обязателен!" — если рег. номер пустой.
* QMessageBox.warning: "Описание обязательно!" — если описание пустое.
* QMessageBox.information: "Обращение сохранено!" — при успешном сохранении.
* QMessageBox.critical: "Не удалось сохранить данные: ..." — при ошибке БД.

PyQt элементы: QDialog, QLineEdit, QComboBox, QTextEdit, QDateEdit, QPushButton, QFormLayout, QMessageBox.

Интерфейс формы добавления/редактирования обращения представлен на рисунке 4.3.

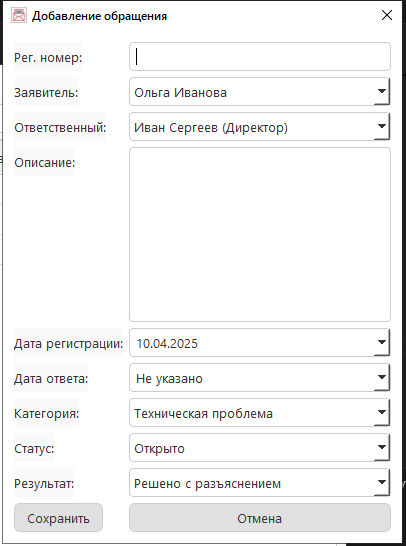


Рисунок 4.3 - Интерфейс формы добавления/редактирования обращения

Форма добавления места работы:

Элементы интерфейса:

* QLineEdit (self.title\_input): Поле ввода названия организации. Проверка: не пустое.
* QPushButton("Сохранить"): Сохраняет новое место работы в БД.
* QPushButton("Отмена"): Закрывает окно без сохранения.

Сообщения:

* QMessageBox.warning: "Название организации не может быть пустым!" — если поле ввода пустое.
* QMessageBox.critical: "Не удалось сохранить место работы: ..." — при ошибке БД.

PyQt элементы: QDialog, QLineEdit, QPushButton, QVBoxLayout, QLabel, QMessageBox.

Интерфейс формы добавления места работы представлен на рисунке 4.4.

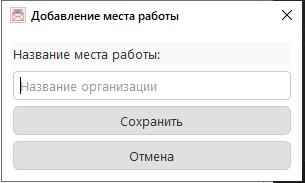


Рисунок 4.4 - Интерфейс формы добавления места работы

Форма добавления адреса проживания:

Элементы интерфейса:

* QLineEdit (self.title\_input): Поле ввода адреса. Проверка: не пустое.
* QPushButton("Сохранить"): Сохраняет новый адрес в БД.
* QPushButton("Отмена"): Закрывает окно без сохранения.

Сообщения:

* QMessageBox.warning: "Адрес не может быть пустым!" — если поле ввода пустое.
* QMessageBox.critical: "Не удалось сохранить адрес: ..." — при ошибке БД.

PyQt элементы: QDialog, QLineEdit, QPushButton, QVBoxLayout, QLabel, QMessageBox.

Интефейс формы добавления адреса представлен на рисунке 4.5.

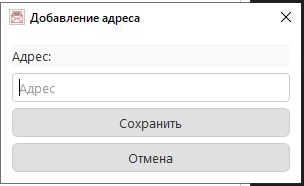


Рисунок 4.5 - Интерфейс формы добавления адреса.

* 1. **Разработка кода программного модуля**

Программное средство представляет собой десктопное приложение для управления обращениями граждан, разработанное с использованием библиотеки PySide6 (Qt для Python) и интегрированное с базой данных PostgreSQL через ORM SQLAlchemy. Оно предоставляет пользователям функционал для регистрации заявителей, добавления, редактирования и удаления обращений, а также просмотра и поиска данных в удобном табличном интерфейсе. Основная цель приложения — обеспечить эффективное управление обращениями граждан, включая учет их персональных данных, ответственных сотрудников, категорий обращений, статусов и результатов обработки.

On\_cell\_double\_clicked:

Описание: Метод on\_cell\_double\_clicked обрабатывает двойной клик по ячейке таблицы, открывая форму редактирования заявителя или обращения. Обновляет таблицу после изменений. Код представлен в листинге 1.

Листинг 1 – Обработчик двойного клика по ячейке таблицы.

def on\_cell\_double\_clicked(self, row, column):

session = Session() # Создание новой сессии для работы с базой данных

reg\_number = self.table.item(row, 0).text() # Извлечение регистрационного номера из первой ячейки строки

appeal = session.query(Appeal).filter\_by(reg\_number=reg\_number).first() # Поиск обращения в базе по рег. номеру

if column == 1: # Проверка, кликнули ли на столбец "Заявитель" (индекс 1)

applicant = appeal.applicant # Получение объекта заявителя, связанного с обращением

dialog = DialogApplicant(self, applicant) # Создание диалогового окна для редактирования заявителя

Продолжение листинга 1

if dialog.exec(): # Запуск диалога и проверка, был ли он принят (нажата "Сохранить")

self.load\_data() # Обновление данных таблицы после редактирования

elif appeal: # Если кликнули на другую ячейку и обращение существует

dialog = DialogAppeal(self, appeal) # Создание диалогового окна для редактирования обращения

if dialog.exec(): # Запуск диалога и проверка, был ли он принят

self.load\_data() # Обновление данных таблицы после редактирования

session.close() # Закрытие сессии базы данных

Save\_appeal:

Описание: Метод save\_appeal сохраняет данные обращения в базу, проверяя обязательные поля. Обновляет таблицу после успешного сохранения. Код представлен в листинге 2.

