Министерство образования и науки Российской Федерации

**Муромский институт (филиал)**

федерального государственного бюджетного образовательного учреждения высшего образования

**«Владимирский государственный университет**

**имени Александра Григорьевича и Николая Григорьевича Столетовых»**

**(МИ ВлГУ)**

Факультет информационных технологий

Кафедра информационных систем

*КУРСОВАЯ*

*РАБОТА*

по курсу Прикладная разработка на java

на тему: Разработка виртуальной модели банкомата на платформе java

Руководитель

к. т. н., доц. каф. ИС

(уч. степень, звание)

Метёлкин А.С.

(оценка) (фамилия, инициалы)

(подпись) (дата)

Члены комиссии Студент ИС-122

(группа)

Юинг В.К.

(подпись) (Ф.И.О.) (фамилия, инициалы)

(подпись) (Ф.И.О.) (подпись) (дата)

Муром 2025

В курсовой работе было разработанно полноценное приложения системы банкомата. Приложение позволяет: для клиента: снимать, пополнять, оплачивать, входить, менять данные; для администратора: инкассация, блокировка банкомата, управление услугами, запуск банкомата.В качестве среды разработки был выбран IntelliJ IDEA, язык программирования Java, графический интерфейс реализован на JavaFX.

• Табл. 8. Ил. 52.

Содержание

[Введение 5](#__RefHeading___Toc500607053)

[1.Анализ технического задания 6](#__RefHeading___Toc3737_53164996)

[1.1 Описание предметной области и существующих особенностей 6](#__RefHeading___Toc3743_53164996)

[1.4 Описание вариантов использования 8](#__RefHeading___Toc3739_53164996)

[1.5 Вопросы безопасности 9](#__RefHeading___Toc3749_53164996)

[1.6 Требования к разрабатываемой системе 10](#__RefHeading___Toc3741_53164996)

[1.7 Обзор и анализ аналогов 11](#__RefHeading___Toc23291_889428090)

[2. Разработка базы данных 13](#__RefHeading___Toc23293_889428090)

[2.1 Таблицы созданной системы для банкомата 14](#__RefHeading___Toc23297_889428090)

[2.2 Процедуры созданной системы для банкомата 17](#__RefHeading___Toc13764_1478057374)

[3. Разработка приложения 19](#__RefHeading___Toc23311_889428090)

[3.1 Проектирование приложения 20](#__RefHeading___Toc23313_889428090)

[3.2 Разработка приложения на языке Java 22](#__RefHeading___Toc23315_889428090)

[3.2.1 Разработка окна авторизации технического сотрудника банкомата 23](#__RefHeading___Toc23317_889428090)

[3.2.2 Разработка сцены технического меню банкомата 26](#__RefHeading___Toc13766_1478057374)

[3.2.3 Разработка сцены инкосаторского меню 27](#__RefHeading___Toc13768_1478057374)

[3.2.4 Разработка сцены управления услугами 34](#__RefHeading___Toc13770_1478057374)

[3.2.5 Разработка сцены блокировки банкомата 39](#__RefHeading___Toc13772_1478057374)

[3.2.6 Разработка сцены ввода номера карты для авторизации. 40](#__RefHeading___Toc13774_1478057374)

[3.2.7 Разработка сцены ввода пин-кода для авторизации. 42](#__RefHeading___Toc13776_1478057374)

[3.2.8 Разработка сцены главного меню клиента 44](#__RefHeading___Toc13778_1478057374)

[3.2.9 Разработка сцены пополнения счета через банкомат 45](#__RefHeading___Toc13780_1478057374)

[3.2.10 Разработка сцены снятия купюр со счета клиента 47](#__RefHeading___Toc13782_1478057374)

[3.2.11 Разработка сцены выбора услуги и выполнение оплаты по ним. 49](#__RefHeading___Toc13784_1478057374)

[3.2.12 Разработка сцены настроек и модуля смены пин-кода 53](#__RefHeading___Toc13786_1478057374)

[3.2.13 Разработка сцены авторизации через биометрию 57](#__RefHeading___Toc13788_1478057374)

[3.3 Разработка прочих модулей для разработки приложения. 59](#__RefHeading___Toc23315_8894280901)

[3.3.1 Разработка модуля автосоздания базы данных. 59](#__RefHeading___Toc13788_14780573741)

[3.3.2 Разработка модуля сессии клиента 61](#__RefHeading___Toc13788_147805737411)

[3.4 Результаты разработки приложения 62](#__RefHeading___Toc167545_1261912567)

[4. Тестирование 63](#__RefHeading___Toc167547_1261912567)

[4.1 Тестирование пользовательского интерфейса 64](#__RefHeading___Toc167549_1261912567)

[4.1.1 Окно авторизации для технического специалиста. 64](#__RefHeading___Toc167551_1261912567)

[4.1.2 Сцена администратора и его меню 64](#__RefHeading___Toc167553_1261912567)

[4.1.3 Сцена инкассации 65](#__RefHeading___Toc167555_1261912567)

[4.1.4 Сцена управления услугами 66](#__RefHeading___Toc167557_1261912567)

[4.1.5 Сцена блоикровки банкомата 66](#__RefHeading___Toc167559_1261912567)

[4.1.6 Сцена запуска привественного окна банкомата 66](#__RefHeading___Toc167561_1261912567)

[4.1.7 Сцена авторизации по номеру карты 67](#__RefHeading___Toc167563_1261912567)

[4.1.8 Сцена биометрии для авторизации 67](#__RefHeading___Toc167563_12619125671)

[4.1.9 Сцена сцены ввода пин-кода 67](#__RefHeading___Toc167563_126191256711)

[4.1.10 Сцена сцены ввода значений для вывода среств 68](#__RefHeading___Toc167563_1261912567111)

[4.1.10 Сцена сцены ввода значений для счета и оплаты услуг 68](#__RefHeading___Toc167563_12619125671111)

[4.2 Тестирование взаимодействия с базой данных 69](#__RefHeading___Toc167565_1261912567)

[4.4. Результаты тестирования 70](#__RefHeading___Toc167569_1261912567)

[Заключение 73](#__RefHeading___Toc167571_1261912567)

[Список литературы 74](#__RefHeading___Toc167573_1261912567)

[ПРИЛОЖЕНИЕ А — Источник исходного кода приложения 75](#__RefHeading___Toc168851_1261912567)

МИВУ 09.03.02 – 00.000 ПЗ

75

4

# Введение

Моделирование работы банкомата является важным аспектом обеспечения надежности, безопасности и эффективности банковских услуг. В условиях стремительного развития цифровых технологий и увеличения объема безналичных операций, банкоматы остаются ключевым элементом инфраструктуры финансовых организаций, обеспечивая клиентам круглосуточный доступ к средствам и сервисам. Создание программной модели работы банкомата позволяет заранее выявлять потенциальные ошибки, оптимизировать пользовательский интерфейс.

Тема данной курсовой работы "Разработка приложения по моделированию работы банкомата" актуальна, так как моделирование позволяет минимизировать риски сбоев в работе оборудования, повысить уровень безопасности транзакций и улучшить качество клиентского сервиса.

Целью работы является создание программного приложения, которое моделирует основные функции банкомата: проверку баланса, снятие и пополнение счета а также обработку ошибочных действий пользователей и внешних сбоев. Приложение должно обеспечивать визуализацию процессов, логирование операций и интеграцию с условной банковской системой для имитации реальных транзакций. Для достижения поставленной цели необходимо решить следующие задачи :

* Спроектировать архитектуру приложения, включая модули взаимодействия с пользователем, обработки запросов и симуляции банковских операций.
* Разработать алгоритмы обработки транзакций, управления наличностью и проверки аутентификации.
* Создать интуитивный интерфейс для имитации взаимодействия клиента с банкоматом.
* Выполнить тестирование и отладку приложения для обеспечения корректности и отказоустойчивости.

# **1.Анализ технического задания**

## 1.1 Описание предметной области и существующих особенностей

Система моделирования работы банкомата предназначена для имитации ключевых функций и процессов, реализуемых в реальных устройствах самообслуживания банков. Она позволяет симулировать взаимодействие клиента с банкоматом, включая аутентификацию через банковскую карту и пин-код, проверку баланса, выполнение операций снятия и пополнения счета, переводы между счетами, а также обработку ошибочных действий и внешних сбоев. Кроме того, система обеспечивает логирование всех операций, контроль остатка наличности в банкомате и интеграцию с условной банковской системой для проверки корректности транзакций.

Основные задачи системы:

1. Моделирование аутентификации клиента (проверка карты, ввод пин-кода, обработка ошибок).
2. Имитация финансовых операций : проверка баланса, снятие и пополнение счета, переводы.
3. Учет наличности в банкомате (контроль остатка банкнот, ограничений на максимальную сумму снятия).
4. Обработка сценариев сбоев : недостаточный баланс, неправильный пин-код, отсутствие связи с банком, технические ошибки.
5. Логирование всех операций (время, тип операции, результат, ошибки).
6. Тестирование пользовательского интерфейса (имитация навигации по меню, ввод данных, отображение подтверждений).
7. Интеграция с эмулятором банковской системы для проверки актуальности данных и корректности транзакций.
8. Существующие особенности моделирования:
9. Обеспечение безопасности операций через имитацию шифрования данных и проверку подлинности.
10. Поддержка различных сценариев взаимодействия (стандартные операции, экстренные ситуации, повторные попытки).
11. Возможность настройки параметров модели (лимиты на операции, курсы валют, количество банкнот).
12. Визуализация процессов для анализа работы банкомата и выявления узких мест в алгоритмах.

Этот подход позволяет создать гибкий инструмент для тестирования функциональности банкоматов, обучения специалистов и разработки безопасных решений в сфере финансовых технологий.

## **1.2 Требования к разрабатываемой системе**

**Проект представляет собой виртуальную модель банкомата, реализованную на Java, с использованием объектно-ориентированного подхода. Система имитирует основные операции реального банкомата, включая авторизацию, управление счетом, ввод/вывод виртуальных купюр и взаимодействие с внешними сервисами (например, оплата услуг). Для обеспечения модульности и тестируемости применяются принципы SOLID, а ключевые компоненты (баланс, транзакции, авторизация) реализуются как независимые модули.**

**Функциональные требования**:

* Возмодность ввода и вывода купюр различных номиналов в банкомат
* Возможность управления услугами банкомата
* Возможность авторизации к счету через ввода номера карты и пин кода
* Возможность пополнения счета через генерацию купюр на счет (Моделирование ввода денежных средств)
* Возможность снятия денежных средств через удаление их из банкомата (Моделирование вывода денежных средств)
* Возможность оплаты какой либо услуги со счета клиента (Ввод нмоера счета, ввод суммы оплаты, получение подтверждения)
* Просмотр своих данных клиентом через систему.
* Возможность смены пин кода клиента через систему

**Не функциональные требования:**

* Обеспечение высокой скорости работы системы при обработке больших объемов данных.
* Обеспечение надежной защиты данных от несанкционированного доступа, потери и повреждения.
* Интуитивно понятный и удобный интерфейс для пользователей с различным уровнем подготовки.
* Предоставление справочной и иной информации для клиентов и сотрудников технической службы банкомата

## 1.3 Обзор и анализ аналогов

**Для разработки виртуальной модели банкомата были проанализированы существующие решения. Основные категории аналогов, их функционал и ограничения представлены в таблице ниже. На их основе определены ключевые преимущества разрабатываемой системы.**

Таблица 1. Таблица сравнения аналогов программ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Критерий / Категория | Коммерческие симуляторы  (Diebold Nixdorf, NCR) | Учебные проекты  (GitHub: ATM Simulator) | Тестовые среды  (IBM FTM, HDFC Test Env.) | Демонстрационные приложения  (ATM Simulation Game) | Разрабатываемая система |
| Целевое назначение | Тестирование ПО банкоматов | Обучение основам банковских операций | Тестирование интеграции и безопасности | Обучение пользователей и геймификация | Учебно-демонстрационное (Java-модель для изучения ООП и банковских процессов) |
| Доступность кода | Закрытый | Открытый (GitHub) | Закрытый | Закрытый | Открытый код + документация |
| Интеграция с API/БД | Реальная (ISO 8583, банковские протоколы) | Эмуляция | Высокий уровень | Только визуализация | Эмуляция + модульность для будущей интеграции |
| Безопасность | Продвинутая (шифрование, защита от атак) | Базовые проверки | Профессиональная | Минимальная | Базовые проверки |
| Обработка сбоев | Полная (потеря связи, нехватка купюр) | Упрощенная | Реалистичная | Отсутствует | Симуляция сбоев (например, недостаток средств в банкомате) |
| Стоимость | Высокая (лицензии) | Бесплатно | Очень высокая | Бесплатно/демо-версии | Бесплатно (учебный проект) |
| Простота настройки | Требует экспертизы | Минимальная | Сложная | Простая | Гибкая конфигурация (файлы настроек, in-memory данные) |
| Моделирование транзакций | Реальное | Базовое | Реальное | Игровое | Детализированное (жадный алгоритм выдачи купюр, логирование) |
| Учебная применимость | Низкая | Средняя | Низкая | Средняя | Высокая (подробные комментарии, тесты) |

Разрабатываемая система объединяет преимущества open-source проектов (доступность, простота) и коммерческих решений (реализм, безопасность). Ее ключевое отличие — фокус на образовательные цели с детальным моделированием банковских процессов, что делает ее оптимальным инструментом для изучения Java и основ fintech.

## 1.4 Выбор технологий и инструментов

На основе требований к системе (безопасность, производительность, удобство разработки) и функциональных потребностей были выбраны следующие технологии и инструменты. В таблицах ниже приведено сравнение альтернатив и обоснование выбора.

1. Сравнение языков программирования

Таблица 2. Сравнение языков программирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерий | Java 17 (LTS) | Java 20 | Java 22 | Java 23 |
| LTS (Long-Term Support) | Поддержка до 2029 | Промежуточный релиз | Промежуточный релиз | Промежуточный релиз |
| Безопасность | Базовые алгоритмы (AES, SHA) | Улучшенный SecureRandom | Поддержка TLS 1.3 | Новые API шифрования (ChaCha20, Poly1305) |
| Многопоточность | Virtual Threads (превью) | Virtual Threads (доработки) | Structured Concurrency (превью) | Оптимизация виртуальных потоков |
| Синтаксис | Паттерны в switch (превью | Паттерны для instanceof | Unnamed Patterns (превью) | Завершенные паттерны, String Templates |
| Производительность | Стабильная | Улучшения в GC (ZGC) | Оптимизация работы с памятью | Ускорение JIT-компиляции |
| Инструменты (IDE) | Полная поддержка | Экспериментальные фичи требуют обновлений | Частичная поддержка новых фич | Полная интеграция с IntelliJ IDEA 2023+ |

Переход на Java 23 обусловлен необходимостью использования современных технологий при сохранении совместимости с исходной версией проекта (Java 17). Несмотря на отсутствие долгосрочной поддержки (LTS), Java 23 предоставляет ключевые улучшения: оптимизированные виртуальные потоки для эффективной многопоточности, новые алгоритмы шифрования (ChaCha20-Poly1305) для защиты данных и завершенные синтаксические паттерны, упрощающие разработку. Эти инновации позволяют демонстрировать передовые практики в образовательном контексте, сохраняя стабильность кода и обратную совместимость с Java 17, что делает версию 23 оптимальным выбором для баланса между актуальностью и надежностью.

1. Сравнение инструментов разработки

Таблица 3. Сравнение инструментов разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | JavaFX | Swing | SWT (Eclipse) |
| Типичное применение | Десктоп-приложения с богатым интерфейсом (банкоматы, CRM, аналитика) | Legacy-системы, инструменты администрирования, простые утилиты | Приложения, требующие интеграции с нативными компонентами ОС |
| Современные функции | Поддержка CSS, анимаций, 3D-графики, FXML | Ограниченная стилизация, устаревший API | Нет встроенной поддержки CSS/анимаций, фокус на нативные виджеты |
| Производительность | Высокая (хардварное ускорение графики) | Средняя (полностью на Java) | Высокая (нативные компоненты) |
| Стилизация | Гибкая (CSS, темы, кастомные компоненты) | Базовые возможности (AWT-стили) | Зависит от ОС, сложное кастомизирование |
| Интеграция с IDE | Полная поддержка в IntelliJ IDEA (Scene Builder, FXML-редактор) | Базовая поддержка | Требует Eclipse IDE для удобной разработки |
| Сообщество | Активное (OpenJFX, сообщество GitHub) | Устаревшее (поддержка в режиме maintenance) | Узкоспециализированное (Eclipse-экосистема) |

JavaFX выбран для проекта, так как он специализирован на создании современных десктоп-приложений с интуитивным и визуально привлекательным интерфейсом. В отличие от Swing, который используется преимущественно в устаревших системах, JavaFX поддерживает CSS-стилизацию, анимации и FXML, что позволяет точно воспроизвести интерфейс реального банкомата (кнопки, экраны ввода, индикацию операций). Утверждение о применении Swing для веб-приложений не совсем корректно: веб-разработка на Java требует других технологий (например, JSF или Spring MVC), тогда как Swing и JavaFX ориентированы исключительно на десктоп. Выбор JavaFX также обусловлен его кроссплатформенностью, активным сообществом и удобством интеграции с IntelliJ IDEA, что ускоряет разработку и тестирование.

