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 |  |  |  | Стальмаков Д.Е. |
|  |  |  | *подпись* |  | *Фамилия И.О.* |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Руководитель практики  от предприятия |  |  |  |  |  |  | Дубинин Д.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](#_Toc195492136)

[2 Анализ предметной области 5](#_Toc195492137)

[2.1 Сущности 5](#_Toc195492138)

[2.2 Роли пользователей 6](#_Toc195492139)

[2.3 Бизнес-процессы: 7](#_Toc195492140)

[2.4 Ограничения 8](#_Toc195492141)

[3 Проектирование программного модуля 9](#_Toc195492142)

[3.1 Проектирование диаграмм 9](#_Toc195492143)

[3.2 Разработка макетов интерфейса программного модуля 12](#_Toc195492144)

[4 Разработка программного модуля 14](#_Toc195492145)

[4.1 Разработка интерфейсов программного модуля 14](#_Toc195492146)

[4.2 Разработка кода программного модуля 19](#_Toc195492147)

[5 Тестирование программного модуля 29](#_Toc195492148)

[5.1 Разработка тест-кейсов 29](#_Toc195492149)

[5.2 Модульное тестирование 31](#_Toc195492150)

[приложения 32](#_Toc195492151)

Приложение А - [спецификация требований 32](#_Toc195492152)

Приложение Б - [репозиторий с документами в системе контроля версий 32](#_Toc195492153)

## **ИССЛЕДОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ ПРЕДПРИЯТИЯ\ОРГАНИЗАЦИИ**

ООО «ДизельАвтоКомплект» — это специализированное предприятие, основной деятельностью которого является продажа запасных частей для крупногабаритной техники. Магазин компании предлагает широкий ассортимент комплектующих, в том числе для таких известных марок, как ЯМЗ (Ярославский моторный завод) и других производителей, зарекомендовавших себя в области грузовой, сельскохозяйственной и промышленной техники.

«ДизельАвтоКомплект» ориентирована на обеспечение владельцев серьезной техники надежными запасными частями. Компания сотрудничает с надежными поставщиками, что гарантирует качество продукции, а статус официального дилера ПАО «Автодизель» (ЯМЗ) подчеркивает высокий уровень доверия со стороны производителей. В наличии всегда имеются востребованные позиции: двигатели, коробки передач, топливные системы, системы охлаждения, подвеска, тормозные узлы и многое другое. В дополнение к продаже запчастей компания занимается производством рукавов высокого давления (РВД), фитингов, цепей противоскольжения.

Помимо реализации автозапчастей, «ДизельАвтоКомплект» предоставляет следующие услуги:

* Производство: изготовление рукавов высокого давления, фитингов и цепей противоскольжения, адаптированных под нужды клиентов.

Компания выделяется на рынке благодаря следующим достоинствам:

* Доступные цены: стоимость продукции и услуг заметно ниже, чем у конкурентов, что делает предложение привлекательным для широкого круга клиентов.
* Профессионализм: в штате работают специалисты с многолетним опытом, гарантирующие качественное обслуживание.
* Гарантия: на все детали распространяется гарантия от производителей, с возможностью обмена или возврата.

В данной организации стоит специальная информационная система, подходящая под данную область – 1С:«Розница». это программа для автоматизации работы магазинов, торговых точек, сетей магазинов и других торговых предприятий. Она помогает управлять ассортиментом, ценами, скидками, остатками товаров на складе, продажами, закупками и другими операциями. 1С:Розница также позволяет вести учет финансов, анализировать продажи и управлять персоналом магазина. Программа содержит модули, предназначенные для кассиров, кладовщиков, бухгалтеров и других специалистов.

К достоинствам данной системы можно отнести:

* Автоматизация всех процессов в магазине: управление ассортиментом, ценообразование, скидки, продажи, закупки и т.д.
* Гибкость настроек: возможность адаптации программы под особенности конкретного магазина или торговой сети.

К недостаткам:

* Высокая стоимость: покупка лицензии и обслуживание программы могут быть дорогими для небольших магазинов и индивидуальных предпринимателей.
* Сложность настройки: если вы хотите настроить 1С:Розницу под свои нужды, вам может потребоваться помощь специалиста, что также может быть затратно.
* Необходимость обучения: хотя интерфейс 1С:Розницы интуитивно понятен, для эффективной работы с программой может потребоваться обучение.

## **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

До внедрения CRM-системы учет отпусков и больничных велся вручную на бумажных носителях и таблицах Excel, что приводило к ошибкам, задержкам в обработке заявок и отсутствию прозрачности, а также сложностью в анализе.

Целью разработки CRM-системы является автоматизация процессов подачи, обработки и учета заявок на отпуска и больничные, а также обеспечение интеграции с Telegram для удобства сотрудников и генерации аналитических отчетов для HR-отдела.

## **Сущности**

Предметная область охватывает процессы управления персоналом, связанными с отпусками и больничными. Ключевые сущности:

Пользователь (User):

* Представляет сотрудника компании или администратора (HR).
* Атрибуты:
  1. user\_id: уникальный идентификатор (Telegram ID для сотрудников).
  2. first\_name: имя сотрудника.
  3. last\_name: фамилия сотрудника.
  4. position: должность (например, «менеджер по продажам», «оператор производства»).
  5. department: подразделение (например, «продажи», «производство»).
  6. email: уникальный email для идентификации и связи.

Заявка (Application):

* Описывает запрос сотрудника на отпуск или больничный.
* Атрибуты:
  1. application\_id: уникальный идентификатор заявки.
  2. user\_id: ссылка на пользователя, подавшего заявку.
  3. start\_date: дата начала отпуска/больничного.
  4. end\_date: дата окончания.
  5. type: тип заявки (например, «ежегодный основной оплачиваемый отпуск», «больничный»).
  6. status: статус заявки («на рассмотрении», «одобрена», «отклонена»).
  7. reason: причина заявки (например, «семейные обстоятельства»).
  8. created\_at: дата создания заявки.
  9. updated\_at: дата последнего обновления.

Лог (Log):

* Фиксирует действия пользователей в системе для обеспечения прозрачности.
* Атрибуты:
  1. log\_id: уникальный идентификатор записи.
  2. user\_id: ссылка на пользователя, выполнившего действие.
  3. action: описание действия (например, «Подача заявки #123», «Одобрение заявки #123»).
  4. timestamp: время выполнения действия.

Отчеты:

* Аналитические данные, генерируемые для HR.
* Типы отчетов:
  1. Заявки за определенный период (например, с 01.01.2025 по 31.12.2025).
  2. Суммарная длительность отпусков/больничных по отделам за год.
  3. История заявок конкретного сотрудника.
* Формат: PDF с поддержкой кириллицы.

## **Роли пользователей**

В системе выделены две основные роли:

Сотрудник:

* Действия:
  1. Регистрация в системе через Telegram-бот (ввод имени, фамилии, должности, отдела, email).
  2. Подача заявки на отпуск или больничный (выбор типа, указание дат и причины).
  3. Просмотр списка своих заявок и их статуса.
  4. Редактирование заявки, если она находится в статусе «на рассмотрении».
* Интерфейс: Telegram-бот с текстовыми сообщениями и клавиатурами.
* Ограничения: сотрудник не имеет доступа к данным других пользователей или админ-панели.

Администратор (HR):

* Действия:
  1. Просмотр и обработка заявок (одобрение, отклонение с указанием причины).
  2. Управление пользователями (редактирование данных, удаление).
  3. Просмотр истории заявок и логов.
  4. Генерация отчетов.
* Интерфейс: десктопная админ-панель на PySide6.
* Привилегии: полный доступ ко всем данным системы.

## **Бизнес-процессы:**

Следующие ключевые бизнес-процессы, которые автоматизируются в CRM-системе:

Регистрация сотрудника:

* Сотрудник запускает Telegram-бот командой /start.
* Последовательно вводит данные: имя, фамилия, должность, отдел, email.
* Система проверяет уникальность email и сохраняет данные в БД.
* После регистрации сотрудник получает доступ к подаче заявок.

Подача заявки:

* Сотрудник выбирает тип заявки («Отпуск» → подтип или «Больничный»).
* Указывает даты начала и окончания, а также причину.
* Система валидирует данные (формат дат, логичность периода).
* Заявка сохраняется в БД со статусом «на рассмотрении».
* HR-отдел уведомляется через Telegram.

Обработка заявки:

* Администратор через админ-панель просматривает список заявок.
* Для каждой заявки доступны действия: одобрить или отклонить (с указанием причины).
* При изменении статуса заявки:
  1. Данные обновляются в БД.
  2. Сотрудник получает уведомление через Telegram.
  3. Действие логируется.

Просмотр и редактирование заявок:

* Сотрудник через Telegram-бот может просмотреть список своих заявок.
* Заявки в статусе «на рассмотрении» можно редактировать (изменить даты или причину).
* Изменения сохраняются в БД, HR уведомляется.

Генерация отчетов:

* Администратор выбирает тип отчета (по периоду, отделам, сотруднику).
* Система извлекает данные из БД и формирует PDF-отчет.
* Отчет сохраняется на локальном устройстве.

Логирование:

* Все действия (регистрация, подача/обработка заявок, редактирование пользователей) записываются в таблицу logs.
* Администратор может просматривать логи для аудита.

## **Ограничения**

Для упрощения разработки и соответствия требованиям бизнеса установлены следующие ограничения:

Формат данных:

* Даты вводятся в формате ГГГГ-ММ-ДД.
* Email должен соответствовать стандарту (валидируется через регулярное выражение).
* Типы отпусков ограничены: «ежегодный основной оплачиваемый», «ежегодный дополнительный оплачиваемый», «без сохранения заработной платы».

Временные рамки:

* Новые заявки не могут начинаться в прошлом.
* Дата окончания не может быть раньше даты начала.