Листинг 2 – Сохранение данных обращения.

def save\_appeal(self):

session = Session() # Создание новой сессии для работы с базой данных

try: # Начало блока обработки исключений

if not self.reg\_number\_input.text().strip(): # Проверка, заполнен ли регистрационный номер

raise ValueError("Регистрационный номер обязателен!") # Выброс исключения, если поле пустое

if not self.description\_input.toPlainText().strip(): # Проверка, заполнено ли описание

raise ValueError("Описание обязательно!") # Выброс исключения, если поле пустое

answer\_date = None # Инициализация переменной для даты ответа как None

selected\_date = self.answer\_date.date() # Получение выбранной даты ответа из виджета

if selected\_date != QDate(1900, 1, 1): # Проверка, отличается ли дата от значения по умолчанию

answer\_date = selected\_date.toPython() # Преобразование даты в формат Python, если выбрана

if self.appeal: # Если редактируется существующее обращение

self.appeal = session.merge(self.appeal) # Обновление объекта обращения в сессии

self.appeal.reg\_number = self.reg\_number\_input.text().strip() # Обновление рег. номера

self.appeal.applicant\_id = self.applicant\_combo.currentData() # Обновление ID заявителя

self.appeal.employee\_id = self.employee\_combo.currentData() # Обновление ID сотрудника

Продолжение листинга 2

self.appeal.description = self.description\_input.toPlainText().strip() # Обновление описания

self.appeal.registration\_date = self.registration\_date.date().toPython() # Обновление даты регистрации

self.appeal.answer\_date = answer\_date # Обновление даты ответа (или None)

self.appeal.category\_id = self.category\_combo.currentData() # Обновление ID категории

self.appeal.status\_id = self.status\_combo.currentData() # Обновление ID статуса

self.appeal.result\_id = self.result\_combo.currentData() # Обновление ID результата

else: # Если создается новое обращение

new\_appeal = Appeal( # Создание нового объекта Appeal

reg\_number=self.reg\_number\_input.text().strip(), # Установка рег. номера

applicant\_id=self.applicant\_combo.currentData(), # Установка ID заявителя

employee\_id=self.employee\_combo.currentData(), # Установка ID сотрудника

description=self.description\_input.toPlainText().strip(), # Установка описания

registration\_date=self.registration\_date.date().toPython(), # Установка даты регистрации

answer\_date=answer\_date, # Установка даты ответа (или None)

category\_id=self.category\_combo.currentData(), # Установка ID категории

status\_id=self.status\_combo.currentData(), # Установка ID статуса

result\_id=self.result\_combo.currentData() # Установка ID результата

)

session.add(new\_appeal) # Добавление нового обращения в сессию

session.commit() # Фиксация изменений в базе данных

QMessageBox.information(self, "Успех", "Обращение сохранено!") # Отображение сообщения об успехе

if self.parent\_window: # Проверка наличия родительского окна

self.parent\_window.load\_data() # Обновление таблицы в главном окне

self.accept() # Закрытие диалогового окна с подтверждением

except ValueError as ve: # Обработка исключений ValueError (ошибки валидации)

QMessageBox.warning(self, "Ошибка", str(ve)) # Отображение предупреждения с текстом ошибки

Продолжение листинга 2

except SQLAlchemyError as e: # Обработка исключений SQLAlchemy (ошибки базы данных)

session.rollback() # Откат изменений при ошибке

QMessageBox.critical(self, "Ошибка", f"Не удалось сохранить данные: {str(e)}") # Сообщение об ошибке

finally: # Блок, выполняющийся в любом случае

session.close() # Закрытие сессии базы данных

Save\_applicant:

Описание: Метод save\_applicant сохраняет данные заявителя в базу, проверяя заполненность полей. Обновляет таблицу после сохранения. Код представлен в листинге 3.

Листинг 3 – Сохранение данных заявителя.

def save\_applicant(self):

full\_name = self.full\_name\_input.text().strip() # Извлечение и очистка ФИО из поля ввода

phone = self.phone\_input.text().strip() # Извлечение и очистка телефона из поля ввода

email = self.email\_input.text().strip() # Извлечение и очистка email из поля ввода

place\_id = self.place\_work\_combo.currentData() # Получение ID места работы из выпадающего списка

address\_id = self.address\_combo.currentData() # Получение ID адреса из выпадающего списка

if not full\_name or not phone or not email or place\_id is None or address\_id is None: # Проверка заполненности всех полей

QMessageBox.warning(self, "Ошибка", "Заполните все поля!") # Предупреждение, если есть пустые поля

return # Выход из метода при ошибке валидации

session = Session() # Создание новой сессии для работы с базой данных

try: # Начало блока обработки исключений

if self.applicant: # Если редактируется существующий заявитель

self.applicant = session.merge(self.applicant) # Обновление объекта заявителя в сессии

self.applicant.full\_name = full\_name # Обновление ФИО

self.applicant.phone = phone # Обновление телефона

self.applicant.email = email # Обновление email

self.applicant.place\_work\_id = place\_id # Обновление ID места работы

self.applicant.address\_id = address\_id # Обновление ID адреса

message = "Заявитель обновлен!" # Сообщение для случая редактирования

else: # Если создается новый заявитель

new\_applicant = Applicant( # Создание нового объекта Applicant

full\_name=full\_name, # Установка ФИО

phone=phone, # Установка телефона

email=email, # Установка email

Продолжение листинга 3

place\_work\_id=place\_id, # Установка ID места работы

address\_id=address\_id # Установка ID адреса

)

session.add(new\_applicant) # Добавление нового заявителя в сессию

message = "Заявитель добавлен!" # Сообщение для случая добавления

session.commit() # Фиксация изменений в базе данных

QMessageBox.information(self, "Успех", message) # Отображение сообщения об успехе

