

МИНИСТЕРСТВО ОБРАЗОВАНИЯ НИЖЕГОРОДСКОЙ ОБЛАСТИ  
Государственное бюджетное профессиональное образовательное учреждение  
НИЖЕГОРОДСКИЙ РАДИОТЕХНИЧЕСКИЙ КОЛЛЕДЖ

**КУРСОВАЯ РАБОТА**

**МДК 02.01 Информационные технологии и платформы разработки  
информационных систем**

**Тема: «Разработка информационной системы для настольного  
клуба»**

Выполнил студент  
группы 4ИСиП-19-1

Танцев В. В.  
ФИО

Проверил преподаватель  
Гутянская Е.М.

Проект защищен с оценкой

\_\_\_\_\_

Дата защиты \_\_\_\_\_

Подпись \_\_\_\_\_

2022 г.

## Оглавление

Введение .....	3
1. Теоретические основы разрабатываемой темы.....	3
1.1. Анализ проектируемой системы .....	4
1.2. Обоснование выбора средств разработки информационной системы....	4
2. Разработка технического задания.....	6
2.1. Назначение и цели создания системы.....	6
2.2. Характеристика объектов автоматизации .....	6
2.3. Требования к структуре системы.....	6
2.4. Требования к функциям, выполняемым системой.....	8
2.5. Требования к видам обеспечения.....	8
3. Проектирование информационной системы .....	9
3.1. Диаграмма прецедентов (вариантов использования).....	9
3.3. Информационно логистическая модель базы данных.....	11
4. Разработка информационной системы .....	15
4.1. Описание структуры системы.....	15
4.2. Разработка файла подключения к базе данных.....	15
4.3. Разработка раздела «Авторизация пользователя» .....	16
4.4. Разработка раздела «Профиль сотрудника».....	20
4.5. Разработка раздела «Добавление нового товара».....	25
4.6. Разработка раздела «Просмотр истории входа» .....	28
4.7. Разработка раздела «Создание заказа» .....	30
4.8. Разработка раздела «Формирование заказа».....	37
4.9. Разработка раздела «Закрытие заказа» .....	45
5. Тестирование информационной системы.....	50
6. Руководство пользователя.....	52
6.1. Страница авторизации пользователя.....	52
6.2. Профиль сотрудника .....	52
6.3. Добавление нового товара .....	53
6.4. Создание нового заказа.....	53
6.5. Формирование заказа .....	54
6.6. Просмотр истории входа.....	56
6.7. Закрытие заказа.....	56
7. Заключение.....	57
Список использованных источников.....	58

## **Введение**

В настоящее время компьютерные технологии нашли широкое применение в различных областях. Огромное значение они имеют и в сфере сотовой связи. Работа современной системы сотовой связи не может быть возможна без использования компьютеров. Сейчас они используются уже не только как вычислительные машины, но и как средства связи.

Информационной системой (ИС) называется комплекс, включающий вычислительное и коммуникационное оборудование, программное обеспечение, лингвистические средства, информационные ресурсы, а также персонал, обеспечивающий поддержку динамической информационной модели предметной области для удовлетворения информационных потребностей пользователей.

В ИС часть функций управления и обработки данных выполняется компьютерами, а часть человеком.

Современный мир информационных технологий трудно представить себе без возможности обработки больших объёмов информации. Такие объёмы информации удобно обрабатывать с помощью баз данных. Практически все системы в той или иной степени связаны с долговременным хранением и обработкой информации. Фактически, информация становится фактором, определяющим эффективность любой сферы деятельности. Увеличились информационные потоки и повысились требования к скорости обработки данных. Большинство операций не может быть выполнено вручную. Любые административные решения требуют более чёткой и точной оценки текущей ситуации и возможных перспектив её изменения.

В данной курсовой работе необходимо разработать модуль информационной системы для настольного клуба, который позволит выполнять управление и создавать заказы. Он должен обеспечивать просмотр, обработку и выборку данных из базы данных.

## **1. Теоретические основы разрабатываемой темы**

### **1.1. Анализ проектируемой системы**

Вымышленный настольный клуб «6 Граней» является небольшой организацией, которая начала работать год назад. Организацией управляет администратор Федёров Иван Николаевич, в его подчинение входят 2 менеджера и два консультанта

«6 Граней» также располагает огромным выбором аренды на настольные игры. Цены на аренду выгодные.

Менеджер заходя в свой профиль способен сформировать новый заказ на аренду, добавить новую настольную игру в список или иной товар, и закрыть заказ на аренду, приняв настольные игры обратно.

Администратор после входа в профиль просмотреть информацию об истории входа сотрудников.

Консультант после входа в свой профиль может сформировать новый заказ на аренду, выбрав нужные товары, и закрыть заказ на аренду, приняв настольные игры обратно.

В данной курсовой работе будет разработан модуль автоматизированной информационной системы для настольного клуба «6 Граней». Она позволит сотрудникам клуба настольных игр производить различные операции с заказами на аренду, пользователями и другим функционалом программы.

### **1.2. Обоснование выбора средств разработки информационной системы.**

Python – это универсальный современный ЯП высокого уровня, к преимуществам которого относят высокую производительность программных решений и структурированный, хорошо читаемый код. Синтаксис Питона максимально облегчен, что позволяет выучить его за сравнительно короткое время. Ядро имеет очень удобную структуру, а широкий перечень встроенных библиотек позволяет применять внушительный набор полезных функций и возможностей. ЯП может использоваться для написания прикладных приложений, а также разработки WEB-сервисов.

Python может поддерживать широкий перечень стилей разработки приложений, в том числе, очень удобен для работы с ООП и функционального программирования.

Один из самых популярных интерпретаторов языка – CPython, написанный на Си. Распространяется эта среда разработки бесплатно по свободной лицензии. Интерпретатор поддерживает большинство популярных платформ.

Питон активно развивается. Примерно раз в 2 года выходят обновления. Важной особенностью языка является отсутствие таких стандартов кодировки как ANSI, ISO и некоторых других, они работают благодаря интерпретатору.

ЯП имеет четко структурированное семантическое ядро и достаточно простой синтаксис. Все, что пишется на этом языке, всегда легко читаемо. В случае необходимости передать аргументы язык использует функцию call-by-sharing.

Набор операторов в языке вполне стандартен. Удобная особенность синтаксиса – это форматирование текста кода при помощи разбивки их на блоки с помощью отступов, которые создают нажатием клавиш «Space» и «Tab». В синтаксисе отсутствуют фигурные или операторные скобки, обозначающие начало и конец блока. Такое решение заметно сокращает количество строк тела программы и приучает программиста соблюдать хороший стиль и аккуратность при написании кода.

## **2. Разработка технического задания**

### **2.1. Назначение и цели создания системы**

#### **2.1.1. Назначение системы**

Модуль информационной системы настольного клуба создается с целью обеспечения:

1. Создания менеджером и консультантом заказа и его формирование через добавление настольных игр в корзину с товарами, и присвоение этого заказа клиенту, вывод чека на печать, подсчёт стоимости за аренду;
2. Приём настольных игр из аренды обратно в настольный клуб.
3. Просмотра администратором истории входа сотрудников, добавление новых товаров.

#### **2.1.2. Цели создания системы**

1. Полный контроль над рабочей деятельности настольного клуба;
2. Учёт товаров, и информации о том, у каких клиентов они находятся в аренде;
3. Легкий способ получения полных отчетов информации о заказе на аренду.

### **2.2. Характеристика объектов автоматизации**

Автоматизируется процесс записи и хранения данных о товарах, сотрудниках и аренде в настольном клубе «6 Граней» и полной детализации его рабочей деятельности. Информация о сотрудниках, заказах и т.д. хранится в соответствующих таблицах БД и заносятся в них путем ввода данных через приложение, либо напрямую через базу данных.

### **2.3. Требования к структуре системы**

#### **2.3.1. Требования к структуре и функционированию системы**

В системе предлагается выделить следующие функциональные модули:

1. Модуль «Формирование заказов», который предназначен для хранения и обработки информации по конкретным заказам, либо датам заказов, а также для создания новых заказов.

2. Модуль «Выбор заказов», который предназначен для хранения и обработки информации по конкретным заказам и их статусам.

### **2.3.2. Требования к способам и средствам связи для информационного обмена между компонентами системы**

Взаимодействие компонентов системы осуществляется стандартными средствами платформы, на которой разработана система.

### **2.3.3. Требования к режимам функционирования системы**

Система должна поддерживать основной режим функционирования, в котором подсистемы приложения выполняют все свои основные функции.

В основном режиме функционирования Система должна обеспечивать:

1. работу пользователей режиме – 24 часов в день, 7 дней в неделю (24x7);
2. выполнение своих функций – сбор, обработка и загрузка данных; хранение данных, предоставление отчетности.

### **2.3.4. Требования по диагностированию системы**

Для обеспечения высокой надежности функционирования, как системы в целом, так и её отдельных компонентов должно обеспечиваться выполнение требований по диагностированию ее состояния.

### **2.3.5. Требования к надежности**

Уровень надежности должен достигаться согласованным применением организационных, организационно-технических мероприятий и программно-аппаратных средств.

При работе системы возможны следующие аварийные ситуации, которые влияют на надежность работы системы:

1. сбой в электроснабжении сервера;
2. сбой в электроснабжении рабочей станции пользователей системы;
3. сбой в электроснабжении обеспечения локальной сети (поломка сети);
4. сбои программного обеспечения сервера.

Проверка выполнения требований по надежности должна производиться на этапе проектирования расчетным путем, а на этапах испытаний и эксплуатации - по методике Разработчика, согласованной с Заказчиком.

### **2.3.6. Требования к защите информации**

1. Обеспечение информационное безопасности Системы должно удовлетворять следующим требованиям:
2. Защита Системы должна обеспечиваться комплексом программно-технических средств и поддерживающих их организационных мер.
3. Защита Системы должна обеспечиваться на всех технологических этапах обработки информации и во всех режимах функционирования, в том числе при проведении ремонтных и регламентных работ.
4. Программно-технические средства защиты не должны существенно ухудшать основные функциональные характеристики Системы (надежность, быстродействие, возможность изменения конфигурации).

### **2.4. Требования к функциям, выполняемым системой**

1. Хранение и обработка информации о сотрудниках;
2. Внесение изменений в БД;

### **2.5. Требования к видам обеспечения**

#### **2.5.1. Требования к программному обеспечению**

Доступные аппаратные средства:

1. ПК с доступом в интернет.

Доступные программные средства:

1. PostgreSQL-сервер;
2. Web-браузер (Google Chrome, Mozilla Firefox и т.д.).

#### **2.5.2. Требования к техническому обеспечению**

Система должна быть реализована с использованием специально выделенных серверов Заказчика.



### **3. Проектирование информационной системы**

UML – унифицированный язык моделирования (Unified Modeling Language) – это система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования. Его можно использовать для визуализации, спецификации, конструирования и документирования программных систем.

#### **3.1. Диаграмма прецедентов (вариантов использования)**

Используется для представления функциональной модели информационной системы.

Прецеденты (варианты использования — Use Cases) — это подробные процедурные описания вариантов использования системы всеми заинтересованными лицами, а также внешними системами, т. е. всеми, кто (или что) может рассматриваться как актёры (actors) — действующие лица. По сути, это своего рода алгоритмы работы с системой с точки зрения внешнего мира.

Прецеденты являются основой функциональных требований к системе, позволяют описывать границы проектируемой системы, ее интерфейс, а затем выступают как основа для тестирования системы заказчиком с помощью приемочных тестов.