1. Сравнение СУБД баз данных
2. Таблица 4. Сравнение СУБД баз данных

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | Red Expert | MySQL | PostgresSQL |
| Тип СУБД | Реляционная (SQL), ACID-транзакции | Реляционная (SQL), ACID-транзакции | Реляционная (SQL), ACID-транзакции |
| Безопасность | Высокая (шифрование на уровне данных, ролевой доступ, аудит транзакций) | Базовые настройки (SSL, ролевой доступ) | Расширенные функции (SSL, SE-PostgreSQL, row-level security) |
| Производительность | Оптимизирована для сложных транзакций (финансовые операции) | Высокая для OLTP, слабее в аналитических запросах | Высокая для сложных запросов и аналитики |
| Транзакции | Полная поддержка ACID, изоляция уровня SERIALIZABLE | ACID, но уровень изоляции по умолчанию REPEATABLE READ | ACID, гибкие уровни изоляции (включая SERIALIZABLE) |
| Интеграция с Java | Нативная поддержка JDBC, ORM-фреймворков | JDBC, Hibernate, Spring Data | JDBC, Hibernate, расширенные типы данных (JSONB, геоданные) |
| Сообщество/поддержка | Коммерческая поддержка, фокус на банковские системы | Огромное open-source сообщество | Активное сообщество, поддержка enterprise-решений |

**Red Expert выбран для проекта виртуального банкомата, так как он специализирован на задачах, требующих максимальной безопасности и надежности транзакций. В отличие от MySQL, который оптимизирован для OLTP, но имеет упрощенные настройки безопасности, Red Expert предоставляет встроенное шифрование данных на уровне полей и детальный аудит операций, что критично для финансовых систем. PostgreSQL, хотя и поддерживает расширенные функции (например, row-level security), требует ручной настройки для достижения уровня безопасности, который Red Expert предлагает «из коробки».**

# **2. Проектирование программы**

## **2.1 Описание вариантов использования**

Проектируемая система предполагает два ключевых типа пользователей, каждый из которых взаимодействует с банкоматом в рамках специфических сценариев. Варианты использования детализируют действия участников, обеспечивая понимание функциональных требований и логики работы приложения.

1. Технический работник банковской системы

Технический специалист отвечает за подготовку банкомата к эксплуатации, управление его состоянием и устранение нештатных ситуаций. Процесс начинается с запуска приложения, в ходе которого система инициализирует виртуальные модули (базу данных, интерфейс, сервисы). Далее работник проходит авторизацию в администраторском меню, используя уникальные учетные данные. Это обеспечивает защиту от несанкционированного доступа к критическим функциям.

После входа в систему специалист проверяет состояние инкассации — анализирует остаток виртуальных купюр, имитирующих наличность в банкомате. Данный этап включает возможность ручного пополнения или изъятия средств для балансировки. Отдельная вкладка позволяет управлять доступностью услуг (платежи, переводы), активируя или деактивируя их в зависимости от требований банка.

Перед переходом в рабочий режим выполняется тестирование на демо-аккаунте. Это позволяет убедиться в корректности обработки транзакций, логирования операций и работы интерфейса. В случае обнаружения сбоев технический работник может выполнить перезапуск приложения без потери данных. Финальным этапом является окончательный запуск банкомата, переводящий систему в режим обслуживания клиентов.

2. Клиент банка

Клиент взаимодействует с банкоматом для выполнения финансовых операций без участия персонала. Начальным этапом является авторизация — пользователь вводит номер карты и PIN-код, после чего система проверяет данные в виртуальной базе. Успешная аутентификация открывает доступ к меню операций, включающему:

* Проверку баланса — отображение текущего состояния счета;
* Снятие средств — выбор суммы с последующей симуляцией выдачи купюр (система проверяет доступность средств и номиналов);
* Пополнение счета — генерация виртуальных купюр, увеличивающих баланс;
* Оплату услуг — ввод реквизитов получателя и суммы с подтверждением транзакции;
* Смену PIN-кода — двухэтапная проверка (старый → новый код) с шифрованием данных.

Каждая операция сопровождается интерактивными подсказками, направляющими пользователя. Например, при снятии средств система запрашивает сумму, проверяет ее корректность, а затем выводит информацию о комиссии (если применимо). После завершения действия клиент получает виртуальный чек с деталями операции (дата, сумма, остаток), который сохраняется в истории транзакций.

Система автоматически завершает сеанс при бездействии или после явного выхода, возвращаясь в исходное состояние. Это предотвращает несанкционированный доступ к счету. В случае ошибок (недостаток средств, неверный PIN) выводятся соответствующие уведомления с рекомендациями, например: «Превышен лимит попыток ввода. Обратитесь в поддержку».

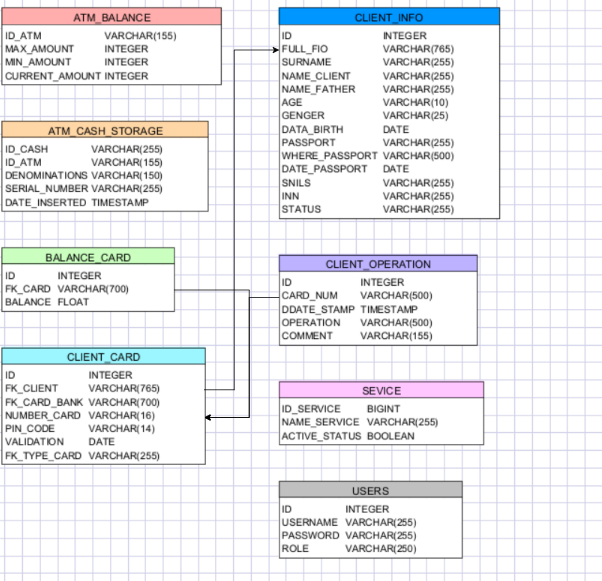
## 2.2 Проектирование базы данных

Основной целью при разработке базы данных было создание структуры, обеспечивающей эффективное хранение и обработку данных, связанных с моделью банкомата. Это включает информацию о клиентах, банковских картах, транзакциях, остатке наличности в банкомате, логах операций и параметрах безопасности. Для моделирования реальных сценариев взаимодействия с банкоматом были определены две роли:

Клиент банка : ограниченный доступ (проверка баланса, снятие/пополнение счета, оплата счетов).

Администратор банкомата : расширенные права (управление наличностью, настройка параметров, просмотр логов).

Для реализации базы данных использовалась система Red Expert — отечественное программное обеспечение, предоставляющее удобные инструменты для проектирования таблиц, связей, пользовательских ролей и настройки безопасности. ER-диаграмма (Entity-Relationship Diagram) графически отображает структуру базы данных, включая сущности, их атрибуты и взаимосвязи.

Рисунок 1 — ER-диаграмма таблиц в базе данных для банкомата

На основе ER-диаграммы были разработаны следующие таблицы базы данных, каждая из которых выполняет определенную функцию в моделировании работы банкомата:

Таблица 5. Таблица ATM\_BALANCE. Хранит информацию о балансе банкомата

|  |  |
| --- | --- |
| Название поля | Значение |
| ID\_ATM | Первичный ключ таблицы |
| MAX\_AMOUNT | Значение сколько максимально можно хранить в банкомате наличных (Сумма всех номиналов в банкомате) |
| MIN\_AMOUNT | Значение сколько минимально должен хранить банкомат для полноценной работы банкомата |
| CURRENT\_AMOUNT | Текущее кол-во денежных средств в банкомате (Сумма) |

Таблица 6. Таблица ATM\_CASH\_STORAGE. Хранит купюры в виде строк

|  |  |
| --- | --- |
| Название поля | Значение |
| ID\_CASH | Первичный ключ таблицы |
| ID\_ATM | Номер банкомата где храниться |
| DENOMINATIONS | Номинал купюры |
| SERIAL\_NUMBER | Уникальный номер купюры |
| DATE\_INSERTED | Дата входа в банкомат |

Таблица 7. Таблица BALANCE\_CARD. Хранит баланс счета банковских карт клиентов.

|  |  |
| --- | --- |
| Название поля | Значение |
| ID | Первичный ключ таблицы |
| FK\_CARD | Номер банковской карточки клиента |
| BALANCE | Баланс карточки |

Таблица 8. Таблица CLIENT\_CARD. Хранит данные банковских карт клиентов

|  |  |
| --- | --- |
| Название поля | Значение |
| ID | Первичный ключ таблицы |
| FK\_CLIENT | Данные клиента карты (ФИО) |
| FK\_CARD\_BANK | Название банка карты |
| NUMBER\_CARD | Номер карты из 16 цифр |
| PIN\_CODE | Ключ для входа из 4 цифр |
| VALIDATION | Срок действия банковской карты |
| FK\_TYPE\_CARD | Тип банковской карты |

Таблица 9. Таблица CLIENT\_INFO. Хранит данные клиента банковской системы

|  |  |
| --- | --- |
| Название поля | Значение |
| ID | Первичный ключ таблицы |
| FULL\_FIO | Полное предсталвение клиента (ФИО) |
| SURNAME | Фамилия клиента |
| NAME\_CLIENT | Имя клиента |
| NAME\_FATHER | Отчество клиента |
| AGE | Возраст клиента |
| GENDER | Пол клиента |
| DATA\_BIRTH | Дата рождения клиента |
| PASSPORT | Номер паспорта клиента |
| WHERE\_PASSPORT | Место получения клиента |
| DATE\_PASSPORT | Дата получения паспорта |
| SNILS | Номер снилса |
| INN | Номер ИНН |
| STATUS | Статус клиента (Действующий, Отключенный, Заблокированный) |

Таблица 10. Таблица CLIENT\_OPERATION. Хранит логгированные данные об операциях карт в банкомате.

|  |  |
| --- | --- |
| Название поля | Значение |
| ID | Первичный ключ таблицы |
| CARD\_NUM | Номер карты клиента |
| DDATE\_STAMP | Время выполения операции |
| OPERATION | Тип операции |
| COMMENT | Результат операции |

Таблица 11. Таблица SEVICE. Хранит услуги для банкомата

|  |  |
| --- | --- |
| Название поля | Значение |
| ID\_SERVICE | Первичный ключ таблицы |
| NAME\_SERVICE | Наименование сервиса |
| ACTIVE\_STATUS | Статус активности (Доступен/Отключен) |

Таблица 12. Таблица USERS. Хранит данные для авторизации технических работников банковской ситемы.

|  |  |
| --- | --- |
| Название поля | Значение |
| ID | Первичный ключ таблицы |
| USERNAME | Уникальный индетификатор для входа |
| PASSWORD | Пароль для авторизации в систему |
| ROLE | Роль сотрудника |

### 2.2.1 Процедуры созданной системы для банкомата

Процедуры в базе данных реализуют бизнес-логику моделирования работы банкомата, обеспечивая безопасность, целостность данных и управление операциями. Ниже приведены ключевые группы процедур и их краткое описание:

1. Аутентификация и проверка данных

* + - AUTHENTICATE\_USER — проверяет учетные данные пользователя (логин/пароль) для входа в систему (администратор или сотрудник).
    - CHECK\_PIN\_CODE — сравнивает введенный ПИН-код с значением в базе данных.
    - CHECK\_CARD\_IN\_DATABASE — проверяет наличие карты в системе по её номеру.

2. Управление наличностью в банкомате

* + - ADD\_CASH\_TO\_STORAGE — добавляет банкноты в кассеты банкомата (имитация пополнения).
    - GET\_ATM\_CASH\_STORAGE — возвращает список банкнот в банкомате с их номиналом и серийными номерами.
    - GET\_ATM\_CASH\_QUANTITY — отображает количество банкнот по номиналам.
    - UPDATE\_CURRENT\_AMOUNT — обновляет текущий баланс банкомата после операций снятия или пополнения.
    - DELETE\_ATM\_CASH\_BY\_SERIAL — удаляет банкноту по серийному номеру (например, при выводе из банкомата).

3. Операции с банковскими картами

* + - GET\_BALANCE — возвращает текущий баланс карты.
    - UPDATE\_BALANCE — корректирует баланс карты после снятия, пополнения или перевода.
    - GET\_CLIENT\_INFO — извлекает персональные данные клиента по номеру карты (ФИО, паспортные данные и т.д.).
    - GET\_CLIENT\_OPERATIONS — возвращает историю операций клиента (снятие, пополнение, ошибки).

4. Логирование и отчетность

* + - LOG\_OPERATION — записывает операции клиента в журнал (например, успешное снятие средств или ошибку ввода ПИН-кода).
    - GET\_TOTAL\_AMOUNT\_IN\_ATM — рассчитывает общую сумму наличности в банкомате.
    - LOAD\_ATM\_BALANCE — загружает параметры баланса банкомата (минимальный/максимальный остаток).

5. Обслуживание банкомата

* + - GET\_ACTIVE\_SERVICES — возвращает список доступных услуг банкомата (например, «Снятие наличных», «Проверка баланса»).
    - UPDATE\_SERVICE\_STATUS — включает/выключает конкретную услугу (например, отключение пополнения при техническом обслуживании).
    - INSERT\_SERVICE — добавляет новую услугу в систему.
    - DELETE\_SERVICE\_BY\_NAME — удаляет услугу по названию.

6. Вспомогательные процедуры

* + - GET\_MAX\_ATM\_CASH\_ID — возвращает максимальный идентификатор банкноты (для генерации новых записей).
    - CHECK\_NOMINAL\_EXISTS — проверяет наличие номинала в справочнике (DIC\_NOMINAL).
    - COUNT\_SERVICES\_BY\_NAME — подсчитывает количество активных услуг.

## 2.3 Проектирование приложения.

Приложение спроектировано с использованием объектно-ориентированного подхода, где каждый функциональный модуль инкапсулирован в отдельный класс Java. Архитектура строится на принципах модульности и разделения ответственности: классы отвечают за формирование графических интерфейсов (сцен), обработку пользовательских действий и бизнес-логику. Взаимодействие между компонентами организовано через события, инициируемые кнопками, что обеспечивает динамическую навигацию между экранами.

Приложение построено на основе модульной объектно-ориентированной архитектуры, где каждый компонент инкапсулирует строго определенную функциональность. Система разделена на три ключевых слоя: презентационный (интерфейс), бизнес-логика (обработка операций) и уровень данных (хранение и управление информацией). Взаимодействие между слоями организовано через событийно-ориентированную модель, что обеспечивает гибкость и масштабируемость.

Презентационный слой реализован с использованием графических сцен, каждая из которых отвечает за отображение конкретного состояния интерфейса (авторизация, меню операций, управление транзакциями). Элементы интерфейса (кнопки, поля ввода, текстовые блоки) динамически генерируются и связываются с обработчиками событий. Переход между сценами осуществляется через централизованный механизм навигации, который заменяет текущее представление в зависимости от действий пользователя. Например, нажатие кнопки "Снять средства" инициирует загрузку новой сцены, скрывая предыдущую, без перезапуска основного окна приложения.

Слой бизнес-логики содержит модули, отвечающие за выполнение операций: аутентификацию, проведение транзакций, валидацию данных, управление балансом. Эти модули изолированы от интерфейса и взаимодействуют с презентационным слоем через абстрактные интерфейсы, что позволяет легко модифицировать или заменять компоненты. Например, при запросе на снятие средств интерфейс передает сумму в сервис транзакций, который проверяет доступность денег, применяет комиссии и возвращает результат для отображения. Все операции выполняются атомарно, гарантируя целостность данных даже в случае сбоев.

Уровень данных обеспечивает взаимодействие с хранилищем информации, включая виртуальные счета пользователей, историю операций и настройки банкомата. Для доступа к данным используется абстракция репозитория, которая скрывает детали реализации (работа с SQL-запросами или in-memory структурами). Это позволяет подключать различные источники данных без изменения кода бизнес-логики. Например, модуль управления балансом запрашивает актуальное состояние счета через унифицированный API, не зная, хранятся ли данные в реляционной базе или временном кэше.

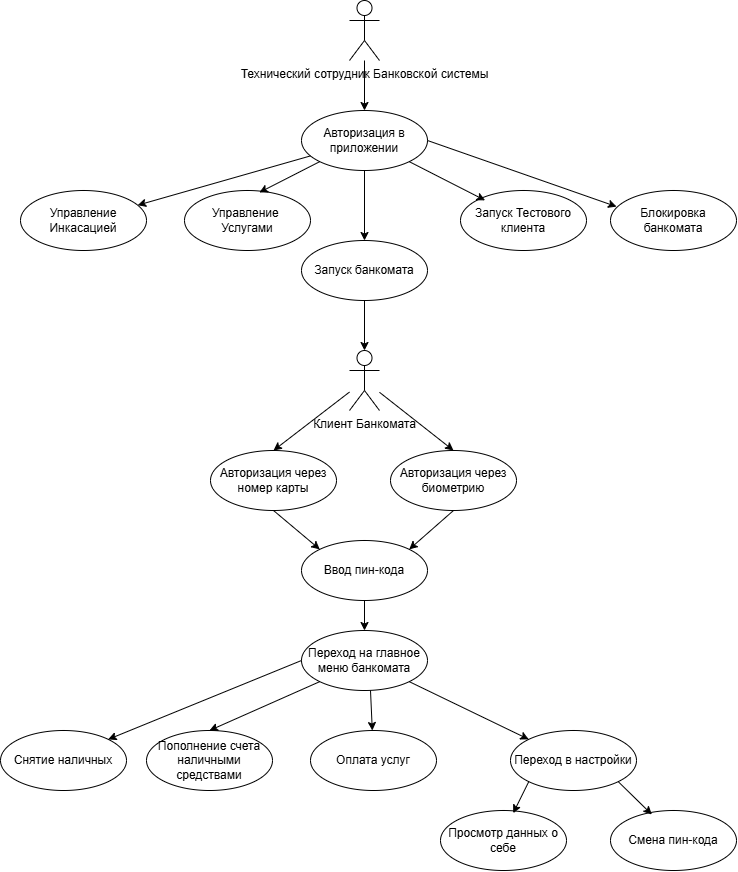
Взаимодействие компонентов строится на принципах слабой связанности. События, инициируемые пользователем (нажатие кнопки, ввод текста), передаются в контроллеры, которые делегируют выполнение соответствующим сервисам. Результаты операций возвращаются в виде объектов-ответов, содержащих данные или коды ошибок. Например, при попытке ввода неверного PIN-кода сервис аутентификации генерирует ошибку, которая преобразуется в понятное уведомление на интерфейсе.