if self.parent\_window: # Проверка наличия родительского окна

self.parent\_window.load\_data() # Обновление таблицы в главном окне

self.accept() # Закрытие диалогового окна с подтверждением

except SQLAlchemyError as e: # Обработка исключений SQLAlchemy (ошибки базы данных)

session.rollback() # Откат изменений при ошибке

QMessageBox.critical(self, "Ошибка", f"Не удалось сохранить данные: {str(e)}") # Сообщение об ошибке

finally: # Блок, выполняющийся в любом случае

session.close() # Закрытие сессии базы данных

1. **Тестирование программного модуля**
   1. **Разработка тест-кейсов**

Тест-кейс (или тестовый сценарий) — это документ, описывающий набор условий, действий и ожидаемых результатов, которые определяют, выполнена ли функциональность программного обеспечения корректно. Он позволяет тестировщикам последовательно проверять определённые аспекты приложения, обеспечивая структурированный подход к тестированию.

Тест-кейс 1: Добавление нового заявителя

Описание: Этот тест проверяет функциональность добавления нового заявителя через форму DialogApplicant. Успешное добавление заявителя является ключевым шагом для последующего создания обращений. Тест подтверждает, что при вводе корректных данных заявитель сохраняется в базе данных, а таблица в главном окне обновляется. Тест – кейс представлен на рисунке 5.1.

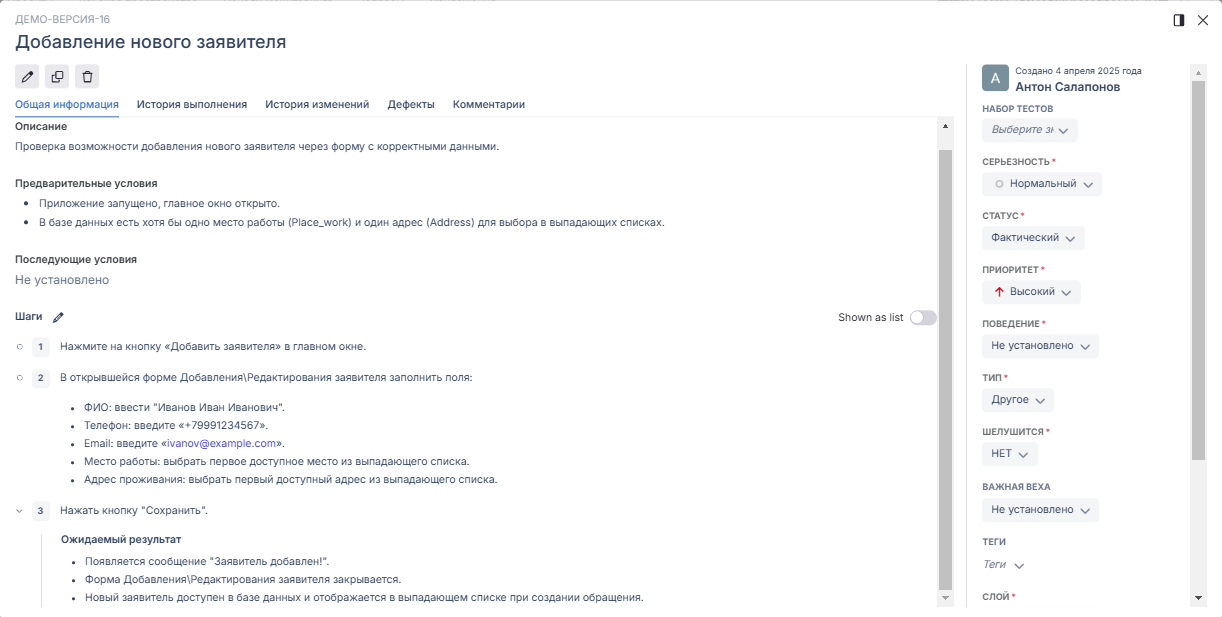


Рисунок 5.1 – Тест-кейс 1: Добавление нового заявителя

Фактический результат тест – кейса 1 представлен на рисунках 5.2 -5.3.

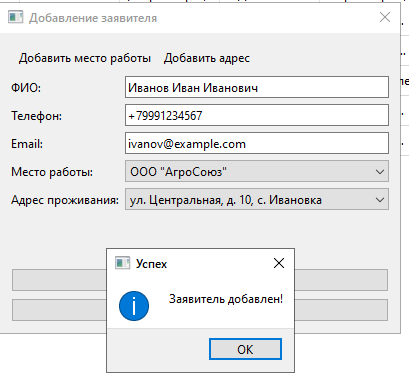


Рисунок 5.2 – Фактический результат тест – кейса 1.