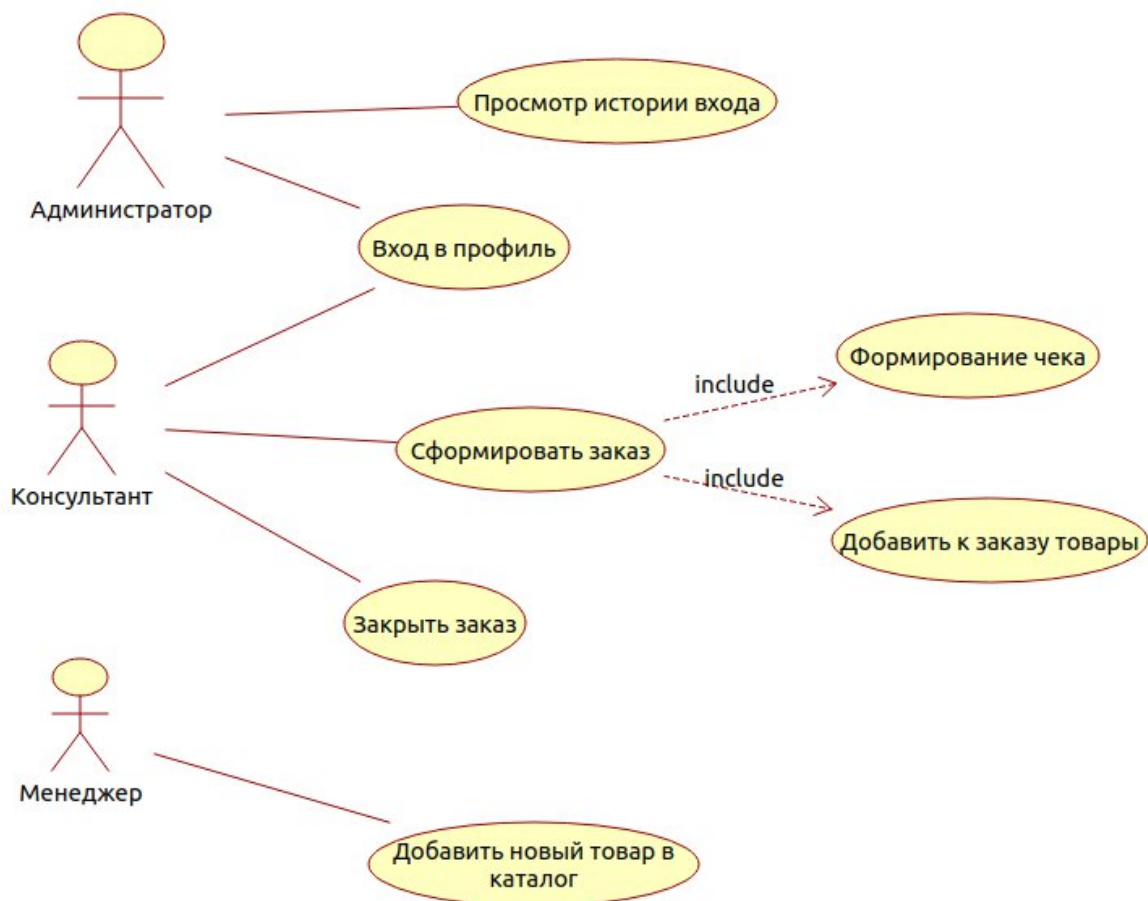


Рис. 1. Диаграмма прецедентов.

### 3.1. 3.2. Диаграмма последовательности

Диаграмма последовательности — диаграмма, на которой показано взаимодействие объектов (обмен между ними сигналами и сообщениями), упорядоченное по времени, с отражением продолжительности обработки и последовательности их проявления.

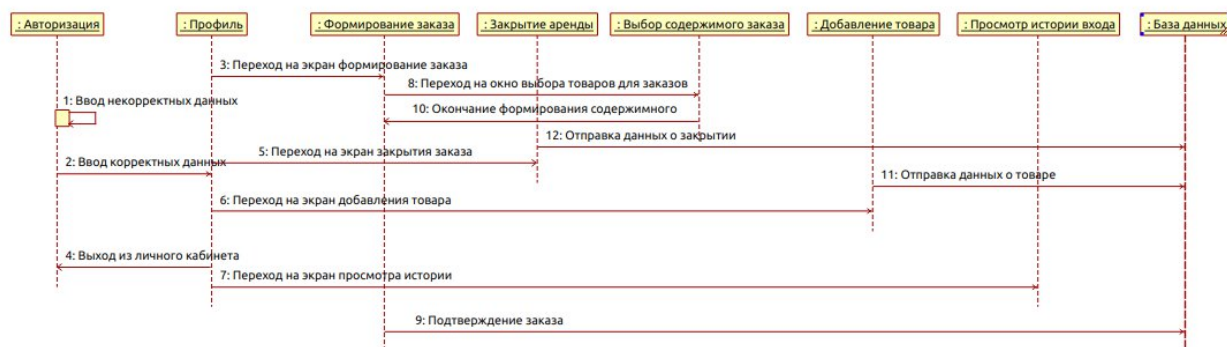


Рис. 2. Диаграмма последовательности

### 3.3. Информационно логистическая модель базы данных

Информационно-логическая модель отображает данные предметной области в виде совокупности информационных объектов и связей между ними. Эта модель представляет данные, подлежащие хранению в базе данных.

#### 3.3.1. Описание структуры БД

В нижеследующей модели описаны поля всех таблиц БД, используемых разработанной информационной системой.

Таблица 1. Описание атрибутов таблицы Клиенты

№ п/п	Название	Идентификатор	Тип	Знач. по умолчанию	Обяз-ное поле?	Признак ключа
1.	Ид. № Клиента	id	N	–	+	PK
2.	ФИО	fullname	C	–	+	-
3.	Паспорт	passport	C	–	-	–
4.	День рождения	birthday_date	D	–	–	–
5.	Адрес	address	C	-	-	–
6.	Эл. почта	email	C	-	-	-
7.	Пароль	password	C	-	-	-

Таблица 2. Описание атрибутов таблицы Сотрудники

№ п/п	Название	Идентификатор	Тип	Знач. по умолчанию	Обяз-ное поле?	Признак ключа
1.	Ид. № Сотрудника	id	N	–	+	PK
2.	ФИО	fullname	C	–	+	-
3.	Позиция	position	C	–	-	–
4.	Последний вход	Last_entry	D	–	–	–

5.	Статус	status	C	-	-	-
6.	Логин	login	C	-	-	-
7.	Пароль	password	C	-	-	-

Таблица 3. Описание атрибутов таблицы Товары

№ п/п	Название	Идентификатор	Тип	Знач. по умолчанию	Обяз-ное поле?	Признак ключа
1.	Ид. № Товара	id	N	-	+	PK
2.	Наименование	title	C	-	+	-
3.	Изображение	img	C	-	-	-
4.	Стоимость за час	cost_per_hour	N	-	-	-
5.	Полная цена	full_price	N	-	-	-
6.	Описание	description	C	-	-	-
7.	Оставшееся кол-во	remaining_amount	N	-	-	-
8.	Категория	category	C	-	-	-

Таблица 4. Описание атрибутов таблицы Заказы

№ п/п	Название	Идентификатор	Тип	Знач. по умолчанию	Обяз-ное поле?	Признак ключа
1.	Ид. № Заказа	id	N	-	+	PK
2.	Дата создания	create_date	D	-	+	-
3.	Время создания	create_time	T	-	-	-
4.	Ид. № Клиента	client_id	N	-	-	FK(Клиенты)
5.	Дата закрытия	close_date	D	-	-	-

6.	Время аренды	arenda_hour_time	N	-	-	-
7.	Ид. № Сотрудника	employer_id	N	-	-	FK(Сотрудники)
8.	Статус	category	C	-	-	-

Таблица 5. Описание атрибутов таблицы Заказ товары

№ п/п	Название	Идентификатор	Тип	Знач. по умолчанию	Обяз-ное поле?	Признак ключа
1.	Ид. № Заказа	order_id	N	—	+	PK, FK(Заказы)
2.	Ид. № Товара	goods_id	N	—	+	PK, FK(Товары)

Таблица 6. Описание атрибутов таблицы История входа

№ п/п	Название	Идентификатор	Тип	Знач. по умолчанию	Обяз-ное поле?	Признак ключа
1.	Ид. № Попытки входа	id	N	—	+	PK
2.	Логин	login	C	—	+	—
3.	Тип входа	entry_type	C	-	-	-
4.	Время входа	entry_time	D	-	-	-

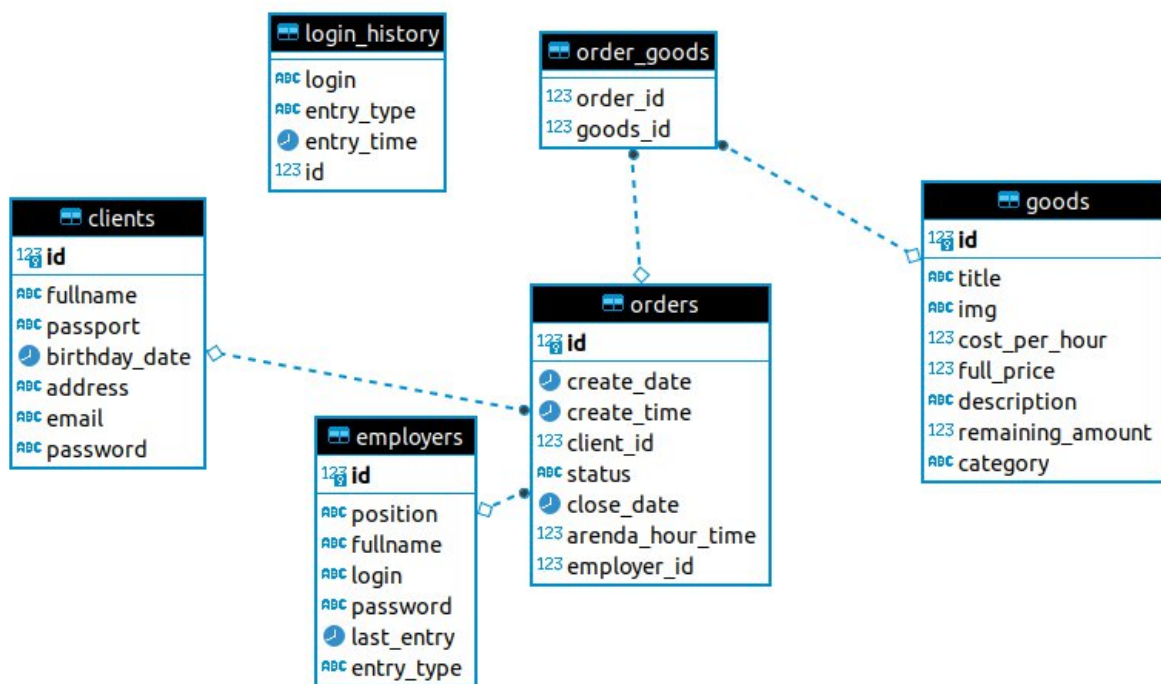


Рис. 3. Информационно-логистическая модель базы данных.

## **4. Разработка информационной системы**

### **4.1. Описание структуры системы**

В магазине работают администратор, менеджер и консультант.

Консультант после входа в профиль может сформировать новый заказ, а также принять для закрытия старый.

Менеджер заходя в свой профиль может сформировать новый заказ, и принять для закрытия старый, и добавить новый товар.

Оператор заходя в свой профиль способен сформировать новый заказ, просмотреть информацию о клиентах и зарегистрировать нового клиента.

Администратор после входа в профиль может посмотреть информацию об истории входа сотрудников.

Система будет спроектирована с учётом паттерна MVC.

MVC расшифровывается как «модель-представление-контроллер» (от англ. model-view-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения.

Компоненты MVC:

- Модель — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете To-Do приложение, код компонента `model` будет определять список задач и отдельные задачи.
- Представление — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента `view` определяет внешний вид приложения и способы его использования.
- Контроллер — этот компонент отвечает за связь между `model` и `view`. Код компонента `controller` определяет, как сайт реагирует на действия пользователя. По сути, это мозг MVC-приложения.