Безопасность реализована через шифрование критических данных (например, PIN-кодов) на уровне бизнес-логики и валидацию вводимых значений перед их обработкой. Механизмы защиты от несанкционированного доступа включают автоматическое завершение сеанса при бездействии, ограничение попыток ввода и журналирование всех операций для последующего аудита.

Обработка ошибок интегрирована во все слои приложения. Некорректные действия пользователя, сбои в работе сервисов или проблемы с данными перехватываются глобальными обработчиками, которые формируют стандартизированные сообщения об ошибках и восстанавливают стабильное состояние системы. Например, при потере соединения с хранилищем данных приложение переключается в аварийный режим, уведомляя пользователя о временной недоступности сервиса.

Архитектура приложения обеспечивает высокую сопровождаемость: добавление новых функций (например, поддержка дополнительных валют) требует создания отдельных модулей без изменения существующего кода. Тестирование упрощается за счет изолированности компонентов — каждый слой может быть проверен автономно с использованием mock-объектов. Такой подход позволяет сохранить четкость структуры даже при масштабировании системы.

Чтобы визуализировать работу тех или иных ролей стоит создать диаграму прецедентов. Она позволит наглядно показать как будет работать авторизация за администратора или пользователя.

Рисунок 2 — Диаграмма прецедентов в приложении

Когда мы спроектировали действия пользователей и логигу приложения, можно начинать реализацию приложения. При разработке стоит учитывать не только задачу, но и методы которые будем использовать также быть обоснованны и реализованны до ума.

# **3. Разработка приложения**

Разработка приложения по моделированию работы банкомата включала два ключевых этапа: проектирование пользовательского интерфейса (UI) и реализацию внутренней логики , имитирующей реальные процессы работы банкомата. Эти этапы были тесно взаимосвязаны, так как интерфейс разрабатывался с учетом функциональных требований (например, аутентификация клиента, выполнение транзакций), а логика приложения адаптировалась под взаимодействие с интерфейсом для обеспечения корректной работы всех сценариев. Ниже описаны этапы создания приложения с акцентом на моделирование работы банкомата.

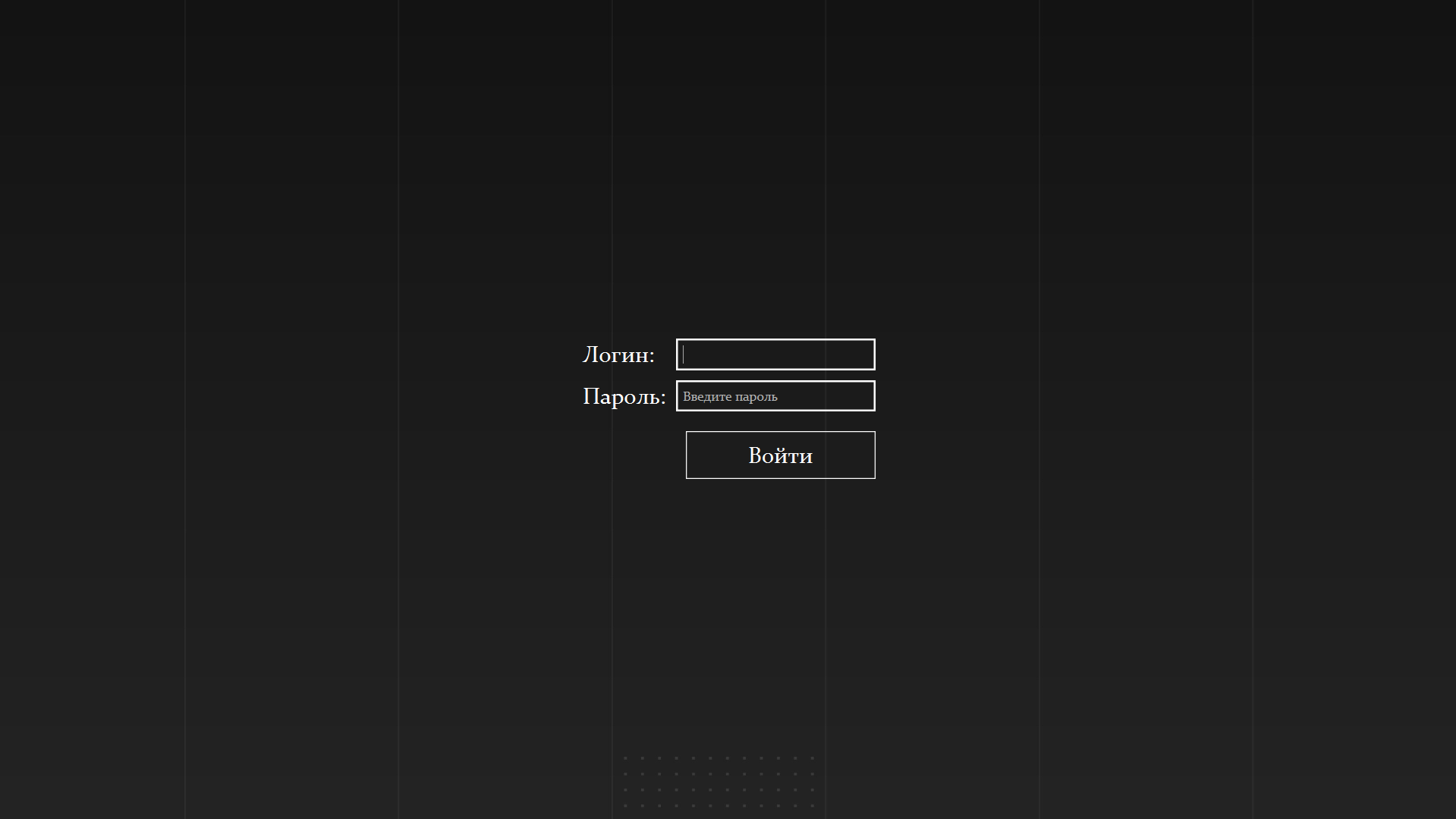
## 3.2 Разработка приложения на языке Java

Выбор Java 23 в качестве языка программирования обусловлен её широким применением в разработке надёжных и масштабируемых приложений, особенно в финансовой сфере. Java обеспечивает кроссплатформенность, поддержку многопоточности (например, виртуальные потоки) и интеграцию с библиотеками безопасности, что критично для моделирования работы банкомата. Для реализации графического интерфейса использовалась библиотека JavaFX , позволяющая создавать динамичные сцены с визуализацией состояния банкомата и взаимодействием с пользователем.

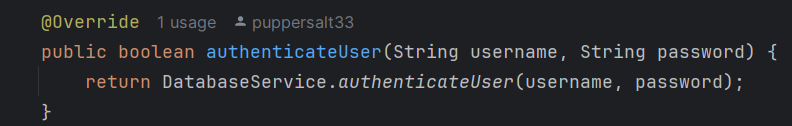
В ходе разработки необходимо реализовать:

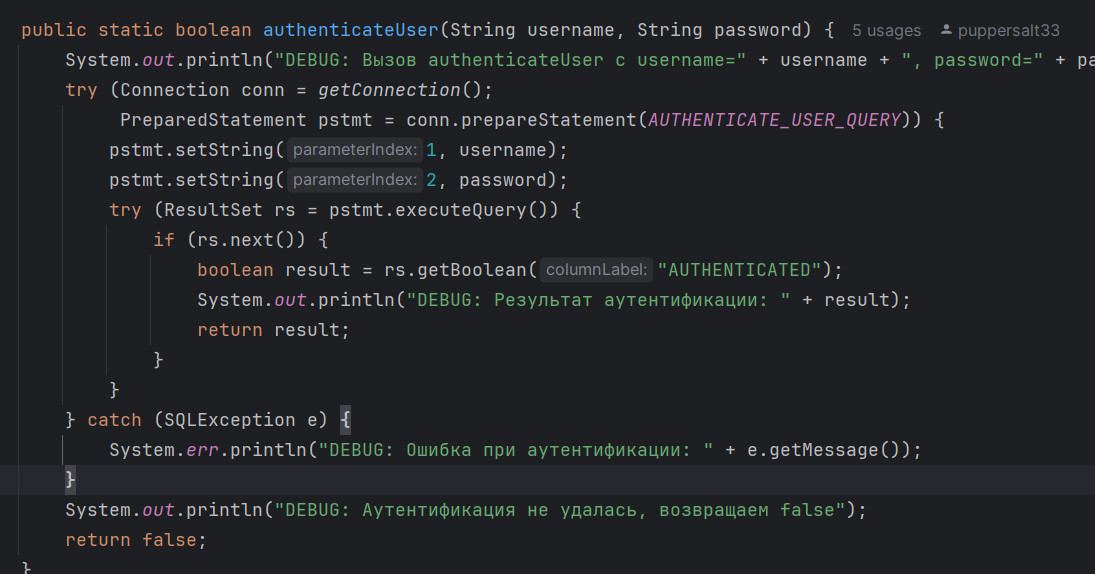
* Окно авторизации технического сотрудника банкомата
* Сцена администраторского меню
* Сцена инкассации
* Сцена настройки услуг
* Сцена блокировки банкомата
* Окно с привествием клиента
* Сцена для ввода номера карты
* Сцена для прохождения биометрии
* Сцена для ввода пин-кода
* Сцена главного меню банкомата
* Модуль из классов с пополнением счета
* Модуль из классов со снятием наличных со счета
* Модуль выбора и оплаты услуг
* Модуль с проверкой баланса, настроек
* Модуль смены пин-кода

### 3.2.1 Разработка окна авторизации технического сотрудника банкомата

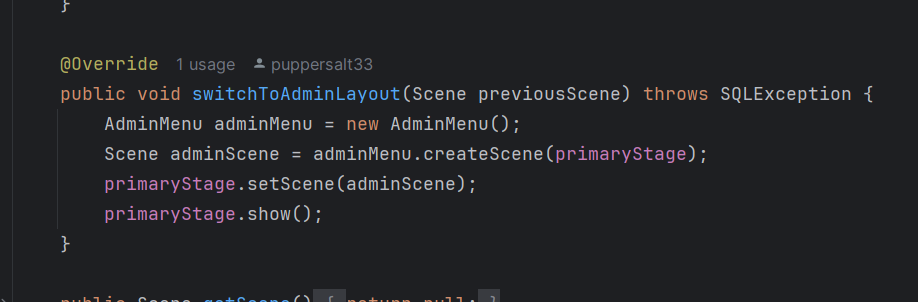
Рисунок 3 — Окно авторизации для админ меню

Окно авторизациии, это первое что встречает технического сотрудника банка. Сцена построенна следующим образом. Создается сцена с раположением полей и кнопок. После нажатия на кнопку войти, происходит событие с вызовом процедуры.

Рисунок 4 — Вызов функции authenticateUser

Рисунок 5 — Структура метода authenticateUser.

После обработки запроса, возвращает данные, если они верные, то идет смена сцены, иначе выведет ошибку.

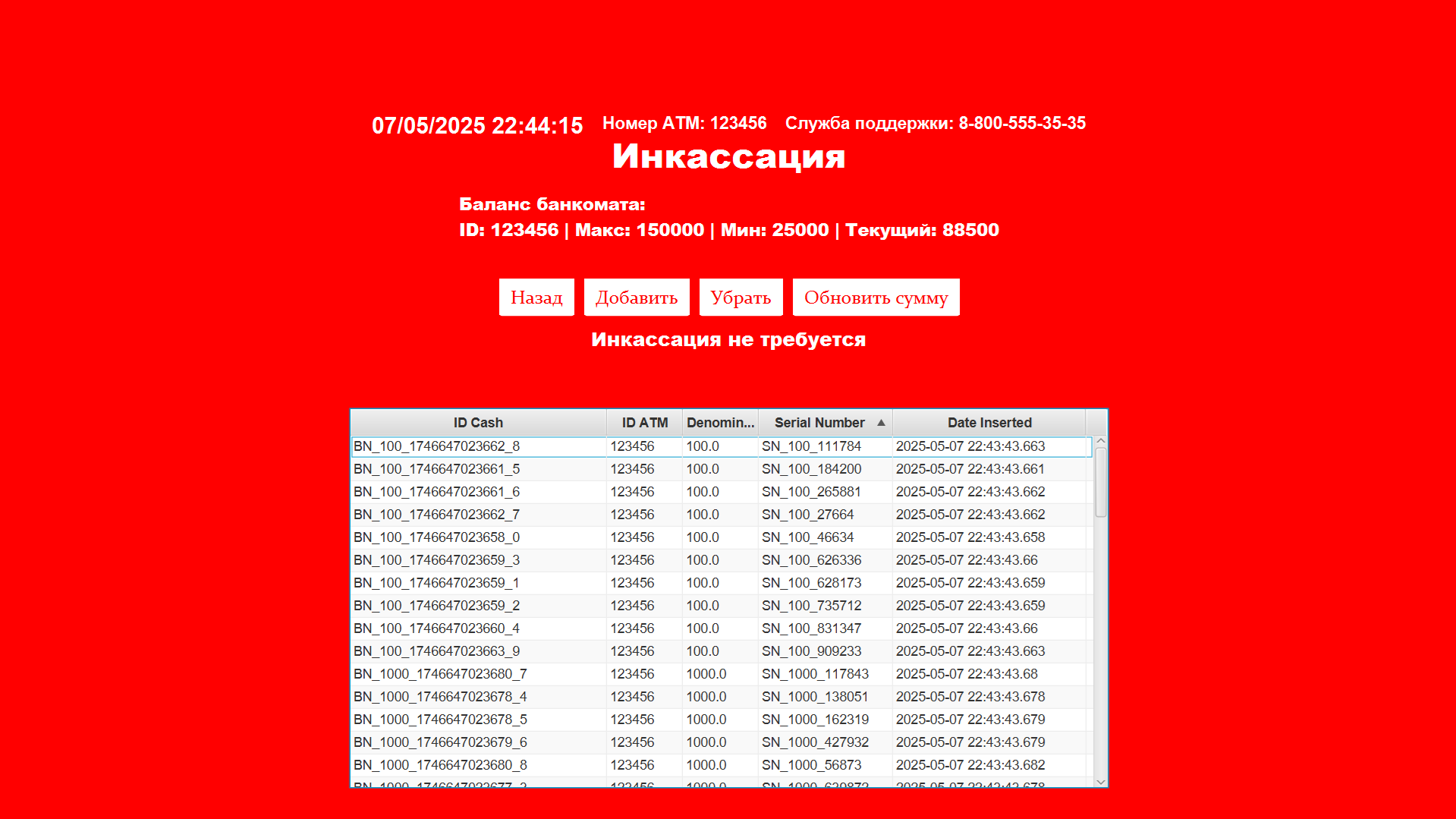
Рисунок 6 — Структура функции смены сцены на AdminMenu

### 3.2.2 Разработка сцены технического меню банкомата

Рисунок 7 — Сцена меню настройки банкомата

При входе за сотрудника банковской системы, сцена встречает меню с интерактивными кнопками.Расположение кнопок и наполнение сцены аналогичны что и с другими сценами. Логика нажатия кнопок привязана к событию, которые просто меняют сцену по аналогии с окном авторизации.

### 3.2.3 Разработка сцены инкосаторского меню

Рисунок 8 — Сцена инкассаторского меню управления

Сцена встречает несколькими label с информациией о наличных в банкомате. Далее идут кнопки управления наличными, а именно ввод денежных средств, вывод их из банкомата и обновления статуса, для актулизации данных. В конце идет таблица, данные которые получает из базы данных, и выводит все купюры хранящиеся в банкомате.

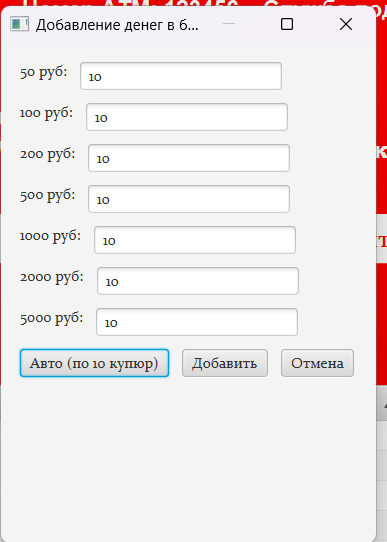
Вывод времени, номера банкомата и телефона, записанно в отдельном классе и выводиться простым вызовом метода класса. Ниже приведен подробный код реализации реального времени и вывода на сцену:

 Рисунок 9 — Метод вывода label на сцену в отдельном классе

Рисунок 10 — Включенныые методы для вычисления реального времени в том же классе.