Статусы заявок:

* Заявка может находиться только в одном из трех состояний: «на рассмотрении», «одобрена», «отклонена».
* Редактирование возможно только для заявок «на рассмотрении».

Уникальность:

* Email сотрудника уникален в системе.
* Telegram ID сотрудника используется как первичный ключ (user\_id).

Интеграция:

* Уведомления отправляются только через Telegram.
* PDF-отчеты используют шрифт DejaVuSans для поддержки кириллицы.

## **ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО МОДУЛЯ**

## **Проектирование диаграмм**

* 1. IDEF0

Диаграммы IDEF0 моделируют бизнес-процессы системы, показывая функции, их входы, выходы, управляющие элементы и механизмы выполнения. Уровень 0 дает общий обзор, уровень 1 детализирует процессы.

Диаграмма A-0 представляет общий взгляд на crm-систему управления заявками сотрудников. Диаграмма представлена на рисунке 3.1.

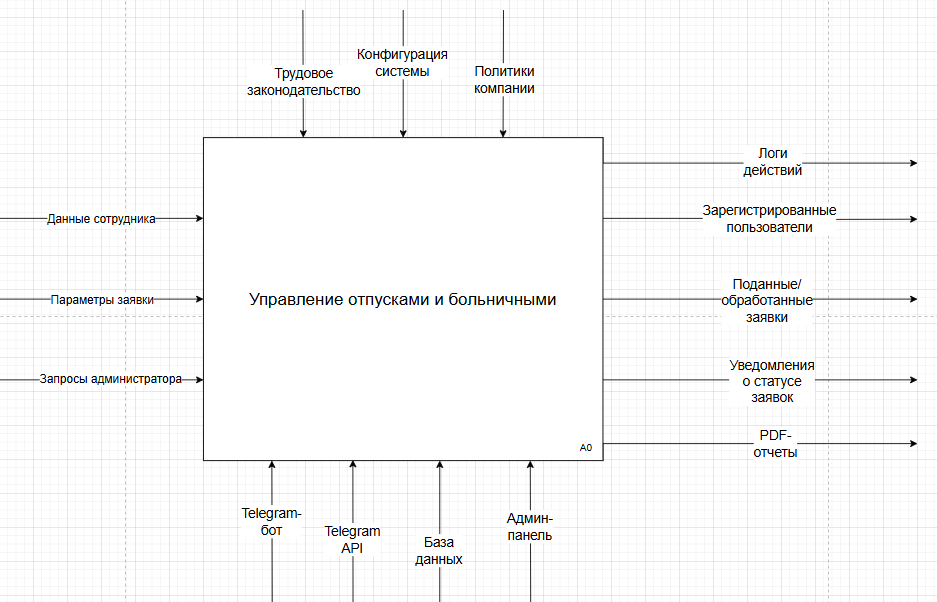
1. ****
2. Рисунок 3.1 - Диаграмма A-0

Диаграмма A0 детализирует основную функцию из A-0, разбивая ее на шесть ключевых подпроцессов. Диаграмма показана на рисунке 3.2.

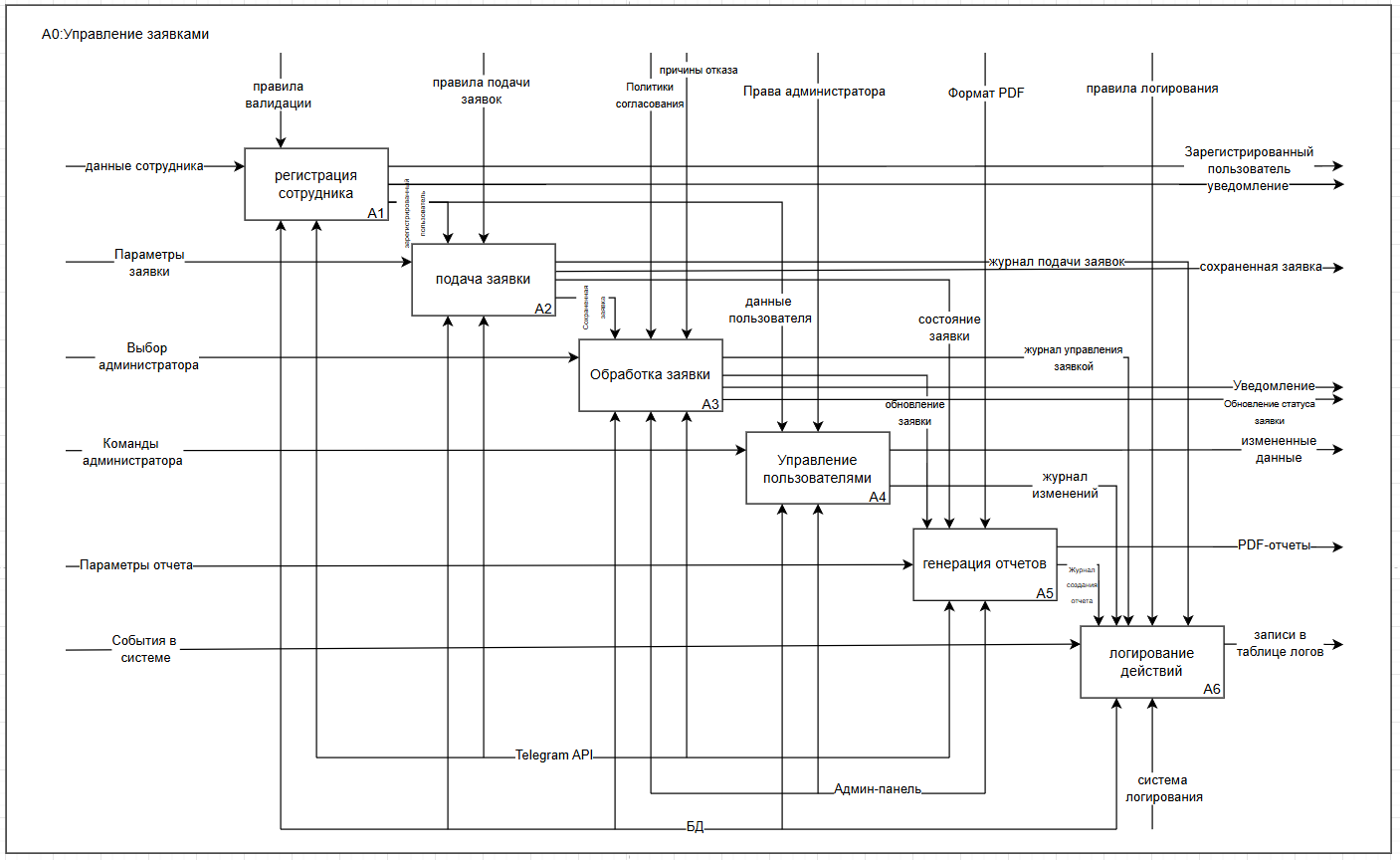
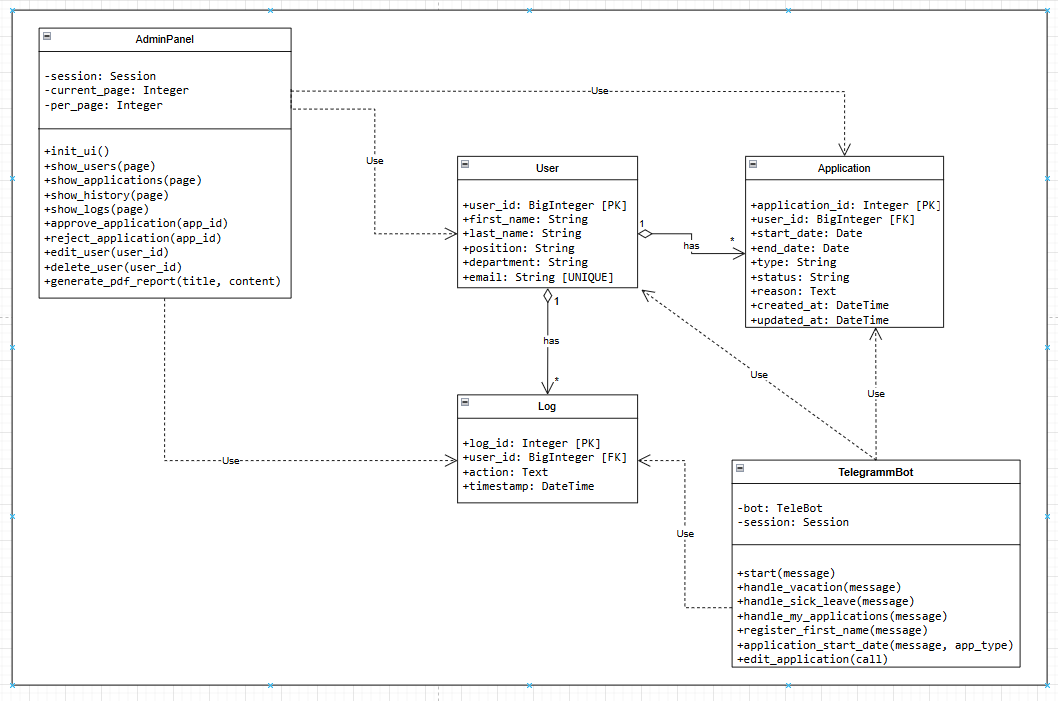
1. ****
2. Рисунок 3.2 - Диаграмма A0
   1. Диаграмма классов

Диаграмма классов предоставлена на рисунке 3.3. На ней отображена структура данных системы, показывая, как организованы сущности и какие действия с ними можно выполнять.