### **4.2. Разработка файла подключения к базе данных**

Создаем класс `DB`. В этом файле опишем подключение к базе данных. И далее в процессе разработки использовать этот файл к другим `python` классам.

### **Листинг класса Db**

```
class Db:
    def __init__(self):
        """Инициализация соединения"""
        self.connection = psycopg2.connect(
            dbname='coursework_prdb',
            user='admin',
            password='qwerty',
            host='localhost'
        )
        self.connection.autocommit = True
        self.cursor = self.connection.cursor(cursor_factory=RealDictCursor)
        self.simple_cursor = self.connection.cursor()
        print('successful connect to db')

db = Db()
```

### **4.3. Разработка раздела «Авторизация пользователя»**

Данный раздел разработан с целью обеспечения разграничения доступа сотрудников к функционалу приложения. Суть заключается в сравнении данных, указанных пользователем на форме авторизации со значениями, хранящимися в базе и, в случае их совпадения, предоставление пользователю доступа к его личному кабинету. Создаем классы MVC для Authorize, в которых будет располагаться код основной формы авторизации. Так же создана функция скрытия/показа пароля, запрет ввода пароля, после трех неверных попыток ввода данных на 10 с. и вызов диалога с капчей

#### **Листинг класса AuthorizeController**

```
class AuthorizeController(QObject):
    def __init__(self, view, model):
        super().__init__()

        self._model = model
        self._view = view

    @pyqtSlot(str, str)
    def authorize(self, login, password):
        """авторизация"""
        if not (login and password):
            self._view.show_error("Заполните все поля")
            return
```



```

candidate = self._model.verify_credentials(login, password)
if candidate:
    model = PersonalAccountModel(candidate)
    view = PersonalAccountView(model)
    view.show()
    self._view.close()
else:
    self._view.show_error("Неверный логин или пароль")
    self._model.try_auth += 1

@pyqtSlot()
def create_capcha(self):
    """Создать капчу"""
    self._view.on_capcha()

from views.CapchaDialogView import CapchaDialogView
model = CapchaDialogModel()
view = CapchaDialogView(model, self._view)
view.show()

```

### **Листинг класса AuthorizeModel**

```

class AuthorizeModel(QObject):

    user_authorized = pyqtSignal(bool)
    block_auth = pyqtSignal()

    def __init__(self):
        super().__init__()
        self._user_model = UserModel()
        self._cur_user = None
        self._try_auth = 0

    @property
    def cur_user(self):
        return self._cur_user

    @cur_user.setter
    def cur_user(self, value):
        self._cur_user = value

    def verify_credentials(self, login, password):
        """Проверить данные для входа"""
        candidate = self._user_model.get_user(login, password)
        input_type = 'успешно' if candidate else 'не успешно'
        time = datetime.datetime.now()
        self.add_to_history(login, input_type, time)

```

```

return candidate

def add_to_history(self, login, input_type, time):
    """Добавить попытку в историю входа"""
    db.simple_cursor.execute(f'INSERT INTO login_history '
                              f'("login", "entry_type", "entry_time") '
                              f'VALUES (%s, %s, %s)',
                              (
                                  login,
                                  input_type,
                                  time
                              ))

# номер попытки авторизоваться
@property
def try_auth(self):
    return self._try_auth

@try_auth.setter
def try_auth(self, value):
    self._try_auth = value
    if self._try_auth >= 4:
        self.block_auth.emit()

```

### Листинг класса **AuthorizeView**

```

class AuthorizeView(QMainWindow):

    def __init__(self, model):
        super().__init__()

        self._model = model
        self._controller = AuthorizeController(self, self._model)
        self._ui = Ui_MainWindow()
        self.setAttribute(Qt.WA_DeleteOnClose)
        self._ui.setupUi(self)
        self.init_slots()
        self.init_data()

    def init_slots(self):
        """Инициализация слотов"""
        self._ui.show_pass_box.stateChanged.connect(self.on_show_pass_changed)
        self._ui.auth_button.clicked.connect(self.on_authorize_click)
        self._model.user_authorized.connect(self.on_authorize_result)
        self._model.block_auth.connect(self._controller.create_capcha)

    def init_data(self):

```

```

        """Инициализация данных"""
        pass
        # self._ui.login_edit.setText("fedorov@namecomp.ru")
        # self._ui.pass_edit.setText("8ntwUp")

    def show_error(self, text):
        """Показать ошибку"""
        msg_box = QMessageBox()
        msg_box.setText(text)
        msg_box.setIcon(QMessageBox.Critical)
        msg_box.setWindowTitle("Ошибка авторизации")
        msg_box.exec()

    @pyqtSlot(int)
    def on_show_pass_changed(self, is_checked):
        """Слот для реакции на чекбокс о показе пароля"""
        mode = QLineEdit.Normal if is_checked else QLineEdit.Password
        self._ui.pass_edit.setEchoMode(mode)

    @pyqtSlot()
    def on_authorize_click(self):
        """Клик для авторизации"""
        login = self._ui.login_edit.text()
        password = self._ui.pass_edit.text()
        self._controller.authorize(login, password)

    @pyqtSlot(bool)
    def on_authorize_result(self, value):
        """Результат авторизации"""
        print(value)

    def on_capcha(self):
        """Слот при открытии капчи"""
        self._ui.auth_button.setDisabled(True)

    def on_block_auth(self):
        """Слот при блокировании возможности входа"""
        timer = QTimer()
        self._ui.auth_button.setDisabled(True)
        timer.singleShot(10000, self.after_block_auth)

    def on_good_capcha(self):
        """При успешном вводе капчи"""
        self._ui.auth_button.setDisabled(False)

    def after_block_auth(self):

```

```

"""После блока возможности авторизоваться"""
self._ui.auth_button.setEnabled(True)

```

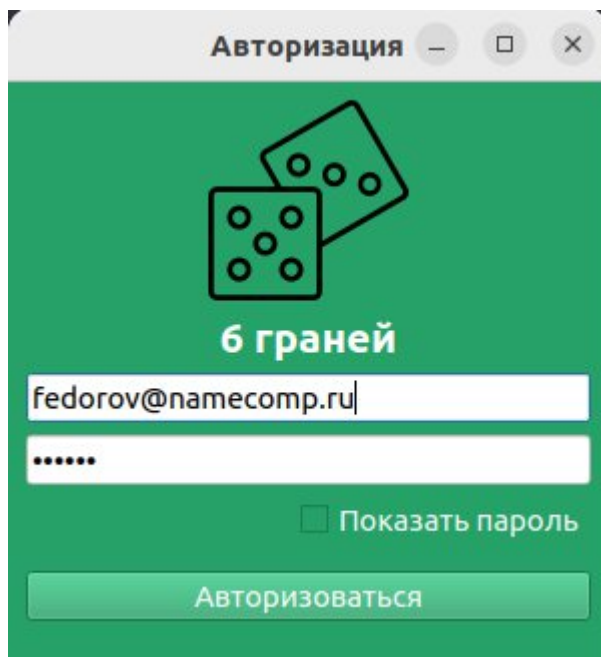


Рис. 4. Форма авторизации сотрудников

#### 4.4. Разработка раздела «Профиль сотрудника»

Эту страницу будут видеть после успешной авторизации все сотрудники службы такси. В зависимости от их должности определенные кнопки будут скрыты. Раздел профиль сотрудника является переходным для дальнейшего функционала в зависимости от должности сотрудника. Также в данном разделе сотруднику будет показано его ФИО, должность, а также его фотография. В верхнем меню можно выйти из аккаунта и из приложения

##### Листинг класса **PersonalAccountController**

```

class PersonalAccountController(QObject):
    def __init__(self, view, model):
        super().__init__()

        self._model = model
        self._view = view

    @pyqtSlot()
    def open_history(self):
        """Открыть окно с историей входа"""
        model = LoginHistoryModel()
        view = LoginHistoryView(model, self._view)
        view.show()

```

```

@pyqtSlot()
def create_order(self):
    """Открыть окно с созданием заказа"""
    model = CreateOrderModel(self._model.user)
    view = CreateOrderView(model, self._view)
    view.show()

```

```

@pyqtSlot()
def close_order(self):
    """Открыть окно с закрытием заказа"""
    model = CloseOrderModel()
    view = CloseOrderView(model, self._view)
    view.show()

```

```

@pyqtSlot()
def add_good(self):
    """Открыть окно с добавлением товара"""
    model = AddGoodModel()
    view = AddGoodView(model, self._view)
    view.show()

```

```

@pyqtSlot()
def deauth(self):
    """Деавторизация"""
    from views.AuthorizeView import AuthorizeView # for avoid circular import
    model = AuthorizeModel()
    view = AuthorizeView(model)
    self._view.close()
    view.show()

```

```

@pyqtSlot()
def close(self):
    """Закрытие экрана"""
    self._view.close()

```

## Листинг класса **PersonalAccountModel**

```

class PersonalAccountModel(QObject):

```

```

    def __init__(self, user):
        super().__init__()
        self._user_model = UserModel()
        self._cur_user = user

```

```

    @property
    def user(self):
        """Текущий пользователь"""

```

```
return self._cur_user
```

### Листинг класса **PersonalAccountView**

```
class PersonalAccountView(QMainWindow):
```

```
    def __init__(self, model: PersonalAccountModel):
        super().__init__()

        self._model = model
        self._controller = PersonalAccountController(self, self._model)
        self._ui = Ui_MainWindow()

        self.setAttribute(Qt.WA_DeleteOnClose)
        self._ui.setupUi(self)
        self.init_slots()
        self.init_data()

    def init_slots(self):
        """Инициализация слотов"""
        self._ui.see_history.clicked.connect(self._controller.open_history)
        self._ui.create_order_btn.clicked.connect(self._controller.create_order)
        self._ui.deauth_btn.triggered.connect(self._controller.deauth)
        self._ui.exit_btn.triggered.connect(self._controller.close)
        self._ui.close_order_btn.clicked.connect(self._controller.close_order)
        self._ui.add_good_btn.clicked.connect(self._controller.add_good)
        # self._ui.create_order_btn.clicked.connect(self._controller)

    def init_data(self):
        """Инициализация данных"""
        fio = self._model.user['fullname']
        position = self._model.user['position']

        lastname = fio.split()[0]
        photo = QPixmap(f"./img/{lastname}.jpeg")

        self._ui.photo_label.setPixmap(photo)
        self._ui.fio_label.setText(fio)
        self._ui.position_label.setText(position)

        if position != 'Администратор':
            self._ui.see_history.hide()
        else:
            self._ui.create_order_btn.hide()
            self._ui.close_order_btn.hide()

        if position != 'Менеджер':
```

```
self._ui.add_good_btn.hide()
```