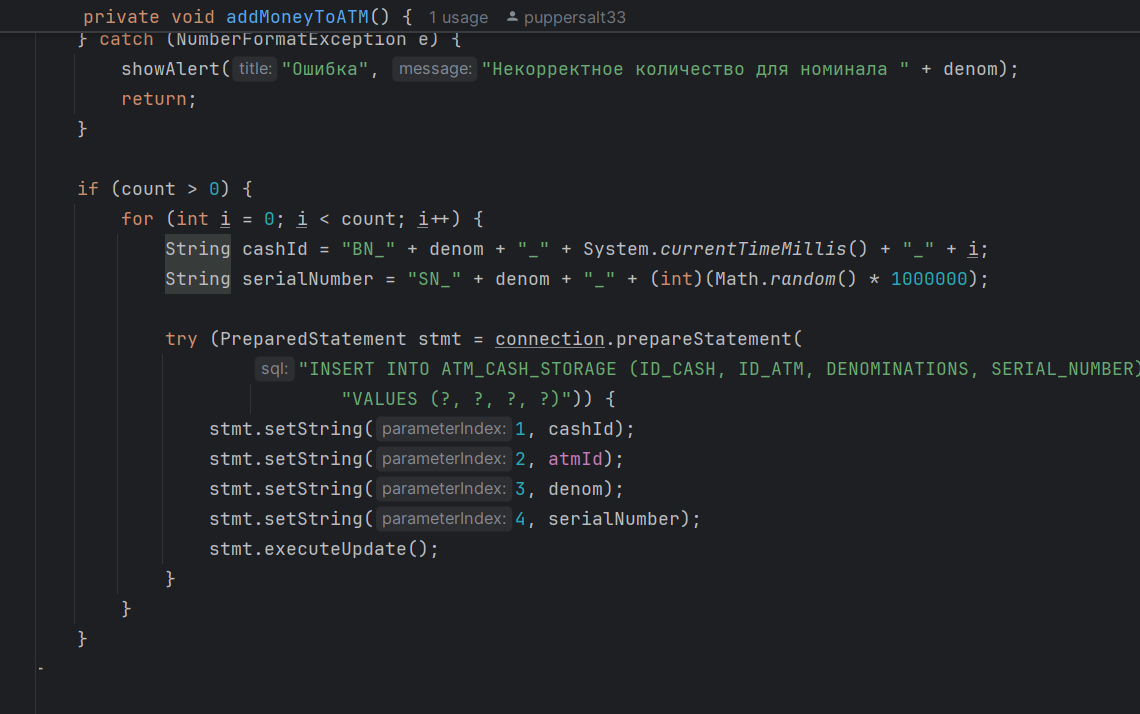
Кнопки управления также вызвают событие, в котором вызвается различные сцены из других классов. Кнопка «назад» выполняет функцию возвращения на админ меню, после нажатия окно не закрывается, а просто меняют сцену из класса Инкосации обратно на AdminWindow.

Кнопка «Добавить» вызвает модальное (Алерт) окно в котором расположены различные поля и номиналы, а также кнопки управления.

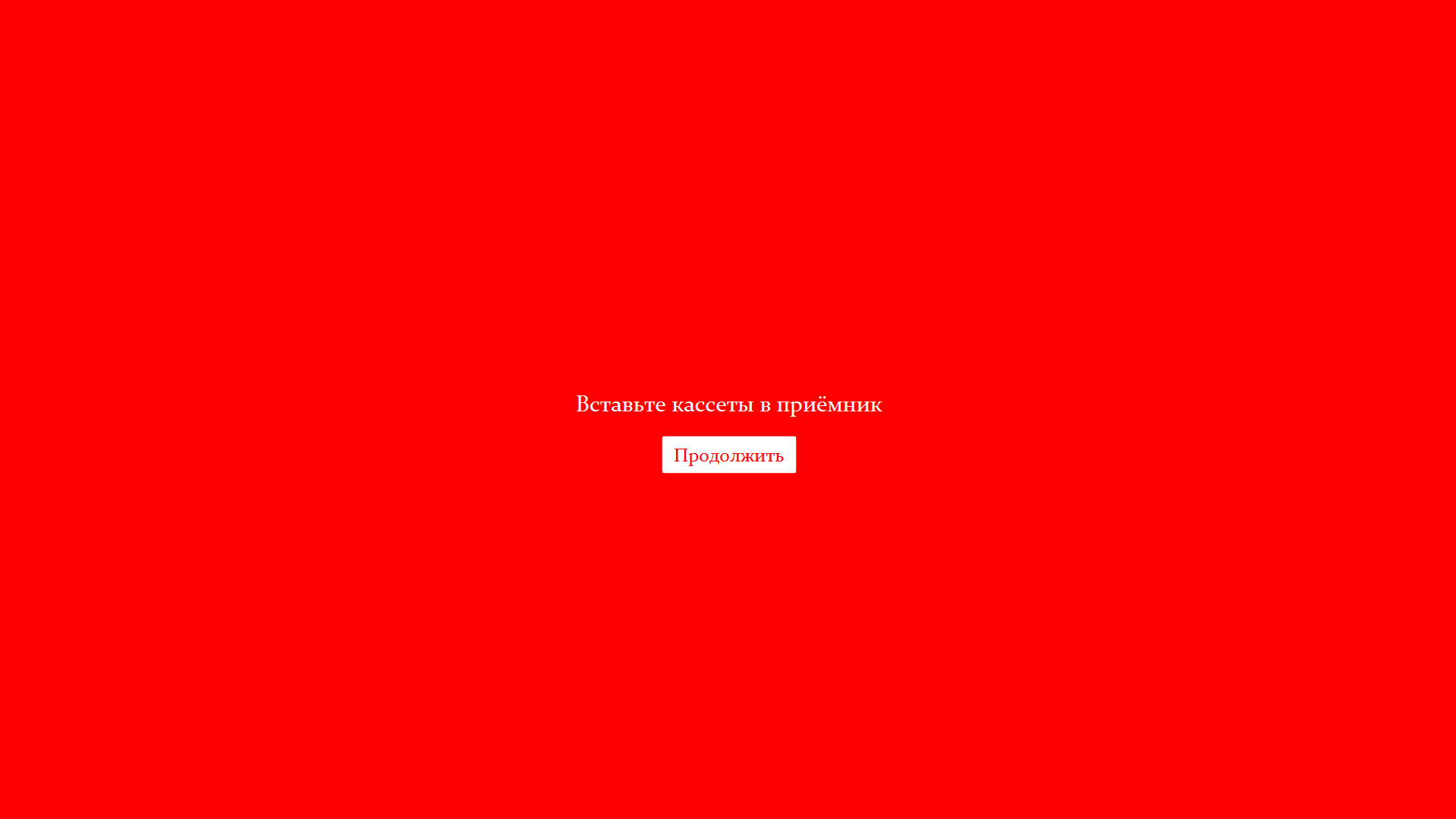
Рисунок 11 — Модальное окно добавления купюр.

Окно предлагает сотруднику указать, какие именно купюры необходимо создать и подготовить к пополнению в банкомат. Это делает более гибким и удобным моделирование ситуации пополения банкомата купюр. Кнопка «Авто» автоматически заполняет поля значением 10, что значит что будет указано что к созданию купюр будет по 10 шт. каждого номинала. Кнопка «Отмена» закрывает модальное окно и прерывает операцию ввода купюр.

Кнопка «Добавить» выполняет основную функцию добавления. После ввода значений в поля и нажатия кнопки, просиходит процесс генерации купюр, при этом также запускается процесс смены окна. Когда мы создаем купюры, мы загатавливаем ей начало серийного номера «BN», а после к ней добавляем случайное сгенерированное число, с уловием того что данное число является уникальным для банкомата. После чего мы сохраняем их в базе данных в соотвестующей таблице

Рисунок 12 — Алгоритм добавления генерации купюр и сохранением их в базе данных.

Сразу же после этого мы переходим на другую сцену с иммитацией ввода банковских кассет в банкомат. Эта сцена находится в другом классе в котором кроме текста и кнопки более не происходит.

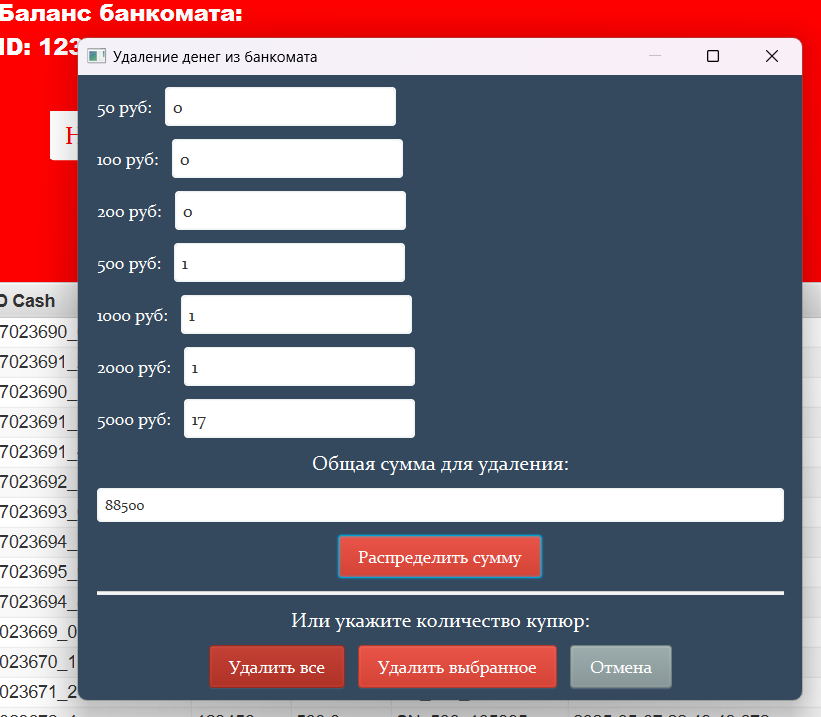
Рисунок 13 — Окно иммитации ввода кассет в банкомат

После нажатия на кнопку «Продолжить» идет переход на сцену загрузки денег и проверки денежных средств. Проверка эта номинальная, но для моделировании и имитации подходит безупречно.

Рисунок 14 — Окно имитации загрузки и проверки денежных средств.

После этого, сцена переходит обратно на сцену инкассации , что делает данные действия многоповторяемыми.

Далее перейдем к кнопке «Удалить», после нажатия на нее, событие вызывает функцию модального окна из класса RemoveMoneyDialog. Выглядит это следующим образом.

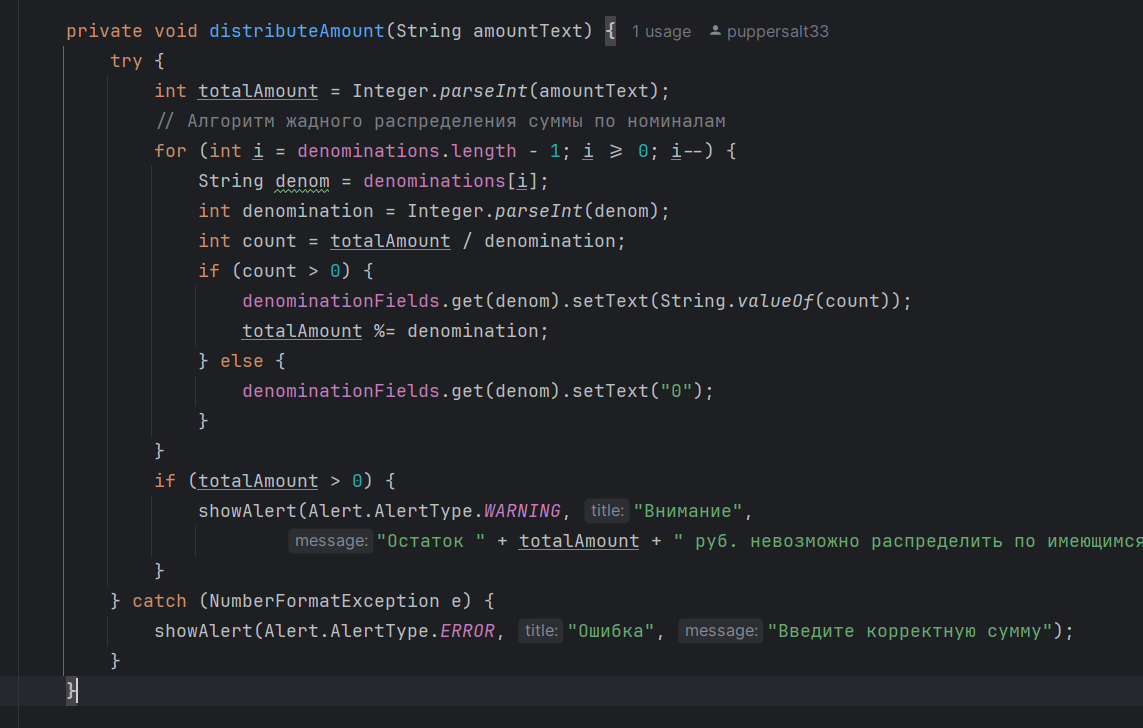
Рисунок 15 — Модальное окно удаления купюр в сцене инкассации.

Расмотри поподробнее. На окне представлены такие поля и номиналы с указанием кол-ва штук. Ниже есть поле для ввода конкретной суммы для удаления суммы из купюры. После чего идет кнопка «Распределить сумму» которая выполняет функцию автоматического распределения купюр к этой сумме. В программе используется жадный алгоритм который принимает строку amountText, представляющую сумму для распределения. Предполагает наличие:

* + - * Массива denominations (строковые представления номиналов, например ["10", "50", "100"]).
      * Карты denominationFields (связывает номинал с текстовым полем интерфейса).

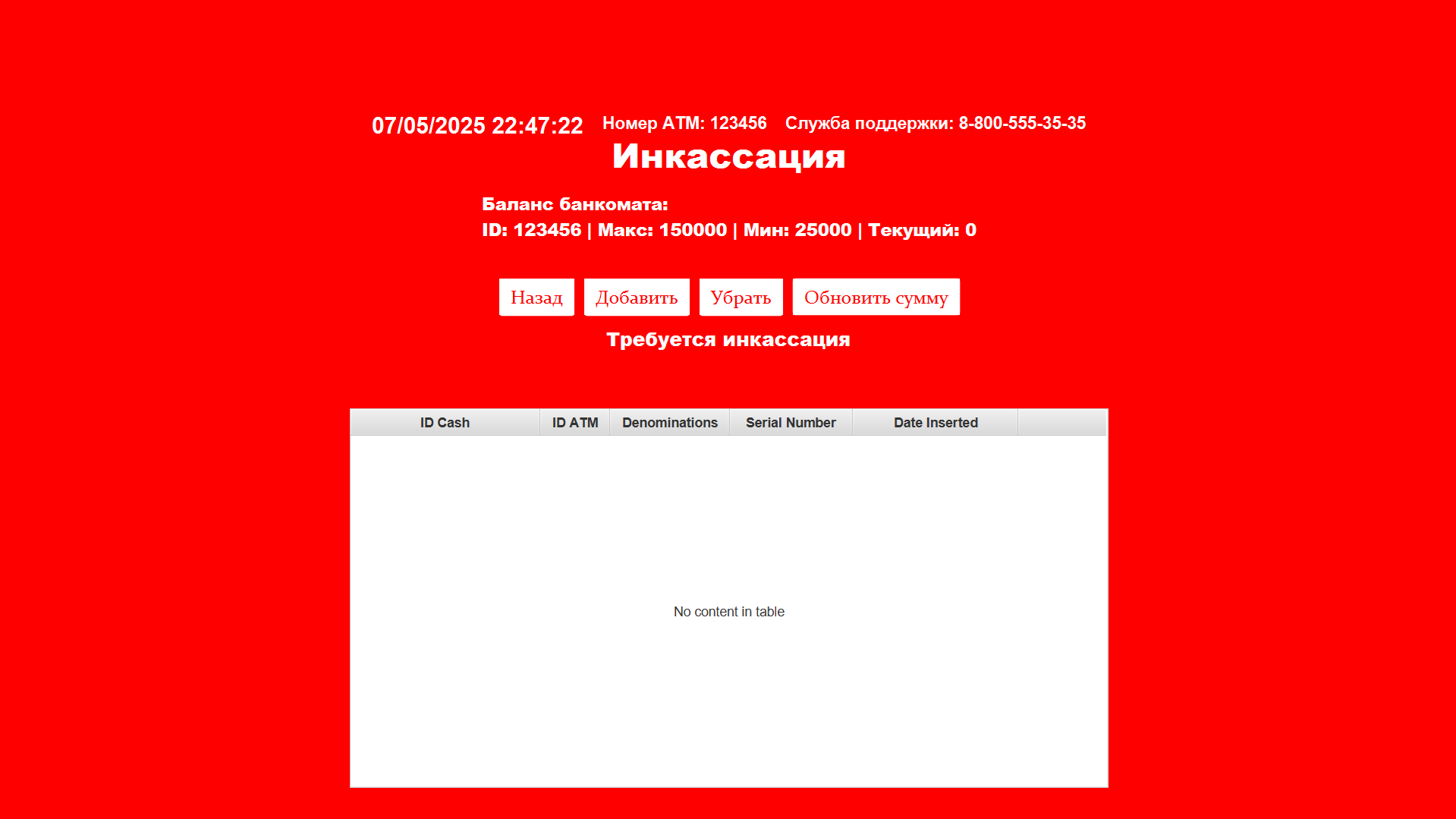
После чего пытается преобразовать входную строку в целое число. При ошибке (NumberFormatException) выводит сообщение о некорректном вводе. Далее идет алгоритм, который итерируется по массиву denominations с конца. Для каждого номинала:

1. Вычисляет максимальное количество купюр/монет:
2. Обновляет соответствующее текстовое поле:
3. Уменьшает оставшуюся сумму:
4. Если count == 0, Поле заполняется "0".