1. ****
2. Рисунок 3.3 - Диаграмма классов
   1. Диаграмма вариантов использования

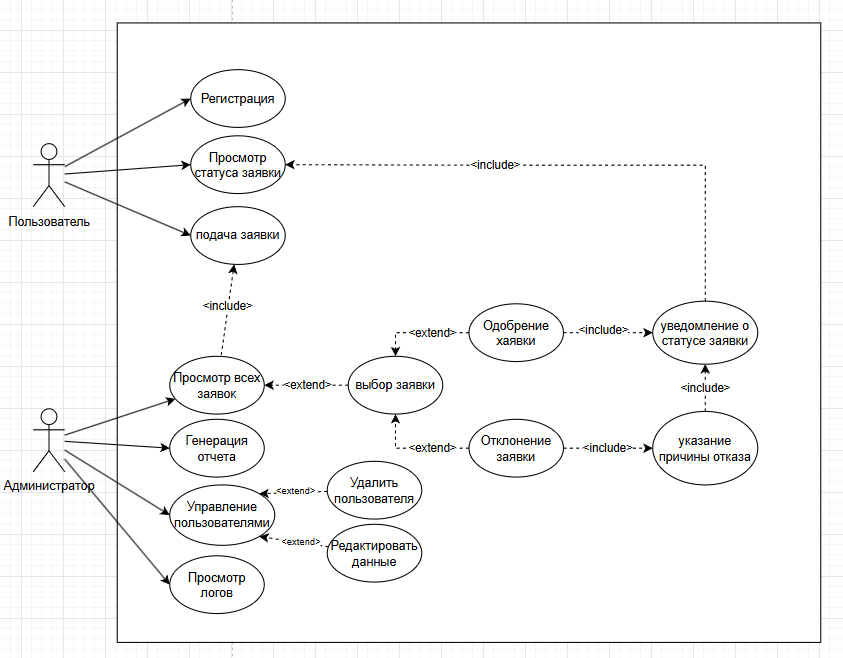
Данная диаграмма представлена на рисунке 3.4. Диаграмма включает двух акторов, представляющих пользователей системы:

1. Пользователь

* Представляет сотрудника, который использует Telegram-бот для подачи заявок на отпуск или больничный.
* Пользователь взаимодействует с системой через интерфейс Telegram, выполняя такие действия, как регистрация, подача заявок и просмотр статуса заявок.

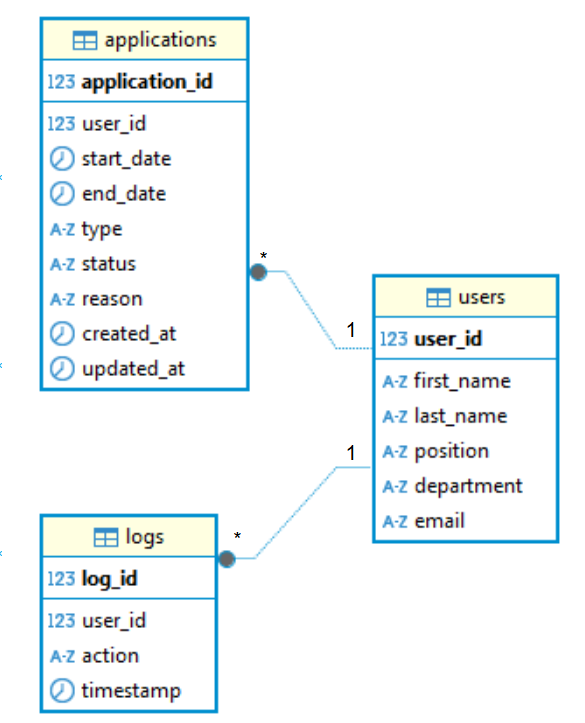
1. Администратор

* Представляет HR-специалиста, который управляет заявками сотрудников.
* Администратор имеет отдельное приложение, называемое админ-панелью, через которое управляет пользователями (редактирование, удаление), рассматривает заявки (одобрение/отклонение), генерирует отчеты, просматривает логи.



1. Рисунок 3.4 – Диаграмма вариантов использования
   1. ER-диаграмма

Для работы бота потребуется база данных, которая будет содержать таблицы для хранения информации о пользователях, заявках, а также логи действий. ER-диаграмма показана на рисунке 3.5.



1. Рисунок 3.5 – ER-диаграмма

## **Разработка макетов интерфейса программного модуля**

Макеты интерфейсов включают:

Админ-панель:

* 1. Главное окно:
* Навигация: кнопки «Пользователи», «Заявки», «История заявок», «Отчеты», «Логи».
* Фильтры: поиск по имени/ID, фильтр по статусу заявок.
* Область содержимого: список пользователей/заявок/логов с пагинацией.
* Кнопки действий: «Одобрить», «Отклонить», «Редактировать», «Удалить».
  1. Диалог редактирования пользователя:
* Поля: ID (только чтение), имя, фамилия, должность, подразделение, email.
* Кнопка «Сохранить».
  1. Диалог выбора периода для отчетов:
* Поля: даты начала и конца.
* Кнопки «OK», «Отмена».

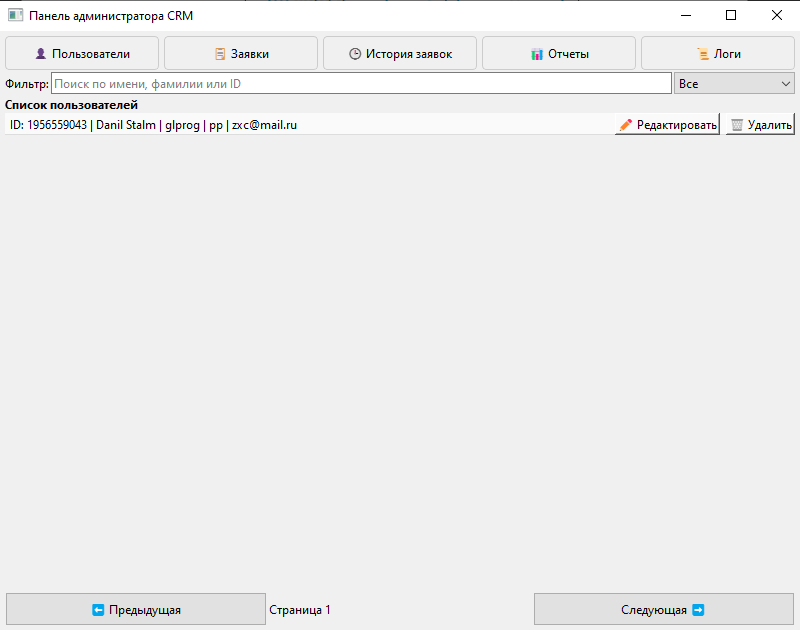
Telegram-бот:

* 1. Меню:
* Кнопки: «Отпуск», «Больничный», «Мои заявки», «В главное меню».
  1. Подменю отпуска:
* Кнопки: «Ежегодный основной», «Дополнительный», «Без сохранения», «В главное меню».
  1. Просмотр заявки:
* Инлайн-кнопка «Редактировать» для заявок «на рассмотрении»

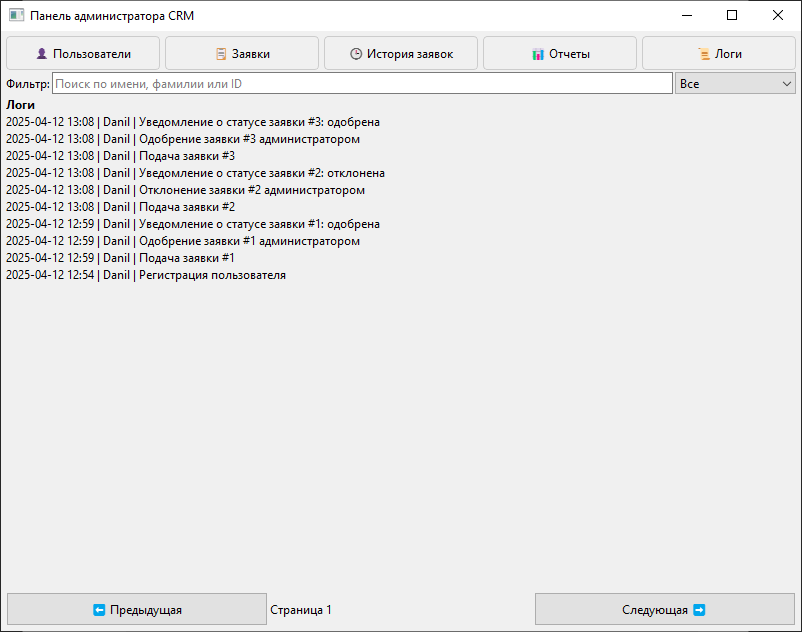
## **РАЗРАБОТКА ПРОГРАММНОГО МОДУЛЯ**

## **Разработка интерфейсов программного модуля**

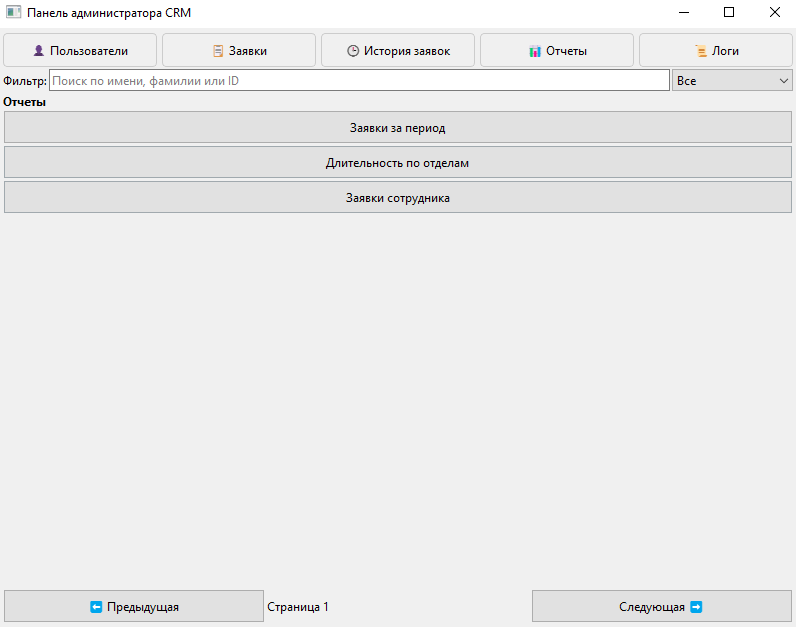
Админ-панель реализована как графическое приложение с использованием библиотеки PySide6. Реализованное приложение показано на рисунках 4.1 – 4.5.