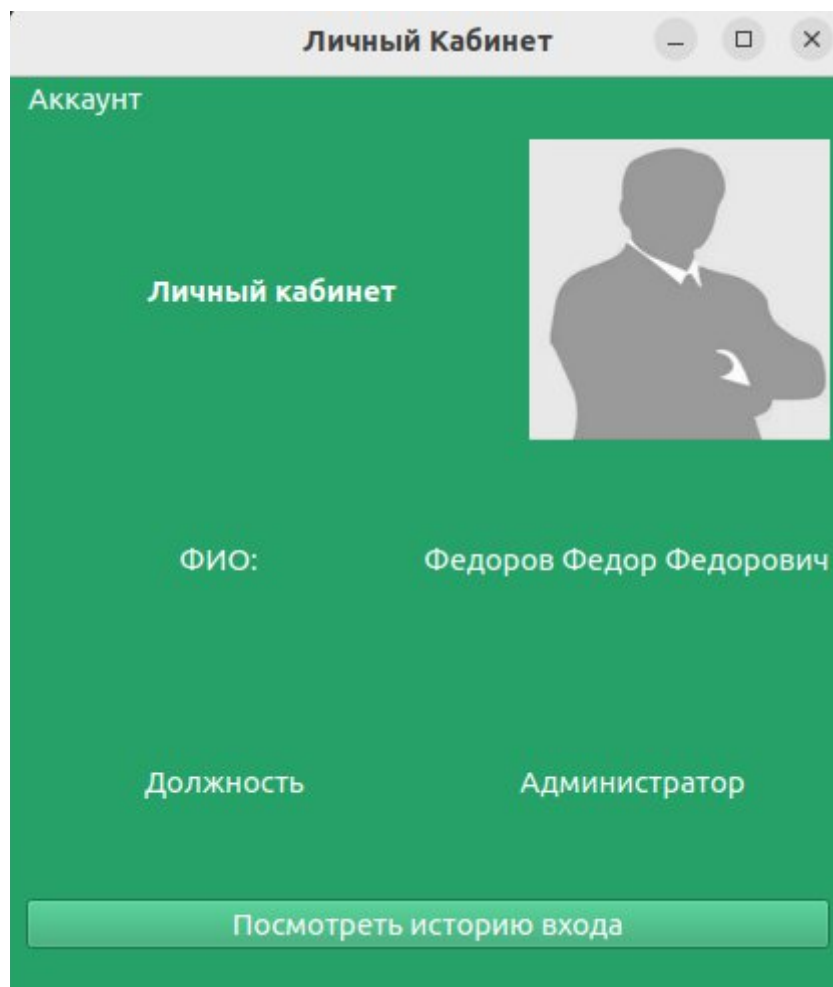


Рис. 5. Окно профиля менеджера

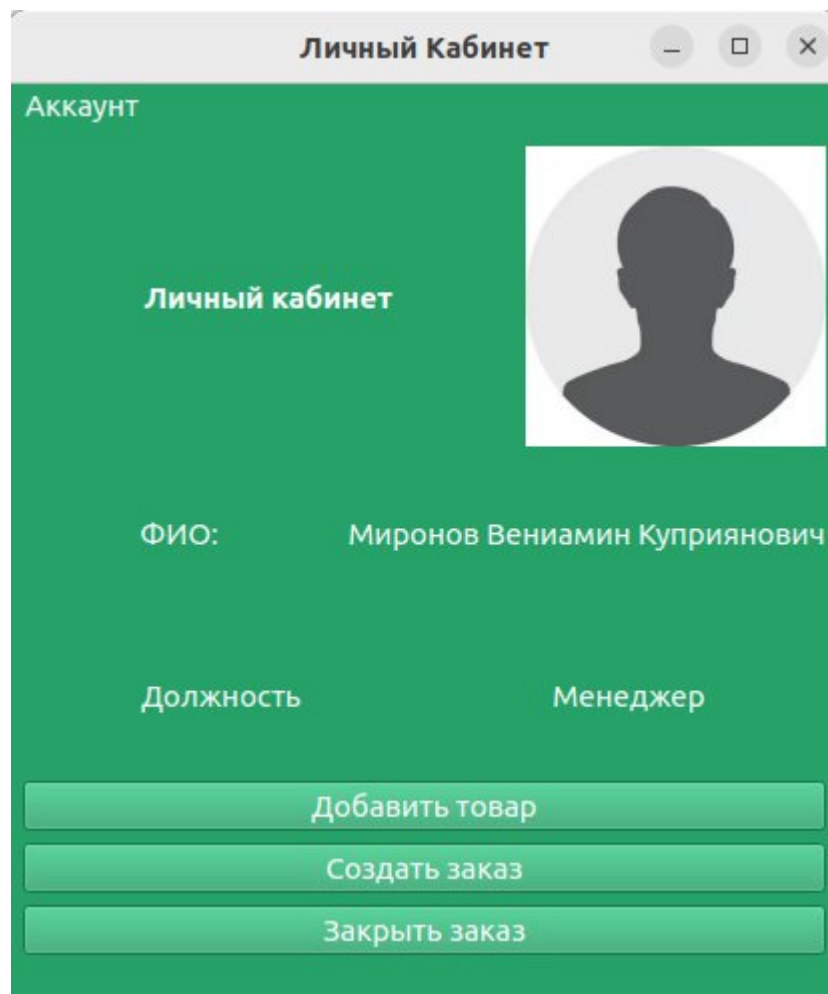


Рис. 6. Окно профиля менеджера



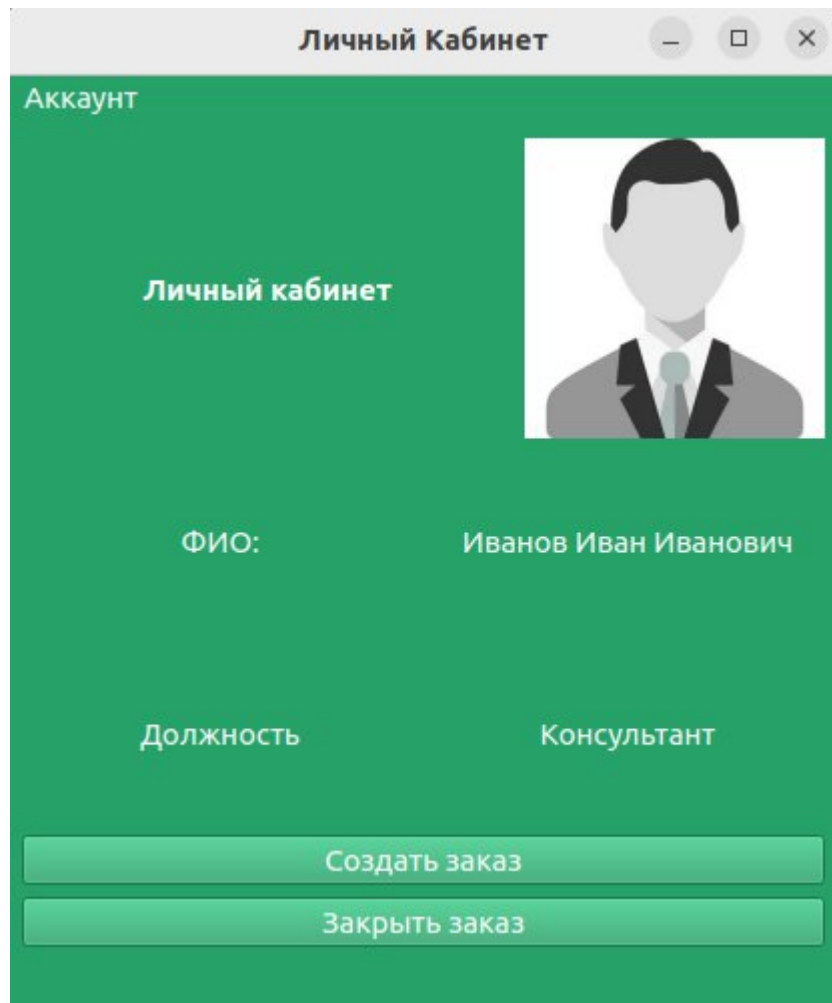


Рис. 7. Окно профиля консультанта

#### 4.5. Разработка раздела «Добавление нового товара»

Данный раздел нужен для добавления нового товара в базу данных настольного клуба. Этим функционалом может пользоваться только менеджер.

##### Листинг класса AddGoodController

```
class AddGoodController(QObject):
    def __init__(self, view, model):
        super().__init__()

        self._model = model
        self._view = view

    @pyqtSlot(dict)
    def add_good(self, good):
        """Добавить товар"""
        self._model.add_good(good)
```

```

@pyqtSlot()
def on_added_good(self):
    """Слот при успешном добавлении товара"""
    self._view.close()

```

### **Листинг класса AddGoodModel**

```

class AddGoodModel(QObject):

    good_added = pyqtSignal()

    def __init__(self):
        super().__init__()
        self.db = db

    def get_last_good_id(self):
        self.db.cursor.execute('SELECT MAX("id") FROM goods')
        result = self.db.cursor.fetchone()['max']
        return result

    def add_good(self, good):
        if " in (good['title'], good['hour_cost'], good['full_cost'], good['category']):
            return

        good['id'] = self.get_last_good_id() + 1

        db.cursor.execute(f'insert into goods '
                          f'VALUES '
                          f'('
                          f'%s, %s, %s, %s, %s, %s, %s, %s'
                          f')',
                          (
                              good['id'],
                              good['title'],
                              good['img'],
                              good['hour_cost'],
                              good['full_cost'],
                              good['description'],
                              good['count'],
                              good['category']
                          )
                          )

        self.good_added.emit()

```

### **Листинг класса AddGoodView**

```

class AddGoodModel(QObject):

```

```

good_added = pyqtSignal()

def __init__(self):
    super().__init__()
    self.db = db

def get_last_good_id(self):
    """Получение id последнего товара"""
    self.db.cursor.execute('SELECT MAX("id") FROM goods')
    result = self.db.cursor.fetchone()['max']
    return result

def add_good(self, good):
    """Добавить товар в БД1"""
    if " in (good['title'], good['hour_cost'], good['full_cost'], good['category']):
        return

    good['id'] = self.get_last_good_id() + 1

    db.cursor.execute(f'insert into goods '
                      f'VALUES '
                      f'('
                      f'%s, %s, %s, %s, %s, %s, %s, %s'
                      f')',
                      (
                          good['id'],
                          good['title'],
                          good['img'],
                          good['hour_cost'],
                          good['full_cost'],
                          good['description'],
                          good['count'],
                          good['category']
                      )
                    )
    self.good_added.emit()

```

Рис. 8. Окно раздела «Добавление товара»

#### 4.6. Разработка раздела «Просмотр истории входа»

Данным функционалом может обладать Администратор. Он нужен для контроля персоналом.

##### Листинг класса `LoginHistoryController`

```
class LoginHistoryController(QObject):
    def __init__(self, view, model):
        super().__init__()

        self._model = model
        self._view = view

    @pyqtSlot(str)
    def filter_by_login(self, value):
        """Фильтр по логину"""
        self._model.history = self._model.get_history_by_login(value)
```

##### Листинг класса `LoginHistoryController`

```
class LoginHistoryController(QObject):
    def __init__(self, view, model):
```

```

super().__init__()

self._model = model
self._view = view

@pyqtSlot(str)
def filter_by_login(self, value):
    """Фильтр по логину"""
    self._model.history = self._model.get_history_by_login(value)

```

### **Листинг класса LoginHistoryModel**

```

class LoginHistoryModel(QObject):

    history_changed = pyqtSignal()

    def __init__(self):
        super().__init__()
        self.db = db
        self._history = self.get_history_from_db()

    @property
    def history(self):
        return self._history

    @history.setter
    def history(self, value):
        self._history = value
        self.history_changed.emit()

    def get_history_from_db(self):
        """Получим историю входа"""
        db.cursor.execute("SELECT * FROM login_history")
        result = self.db.cursor.fetchall()
        return result

    def get_history_by_login(self, login):
        """Получить запись о истории входа по id"""
        db.cursor.execute("SELECT * FROM login_history "
                          f"WHERE login LIKE '%{login}%'")
        result = self.db.cursor.fetchall()
        return result

```

История входа				
Поиск по логину			Поиск	
	Логин	Тип входа	Время	id
27	fedorov@namecomp.ru	успешно	2022-12-02 00:40:26.697006	27
28	mironov@namecomp.ru	успешно	2022-12-02 00:41:37.561552	28
29	Ivanov@namecomp.ru	успешно	2022-12-02 00:42:35.853048	29
30	mironov@namecomp.ru	успешно	2022-12-02 00:46:25.321364	30
31	fedorov@namecomp.ru	успешно	2022-12-02 22:00:05.509261	31
32	fedorov@namecomp.ru	успешно	2022-12-02 22:00:11.510146	32
33	fedorov@namecomp.ru	успешно	2022-12-03 23:11:18.811384	33
34	pupkin@namecomp.ru	не успешно	2022-12-03 23:11:46.042422	34
35	rydov@namecomp.ru	не успешно	2022-12-03 23:11:56.371141	35
36	mironov@namecomp.ru	успешно	2022-12-03 23:12:04.938667	36
37	fedorov@namecomp.ru	успешно	2022-12-03 23:12:14.240625	37

Рис. 9. Окно раздела «Регистрация клиента»

#### 4.7. Разработка раздела «Создание заказа»

Данный раздел предназначен для менеджера и консультанта. В нем он может добавить новый заказа в базу данных настольного клуба.