Рисунок 16 — Жадный алгоритм автораспределения купюр

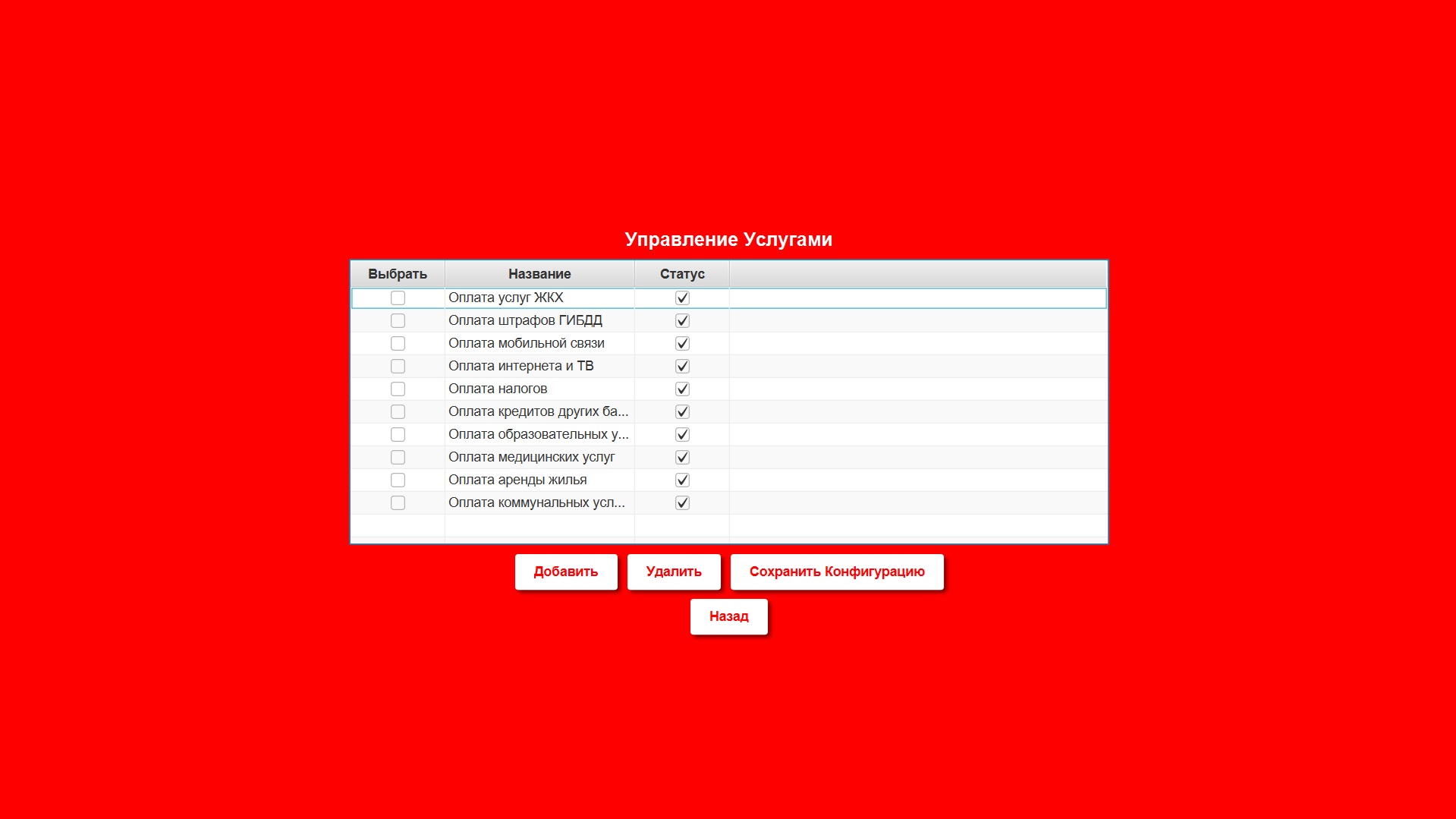
По середине, расположена кнопка «Удалить все», при ее нажатии идет простое удаление всех купюр из таблицы, в которой хранится все купюры. После идет кнопка «Назад» которая закрыывает диалоговое окно.

При любом нажатии из кнопок удаления будет происходить процедура удаления выбранных купюр, а также переход на сцену иммитации удаления купюр, аналогичные с добавлением.

Рисунок 17 — Сцена меню инкассации после удаления всех купюр

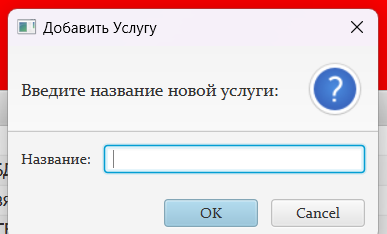
### 3.2.4 Разработка сцены управления услугами

После перехода на сцену с управлением услугами, пользователя встречает простое окно с таблицей всех услуг и кнопками по управлению услугами.

Рисунок 18 — Сцена управления услугами

Данные об услугах подгружаются из базы данных, преобразовываются и выводиться в таблицу TableView. К строкам таблицы также добавлены два checkbox (Галочки) для упралвения состоянием услуги. Первый столбец таблицы отвечает за выбор строки, где 0 (нет галочки) значит строка не выбрана, а 1 (Есть стрелочка), что строка выбрана для операции (например удаления). Далее идет столбец с названиями услуг. После него идет столбец со статусом услуги. В базе данных статус услуги храниться в формате boolean, что означает что услуга либо активна, либо нет.

Ниже идут кнопки управления услугами. Кнопка «Добавить», содержит событие при ее нажатии. Появляется модальное окно, в котором система требует ввести название новой услуги.

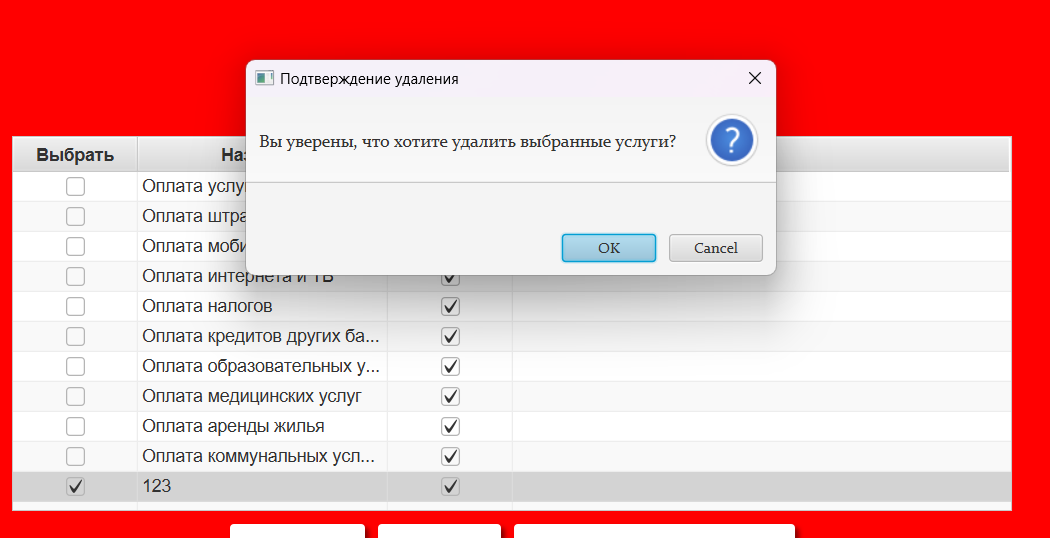
Рисунок 19 — Модальное окно с добавлением новой услуги

После ввода значения новой услуги, пользователь нажимает на кнопку ОК, если уверен в названии, или же отмена если передумал. В случаии отмены, окно закрывается и возвращает сцену в исходный момент.

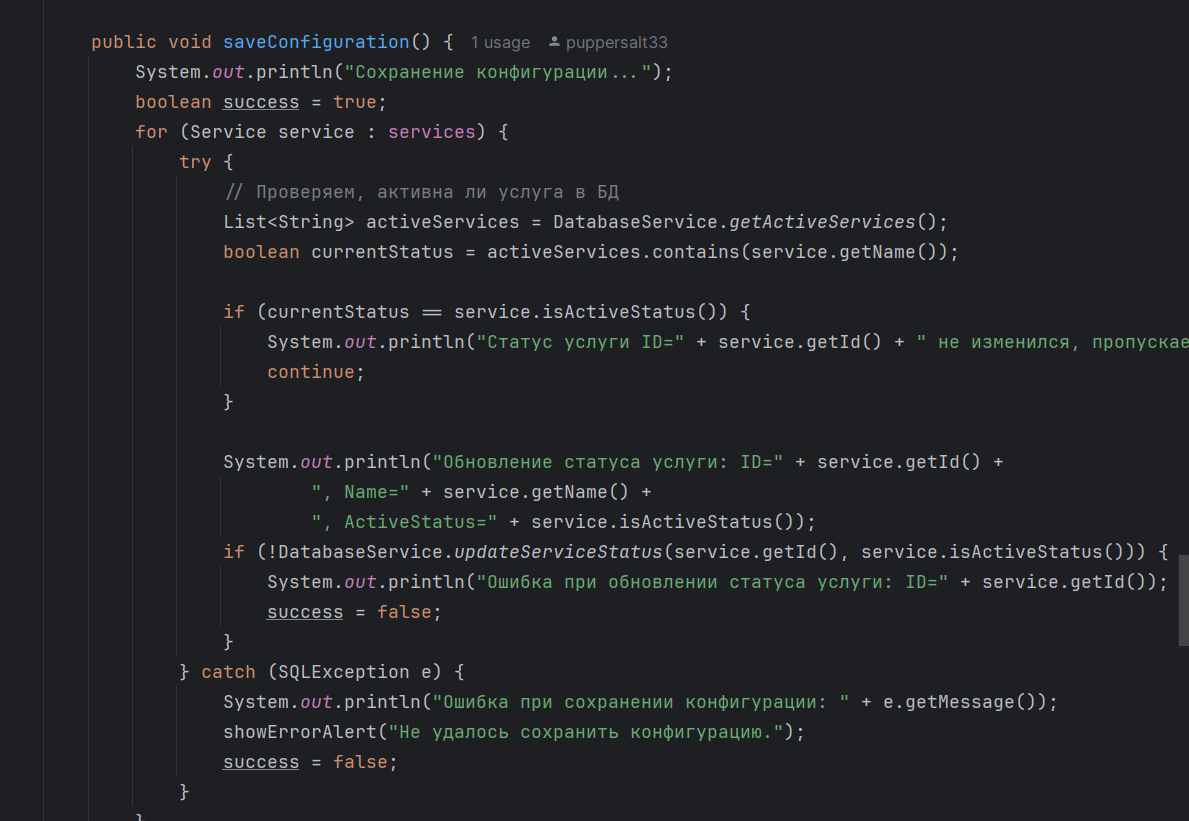
Когда пользователь вводит название новой услуги, событие обрабатывает поле ввода, проверяет на уникальность название (Таким образом защищаемся от дублицирования услуг), получаем новый ID услуги (Учитываем последний ID в таблице) и добавляем услугу в таблицу базы данных. Вот подробный код реализации данного метода:

Рисунок 20 — Функция добавления новой услуги и запись ее в таблицу

Далее идет кнопка «Удалить». После ее нажатия, событие кнопки берет все выбранные (В первом столбце) строки услуг и сохраняет их в переменные. Перед удалением выводиться предупреждение пользователя об удалении. Если пользователь нажал ОК, то алгоритм получает ID данных услуг и удаляет их таблицы, тем самым удаляя их из банкомата. В ином случаии при нажатии отмена, модальное окно вовсе закрывается, завершая процесс удаления.

 Рисунок 21 — Окно предупреждения об удалении услуги

Последней кнопкой в управлении является «Сохранить». Она используется если пользователь решил изменить статус той или иной услуги. Алгоритм берет все строки из таблицы, и обновляет их с теми строками в таблице данных. Вот подробный код как это реализованно:

Рисунок 22 — Функция сохранения и обновления всех услуг.

### 3.2.5 Разработка сцены блокировки банкомата

Сцена предназначена для блокировки банкомата от воздействий клиентов. Инфомрационный посыл гласит о том что банкомат не готов к обслуживанию клиентов. Также на сцене нет кнопок для выхода с этой сцены, тем самым, кроме технического работника, имеющий доступ к клавиатуре и мыши предпологаемоего банкомата, есть возможность перезапуска приложения. Это окно может быть споособом отлова критических ошибок, но без аварийного завершения процесса программы.

Рисунок 23 — Окно блокировки банкомата для ремонта или перезагрузки

### 3.2.6 Разработка сцены ввода номера карты для авторизации.

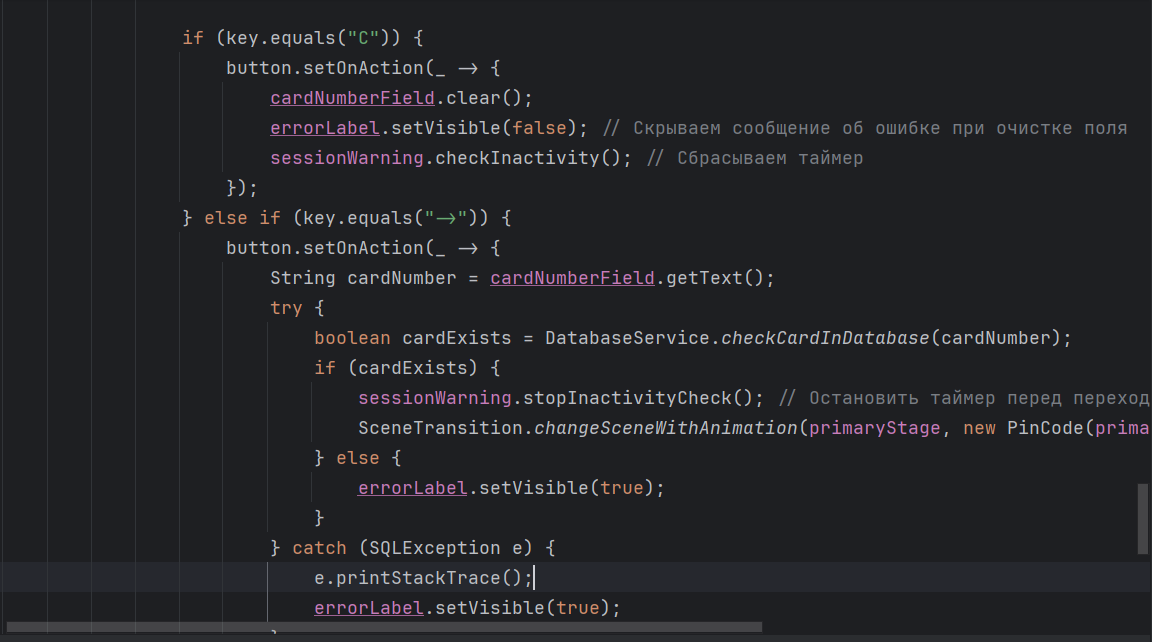
Рисунок 24 — Сцена с вводом номера карты через вирутальную клавиатуру

Пользователя встречает окно вводаномера карты. Снизу представлена вирутальная клавиатура для ввода значений. После ввода номера карты, пользователь нажимает кнопку «>», после чего система проверяет правильность ввода карты, и если такая есть, то переходим на сцену с вводом пинкода.

Поле ввода прописанно так, чтобы туда можно было ввести не более 16 цифр (Стандарт банковских карт), а также защищенно от ввода буквенных значений или знаков. Это сделано для того чтобы клиент, если бы очень захотел не мог бы ввести что то другое кроме номера карты.

Вирутальная клавиатура реализована методом списка кнопок, где каждая кнопка отвечает за свой номер (Кроме кнопки → которое содержит событие). При нажатие из любых цифр, к текущему значению поля добавляется следующая цифра. Кнопка «С» стирает все значение поля.