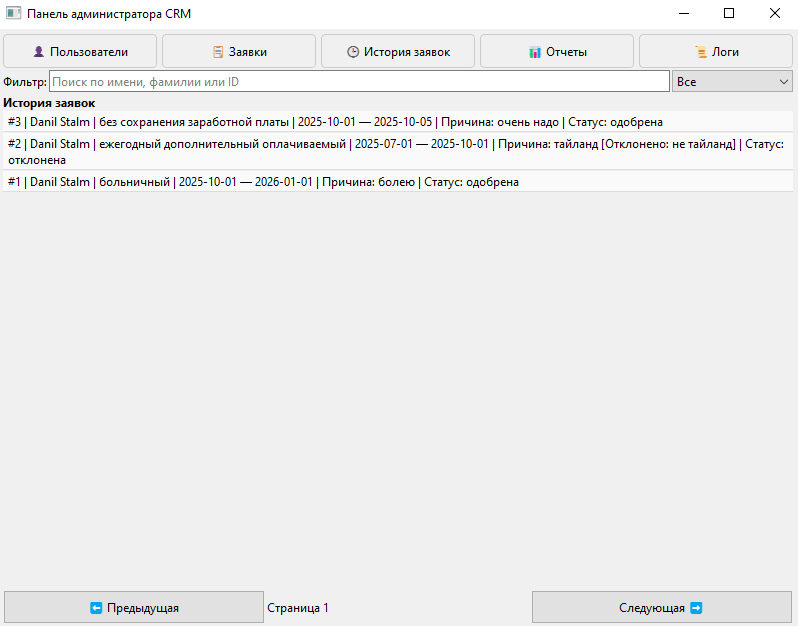
1. Рисунок 4.1 – Окно пользователи



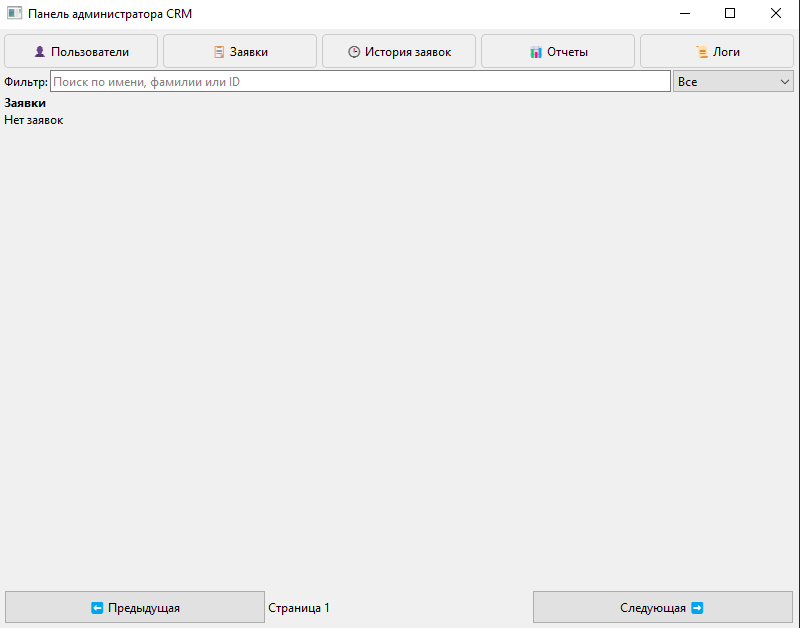
1. Рисунок 4.2 – Окно логи



1. Рисунок 4.3 – Окно отчеты



1. Рисунок 4.4 – Окно история заявок



1. Рисунок 4.5 – Окно заявки

Основной класс AdminPanel отвечает за создание главного окна, навигации, фильтров, отображения данных и диалогов. Код представлен в листинге кода 1.

Листинг 1 - класс AdminPanel

class AdminPanel(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Панель администратора CRM")

self.setGeometry(100, 100, 800, 600)

self.session = SessionFactory()

self.current\_page = 1

self.per\_page = 20

self.init\_ui()

logger.info("Панель администратора инициализирована")

Карточка данных заявки. Код представлен в фрагментах кода 2.

Листинг 2 - метод add\_application\_card

def add\_application\_card(self, app, user, history=False):

card\_widget = QWidget()

card\_layout = QHBoxLayout(card\_widget)

card\_layout.setContentsMargins(0, 0, 0, 0)

card\_text = self.format\_application\_text(app, user)

card\_label = QLabel(card\_text)

Продолжение листинга 2

card\_label.setWordWrap(True)

if not history:

approve\_btn = QPushButton("")

reject\_btn = QPushButton("")

approve\_btn.setFixedSize(40, 40)

reject\_btn.setFixedSize(40, 40)

self.apply\_style(approve\_btn, "approve\_button")

self.apply\_style(reject\_btn, "reject\_button")

approve\_btn.clicked.connect(lambda: self.approve\_application(app.application\_id))

reject\_btn.clicked.connect(lambda: self.reject\_application(app.application\_id))

card\_layout.addWidget(card\_label)

card\_layout.addStretch()

card\_layout.addWidget(approve\_btn)

card\_layout.addWidget(reject\_btn)

else:

card\_layout.addWidget(card\_label)

self.apply\_style(card\_widget, "card")

self.content\_layout.addWidget(card\_widget)

Код диалога редактирования пользователя представлен в листинге кода 3.

Листинг 3 - метод edit\_user

def edit\_user(self, user\_id):

logger.info(f"Редактирование пользователя {user\_id}")

user = self.session.query(User).filter\_by(user\_id=user\_id).first()

dialog = QDialog(self)

dialog.setWindowTitle("Редактировать пользователя")

layout = QVBoxLayout(dialog)

first\_name = QLineEdit(user.first\_name)

last\_name = QLineEdit(user.last\_name)

position = QLineEdit(user.position or "")

department = QLineEdit(user.department or "")

email = QLineEdit(user.email)

layout.addWidget(QLineEdit(f"ID: {user\_id}", readOnly=True))

layout.addWidget(first\_name)

layout.addWidget(last\_name)

layout.addWidget(position)

layout.addWidget(department)

layout.addWidget(email)

save\_btn = QPushButton("Сохранить")

save\_btn.clicked.connect(

lambda: self.save\_user(user\_id, first\_name.text(), last\_name.text(), position.text(), department.text(), email.text(), dialog))

layout.addWidget(save\_btn)

dialog.setStyleSheet("QWidget { padding: 10px; }")

dialog.exec()

Код диалога выбора периода для отчетов представлен в листинге кода 4.

Листинг 4 - метод report\_applications\_period

def report\_applications\_period(self):

logger.info("Генерация отчета: Заявки за период")

dialog = QDialog(self)

dialog.setWindowTitle("Выберите период")

layout = QVBoxLayout(dialog)

start\_date\_edit = QDateEdit(QDate.currentDate())

start\_date\_edit.setCalendarPopup(True)

start\_date\_edit.setDisplayFormat("yyyy-MM-dd")

layout.addWidget(QLabel("Начало периода:"))

layout.addWidget(start\_date\_edit)

end\_date\_edit = QDateEdit(QDate.currentDate())

end\_date\_edit.setCalendarPopup(True)

end\_date\_edit.setDisplayFormat("yyyy-MM-dd")

layout.addWidget(QLabel("Конец периода:"))

layout.addWidget(end\_date\_edit)

button\_box = QHBoxLayout()

ok\_btn = QPushButton("OK")

cancel\_btn = QPushButton("Отмена")

ok\_btn.clicked.connect(dialog.accept)

cancel\_btn.clicked.connect(dialog.reject)

button\_box.addWidget(ok\_btn)

button\_box.addWidget(cancel\_btn)

layout.addLayout(button\_box)

Telegram-бот предоставляет текстовый интерфейс с использованием клавиатур (ReplyKeyboardMarkup и InlineKeyboardMarkup) для взаимодействия с пользователями.

Определение клавиатур представлено в листинге кода 5.