##### Листинг класса CreateOrderController

```
class CreateOrderController(QObject):
    def __init__(self, view, model: CreateOrderModel):
        super().__init__()

        self._model = model
        self._view = view

    @pyqtSlot(str)
    def change_order_id(self, id):
        """Изменение текущего ид заказа"""
        self._model.cur_order = id

    @pyqtSlot(dict)
    def create_order(self, order):
        """создать заказ"""
```

```

        result = self._model.create_order(order)
        if result:
            self.print_pdf(order)

    @pyqtSlot()
    def success_create_order(self):
        """успешное создание заказа"""
        self._view.close()

    @pyqtSlot()
    def add_goods(self):
        """Добавление товаров"""
        model = GoodsModel(self._model)
        model.cart = self._model.cart
        view = GoodsView(model)
        view.show()

    def print_pdf(self, order):
        """Распечатать чек"""
        text = f"""
Номер заказа: {order['id']}
Клиент: {order['client_fullname']}
Количество часов проката: {order['arenda_hours']}

Список товаров:

"""
        cart = self._model.cart
        cart_text = ""
        for good_id in cart:
            good = self._model.get_good_by_id(good_id)
            cart_text += f'{good["title"]}, {good["cost_per_hour"]} руб.)\n'
        text += cart_text

        text += f"Итого: {self._model.get_cart_total_price(int(order['arenda_hours']))}"
        text = text.replace('\n', '\n ')
        label = QLabel(text)
        label.setStyleSheet("background-color: #FFF")
        file_name, _ = QFileDialog.getSaveFileName(self._view, "Сохранить чек в:", "",
                                                    "pdf (*.pdf);;All Files (*)");
        if file_name:
            print_widget(label, file_name)

```

### **Листинг класса CreateOrderModel**

```
class CreateOrderModel(QObject):
```

```

order_changed = pyqtSignal(int)
order_exists = pyqtSignal(bool)
order_create = pyqtSignal(int)
order_empty = pyqtSignal()
cart_changed = pyqtSignal()

def __init__(self, cur_employer=None):
    super().__init__()
    self.db = db
    self._cur_order = self.get_last_order_id() + 1
    self._cart = []
    self.cur_employer = cur_employer

@property
def cart(self):
    return self._cart

@cart.setter
def cart(self, value):
    self._cart = value
    self.cart_changed.emit()

@property
def cur_order(self):
    return self._cur_order

@cur_order.setter
def cur_order(self, value):
    self._cur_order = value
    self.order_changed.emit(value)
    if self._cur_order == "":
        self.order_empty.emit()
    return

    candidate = self.get_order_by_id(self._cur_order)
    self.order_exists.emit(bool(candidate))

def get_last_order_id(self):
    """Получить номер последнего заказа"""
    self.db.cursor.execute('SELECT MAX("id") FROM orders')
    result = self.db.cursor.fetchone()['max']
    return result

def get_order_by_id(self, id):
    """Получить заказ по id"""

```



```

self.db.cursor.execute('SELECT * FROM orders WHERE "id" = %s', (id,))
result = self.db.cursor.fetchone()
return result

def get_good_by_id(self, good_id):
    """Получить товар по id"""
    self.db.cursor.execute(f'select * '
                           f'from goods '
                           f'WHERE '
                           f'"id" = {good_id}'
                           )
    result = self.db.cursor.fetchone()
    return result

def get_clients(self):
    """Получить клиентов"""
    self.db.cursor.execute(f'select * from clients')
    result = self.db.cursor.fetchall()
    return result

def get_client_by_fullname(self, fullname):
    """Получить клиента по ФИО"""
    self.db.cursor.execute(f'select * from clients '
                           f'WHERE "fullname" = \"{fullname}\"')
    result = self.db.cursor.fetchone()
    return result

def create_order(self, order):
    """Создать заказ"""
    if not len(order['cart']):
        return

    order['date_created'] = datetime.date.today()
    order['time_created'] = datetime.datetime.now().time()
    order['client_id'] = self.get_client_by_fullname(order['client_fullname'])['id']
    order['status'] = 'В прокате'
    order['employer_id'] = self.cur_employer["id"]

    self.db.cursor.execute(f'insert into orders '
                           f'( '
                           f'"id", '
                           f'"create_date", '
                           f'"create_time", '
                           f'"client_id", '
                           f'"status", '

```

```

        f"arenda_hour_time", '
        f"employer_id"
    f')
    f'VALUES '
    f(
    f"%s, %s, %s, %s, %s, %s, %s"
    f)',
    (
        order['id'],
        order['date_created'],
        order['time_created'],
        order['client_id'],
        order['status'],
        order['arenda_hours'],
        order['employer_id'],
    )
)

for good_id in order['cart']:
    self.db.cursor.execute("INSERT INTO order_goods "
                           "VALUES (%s, %s)", (order['id'], good_id))
    self.db.cursor.execute(f"UPDATE goods "
                           f"SET "remaining_amount" = "remaining_amount" - 1 "
                           f"WHERE "id" = {good_id}')

self.order_create.emit(order['id'])
return True

def get_cart_total_price(self, hours):
    """Получить итоговую стоимость содержания корзины"""
    total_price = 0
    for good_id in self.cart:
        good = self.get_good_by_id(good_id)
        total_price += good['cost_per_hour'] * hours

    return total_price

```

### Листинг класса CreateOrderView

```

class CreateOrderView(QWidget):

    def __init__(self, model: CreateOrderModel, parent=None):
        super().__init__(parent)

        self._model = model
        self._controller = CreateOrderController(self, self._model)

```

```

self._ui = Ui_Form()
self.setWindowFlags(Qt.Window)

self._ui.setupUi(self)
self.init_slots()
self.init_data()

def init_slots(self):
    """Инициализация слотов"""
    self._model.order_exists.connect(self.on_order_exists)
    self._ui.order_id_edit.textChanged.connect(self._controller.change_order_id)
    self._ui.create_order_btn.clicked\
        .connect(self.on_create_order)
    self._model.order_create.connect(self.on_created_order)
    self._model.order_empty.connect(self.on_order_empty)
    self._model.cart_changed.connect(self.on_order_changed)
    self._ui.add_goods.clicked.connect(self._controller.add_goods)
    self._ui.hours_count_box.textChanged.connect(self.on_order_changed)

def init_data(self):
    """Инициализация данных"""
    stu_id_regx = QRegExp('[0-9]{10}$')
    stu_id_validator = QRegExpValidator(stu_id_regx, self._ui.order_id_edit)
    self._ui.order_id_edit.setValidator(stu_id_validator)
    self._ui.order_id_edit.setText(str(self._model.cur_order))

    for client in self._model.get_clients():
        self._ui.client_box.addItem(client['fullname'])

    self.draw_list()

@pyqtSlot(bool)
def on_order_exists(self, order_exists):
    """Слот для реакции на существовании или отсутствия заказа"""
    if order_exists:
        self._ui.info_label.setText("Заказ с таким номером существует")
        self._ui.create_order_btn.setDisabled(True)
    else:
        self._ui.info_label.setText("")
        self._ui.create_order_btn.setDisabled(False)

@pyqtSlot()
def on_order_empty(self):
    """Слот для реакции на пустой заказ"""
    self._ui.info_label.setText("Введите номер заказа")

```

```

self._ui.create_order_btn.setDisabled(True)

@pyqtSlot()
def on_create_order(self):
    """Слот для приёма сигнала создания заказа"""
    order = dict()
    order['id'] = int(self._ui.order_id_edit.text())
    order['arenda_hours'] = int(self._ui.hours_count_box.text().split()[0])
    order['client_fullname'] = self._ui.client_box.currentText()
    order['cart'] = self._model.cart
    self._controller.create_order(order)

@pyqtSlot(int)
def on_created_order(self, order_id):
    """Слот при успешном создании заказа"""
    msg_box = QMessageBox()
    msg_box.setText(f"Заказ с номером {order_id} сформирован")
    msg_box.setWindowTitle("Успешно")
    msg_box.exec()
    self._controller.success_create_order()

@pyqtSlot()
def on_order_changed(self):
    """Слот при изменении заказа"""
    self.draw_list()

def draw_list(self):
    """Рисование содержания заказа"""
    cart = self._model.cart
    table = self._ui.goods_table
    table.clearContents()

    table.setRowCount(len(cart))

    total_sum = 0
    for i in range(len(cart)):
        good = self._model.get_good_by_id(cart[i])
        table.setItem(i, 0, QTableWidgetItem(str(good['title'])))
        table.setItem(i, 1, QTableWidgetItem(str(good['cost_per_hour'])))
        s = good['cost_per_hour'] * int(self._ui.hours_count_box.text().split()[0])
        total_sum += s
        table.setItem(i, 2, QTableWidgetItem(str(s)))

    self._ui.total_value_lbl.setText(str(total_sum) + " руб.")
    # table.resizeColumnsToContents()

```

table.setSortingEnabled(True)

Наименование	Руб. за час	Сумма
1 Сиггил	100	500
2 Цитадели	200	1000
3 Битва боевых ...	150	750

Итого: 2250 руб.

Рис. 10. Окно раздела «Создание заказа»

Номер заказа: 45462587  
Клиент: Фролов Андрей Иванович  
Количество часов проката: 1

Список товаров:

Similo: Мифы 50 руб.  
Битва боевых магов: Битва на горе Черепламени 150 руб.  
Итого: 200

Рис. 11. Чек заказа после его формирования

#### 4.8. Разработка раздела «Формирование заказа»

При формировании заказа необходимо определить его содержимое. Для этой цели служит окно раздела «Формирование заказа».