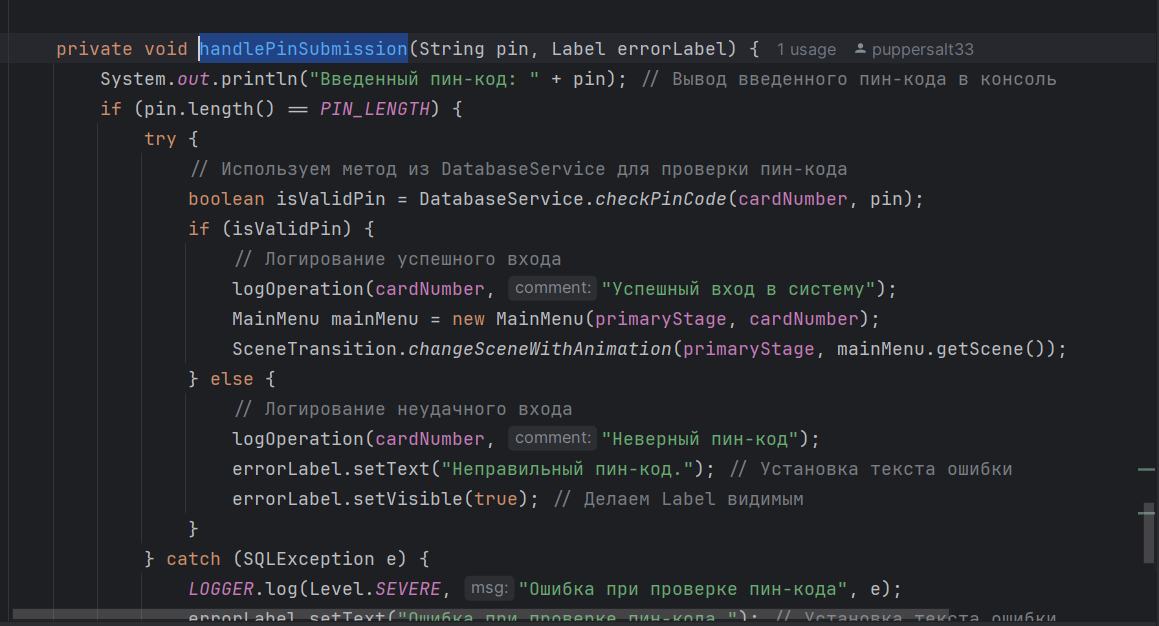
Когда кнопка выполнить нажимается, выполняется запрос в базу данных из процедуры. Процедура возвращает ответ, что если есть такая карта, или нет. В условии далле идет переход на сцену ввода пин-кода.

Рисунок 25 — Функции обработки поля вввода и перехода на новую сцену

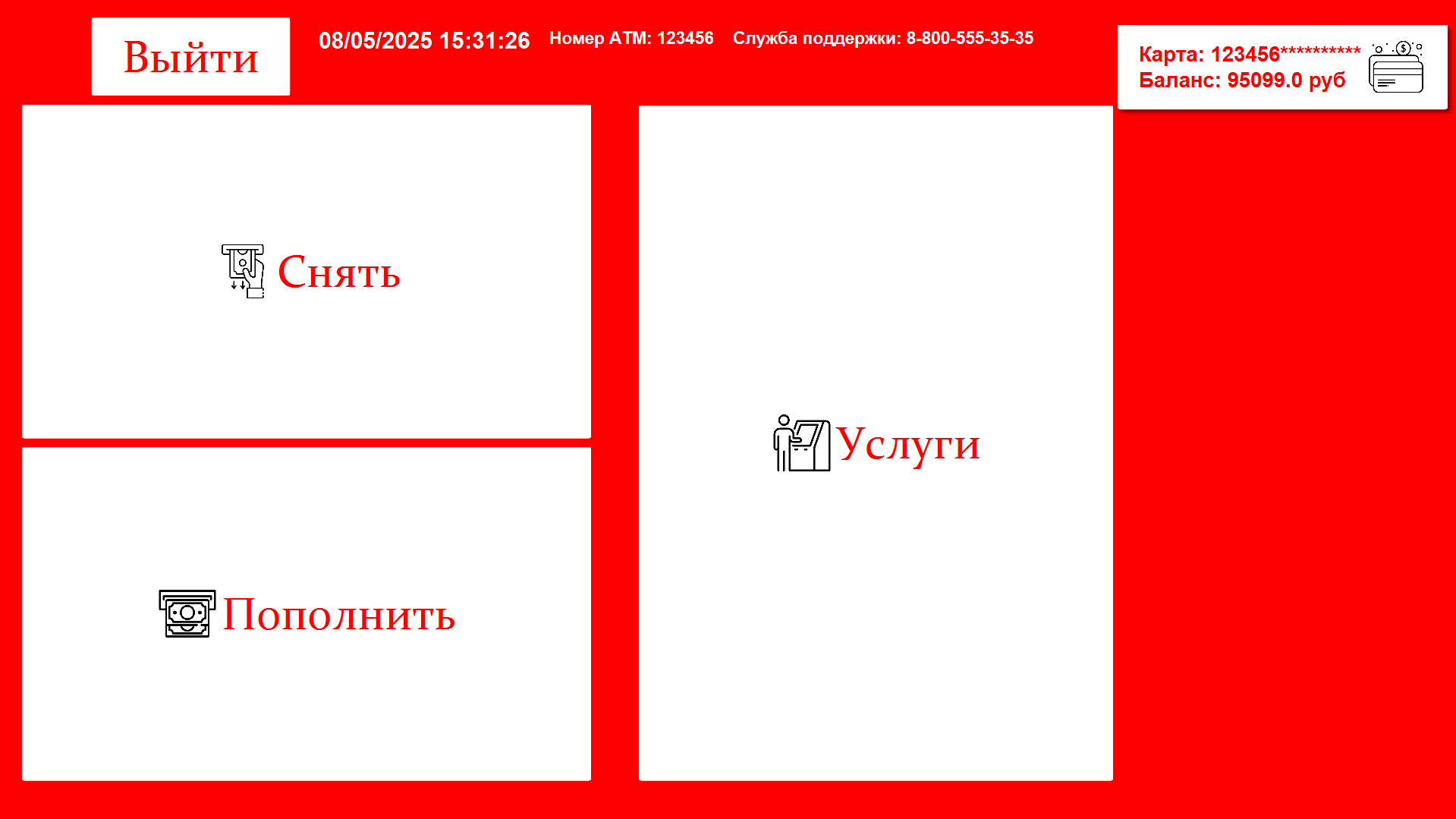
### 3.2.7 Разработка сцены ввода пин-кода для авторизации.

Рисунок 26 — Сцена ввода пин-кода

Сцена аналогична той, что в вводе номера карты. Также есть ограничение на кол-во вводимых цифр равное 4-ым. Также стоит шифровка цифр, для защиты от подглядывания и конфедициальности. Реализованн вхож через функцию handlePinSubmission в которой проверяется и номер карты и пин-код, иначе выводим ошибку.

Рисунок 27 — Функция handlePinSubmission для входа на счет

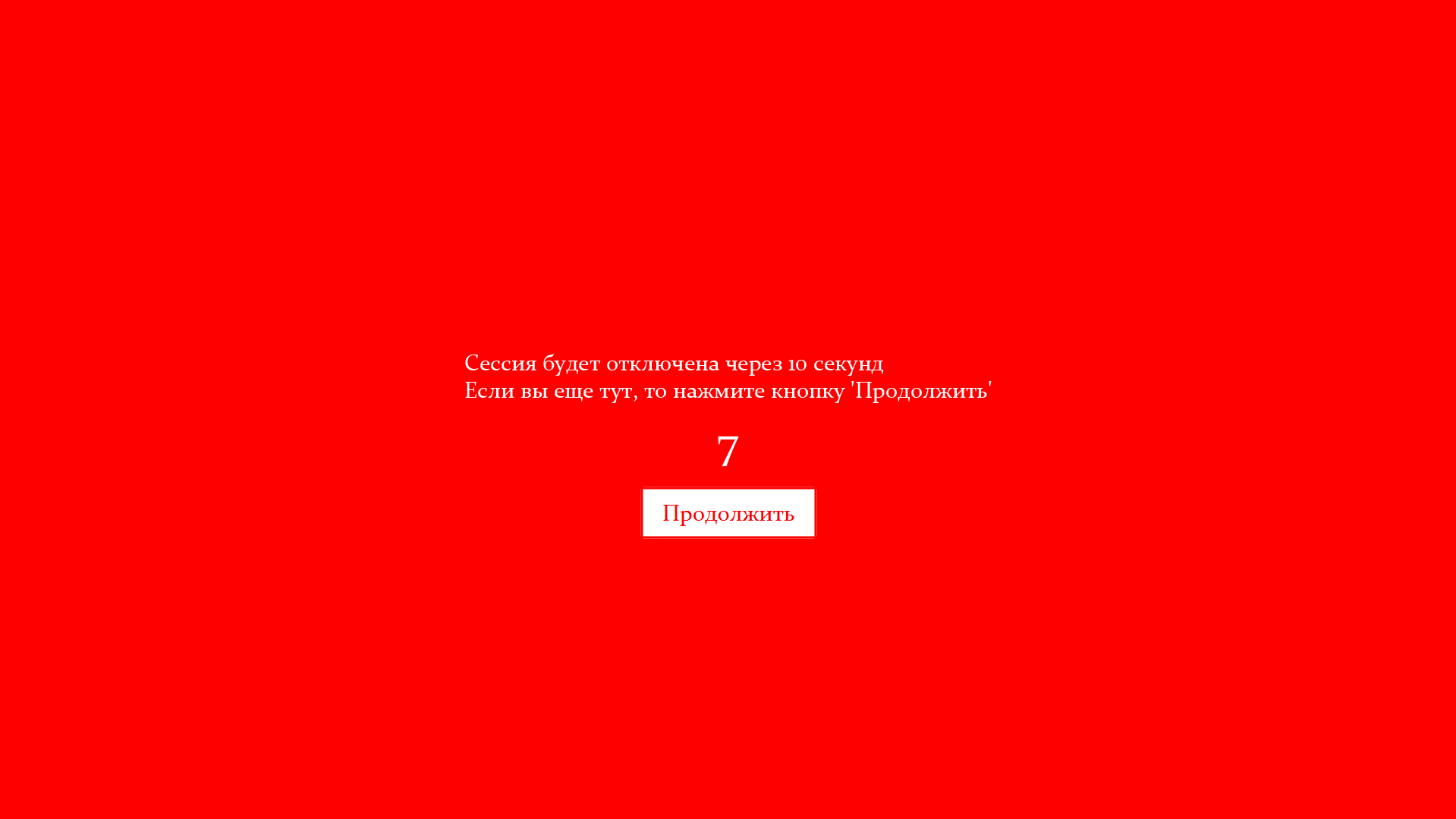
### 3.2.8 Разработка сцены главного меню клиента

Рисунок 28 — Главное меню клиента после авторизации к счету

Главное меню клиента представляет собой сцену из нескольких кнпок и информативной надписи о времени. Здесь представленны основные функции банкомата для клиента. Также в правом вверхнем углу расположена кнопка с зашифрованным номером карты и балансом счета.

Данная методика позволяет клиенту точнее определить как он авторизовался и сколько денежных средств он имеет.

Также в проект был добавлен модуль сессии клиента. Он включается в каждой сцене в начале подгрузки обьектов. Если клиент не выполнит действия (Переход на другую сцену через кнопки), то появиться всплывающее окно с подверждением активности. Окно появляется через 10 секунд бездействия клиента и содержит предупреждение и кнопку «продолжить». После ее нажатия, таймер сбрасывается и начинает круг заново. Если клиент не нажмет на кнопку, то система закроет текущую сцену, и вернется на окно привествия банкомата, а все текущие операции завершаться и отменятся.

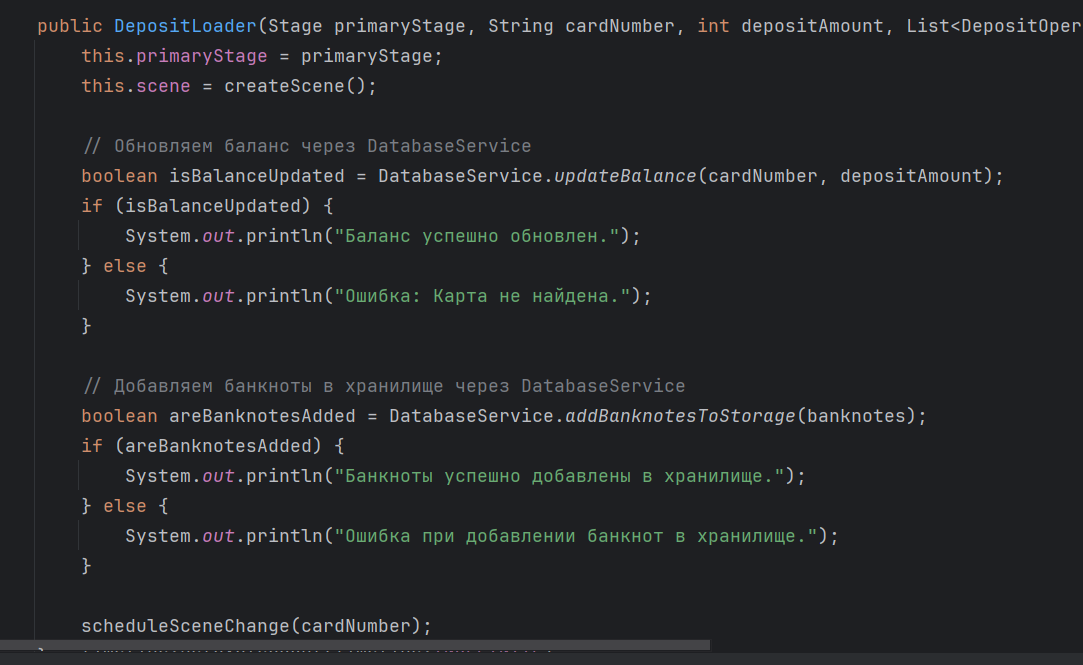
Рисунок 29 — Окно предупреждения о закрытии сессии.

### 3.2.9 Разработка сцены пополнения счета через банкомат

Рисунок 30 — Сцена пополнения счета через банкомат

Когда клиент хочет пополнить свой счет, он нажимает на кнопку «Пополнить» в домашнем меню. После этого открывается сцена где отображенно: Надписи с подсказками, таблица купюр, кнопки с разными номиналами. При нажатии на любую из кнопок, мы создаем купюру и добавляем в лист таблицы эту купюру. Таким образом мы смоделируем ввод денежных средств на счет клиента и сохранением их в банкомат. При создании купюр, мы используем тот же метод создания купюр, какой использовали в модуле инкассации, поэтому ошибок при создании купюр и их уникальности предотвращены до минимума.

После того как клиент ввел достаточно купюр, он нажимает кнпоку «Далее» что отправляет его на сцену с загрузкой. Номинально мы берем список новых купюр и передаем на следующую сцену, где мы сначала обновляем баланс клиента на сумму всех кол-во номиналов, а после добавляет поочередно каждую купюру в банкомат.

Рисунок 31 — Алгоритм добавление купюр и пополнение счета клиента

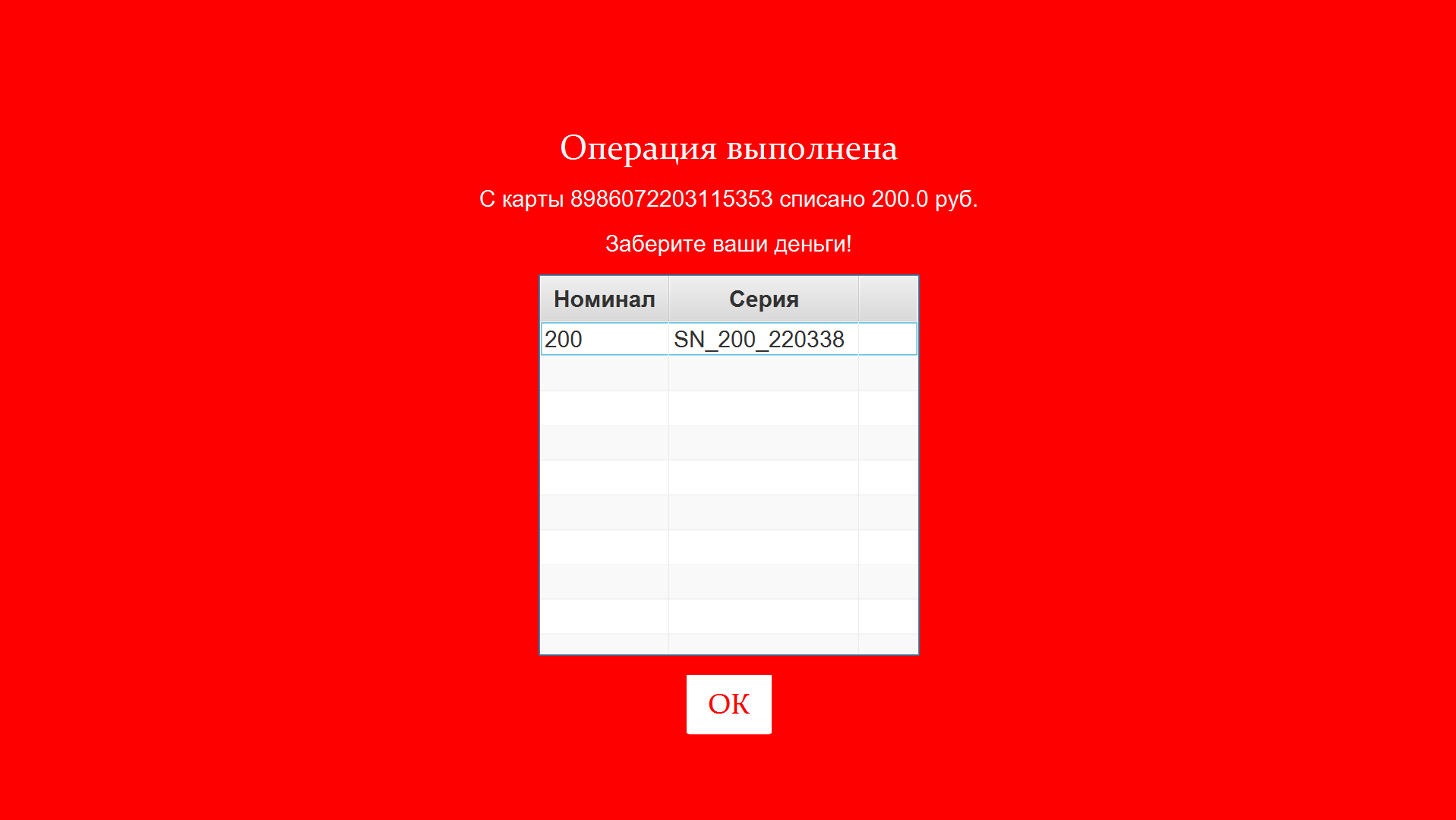
После того как все было загружено и подтверждено, сцена клиента меняется на окно привествия, тем самым завершая сессию.