Листинг 5 – класс Keyboards

class Keyboards:

@staticmethod

def main\_menu():

return types.ReplyKeyboardMarkup(resize\_keyboard=True).add(" В главное меню")

@staticmethod

def action():

return types.ReplyKeyboardMarkup(resize\_keyboard=True).add("🏖️ Отпуск", "🤒 Больничный", "📋 Мои заявки")

@staticmethod

def vacation\_type():

return types.ReplyKeyboardMarkup(resize\_keyboard=True).add(

"🌴 Ежегодный основной оплачиваемый",

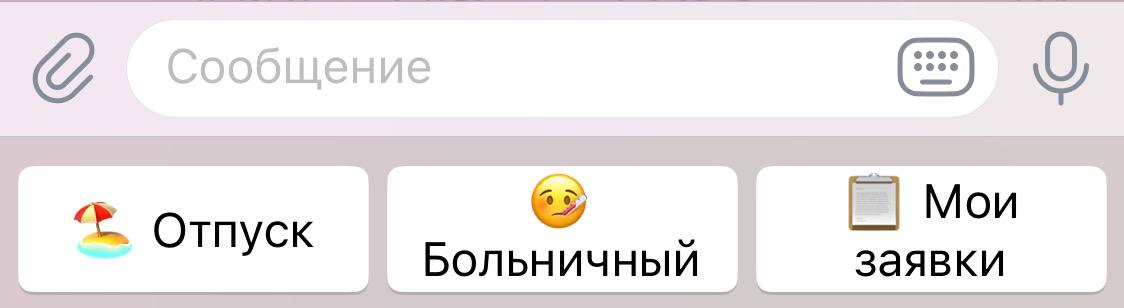
"🌞 Ежегодный дополнительный оплачиваемый",

"🏝️ Без сохранения заработной платы",

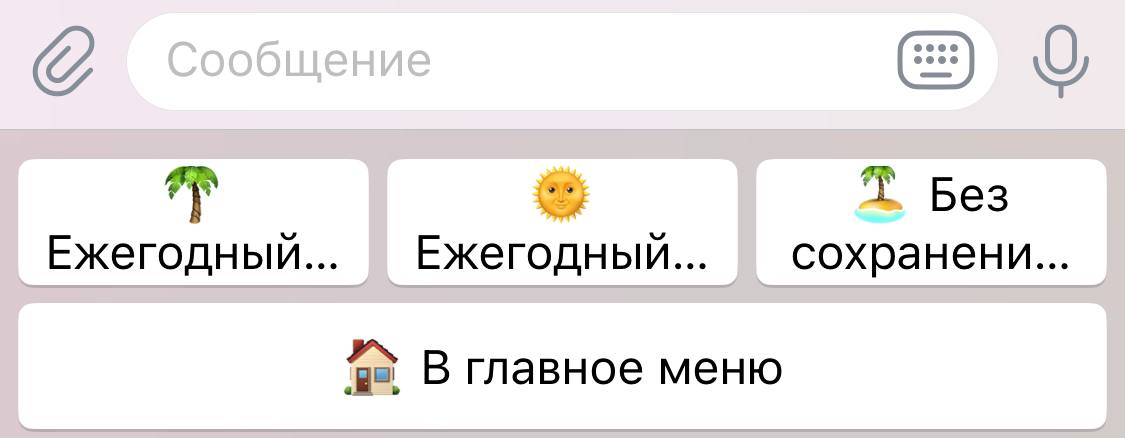
"🏠 В главное меню"

)

Рабочая программа показана на рисунках 4.6 – 4.7.



1. Рисунок 4.6 – Телеграмм бот



1. Рисунок 4.7 - Телеграмм бот

## **Разработка кода программного модуля**

Админ-панель реализована в классе AdminPanel. Основные методы отвечают за отображение данных, управление пользователями и заявками, генерацию отчетов и логирование.

Метод init\_ui инициализирует пользовательский интерфейс админ-панели, создает главный контейнер с вертикальным макетом, навигационную панель, фильтры, область содержимого с прокруткой и пагинацию, подключает кнопки к соответствующим методам отображения данных, устанавливает стили. Представлено в фрагменте кода 6.

Листинг 6 – метод init\_ui

def init\_ui(self):

main\_widget = QWidget()

self.setCentralWidget(main\_widget)

main\_layout = QVBoxLayout(main\_widget)

main\_layout.setContentsMargins(5, 5, 5, 5)

main\_layout.setSpacing(2)

# Навигация

Продолжение листинга 6

nav\_layout = QHBoxLayout()

nav\_layout.setSpacing(5)

self.btn\_users = QPushButton("👤 Пользователи")

self.btn\_applications = QPushButton("📋 Заявки")

self.btn\_history = QPushButton("🕒 История заявок")

self.btn\_reports = QPushButton("📊 Отчеты")

self.btn\_logs = QPushButton("📜 Логи")

for btn in [self.btn\_users, self.btn\_applications, self.btn\_history, self.btn\_reports, self.btn\_logs]:

self.apply\_style(btn, "nav\_button")

nav\_layout.addWidget(btn)

main\_layout.addLayout(nav\_layout)

# Фильтры

filter\_layout = QHBoxLayout()

self.search\_input = QLineEdit()

self.search\_input.setPlaceholderText("Поиск по имени, фамилии или ID")

self.status\_filter = QComboBox()

self.status\_filter.addItems(["Все", ApplicationStatus.PENDING, ApplicationStatus.APPROVED, ApplicationStatus.REJECTED])

self.search\_input.textChanged.connect(lambda: self.show\_applications(page=1))

self.status\_filter.currentTextChanged.connect(lambda: self.show\_applications(page=1))

filter\_layout.addWidget(QLabel("Фильтр:"))

filter\_layout.addWidget(self.search\_input)

filter\_layout.addWidget(self.status\_filter)

main\_layout.addLayout(filter\_layout)

# Область содержимого

self.content\_widget = QWidget()

self.content\_layout = QVBoxLayout(self.content\_widget)

self.content\_layout.setSpacing(1)

self.content\_layout.setContentsMargins(0, 0, 0, 0)

self.content\_layout.setAlignment(Qt.AlignTop)

self.scroll = QScrollArea()

self.scroll.setWidget(self.content\_widget)

self.scroll.setWidgetResizable(True)

self.scroll.setStyleSheet("QScrollArea { border: none; } QWidget { margin: 0px; }")

main\_layout.addWidget(self.scroll)

# Пагинация

pagination\_layout = QHBoxLayout()

self.prev\_page\_btn = QPushButton("⬅️ Предыдущая")

self.next\_page\_btn = QPushButton("Следующая ➡️")

self.page\_label = QLabel("Страница 1")

self.apply\_style(self.prev\_page\_btn, "action\_button")

self.apply\_style(self.next\_page\_btn, "action\_button")

self.apply\_style(self.page\_label, "label")

self.prev\_page\_btn.clicked.connect(self.prev\_page)

self.next\_page\_btn.clicked.connect(self.next\_page)

Продолжение листинга 6

pagination\_layout.addWidget(self.prev\_page\_btn)

pagination\_layout.addWidget(self.page\_label)

pagination\_layout.addWidget(self.next\_page\_btn)

main\_layout.addLayout(pagination\_layout)

# Подключение кнопок

self.btn\_users.clicked.connect(lambda: self.show\_users(page=1))

self.btn\_applications.clicked.connect(lambda: self.show\_applications(page=1))

self.btn\_history.clicked.connect(lambda: self.show\_history(page=1))

self.btn\_reports.clicked.connect(self.show\_reports)

self.btn\_logs.clicked.connect(lambda: self.show\_logs(page=1))

self.show\_applications(page=1)

Метод show\_users - Отображает список пользователей с пагинацией, очищает область содержимого и добавляет заголовок, запрашивает данные из таблицы users с учетом смещения, для каждого пользователя создает карточку с информацией и кнопками «Редактировать» и «Удалить», обновляет метку текущей страницы. Представлено в листинге кода 7.

Листинг 7 – интод show\_users

def show\_applications(self, page=1):

logger.info(f"Показ заявок, страница {page}")

self.current\_page = page

self.clear\_content()

self.add\_header("Заявки")

offset = (page - 1) \* self.per\_page

query = self.session.query(Application, User).join(User)

search\_text = self.search\_input.text().lower()

if search\_text:

query = query.filter(

(User.first\_name.ilike(f"%{search\_text}%")) |

(User.last\_name.ilike(f"%{search\_text}%")) |

(Application.application\_id == search\_text if search\_text.isdigit() else False)

)

status = self.status\_filter.currentText()

if status != "Все":

query = query.filter(Application.status == status)

else:

query = query.filter(Application.status == ApplicationStatus.PENDING)

applications = query.limit(self.per\_page).offset(offset).all()

if not applications:

no\_data = QLabel("Нет заявок")

self.apply\_style(no\_data, "label")

self.content\_layout.addWidget(no\_data)

for app, user in applications:

Продолжение листинга 7

self.add\_application\_card(app, user, history=(app.status != ApplicationStatus.PENDING))

self.page\_label.setText(f"Страница {page}")

Метод show\_applications - отображает список заявок с фильтрацией по поиску и статусу, формирует запрос к БД с учетом текста поиска и выбранного статуса. Представлено в листинге кода 8.

Листинг 8 – метод show\_applications

def show\_applications(self, page=1):

logger.info(f"Показ заявок, страница {page}")

self.current\_page = page

self.clear\_content()

self.add\_header("Заявки")

offset = (page - 1) \* self.per\_page

query = self.session.query(Application, User).join(User)

search\_text = self.search\_input.text().lower()

if search\_text:

query = query.filter(

(User.first\_name.ilike(f"%{search\_text}%")) |

(User.last\_name.ilike(f"%{search\_text}%")) |

(Application.application\_id == search\_text if search\_text.isdigit() else False)

)

status = self.status\_filter.currentText()

if status != "Все":

query = query.filter(Application.status == status)

else:

query = query.filter(Application.status == ApplicationStatus.PENDING)

applications = query.limit(self.per\_page).offset(offset).all()

if not applications:

no\_data = QLabel("Нет заявок")

self.apply\_style(no\_data, "label")

self.content\_layout.addWidget(no\_data)

for app, user in applications:

self.add\_application\_card(app, user, history=(app.status != ApplicationStatus.PENDING))

self.page\_label.setText(f"Страница {page}")

Метод approve\_application - меняет статус заявки на «одобрена» для заявок «на рассмотрении», логирует действие, отправляет уведомление через notify\_user и обновляет список заявок, обрабатывает ошибки с откатом транзакции и выводом сообщения. Представлен в листинге кода 9.

Листинг 9 – метод approve\_application

def approve\_application(self, app\_id):

logger.info(f"Одобрение заявки #{app\_id}")

Продолжение листинга 9

try:

app = self.session.query(Application).filter\_by(application\_id=app\_id).first()

if app and app.status == ApplicationStatus.PENDING:

app.status = ApplicationStatus.APPROVED

self.log\_action(app.user\_id, f"Одобрение заявки #{app\_id} администратором")

self.session.commit()

notify\_user(app\_id, ApplicationStatus.APPROVED)

QMessageBox.information(self, "Успех", f"Заявка #{app\_id} одобрена")

else:

logger.warning(f"Заявка #{app\_id} не найдена или не в статусе PENDING")

QMessageBox.warning(self, "Предупреждение", f"Заявка #{app\_id} не может быть одобрена")

self.show\_applications(self.current\_page)

except Exception as e:

self.session.rollback()

logger.error(f"Ошибка при одобрении заявки #{app\_id}: {e}")

QMessageBox.critical(self, "Ошибка", f"Не удалось одобрить заявку: {str(e)}")

Метод reject\_application - запрашивает причину отклонения через диалог, обновляет статус на «отклонена», добавляет причину в поле reason, логирует действие, уведомляет пользователя и обновляет интерфейс. Представлен в листинге кода 10.