### Листинг класса GoodsController.

```
class GoodsController(QObject):
    def __init__(self, view, model):
        super().__init__()

        self._model = model
        self._view = view

    @pyqtSlot(str)
    def filter_by_login(self, value):
        """Фильтрация по логину"""
        self._model.history = self._model.get_history_by_login(value)

    @pyqtSlot(str)
    def sort_type_change(self, value):
        """Изменение порядка сортировки"""
        if value == "По возрастанию":
            self._model.sort_type = "ASC"
        else:
            self._model.sort_type = "DESC"

    @pyqtSlot(str)
    def sort_field_change(self, value):
        """Изменение поля сортировки"""
        if value == "Имя":
            self._model.sort_field = "title"
        elif value == "Цена":
            self._model.sort_field = "cost_per_hour"

    @pyqtSlot(str)
    def good_category_change(self, value):
        """Изменение категории"""
        if value == "Все категории":
            self._model.goods_type = ""
        else:
            self._model.goods_type = value

    @pyqtSlot()
    def next_click(self):
        """Следующая страница"""
        self._model.offset += self._model.limit
        goods_len = len(self._model.get_goods())
        if not goods_len:
            self._model.offset -= self._model.limit

    @pyqtSlot()
```

```

def prev_click(self):
    """Предыдущая страница"""
    if (self._model.offset - self._model.limit) < 0:
        self._model.offset = 0
    else:
        self._model.offset -= self._model.limit

@pyqtSlot(str)
def good_name_change(self, value):
    """Изменение фильтра товара по имени"""
    self._model.filter_good_name = value

@pyqtSlot(int)
def add_good(self, value):
    """Добавление модели"""
    self._model.add_to_cart(value)

@pyqtSlot()
def complete(self):
    """При завершении выбора"""
    self._model.createOrderModel.cart = self._model.cart
    self._view.close()

```

### Листинг класса GoodsModel

```

class GoodsModel(QObject):

    goods_changed = pyqtSignal()
    cart_changed = pyqtSignal()

    def __init__(self, create_order_model: CreateOrderModel = None):
        super().__init__()
        self.db = db
        self._filter_good_name = ""
        self._sort_field = "id"
        self._sort_type = "ASC"
        self._goods_type = ""
        self._limit = 6
        self._offset = 0
        self._goods = self.get_goods()
        self._cart = []

        self.createOrderModel = create_order_model

    def add_to_cart(self, good_id):
        """Добавить в корзину"""
        good = self.get_good_by_id(good_id)

```

```

    if good_id not in self._cart:
        if not good["remaining_amount"]:
            return
        self._cart.append(good_id)
    else:
        self._cart.remove(good_id)
    self.cart_changed.emit()

# Различные геттеры и сеттеры для свойств
@property
def cart(self):
    return self._cart

@cart.setter
def cart(self, value):
    self._cart = value

@property
def goods(self):
    return self._goods

@goods.setter
def goods(self, value):
    self._goods = value
    self.goods_changed.emit()

@property
def filter_good_name(self):
    return self._filter_good_name

@filter_good_name.setter
def filter_good_name(self, value):
    self._filter_good_name = value
    self.goods = self.get_goods()

@property
def sort_field(self):
    return self._sort_field

@sort_field.setter
def sort_field(self, value):
    self._sort_field = value
    self.goods = self.get_goods()

@property

```



```

def sort_type(self):
    return self._sort_type

@sort_type.setter
def sort_type(self, value):
    self._sort_type = value
    self.goods = self.get_goods()

@property
def goods_type(self):
    return self._goods_type

@goods_type.setter
def goods_type(self, value):
    self._goods_type = value
    self.goods = self.get_goods()

@property
def offset(self):
    return self._offset

@property
def limit(self):
    return self._limit

@offset.setter
def offset(self, value):
    self._offset = value
    self.goods = self.get_goods()

def get_all_goods(self):
    """Получить все товары"""
    self.db.cursor.execute('SELECT * FROM goods')

    result = self.db.cursor.fetchall()
    return result

def get_goods(self):
    """Получить все товары с учётом параметров"""
    self.db.cursor.execute(f'select * '
                           f'from goods '
                           f'WHERE '
                           f'"title" LIKE \'%{self._filter_good_name}%\' '
                           f'AND '
                           f'"category" LIKE \'%{self._goods_type}%\' '
                           f'ORDER BY {self._sort_field} {self._sort_type} '

```

```

        f'LIMIT {self._limit} '
        f'OFFSET {self._offset}'
    )
    result = self.db.cursor.fetchall()
    return result

def get_good_by_id(self, good_id):
    """получить товар по id"""
    self.db.cursor.execute(f'select * '
                           f'from goods '
                           f'WHERE '
                           f'"id" = {good_id}'
                           )
    result = self.db.cursor.fetchone()
    return result

@cart.setter
def cart(self, value):
    self._cart = value

```

## Листинг класса GoodsView

```
class GoodsView(QWidget):
```

```

    def __init__(self, model: GoodsModel, parent=None):
        super().__init__(parent)

        self._parent = parent
        self._model = model
        self._controller = GoodsController(self, self._model)
        self._ui = Ui_Form()
        self.setAttribute(Qt.WA_DeleteOnClose)
        self._ui.setupUi(self)
        self.init_slots()
        self.init_data()

    def init_slots(self):
        """Инициализация слотов"""
        self._model.goods_changed.connect(self.on_goods_change)
        self._ui.sort_type_box.currentTextChanged.connect(self._controller.sort_type_change)
        self._ui.field_box.currentTextChanged.connect(self._controller.sort_field_change)
        self._ui.type_box.currentTextChanged.connect(self._controller.good_category_change)
        self._ui.next_btn.clicked.connect(self._controller.next_click)
        self._ui.prev_btn.clicked.connect(self._controller.prev_click)
        self._ui.goods_name_edit.textChanged.connect(self._controller.good_name_change)
        self._ui.see_cart.clicked.connect(self.on_see_cart)
        self._model.cart_changed.connect(self.on_cart_changed)

```

```

self._ui.create_order.clicked.connect(self._controller.complete)

def init_data(self):
    """Инициализация данных"""
    self.draw_goods()
    self.on_cart_changed()

    @pyqtSlot()
    def on_goods_change(self):
        """слот для изменения товаров"""
        self.draw_goods()

    def draw_goods(self):
        """Отрисовка товаров"""
        for i in reversed(range(self._ui.goodsGrid.count())):
            self._ui.goodsGrid.itemAt(i).widget().setParent(None)

        goods = self._model.goods[:self._model.limit]

        positions = [(i, j) for i in range(2) for j in range(3)]

        for position, good in zip(positions, goods):
            good_widget = self.get_good_widget(good)

            self._ui.goodsGrid.addWidget(good_widget, *position)

    @pyqtSlot()
    def on_cart_changed(self):
        """Слот для изменений в корзине"""
        self._ui.cart_count.setText(str(len(self._model.cart)))
        self.draw_goods()

    def get_good_widget(self, good):
        """Получить виджет товара"""
        widget = QWidget()
        widget.setMaximumSize(270, 270)

        layout = QVBoxLayout()
        layout.addWidget(QLabel(good["title"]))

        photo_label = QLabel()
        if good["img"]:
            photo = QPixmap(os.path.join("img", good["img"]))
            photo.scaled(150, 50, Qt.KeepAspectRatio, Qt.SmoothTransformation)
            photo_label.setPixmap(photo)
        else:
            photo_label.setText("Нет фото")

```

```

photo_label.setFixedSize(150, 50)
photo_label.setAlignment(Qt.AlignCenter)
layout.addWidget(photo_label)

layout.addWidget(self.get_property_layout("Pyб/час", good["cost_per_hour"]))
layout.addWidget(self.get_property_layout("Категория", good["category"]))

remaining_amount = good["remaining_amount"]
if good["id"] in self._model.cart:
    remaining_amount -= 1
    add_to_cart_text = "Добавлено"
else:
    add_to_cart_text = "Добавить в заказ"

layout.addWidget(self.get_property_layout("Осталось", remaining_amount))

add_to_cart = QPushButton(add_to_cart_text)
add_to_cart.clicked.connect(lambda: self._controller.add_good(good["id"]))
layout.addWidget(add_to_cart)
widget.setLayout(layout)
widget.setStyleSheet("background-color: white; color: black;")

return widget

def get_property_layout(self, name, value):
    """
    Контейнер для горизонтального
    отображения свойства и его значения
    """
    value = str(value)
    widget = QWidget()
    layout = QHBoxLayout()
    layout.addWidget(QLabel(name + ":"))
    layout.addWidget(QLabel(value))
    widget.setLayout(layout)
    return widget

@pyqtSlot()
def on_see_cart(self):
    """При клике на просмотр корзины"""
    msg_box = QMessageBox()
    text = "Список:\n" if len(self._model.cart) else "Список пуст"

    for good_id in self._model.cart:
        text += self._model.get_good_by_id(good_id)['title'] + "\n"

```

```

msg_box.setText(text)
msg_box.setWindowTitle("Заказ")
msg_box.exec()

```

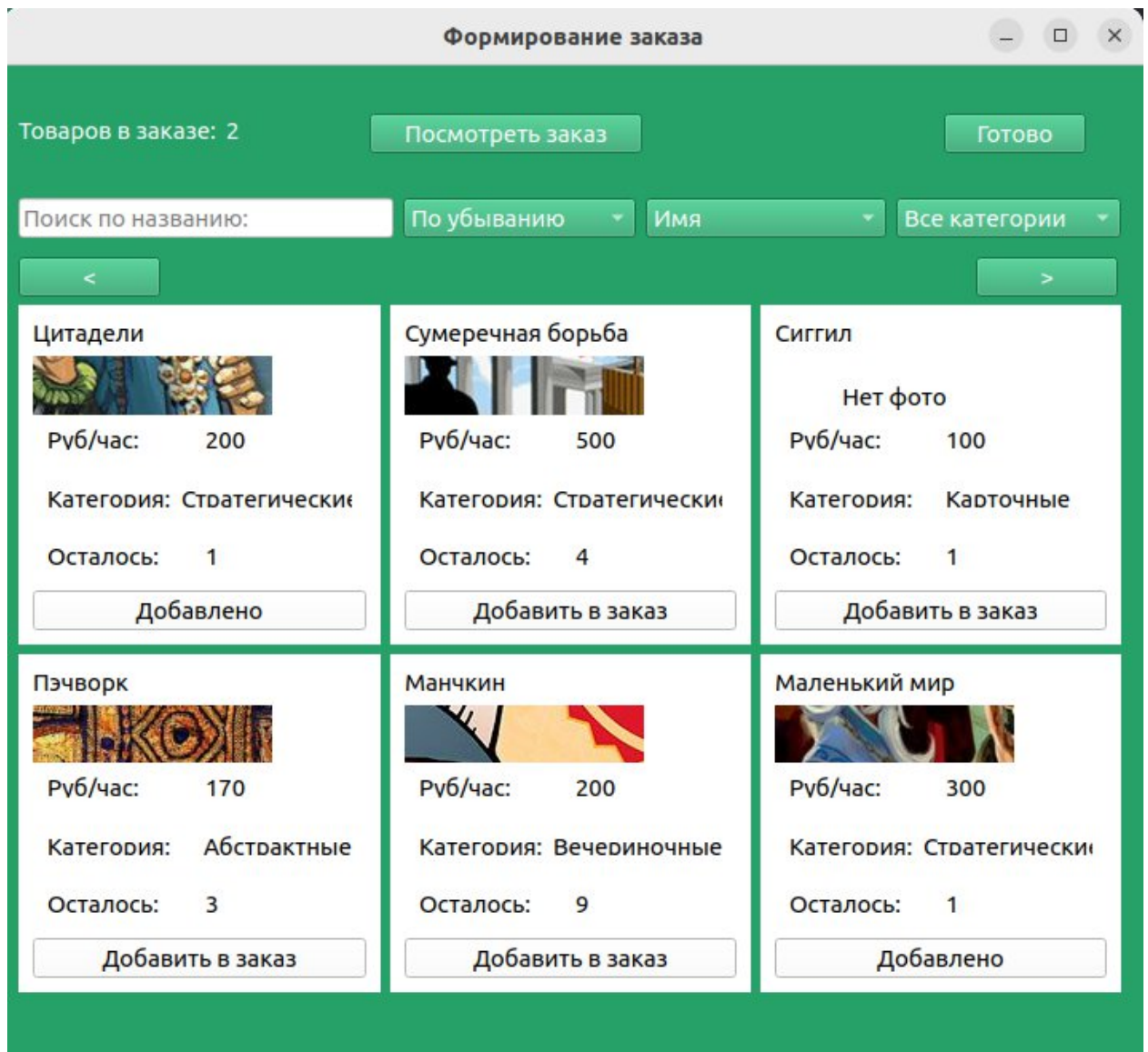


Рис. 12. Окно раздела «Формирование заказа»

#### 4.9. Разработка раздела «Закрытие заказа»

В данном разделе консультант и менеджер смогут закрыть заказ по его номеру, когда клиент вернёт то, что брал в аренду у настольного клуба.