### 3.2.10 Разработка сцены снятия купюр со счета клиента

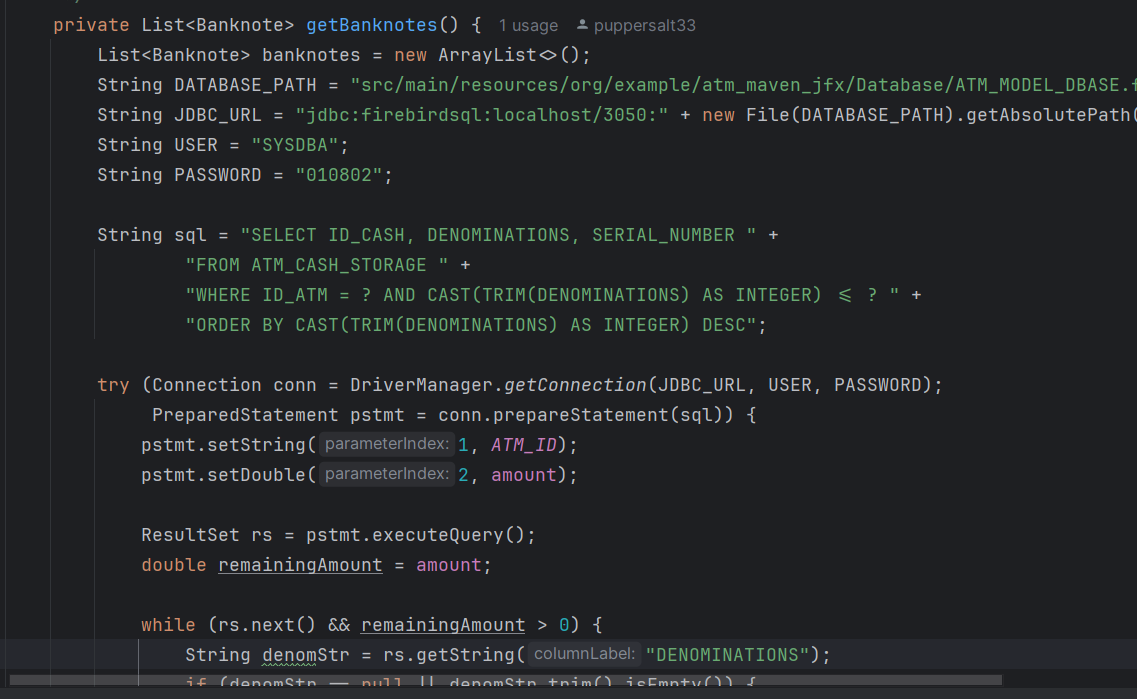
Аналогично со сценой пополнения, при нажатии на кнопку «Снять» открывается следующая сцена со снятием. Она встречает вирутальной клавиатурой и полем для ввода. Также сверху указан баланс счета для помощи клиенту. Клиент нажимая на кнопки вирутальной клавиатуры, вводит неоьходимую сумму для снятия. Поле, как и в воде номера карты или пин-кода защищенно от экранирования. Также добаленно условие о том что клиент не может снимать более 15.000 за одну операцию и не меньше 50. К дополнению условий можно сказать что сумма снятия должна быть кратной 50, для того что бы можно выдать любой набор купбр для этой суммы.

  
Рисунок 32 — Сцена для ввода суммы снятия со счета

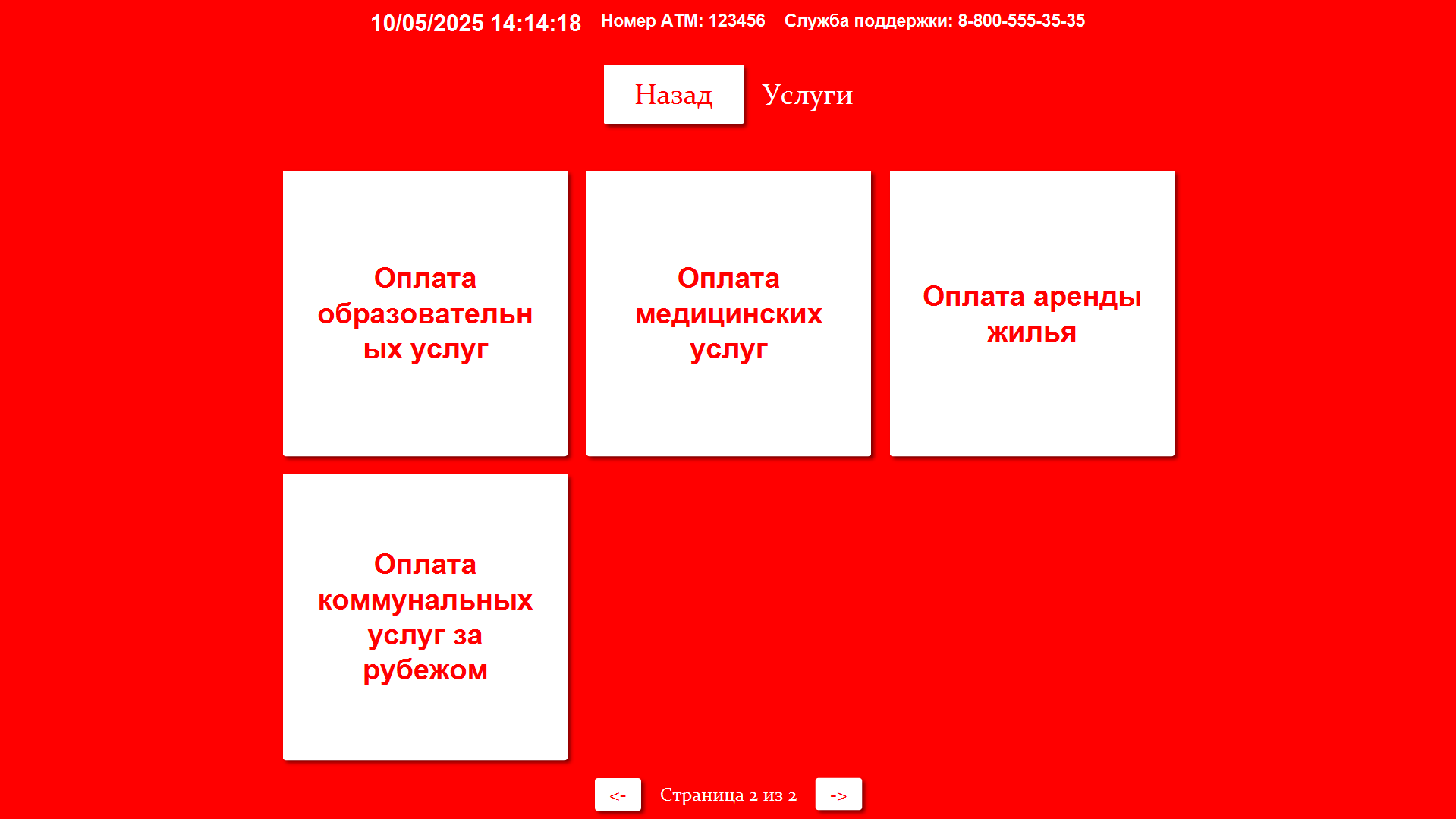
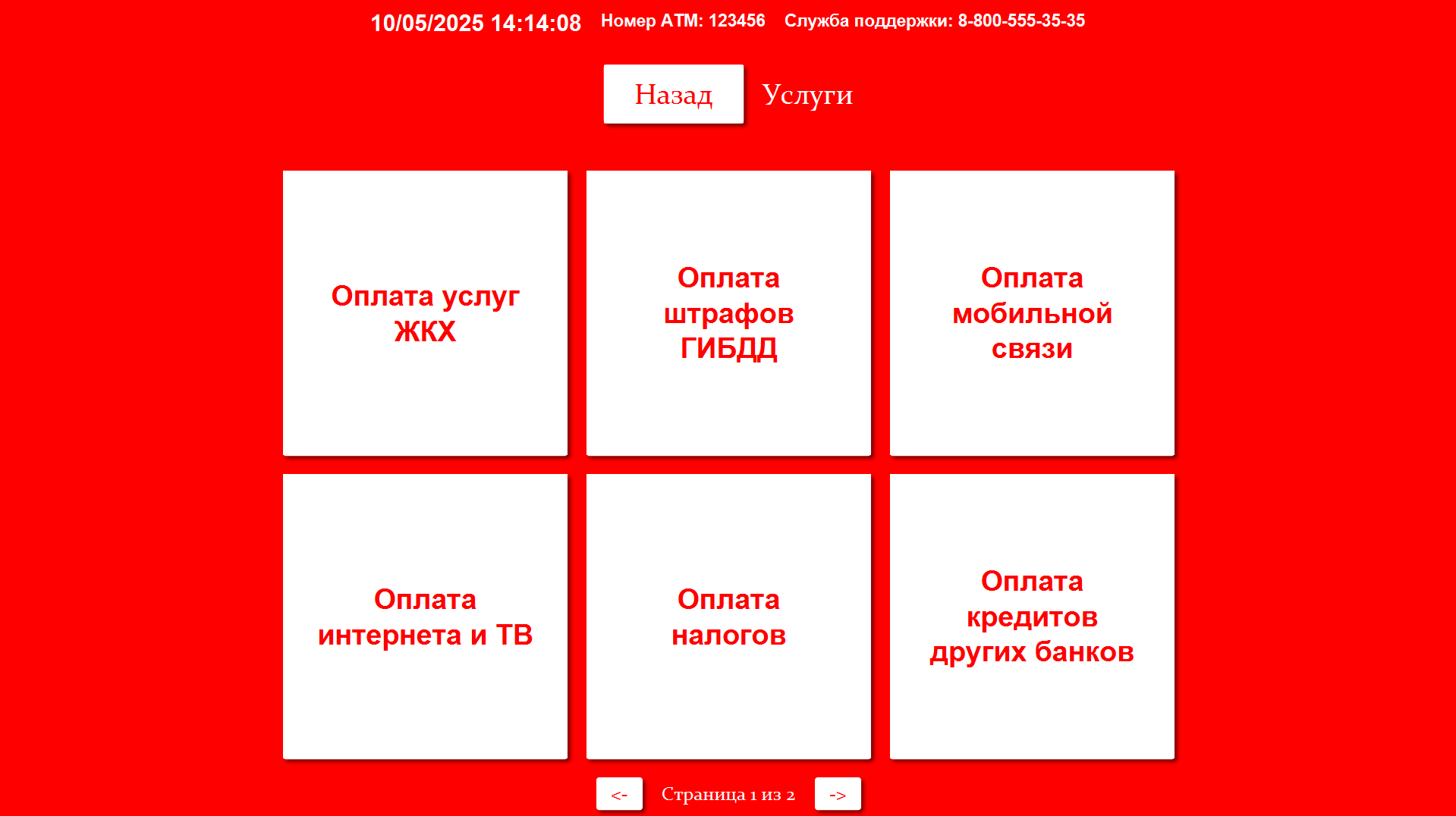
Когда клиент ввел желаемую сумму, он нажимает на кнопку «→” . Если все условия соблюдены, то он переходит на сцену с получением купюр. Система получает сумму для снятия, обращается к таблице с купюрами и подбирает набор купюр. Сначала мы обновляем счет клиента (Вычитаем сумму их баланса), после чего в таблице выводится купюры, которые подходит для суммы.

Рисунок 33 — Сцена с выдачей купюр клиенту

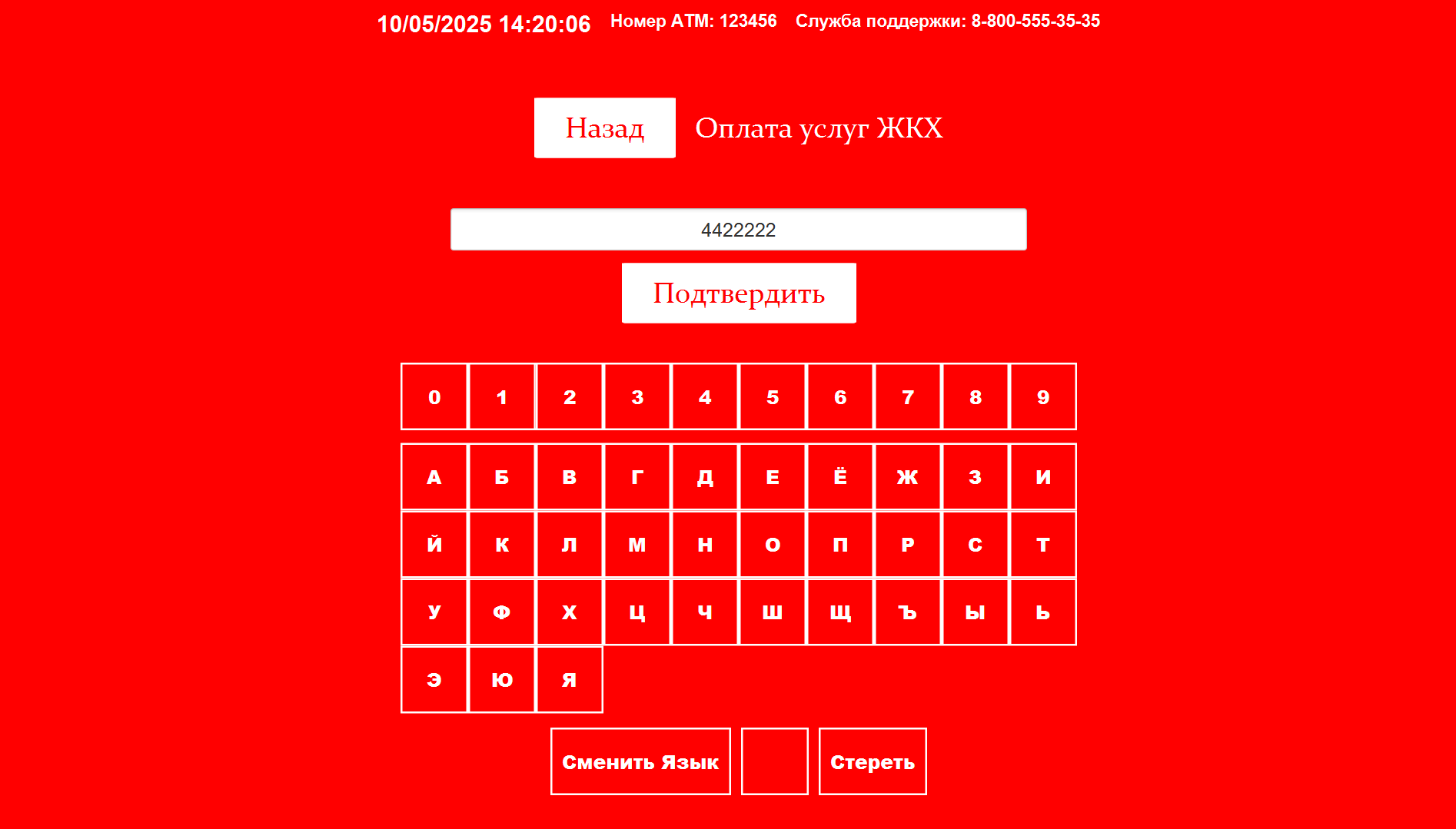
Выбор купюр и выдачей их выполняет функция List<Banknote> getBanknotes(). Мы отправляем запрос в базу данных с указанимем параметров, после чего получаем список купюр. Далее мы эти купюры помещаем в таблицу на сцене.