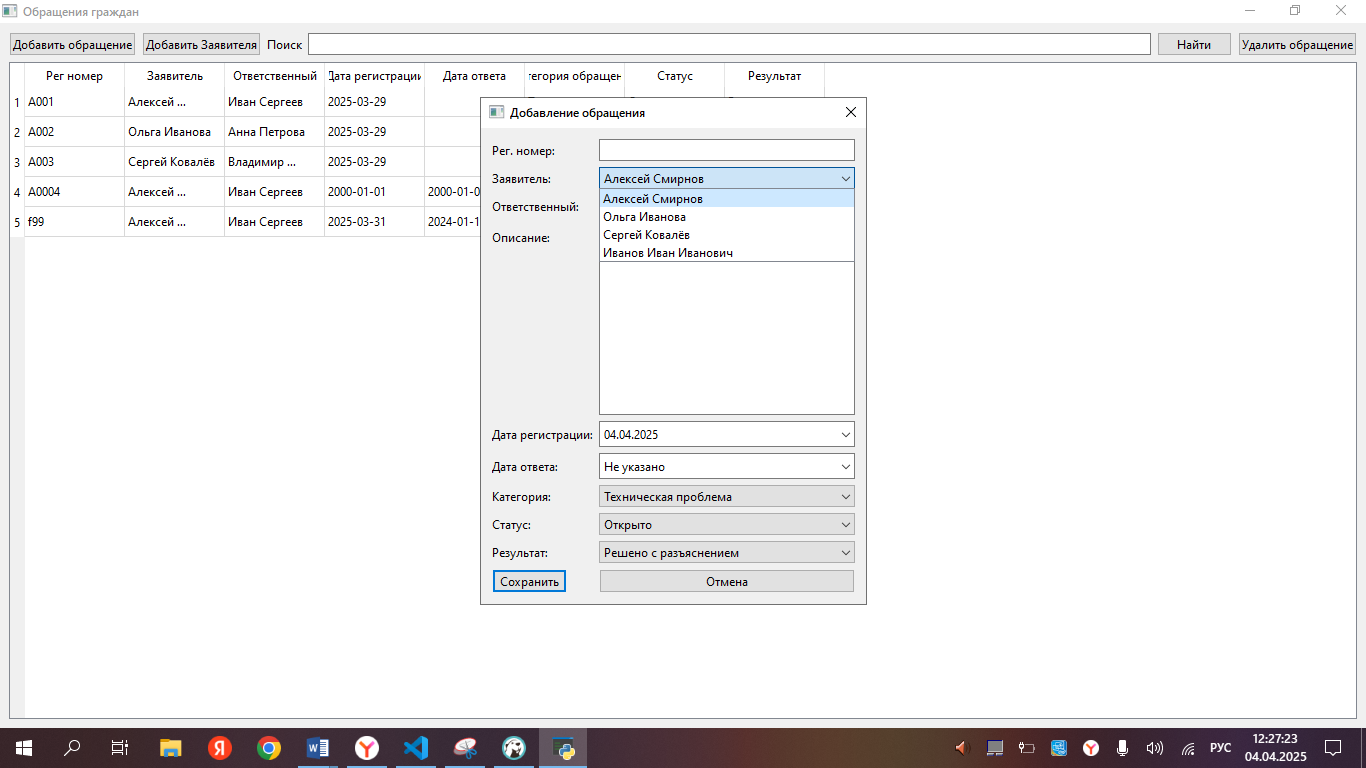


Рисунок 5.3 – Фактический результат тест – кейса 1.

Статус – Пройден успешно.

Тест-кейс №2: Добавление нового обращения

Описание: этот тест проверяет функциональность добавления нового обращения через форму DialogAppeal. Успешное добавление обращения является основной функцией системы. Тест подтверждает, что при вводе корректных данных обращение сохраняется, а таблица в главном окне обновляется. Тест – кейс представлен на рисунке 5.4.

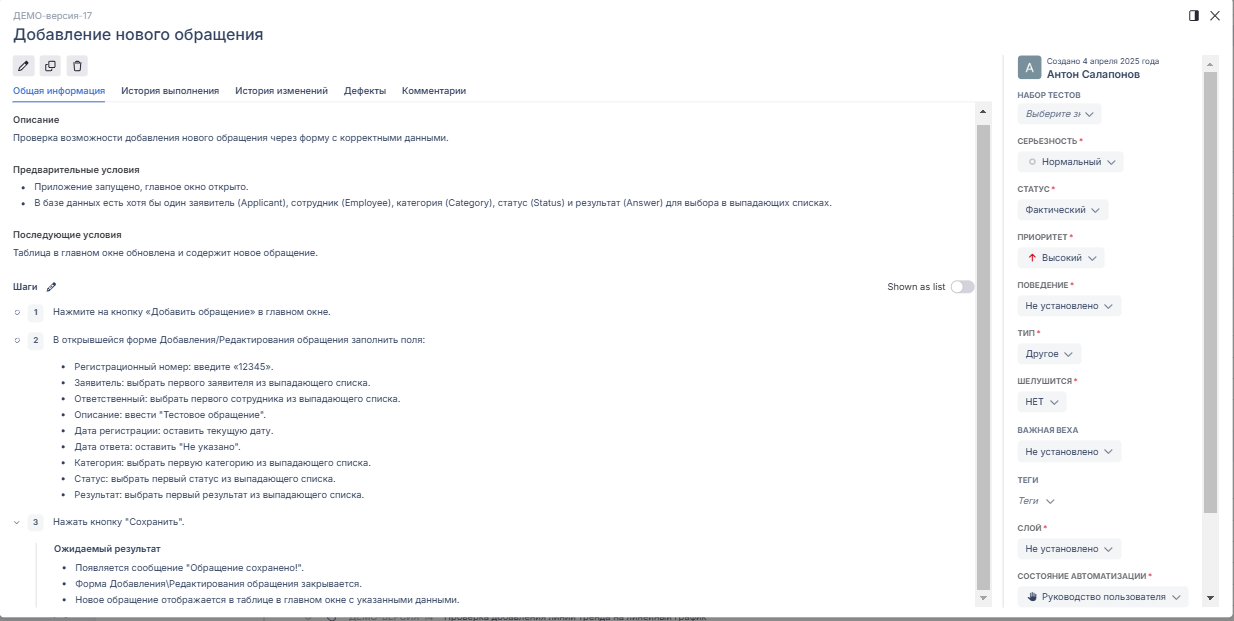


Рисунок 5.4 – Тест-кейс №2: Добавление нового обращения

Фактический результат тест – кейса 2 представлен на рисунках 5.5 – 5.6.