Листинг 10 - метод reject\_application

def reject\_application(self, app\_id):

logger.info(f"Отклонение заявки #{app\_id}")

try:

app = self.session.query(Application).filter\_by(application\_id=app\_id).first()

if app and app.status == ApplicationStatus.PENDING:

reason, ok = QInputDialog.getText(self, "Причина отклонения", "Введите причину:")

if ok:

app.status = ApplicationStatus.REJECTED

app.reason = f"{app.reason or ''} [Отклонено: {reason}]"

self.log\_action(app.user\_id, f"Отклонение заявки #{app\_id} администратором")

self.session.commit()

notify\_user(app\_id, ApplicationStatus.REJECTED)

QMessageBox.information(self, "Успех", f"Заявка #{app\_id} отклонена")

else:

logger.info(f"Отклонение заявки #{app\_id} отменено")

return

Продолжение листинга 10

else:

logger.warning(f"Заявка #{app\_id} не найдена или не в статусе PENDING")

QMessageBox.warning(self, "Предупреждение", f"Заявка #{app\_id} не может быть отклонена")

self.show\_applications(self.current\_page)

except Exception as e:

self.session.rollback()

logger.error(f"Ошибка при отклонении заявки #{app\_id}: {e}")

QMessageBox.critical(self, "Ошибка", f"Не удалось отклонить заявку: {str(e)}")

Метод generate\_pdf\_report - генерирует PDF, создает документ с заголовком и списком строк, использует шрифт DejaVuSans для поддержки кириллицы, сохраняет отчет в файл, обрабатывает ошибки, например, отсутствие шрифта. Представлено в листинге кода 11.

Листинг кода 11 – метод generate\_pdf\_report

def generate\_pdf\_report(self, title, content\_lines):

logger.info(f"Генерация PDF отчета: {title}")

try:

from reportlab.lib.pagesizes import A4

from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer

from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle

from reportlab.pdfbase import pdfmetrics

from reportlab.pdfbase.ttfonts import TTFont

font\_path = "DejaVuSans.ttf"

if os.path.exists(font\_path):

pdfmetrics.registerFont(TTFont("DejaVuSans", font\_path))

else:

raise FileNotFoundError("Шрифт DejaVuSans.ttf не найден")

buffer = io.BytesIO()

doc = SimpleDocTemplate(buffer, pagesize=A4)

styles = getSampleStyleSheet()

styles.add(ParagraphStyle(name='CustomTitle', fontName='DejaVuSans', fontSize=14))

styles.add(ParagraphStyle(name='CustomNormal', fontName='DejaVuSans', fontSize=10))

story = [Paragraph(title, styles['CustomTitle']), Spacer(1, 12)]

for line in content\_lines:

story.append(Paragraph(line, styles['CustomNormal')))

story.append(Spacer(1, 6))

doc.build(story)

buffer.seek(0)

filename = QFileDialog.getSaveFileName(self, "Сохранить отчет", f"{title}.pdf", "Файлы PDF (\*.pdf)")[0]

if filename:

with open(filename, 'wb') as f:

Продолжение листинга 11

f.write(buffer.read())

QMessageBox.information(self, "Успех", "Отчет сохранен")

except Exception as e:

logger.error(f"Ошибка при генерации отчета: {e}")

QMessageBox.critical(self, "Ошибка", "Не удалось сгенерировать отчет")

Метод log\_action - добавляет запись в таблицу logs с указанием user\_id и описания, действия, логирует действие в файл через logger, обрабатывает ошибки с откатом транзакции. Представлено в листинге 12.

Листинг 12 – метод log\_action

def log\_action(self, user\_id, action):

try:

self.session.add(Log(user\_id=user\_id, action=action))

self.session.commit()

logger.info(f"Лог: {action}")

except Exception as e:

self.session.rollback()

logger.error(f"Ошибка при записи лога: {e}")

Telegram-бот реализован с использованием библиотеки telebot. Основные методы (обработчики) отвечают за регистрацию пользователей, подачу и редактирование заявок, а также просмотр заявок.

Метод start - Обрабатывает команду /start, проверяет, зарегистрирован ли пользователь, если зарегистрирован, отображает меню действий, если нет, запрашивает имя и переходит к регистрации. Представлено в листинге кода 13.

Листинг 13 – метод start

@bot.message\_handler(commands=['start'])

def start(message):

chat\_id = message.chat.id

with db\_session() as session:

user = session.query(User).filter\_by(user\_id=chat\_id).first()

if user:

send\_message(chat\_id, "Вы зарегистрированы", Keyboards.action())

else:

send\_message(chat\_id, "Введите имя:", Keyboards.main\_menu())

bot.register\_next\_step\_handler(message, register\_first\_name)

Метод register\_email - завершает процесс регистрации, начатый в start, проверяет уникальность email и его формат, создает нового пользователя в таблице users, логирует действие, при ошибке (например, занятый email) запрашивает повторный ввод, после успешной регистрации отображает меню действий. Представлено в листинге кода 14.

Листинг 14 – метод register\_email

def register\_email(message, first\_name, last\_name, position, department):

chat\_id = message.chat.id

if handle\_main\_menu\_return(message):

return

is\_valid, error = validate\_email(message.text)

if not is\_valid:

send\_message(chat\_id, f" {error}", Keyboards.main\_menu())

bot.register\_next\_step\_handler(message, register\_email, first\_name, last\_name, position, department)

return

with db\_session() as session:

try:

existing\_user = session.query(User).filter\_by(email=message.text).first()

if existing\_user:

send\_message(chat\_id, " Этот email уже зарегистрирован", Keyboards.main\_menu())

logger.warning(f"Попытка регистрации с занятым email: {message.text} для chat\_id {chat\_id}")

bot.register\_next\_step\_handler(message, register\_email, first\_name, last\_name, position, department)

return

new\_user = User(

user\_id=chat\_id,

first\_name=first\_name,

last\_name=last\_name,

position=position,

department=department,

email=message.text

)

session.add(new\_user)

session.flush()

session.add(Log(user\_id=chat\_id, action="Регистрация пользователя"))

session.flush()

saved\_user = session.query(User).filter\_by(user\_id=chat\_id).first()

if not saved\_user:

raise Exception("Пользователь не был сохранен в базе данных")

logger.info(f"Пользователь {chat\_id} успешно зарегистрирован: {first\_name} {last\_name}, {message.text}")

send\_message(chat\_id, "✅ Регистрация завершена", Keyboards.action())

except Exception as e:

logger.error(f"Ошибка при регистрации пользователя {chat\_id}: {str(e)}")

send\_message(chat\_id, f"❌ Ошибка регистрации: {str(e)}. Попробуйте снова с /start", Keyboards.main\_menu())

Метод application\_reason - завершает процесс подачи заявки, создает запись в таблице applications с данными: user\_id, даты, тип, статус «на рассмотрении», причина, логирует действие и отправляет уведомление в HR-чат, возвращает пользователя в главное меню. Представлено в листинге кода 15.

Листинг 15 – метод application\_reason

def application\_reason(message, app\_type, start\_date, end\_date):

chat\_id = message.chat.id

if handle\_main\_menu\_return(message):

return

with db\_session() as session:

app = Application(user\_id=chat\_id, start\_date=start\_date.date(), end\_date=end\_date.date(),

type=app\_type, status="на рассмотрении", reason=message.text)

session.add(app)

session.flush()

app\_id = app.application\_id

session.add(Log(user\_id=chat\_id, action=f"Подача заявки #{app\_id}"))

send\_message(CONFIG["HR\_CHAT\_ID"],

f"Заявка #{app\_id} от {chat\_id}: {app\_type} с {start\_date.date()} по {end\_date.date()}. Причина: {message.text}")

send\_message(chat\_id, " Заявка подана", Keyboards.action())

Метод handle\_my\_applications - обрабатывает запрос на просмотр заявок, пользователя, запрашивает все заявки по user\_id из таблицы applications, сортируя по убыванию ID, создает инлайн-клавиатуру с кнопками для каждой заявки, позволяя просмотреть детали, если заявок нет, информирует пользователя. Представлено в листинге кода 16.

Листинг 16 - метод handle\_my\_applications

@bot.message\_handler(func=lambda m: m.text == " Мои заявки")

def handle\_my\_applications(message):

chat\_id = message.chat.id

with db\_session() as session:

user = session.query(User).filter\_by(user\_id=chat\_id).first()

if not user:

send\_message(chat\_id, "Сначала зарегистрируйтесь с помощью /start")return

apps = session.query(Application).filter\_by(user\_id=chat\_id).order\_by(Application.application\_id.desc()).all()

if not apps:

send\_message(chat\_id, "У вас нет заявок", Keyboards.action())

return

markup = types.InlineKeyboardMarkup()

for app in apps:

btn\_text = f"#{app.application\_id} ({app.type}, {app.status})"

markup.add(types.InlineKeyboardButton(btn\_text, callback\_data=f"view\_{app.application\_id}"))

send\_message(chat\_id, "Ваши заявки:", markup)

Метод edit\_application\_reason - завершает редактирование заявки, обновляет даты, причину и время изменения в таблице applications, логирует действие и уведомляет HR-чат, возвращает пользователя в главное меню. Представлено в листинге кода 17.