##### Листинг класса `CloseOrderController`

```

class CloseOrderController(QObject):
    def __init__(self, view, model):
        super().__init__()

```

```

self._model = model
self._view = view

@pyqtSlot(str)
def change_order_id(self, id):
    """поменять номер заказа"""
    self._model.cur_order = id

@pyqtSlot(str)
def close_order(self, order_id):
    """закрыть заказ"""
    self._model.close_order_by_id(order_id)

@pyqtSlot()
def success_close_order(self):
    """Успешное закрытие заказа"""
    self._view.close()

```

### **Листинг класса CloseOrderModel**

```

class CloseOrderModel(QObject):

    order_exists = pyqtSignal(bool)
    order_already_close = pyqtSignal()
    order_empty = pyqtSignal()
    order_changed = pyqtSignal(int)
    order_close = pyqtSignal(str)

    def __init__(self):
        super().__init__()
        self.db = db
        self._cur_order = ""

    @property
    def cur_order(self):
        return self._cur_order

    @cur_order.setter
    def cur_order(self, value):
        self._cur_order = value
        self.order_changed.emit(value)
        if self._cur_order == "":
            self.order_empty.emit()
        return

    candidate = self.get_order_by_id(self._cur_order)
    self.order_exists.emit(bool(candidate))

```

```

if candidate and candidate['status'].startswith('Закрыт'):
    self.order_already_close.emit()

def get_order_by_id(self, id):
    """Получить заказ по id"""
    self.db.cursor.execute('SELECT * FROM orders WHERE "id" = %s', (id,))
    result = self.db.cursor.fetchone()
    return result

def close_order_by_id(self, order_id):
    """Закрыть заказ по id"""
    self.db.cursor.execute(f'UPDATE orders '
                          f'SET "status" = \'Закрыт\' '
                          f'WHERE "id" = {order_id}')

    self.db.cursor.execute(f'UPDATE goods '
                          f'SET "remaining_amount" = "remaining_amount" + 1 '
                          f'WHERE "id" IN '
                          f'(SELECT "id" FROM order_goods зт '
                          f'WHERE "order_id" = {order_id})')
    self.db.connection.commit()

    self.order_close.emit(order_id)

```

### **Листинг класса CloseOrderView**

```

class CloseOrderView(QWidget):

    def __init__(self, model: CloseOrderModel, parent=None):
        super().__init__(parent)

        self._parent = parent
        self._model = model
        self._controller = CloseOrderController(self, self._model)
        self._ui = Ui_Form()

        self.setWindowFlags(Qt.Window)

        self.setAttribute(Qt.WA_DeleteOnClose)
        self._ui.setupUi(self)
        self.init_slots()
        self.init_data()

    def init_slots(self):
        """Инициализация слотов"""
        self._ui.order_id_edit.textChanged.connect(self._controller.change_order_id)
        self._model.order_empty.connect(self.on_order_empty)

```

```

self._model.order_exists.connect(self.on_order_exists)
self._model.order_already_close.connect(self.on_order_already_close)
self._ui.close_order_btn.clicked\
    .connect(lambda: self._controller.close_order(self._ui.order_id_edit.text()))
self._model.order_close\
    .connect(self.on_created_order)

def init_data(self):
    """Инициализация данных"""
    stu_id_regx = QRegExp('[0-9]{10}$')
    stu_id_validator = QRegExpValidator(stu_id_regx, self._ui.order_id_edit)
    self._ui.order_id_edit.setValidator(stu_id_validator)

@pyqtSlot(bool)
def on_order_exists(self, order_exists):
    """Слот на реагирование при существовании или отсутствии заказа"""
    if not order_exists:
        self._ui.info_label.setText("Заказ с таким номером отсутствует")
        self._ui.close_order_btn.setDisabled(True)
    else:
        self._ui.info_label.setText("")
        self._ui.close_order_btn.setDisabled(False)

@pyqtSlot()
def on_order_empty(self):
    """Слот на реагирование пустого заказа"""
    self._ui.info_label.setText("")
    self._ui.close_order_btn.setDisabled(True)

@pyqtSlot()
def on_order_already_close(self):
    """Слот на реагирование уже закрытого заказа"""
    self._ui.info_label.setText("Заказ уже закрыт")
    self._ui.close_order_btn.setDisabled(True)

@pyqtSlot(str)
def on_created_order(self, order_id):
    """Слот на реагирование успешного закрытия заказа"""
    msg_box = QMessageBox()
    msg_box.setText(f"Заказ с номером {order_id} закрыт")
    msg_box.setWindowTitle("Успешно")
    msg_box.exec()
    self._controller.success_close_order()

```



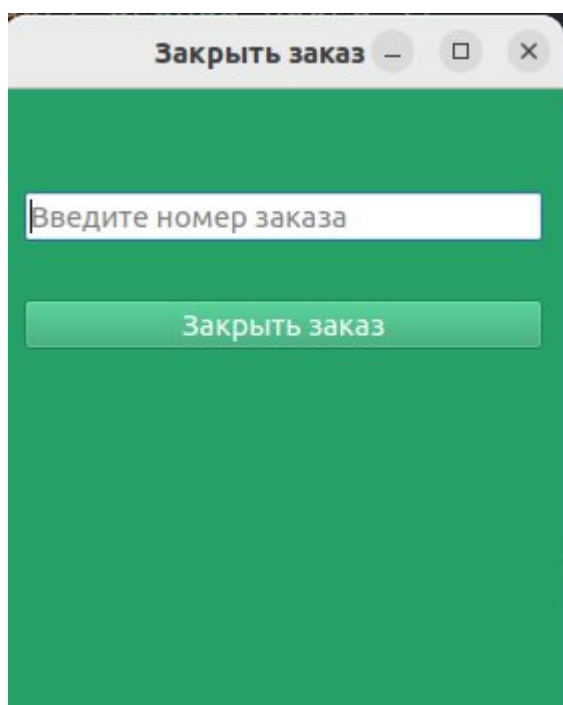


Рис. 13. Окно раздела «Закрытие заказа»

## 5. Тестирование информационной системы

Тестирование – важная часть любой программы контроля качества, поэтому информационная система пройдет этап тестирования, чтобы выявить и искоренить ошибки, мешающие комфортной работе с системой. В момент тестирования приложения были обнаружены некоторые проблемы и неточности.

Для проверки будем использовать автотесты, которые написаны также, как и сам проект на Python с помощью фреймворка unittest.

### Листинг класса Tests

```
class Tests(unittest.TestCase):
```

```
    def setUp(self) -> None:
        """Установка соединения с БД"""
        self.connection = psycopg2.connect(
            dbname='coursework_prdb',
            user='admin',
            password='qwerty',
            host='localhost'
        )
        self.cursor = self.connection.cursor(cursor_factory=RealDictCursor)

    def tearDown(self) -> None:
        """Отключение соединения с БД"""
        self.connection.close()

    def test_authorize_employer(self):
        """Проверка на возможность авторизации сотруднику"""
        self.cursor.execute("SELECT * FROM employers LIMIT 1")
        employer = self.cursor.fetchone()
        if not employer:
            self.fail('no employee found')

        login = employer['login']
        password = employer['password']

        authorize_model = AuthorizeModel()
        result = authorize_model.verify_credentials(login, password)
        self.assertTrue(result)

    def test_non_authorize_employer(self):
        """Проверка авторизации сотрудника при неверных данных"""
        authorize_model = AuthorizeModel()
```

```

result = authorize_model.verify_credentials(
    'nonexistent_email@ne.nel',
    'nonexistent_password'
)
self.assertFalse(result)

def test_filter_goods_category(self):
    """Проверка фильтрации товаров по категории"""
    category = "Стратегические"
    model = GoodsModel()
    model.goods_category = category
    goods = model.get_goods()
    if not len(goods):
        self.fail('no goods found')
    is_all_fits = all([product['category'] == category for product in goods])
    self.assertTrue(is_all_fits)

def test_sort_goods_title_desc(self):
    """Проверка сортировки товаров по имени (по убыванию)"""
    model = GoodsModel()
    model.sort_field = "title"
    model.sort_type = "DESC"
    limit = model.limit

    self.cursor.execute(f"SELECT * from goods ORDER BY title DESC "
                        f"LIMIT {limit}")
    goods_expected = self.cursor.fetchall()
    if not len(goods_expected):
        self.fail('no goods found')

    goods_actual = model.get_goods()

    self.assertEqual(
        [product['id'] for product in goods_expected],
        [product['id'] for product in goods_actual]
    )

```

## 6. Руководство пользователя

### 6.1. Страница авторизации пользователя

Для того, чтобы попасть в личный кабинет и получить доступ ко всему ему функционалу, необходимо совершить вход в систему посредством авторизации на форме приложения.

На форме будет предложено ввести логин и пароль.

После ввода персональных данных необходимо нажать на кнопку «Войти» и, в случае ввода корректных данных пользователь получает доступ к Личному кабинету сотрудника, в противном случае на форму будет выдано предупреждение: «Неверный логин или пароль». При вводе неверных данных более трёх раз подряд, будет вызвано диалоговое окно с вводом капчи. Возможность авторизации будет приостановлена до тех пор, пока не будет введена капча.

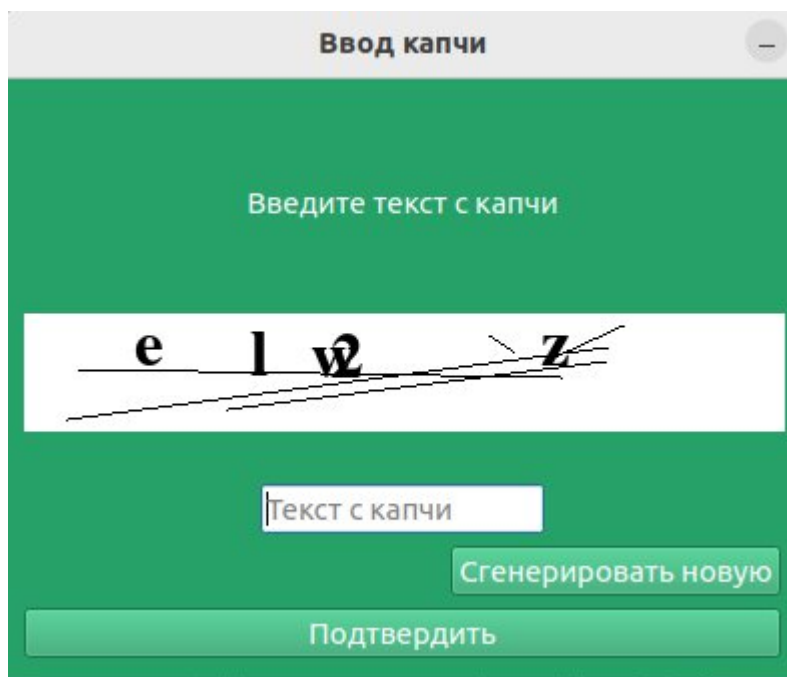


Рис. 14. Окно для ввода капчи

1. Вводим логин и пароль, далее нажимаем на кнопку «Войти» (см. рис. 3);
2. В случае успешной авторизации попадаем на профиль сотрудника (см. рис. 4, 5, 6);

### 6.2. Профиль сотрудника

Находясь на экране «Профиль» сотрудник, в зависимости от его должности, может выбрать любую доступную ему функцию (см. рис. 4, 5, 6).

### 6.3. Добавление нового товара

Находясь на экране «Профиль» сотрудник должен кликнуть по кнопке с названием «Добавить товар», чтобы добавить новый товар. После этого выполняется переход на экран «Добавление товара».

Далее нужно ввести, название товара, его стоимость за час и полную стоимость, описание, выбрать изображение в диалговом окне с выбором файла, указать остаток в настольном клубе, и выбрать категорию, далее нажать на кнопку «Добавить».