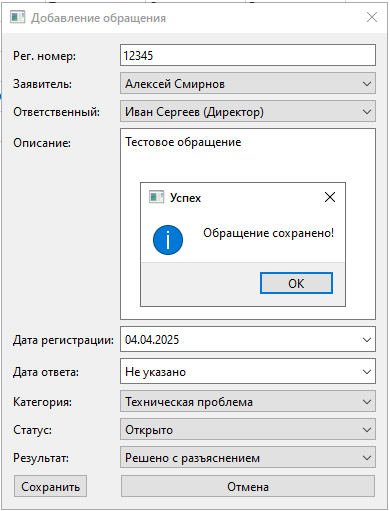


Рисунок 5.5 – Фактический результат тест – кейса 2.

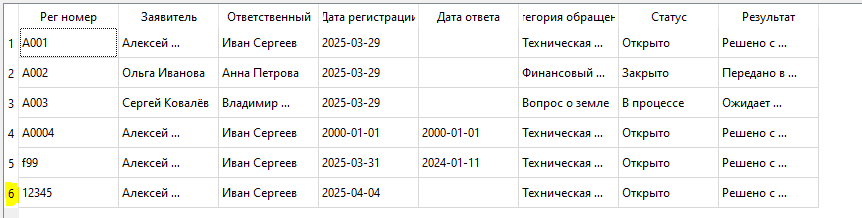


Рисунок 5.6 – Фактический результат тест – кейса 2.

Статус – Пройден успешно.

Тест-кейс №3: Редактирование заявителя через двойной клик.

Описание: Этот тест проверяет функциональность редактирования заявителя через двойной клик на его имя в таблице. Редактирование заявителя важно для обновления данных, которые могут быть использованы в обращениях. Тест подтверждает, что изменения сохраняются, а таблица обновляется. Тест – кейс представлен на рисунке 5.7.

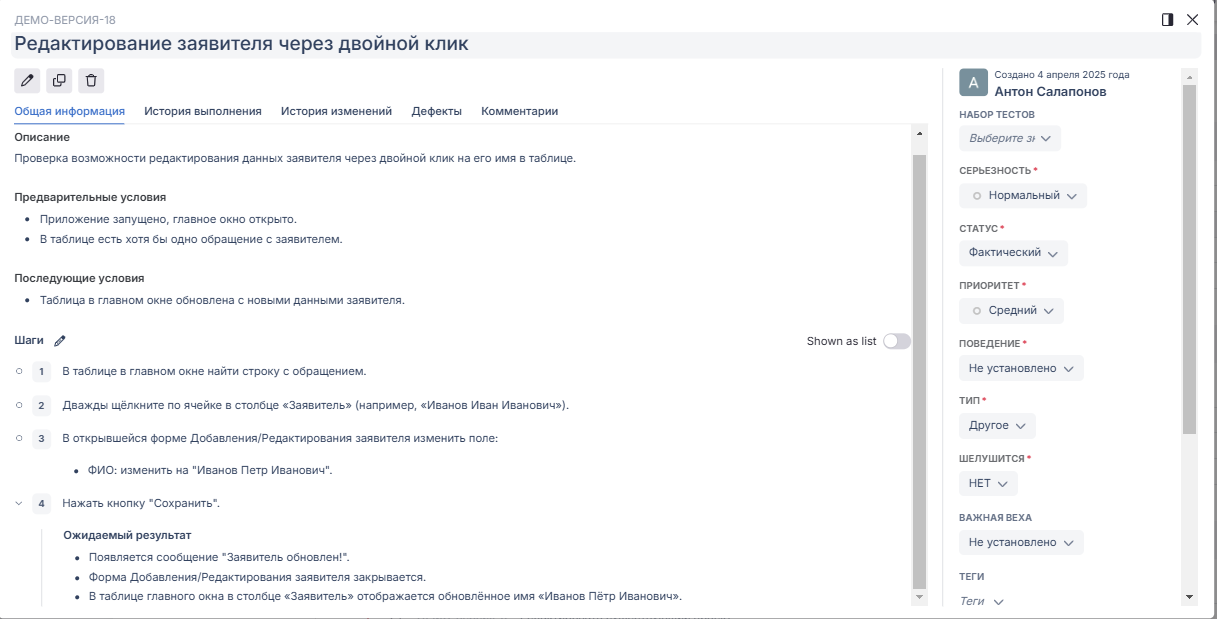


Рисунок 5.7 – Тест-кейс №3: Редактирование заявителя через двойной клик

Фактический результат тест – кейса 3 представлен на рисунках 5.8 – 5.10.

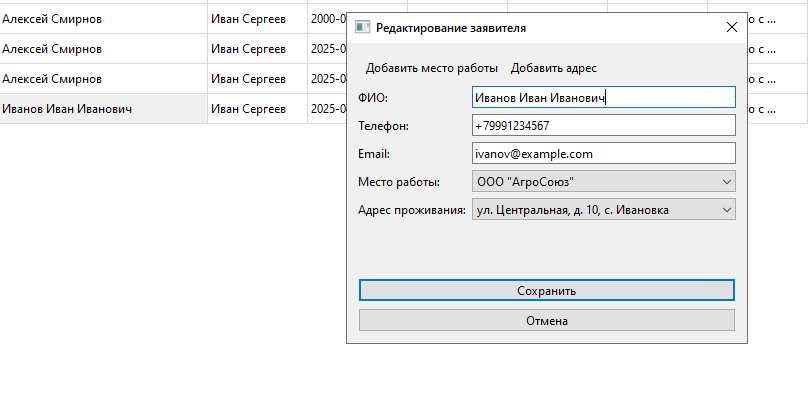


Рисунок 5.8 – Фактический результат тест – кейса 3.