Листинг 17 - метод edit\_application\_reason

def edit\_application\_reason(message, app\_id, start\_date, end\_date):

chat\_id = message.chat.id

if handle\_main\_menu\_return(message):

return

with db\_session() as session:

app = session.query(Application).filter\_by(application\_id=app\_id, user\_id=chat\_id).first()

if app and app.status == "на рассмотрении":

app.start\_date = start\_date.date()

app.end\_date = end\_date.date()

app.reason = message.text

app.updated\_at = datetime.utcnow()

session.add(Log(user\_id=chat\_id, action=f"Редактирование заявки #{app\_id}"))

send\_message(chat\_id, " Заявка обновлена", Keyboards.action())

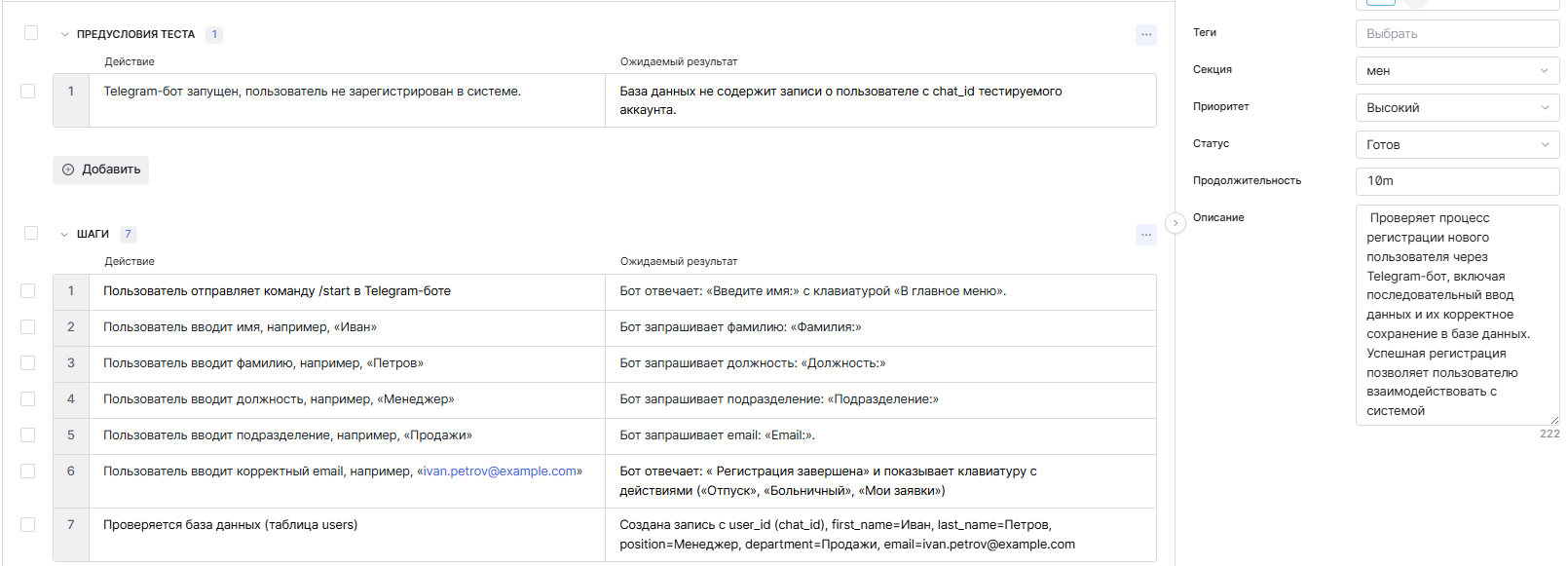
send\_message(CONFIG["HR\_CHAT\_ID"],

f"Заявка #{app\_id} от {chat\_id} обновлена: {app.type} с {app.start\_date} по {app.end\_date}. Причина: {app.reason}")

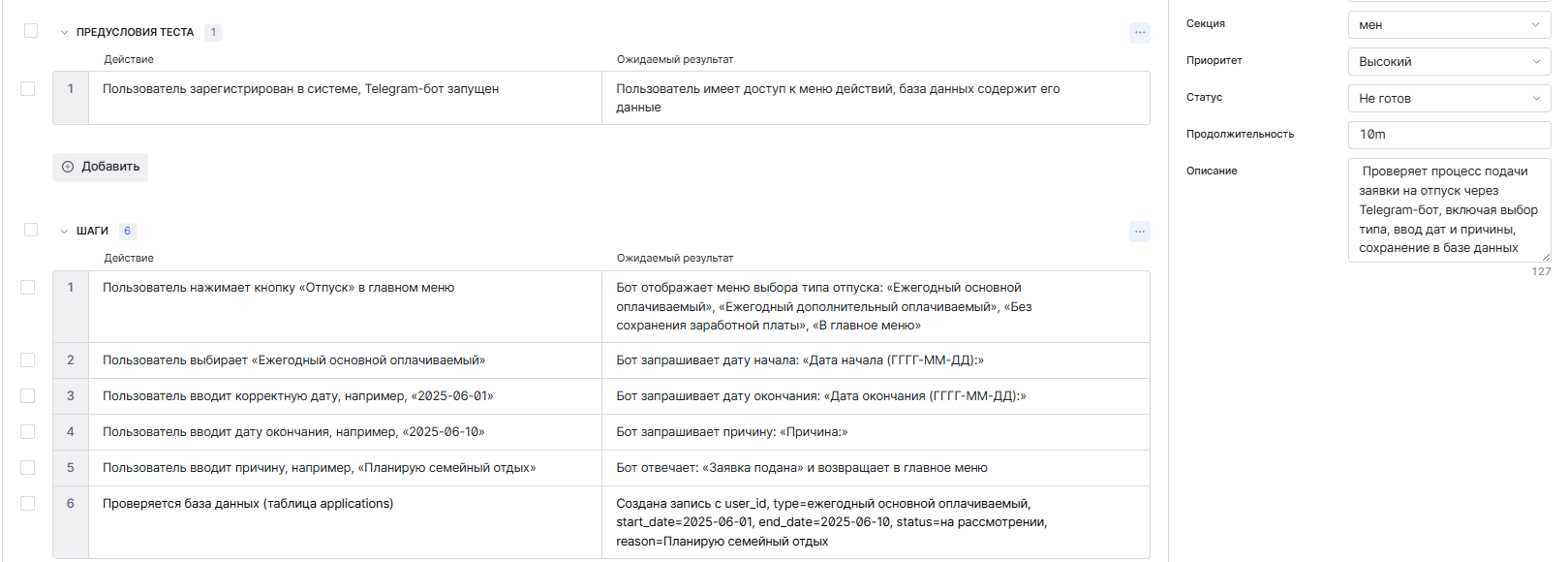
## **ТЕСТИРОВАНИЕ ПРОГРАММНОГО МОДУЛЯ**

## **Разработка тест-кейсов**

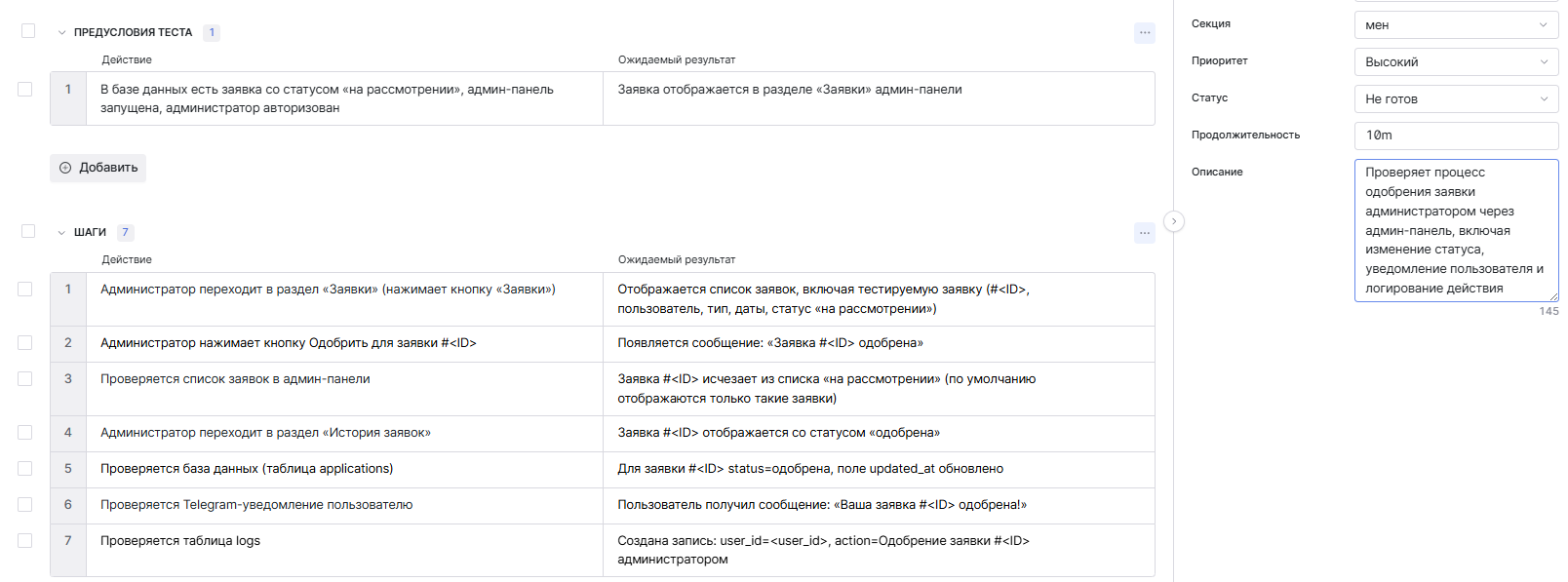
* 1. Регистрация нового пользователя через Telegram-бот