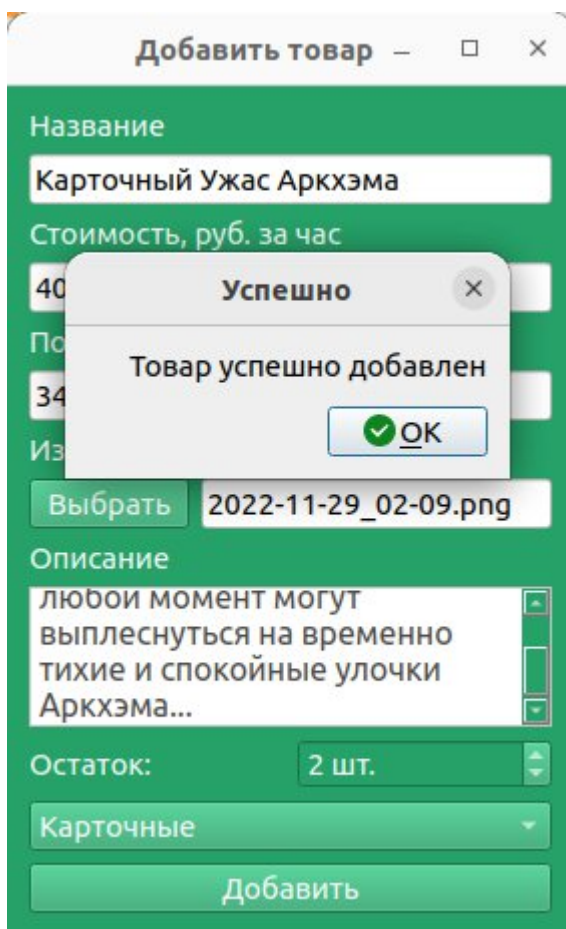


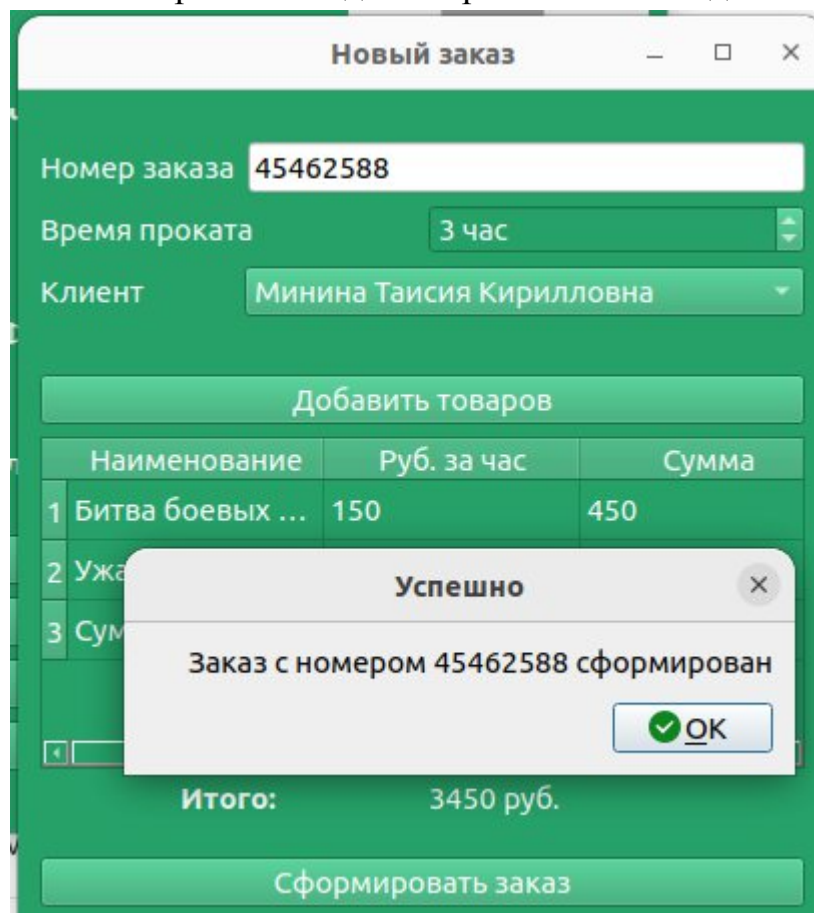
Рис. 15. Добавление нового услуги

### 6.4. Создание нового заказа

Находясь на экране «Профиль» сотрудник должен кликнуть по кнопке с названием «Создать заказ», чтобы создать заказ. После этого выполняется переход на экран «Новый заказа».

На этом экране сотруднику нужно указать номер заказа, время часов проката, выбрать клиента из выпадающего списка, и добавить товаров в

другом окне «Формирование заказа», и нажать на кнопку «Сформировать заказ ». В случае успешного формирования заказа, будет показана надпись «Заказ сформирован», в случае провала будет показана информация об ошибке. Также на данном экране после формирования заказа, генерируется pdf-чек со всеми данными, которые были указаны в заказе. После вызывается диалоговое окно с выбором места для сохранения чека и дальнейшей печати.



Наименование	Руб. за час	Сумма
1 Битва боевых ...	150	450
2 Ужа		
3 Сум		

Рис. 23. Создание нового заказа.

## 6.5. Формирование заказа

Находясь на экране «Новый заказ» сотрудник должен кликнуть по кнопке с названием «Добавить товаров», чтобы добавить товаров заказ. После этого выполняется переход на экран «Формирование заказа».

Здесь мы можем увидеть постраничный список товаров. Для удобства выбора можем выбрать направление сортировки, атрибут, по которому можно сортировать, выбрать категорию товара, а также фильтрация по имени товара. (см. рис. 12)

Чтобы добавить товар в заказ, нужно нажать на кнопку «Добавить в заказ». После этого количество уменьшится на единицу, а кнопка изменит свой текст на «Добавлено»(см. рис. 23). Для того, чтобы убрать товар из

заказа, нужно повторно нажать на эту кнопку. Текст на ней опять станет «Добавить в заказ».

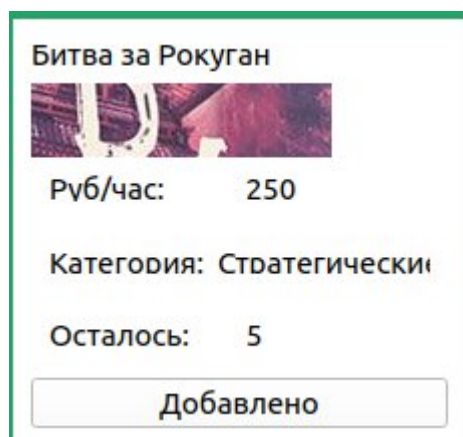


Рис. 24. Добавление нового товара в заказ

После того, как все товары будут выбраны, нужно нажать на кнопку «Готово» в правом верхнем углу. После этого текущее окно будет закрыто, а товары добавятся в таблицу с товарами в окне «Новый заказ».

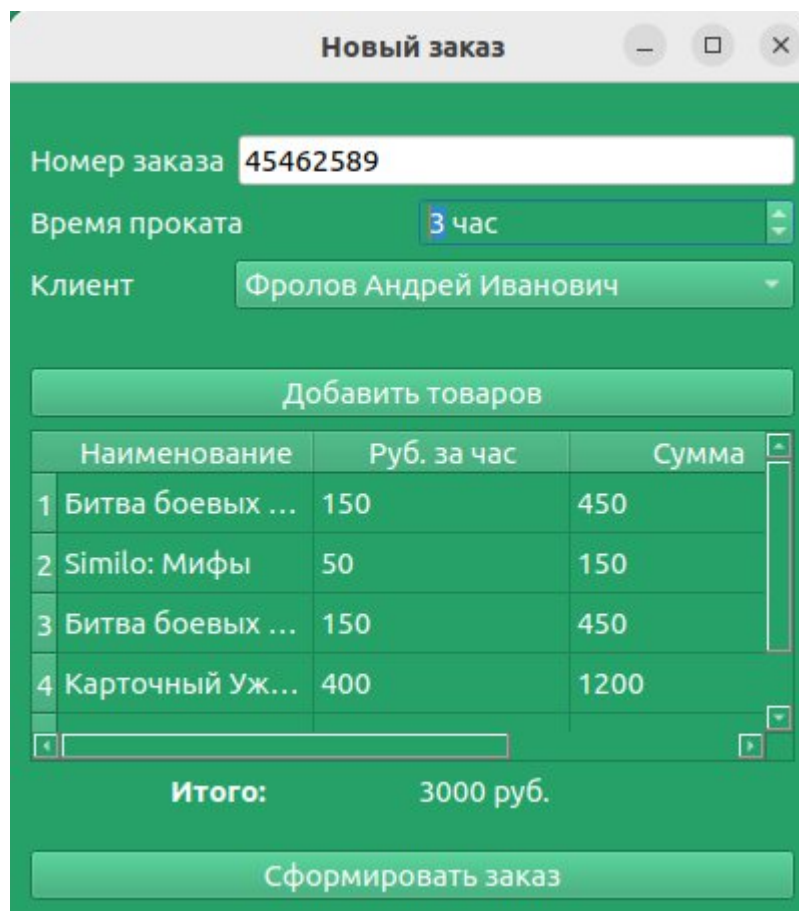


Рис. 25. Список выбранных товаров в окне нового заказа

## 6.6. Просмотр истории входа

Находясь на экране «Профиль» сотрудник должен кликнуть по кнопке с названием «История входа», чтобы просмотреть работников по истории входа. После этого выполняется переход на экран «Просмотр работников по истории входа» (см. рис. 10).

## 6.7. Закрытие заказа

Находясь на экране «Профиль» сотрудник должен кликнуть по кнопке с названием «Закрыть заказ», чтобы закрыть незавершённые заказы. После этого выполняется переход на экран «Закрыть заказ».

На данном окне сотрудники должны написать номер заказа. Если заказ будет уже завершённым или отсутствовать, то система не даст завершить такой заказ, что уменьшает риски ввести неверный номер.

В случае завершения заказа, будет показана надпись «Заказ закрыт». После этого, возвращенные товары станут доступны для следующих заказов.

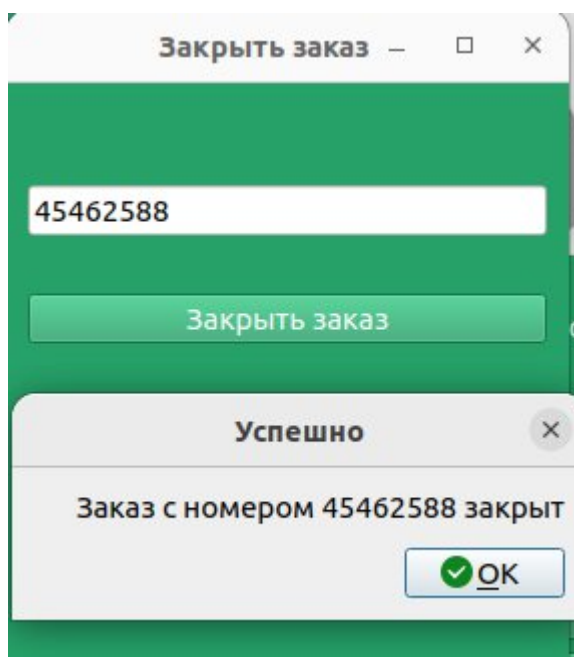


Рис. 26. Завершение заказа



## **7. Заключение**

Итогом данного курсового проекта является приложение для настольного клуба «6 граней».

Разработанный модуль позволяет вести полный контроль за своей рабочей деятельностью внутри настольного клуба, а также управлять заказами и товарами. Проект выполнен в полном соответствии с заданием на курсовое проектирование, весь функционал рабочий и готов для использования.

Таким образом, можно сделать вывод, что во время разработки были достигнуты все цели создания системы и, что её внедрение должно привести к повышению производительности и качества работы настольного клуба.

### **Список использованных источников**

1. PyQt5: первые программы [Электронный ресурс] - Режим доступа: <https://pythonworld.ru/gui/pyqt5-firstprograms.html>
2. Документация Qt for Python [Электронный ресурс] - Режим доступа: <https://doc.qt.io/qtforpython/>
3. Create GUI applications with Python [Электронный ресурс] - Режим доступа: <https://www.pythonguis.com/pyqt5-tutorial/>