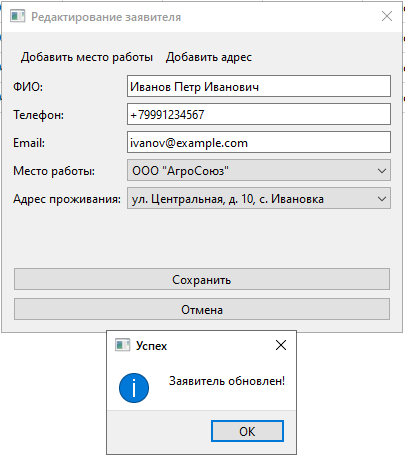


Рисунок 5.9 – Фактический результат тест – кейса 3.

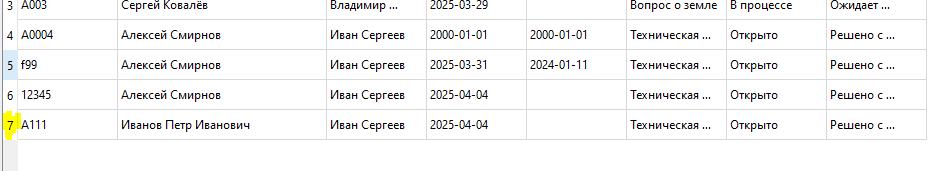


Рисунок 5.10 – Фактический результат тест – кейса 3.

Статус – Пройден успешно.

Тест-кейс №4: Удаление обращения

Описание: Этот тест проверяет функциональность удаления обращения из таблицы. Удаление обращения важно для управления списком обращений. Тест подтверждает, что обращение удаляется из базы данных, а таблица обновляется. Тест – кейс представлен на рисунке 5.11

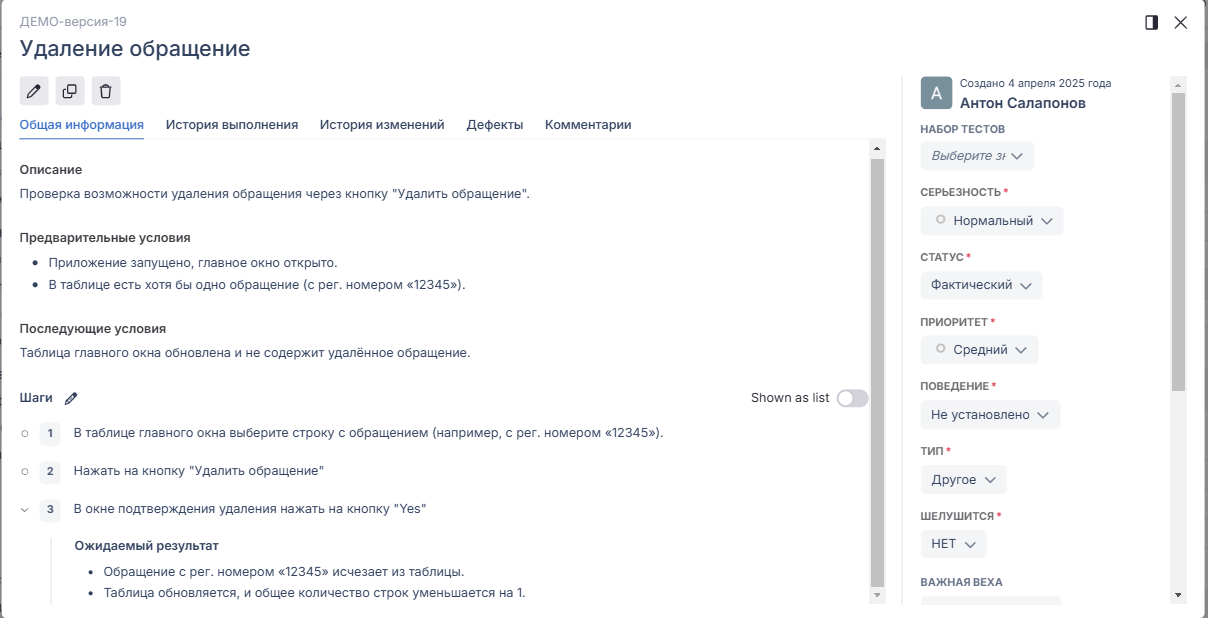


Рисунок 5.11 – Тест-кейс №4: Удаление обращения

Фактический результат тест – кейса 4 представлен на рисунке 5.12 – 5.13.

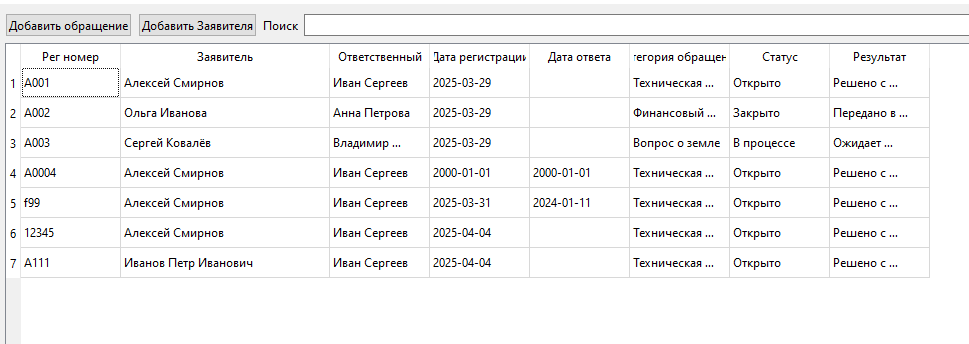


Рисунок 5.12 – Фактический результат тест – кейса 4.