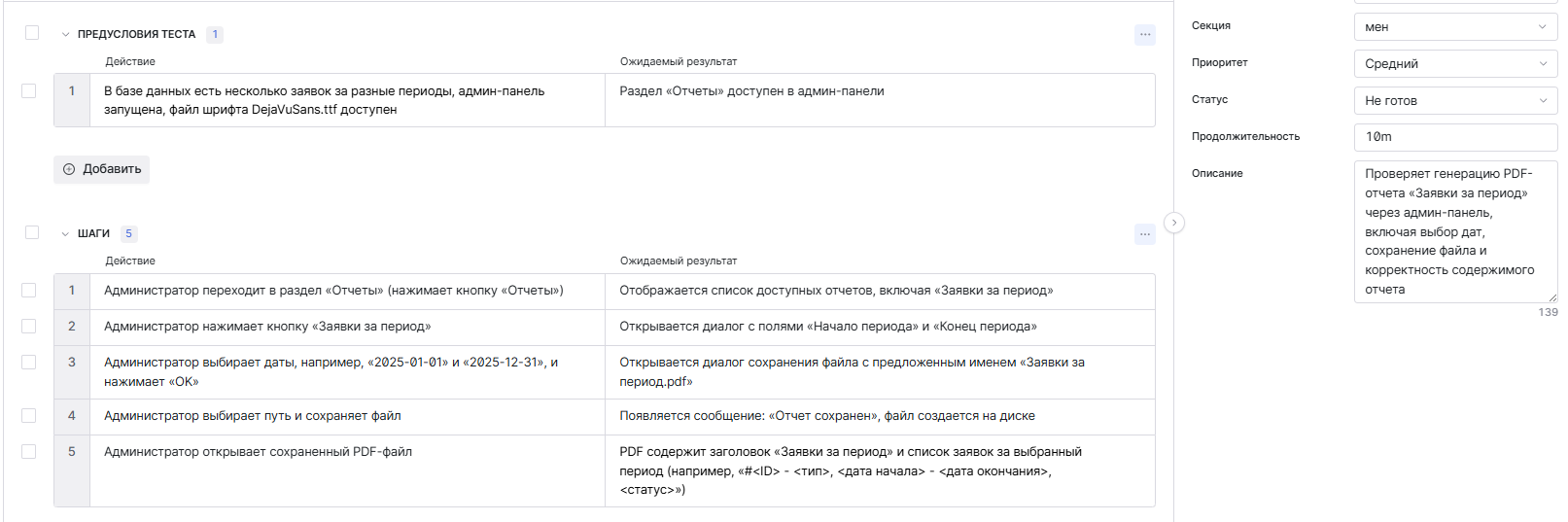
1. Рисунок 5.1 – Тест-кейс 1
   1. Подача заявки на отпуск через Telegram-бот



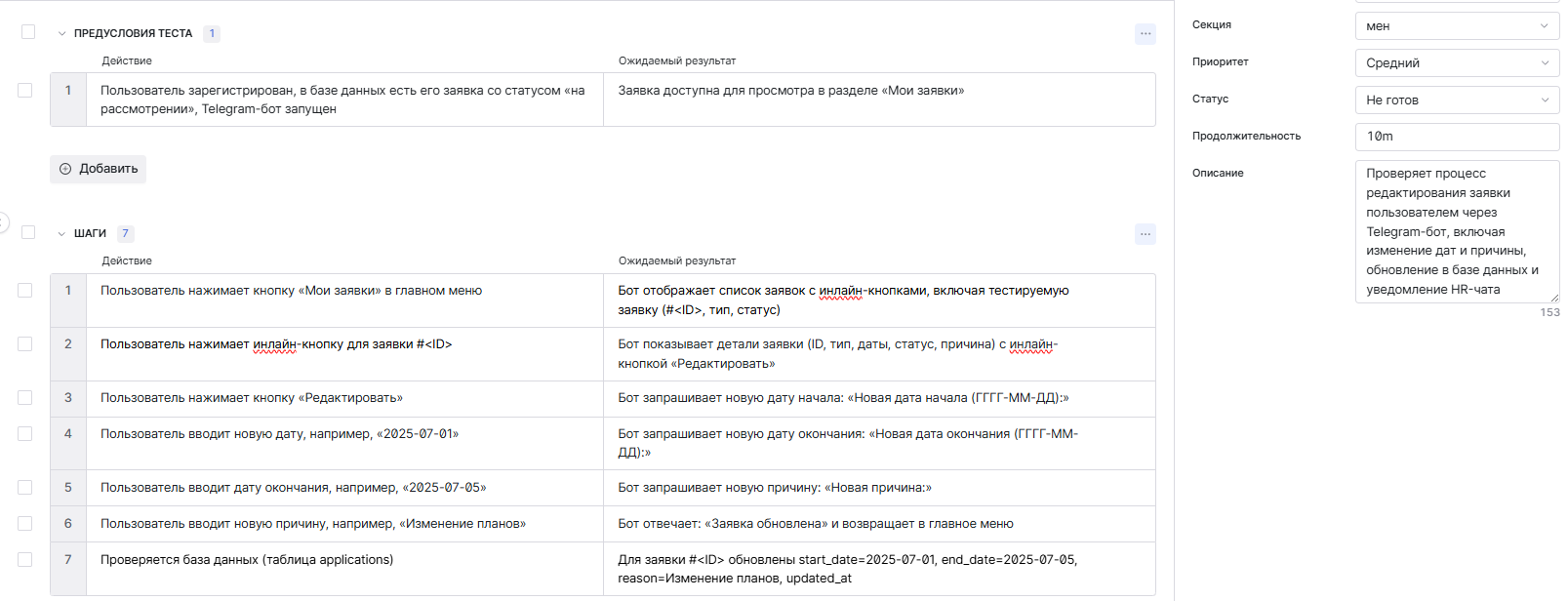
1. Рисунок 5.2 – Тест-кейс 2
   1. Одобрение заявки через админ-панель



1. Рисунок 5.3 – Тест-кейс 3
   1. Генерация отчета «Заявки за период» через админ-панель



1. Рисунок 5.4 – Тест-кейс 4
   1. Редактирование заявки через Telegram-бот



1. Рисунок 5.5 – Тест-кейс 5

## **Модульное тестирование**

Был разработан и выполнен модульный тест для функционала по одобрению заявки. Этот метод изменяет статус заявки на "одобрена", логирует действие, отправляет уведомление пользователю через Telegram и обновляет интерфейс. Модульный тест представлен в листинге кода 18.

Листинг 8 - модульный тест

import unittest

from unittest.mock import Mock, patch

from PySide6.QtWidgets import QApplication

from sqlalchemy.orm import Session

from datetime import date

from admin\_p import AdminPanel, Application, ApplicationStatus

class TestApproveApplication(unittest.TestCase):

@classmethod

def setUpClass(cls):

cls.app = QApplication([])

def setUp(self):

self.panel = AdminPanel()

self.panel.session = Mock(spec=Session)

self.mock\_session = self.panel.session

self.mock\_session.query = Mock()

self.panel.closeEvent = Mock()

self.panel.current\_page = 1

def tearDown(self):

pass

def test\_approve\_application\_success(self):

mock\_app = Application(

application\_id=1,

user\_id=12345,

start\_date=date(2025, 5, 1),

end\_date=date(2025, 5, 10),

type="отпуск",

status=ApplicationStatus.PENDING

)

mock\_query = Mock()

mock\_query.filter\_by.return\_value.first.return\_value = mock\_app

self.mock\_session.query.return\_value = mock\_query

with patch("admin\_p.notify\_user") as mock\_notify:

with patch.object(self.panel, "log\_action") as mock\_log\_action:

with patch.object(self.panel, "show\_applications") as mock\_show\_applications:

self.panel.approve\_application(1)

self.assertEqual(mock\_app.status, ApplicationStatus.APPROVED)

self.mock\_session.commit.assert\_called\_once()

mock\_notify.assert\_called\_once\_with(1, ApplicationStatus.APPROVED)

mock\_log\_action.assert\_called\_once\_with(12345, "Одобрение заявки #1 администратором")

Продолжение листинга 18

mock\_show\_applications.assert\_called\_once\_with(1)

def test\_approve\_application\_already\_processed(self):

mock\_app = Application(

application\_id=1,

user\_id=12345,

start\_date=date(2025, 5, 1),

end\_date=date(2025, 5, 10),

type="отпуск",

status=ApplicationStatus.APPROVED # Заявка уже одобрена

)

mock\_query = Mock()

mock\_query.filter\_by.return\_value.first.return\_value = mock\_app

self.mock\_session.query.return\_value = mock\_query

with patch("PySide6.QtWidgets.QMessageBox.warning") as mock\_warning:

self.panel.approve\_application(1)

self.assertEqual(mock\_app.status, ApplicationStatus.APPROVED) # Статус не изменился

self.mock\_session.commit.assert\_not\_called()

mock\_warning.assert\_called\_once()

self.panel.log\_action.assert\_not\_called()

def test\_approve\_application\_not\_found(self):

mock\_query = Mock()

mock\_query.filter\_by.return\_value.first.return\_value = None # Заявка не найдена

self.mock\_session.query.return\_value = mock\_query

with patch("PySide6.QtWidgets.QMessageBox.warning") as mock\_warning:

self.panel.approve\_application(1)

self.mock\_session.commit.assert\_not\_called()

mock\_warning.assert\_called\_once()

self.panel.log\_action.assert\_not\_called()

@classmethod

def tearDownClass(cls):

cls.app.quit()

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

Результат выполнения кода представлен на рисунке 5.1

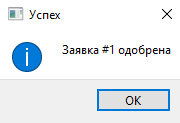


Рисунок 5.1 – Результат модульного тестирования

# Приложения

# Приложение А – спецификация требований

# Приложение Б – ссылка на систему контроля версий

1. https://github.com/sttdd/Pp.git