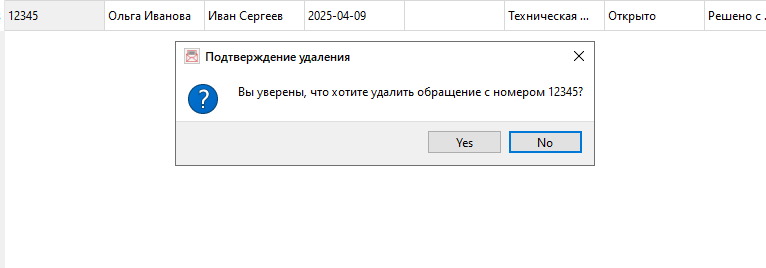


Рисунок 5.13 – Фактический результат тест – кейса 4.

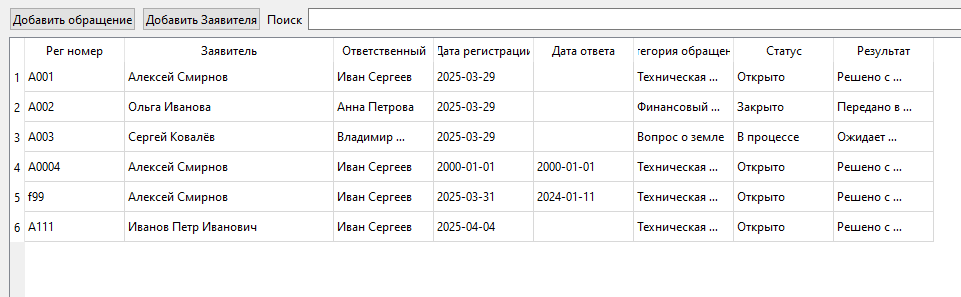


Рисунок 5.14 – Фактический результат тест – кейса 4.

Статус – Пройден успешно.

Тест-кейс №5: Поиск обращений по имени заявителя

Описание: Этот тест проверяет функциональность поиска обращений по имени заявителя или сотрудника. Поиск является важной функцией для быстрого доступа к нужным данным. Тест подтверждает, что при вводе текста в поле поиска таблица фильтруется корректно. Тест – кейс представлен на рисунке 5 .15.

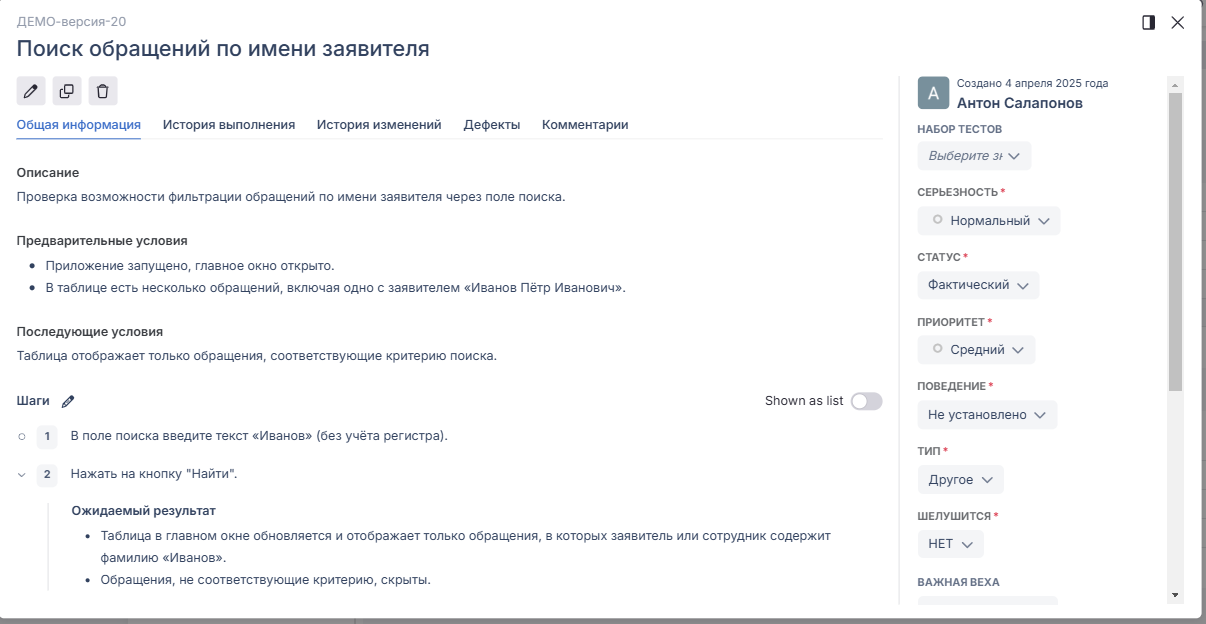


Рисунок 5.14 – Тест-кейс №5: Поиск обращений по имени заявителя

Фактический результат тест – кейса 5 представлен на рисунке 5.15.

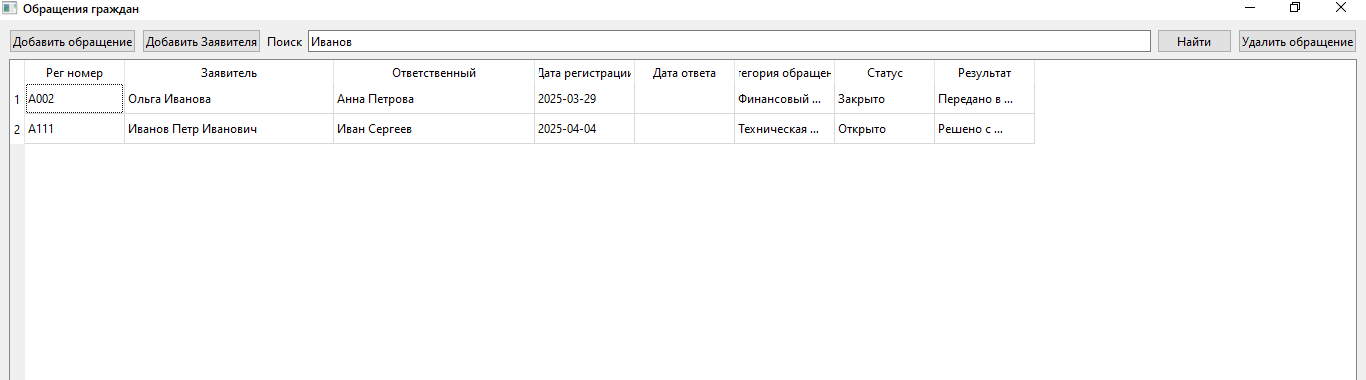


Рисунок 5.15 – Фактический результат тест – кейса 5.

Статус – Пройден успешно.

* 1. **Модульное тестирование**

Модульное тестирование (Unit Testing) — это процесс тестирования отдельных модулей или компонентов программы с целью проверки их корректности. Под "модулем" обычно понимается минимальная изолированная часть кода, которая выполняет определенную функцию, например, метод, функция или класс. Основная идея модульного тестирования заключается в том, чтобы убедиться, что каждая единица кода работает так, как ожидается, независимо от других частей программы.

Модульное тестирование приложения "Обращения граждан" проводилось с использованием библиотеки pytest и инструментов mocking (unittest.mock) для проверки корректности работы основных классов и методов. Тестирование охватывает ключевые функциональности основного окна приложения (MainWindow), а также диалоговых окон для работы с заявителями (DialogApplicant), обращениями (DialogAppeal), местами работы (DialogAddWorkPlace) и адресами (DialogAddAddress).

Цели тестирования

* Проверка корректности загрузки данных в таблицу.
* Тестирование функциональности поиска обращений.
* Проверка удаления обращений из базы данных.
* Валидация сохранения новых и обновленных данных о заявителях и обращениях.
* Тестирование добавления новых мест работы и адресов.
* Проверка обработки ошибок валидации данных.

Обзор тестов

* **test\_load\_data**:Проверяет корректность загрузки данных об обращениях в таблицу QTableWidget. Успешно подтверждает, что данные отображаются в таблице с правильными значениями.
* **test\_search:** Тестирует функциональность поиска по тексту в поле ввода. Успешно проверяет фильтрацию данных и их последующую загрузку в таблицу.
* **test\_delete\_appeal:** Проверяет удаление обращения из базы данных с подтверждением через диалоговое окно. Успешно подтверждает вызов методов удаления и обновления таблицы.
* **test\_save\_applicant\_new**: Тестирует сохранение нового заявителя. Успешно подтверждает добавление записи в сессию и завершение транзакции.
* **test\_save\_appeal\_new:** Проверяет сохранение нового обращения. Успешно подтверждает корректное создание объекта и его сохранение в базе данных.
* **test\_open\_add\_work\_place:** Тестирует добавление нового места работы через диалоговое окно. Успешно проверяет обновление выпадающего списка после добавления.
* **test\_open\_add\_address**: Проверяет добавление нового адреса через диалоговое окно. Успешно подтверждает обновление выпадающего списка адресов.
* **test\_save\_appeal\_validation\_error**: Тестирует обработку ошибки валидации при пустом регистрационном номере обращения. Успешно подтверждает вывод предупреждения пользователю.
* **test\_save\_applicant\_validation\_error:** Проверяет обработку ошибки валидации при пустом поле ФИО заявителя. Успешно подтверждает вызов сообщения об ошибке.

Результаты тестирования представлены в таблице 1.

Таблица 1 – Результаты модульного тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| Цель тестирования | Тест | Описание | Результат |
| Проверка корректности загрузки данных в таблицу | test\_load\_data | Проверяет загрузку данных об обращениях в QTableWidget с правильными значениями. | Passed |
| Тестирование функциональности поиска обращений | test\_search | Тестирует поиск по тексту с фильтрацией данных и загрузкой в таблицу. | Passed |
| Проверка удаления обращений из базы данных | test\_delete\_appeal | Проверяет удаление обращения с подтверждением через диалоговое окно и обновлением таблицы. | Passed |
| Валидация сохранения новых и обновленных данных о заявителях и обращениях | test\_save\_applicant\_new | Тестирует сохранение нового заявителя с добавлением записи в сессию и завершением транзакции. | Passed |
| Валидация сохранения новых и обновленных данных о заявителях и обращениях | test\_save\_appeal\_new | Проверяет сохранение нового обращения с корректным созданием объекта и сохранением в базе. | Passed |
| Тестирование добавления новых мест работы | test\_open\_add\_work\_place | Тестирует добавление нового места работы и обновление выпадающего списка. | Passed |
| Тестирование добавления новых адресов | test\_open\_add\_address | Проверяет добавление нового адреса и обновление выпадающего списка адресов. | Passed |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| Проверка обработки ошибок валидации данных | test\_save\_appeal\_validation\_error | Тестирует обработку ошибки при пустом регистрационном номере обращения с выводом предупреждения. | Passed |
| Проверка обработки ошибок валидации данных | test\_save\_applicant\_validation\_error | Проверяет обработку ошибки при пустом поле ФИО заявителя с вызовом сообщения об ошибке. | Passed |

# **ПРИЛОЖЕНИЯ**

# **Приложение А – Спецификация требований**

# **Приложение Б – Репозиторий с документами в системе контроля версий**

**Ссылка на репозиторий проекта**

https://github.com/TommiFSB/Production-practice.git