Рисунок 34 — Функция по получению купюр по сумме

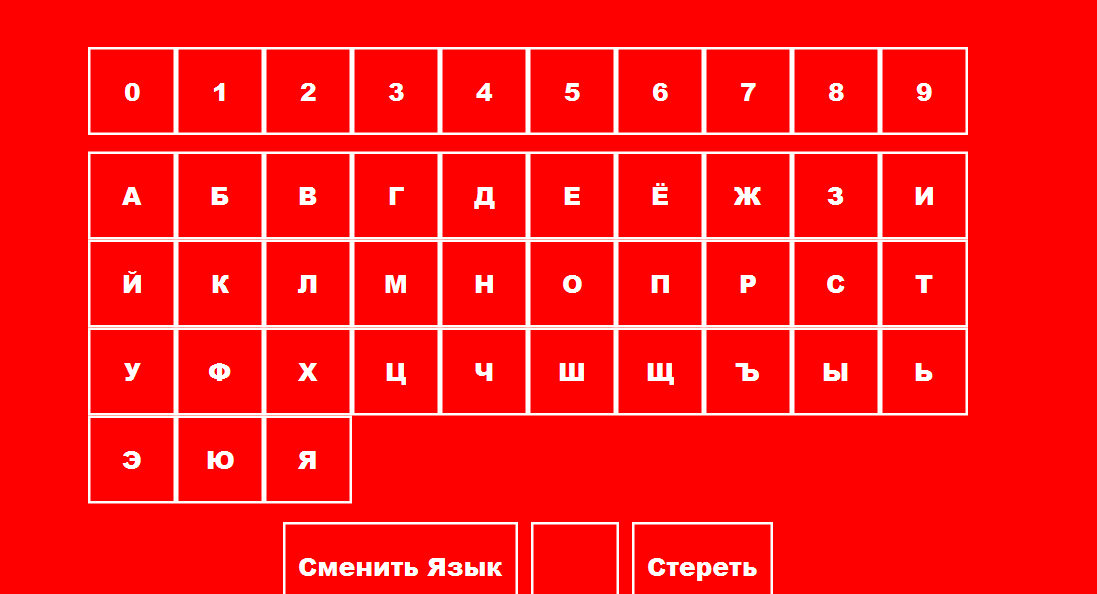
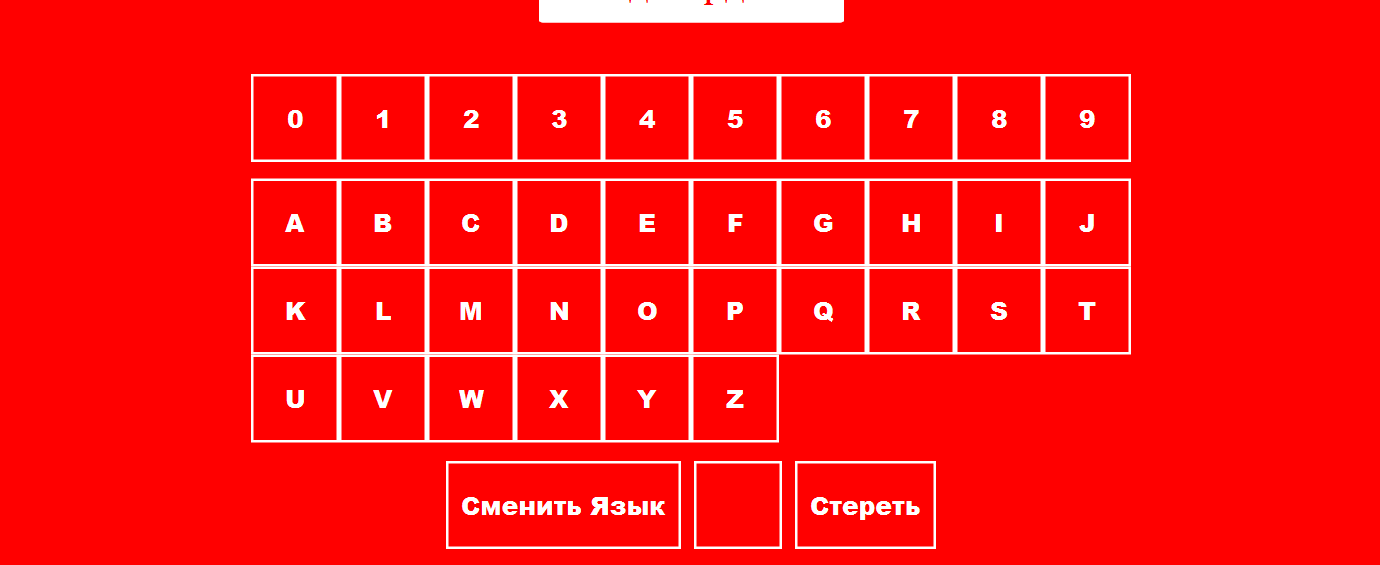
### 3.2.11 Разработка сцены выбора услуги и выполнение оплаты по ним.

Рисунок 35-36 — Сцена с выбором доступных услуг.

После перехода клиента на сцену услуги, в системе банкомата идет подгрузка активных услуг. Сцена разделяет на карточки услуги на 6 частей. Если услуг более 6-ти то создается следующая страница с услугами. Каждая услуга представляет кнопку, которая отправляет сцену на определенную универсальную сцену с указанием номера счета услуги.

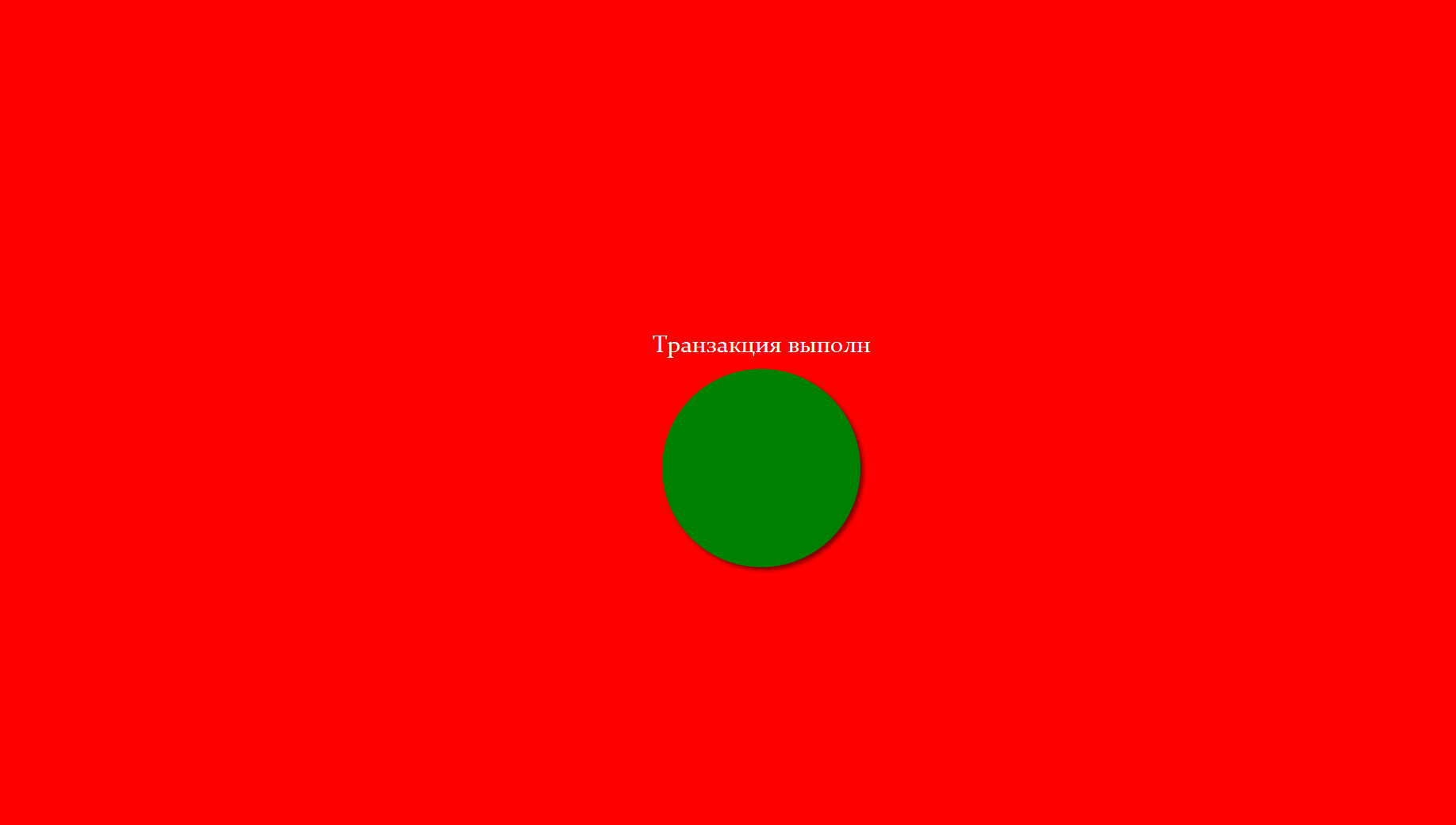
Рисунок 37 — Сцена с вводом номера счета для оплаты услуги

К этой сцене была создана новая виртуальная клавиатура. Помимо цифр были добавлены и буквенные символы. Также кнопки управления клаивитуры содержит кнопку смена языка и проблела, а также и стирания поля. Создание и наполнение клавиатуры аналогично с создаем клавиатуры в других классах, но для данной было реализованно следующее. Помимо букв на русском языке, считается и требование с добавлением букв английского алфавита. При нажатиие на смену, мы меняем наполение кнопок с списка с русскими буквами на английский. Учитывая в том что в английском алфавите букв меньше чем в русском (23 буквы против 33) мы проверяем кол-во строк и уравниваем их на сцене.

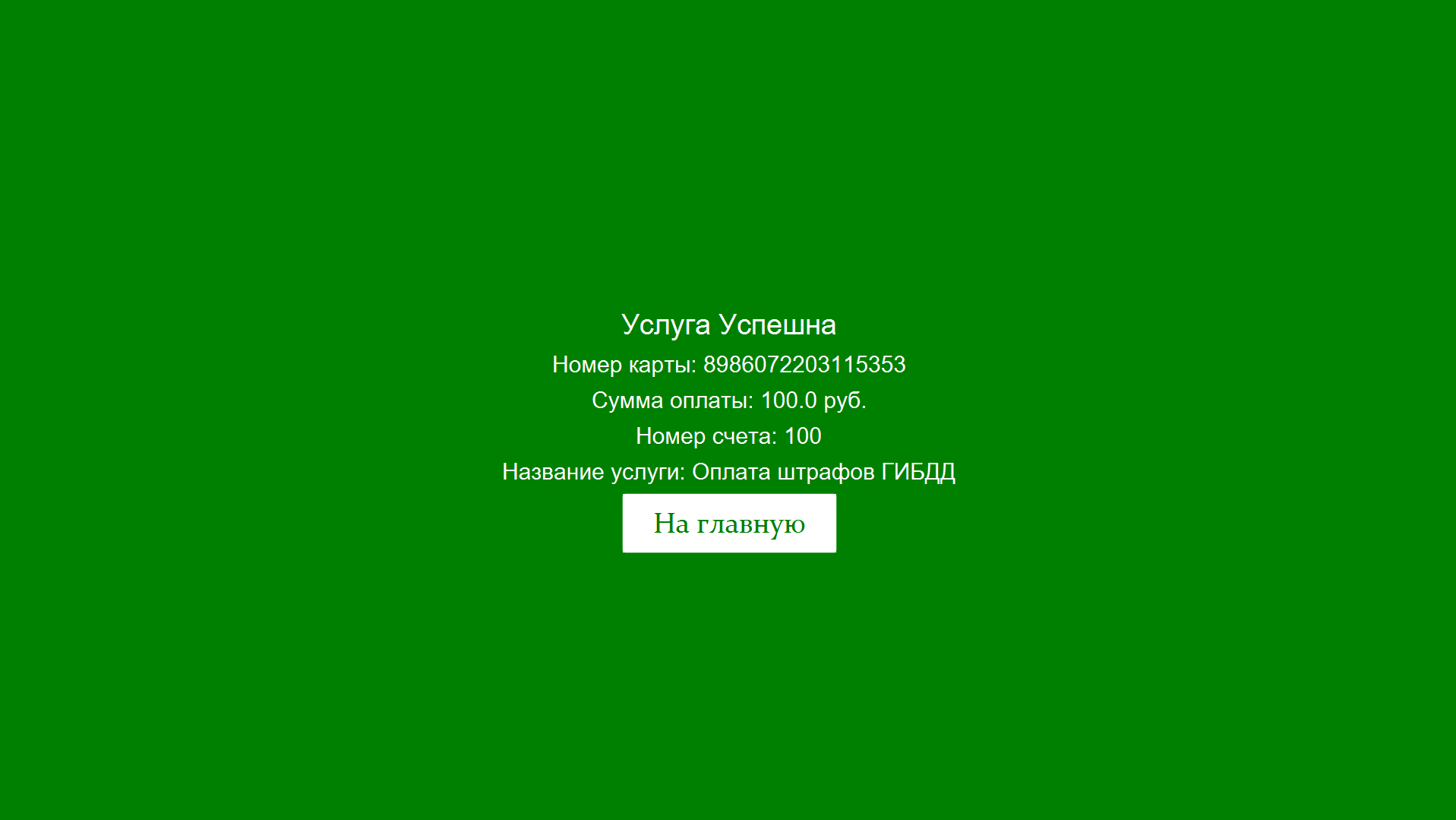
Рисунок 38-39 — Смена раскладки на виртуальной клавиатуре.

В завиисмости от услуги, можно моделировать особенности ввода счета. Но для простой модилирования оплаты услуги достаточно и ввода цифренного номера счета. После ввода его в поле, клиент нажимает на кнопку «Продолжить» и попадет на сцену с вводом суммы оплаты. Эта сцена индентична с той что мы вводили в вводе карты, пин-кода, или снятия со счета, так что ничего нового в ней не добавлено.

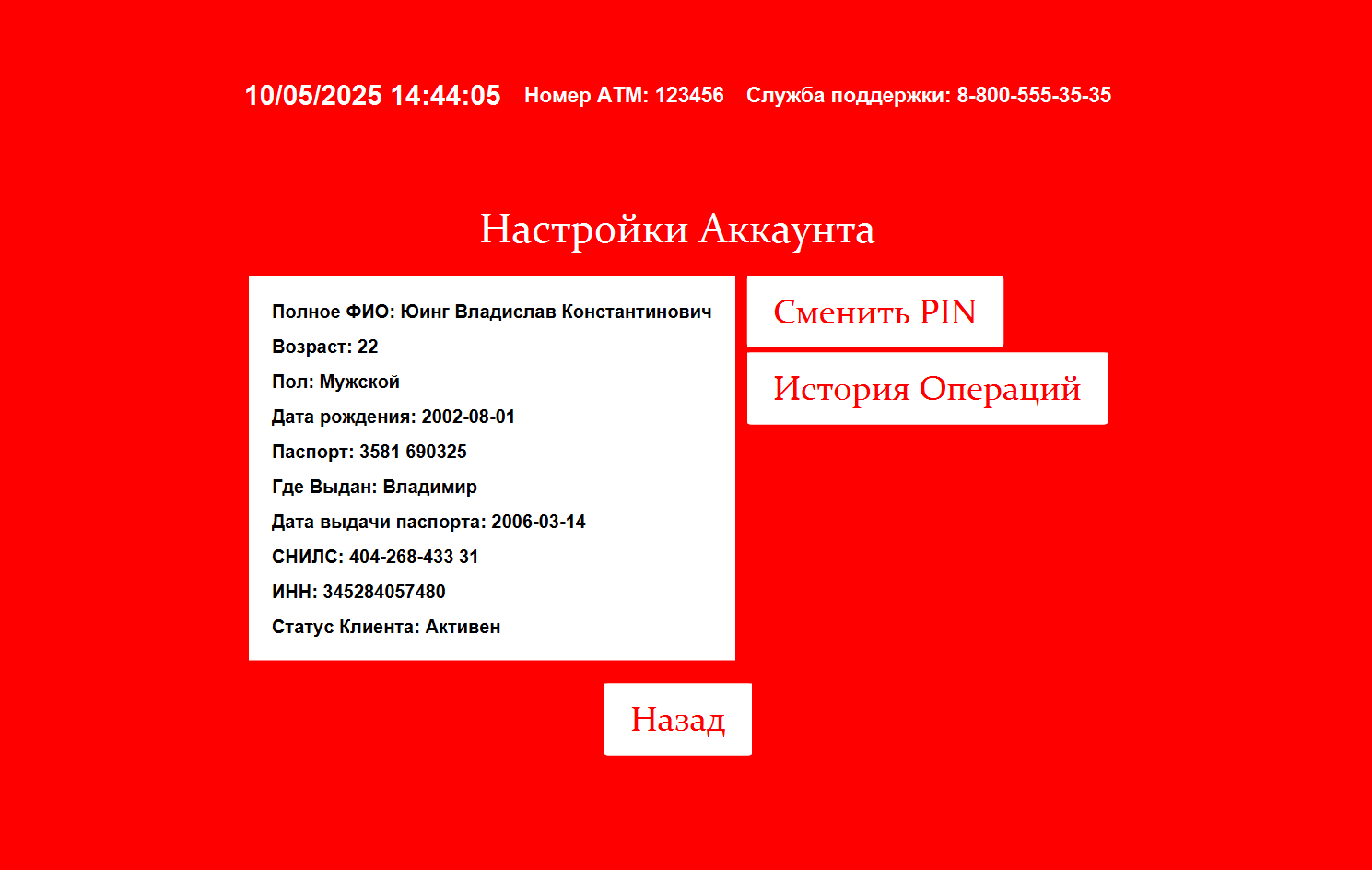
После того как мы ввели сумму оплаты, сцена переходит на страницу загрузки оплаты. Интерактивный кружек заполняется на сцене, пока на фоне идет процесс обращения к базе данных с обновлением баланса клиента и созданием лога об оплате услуги.

Рисунок 40 — Загрузочная сцена с оплатой услуги

Если все хорошо, и операции завершены, то пользователь попадает на сцену с чеком об операции. В ней будет указано какой счет, сколько оплатиили и название услуги. Снизу него есть кнопка на главную, которая возвращает сцену на домашнеее меню клиента.

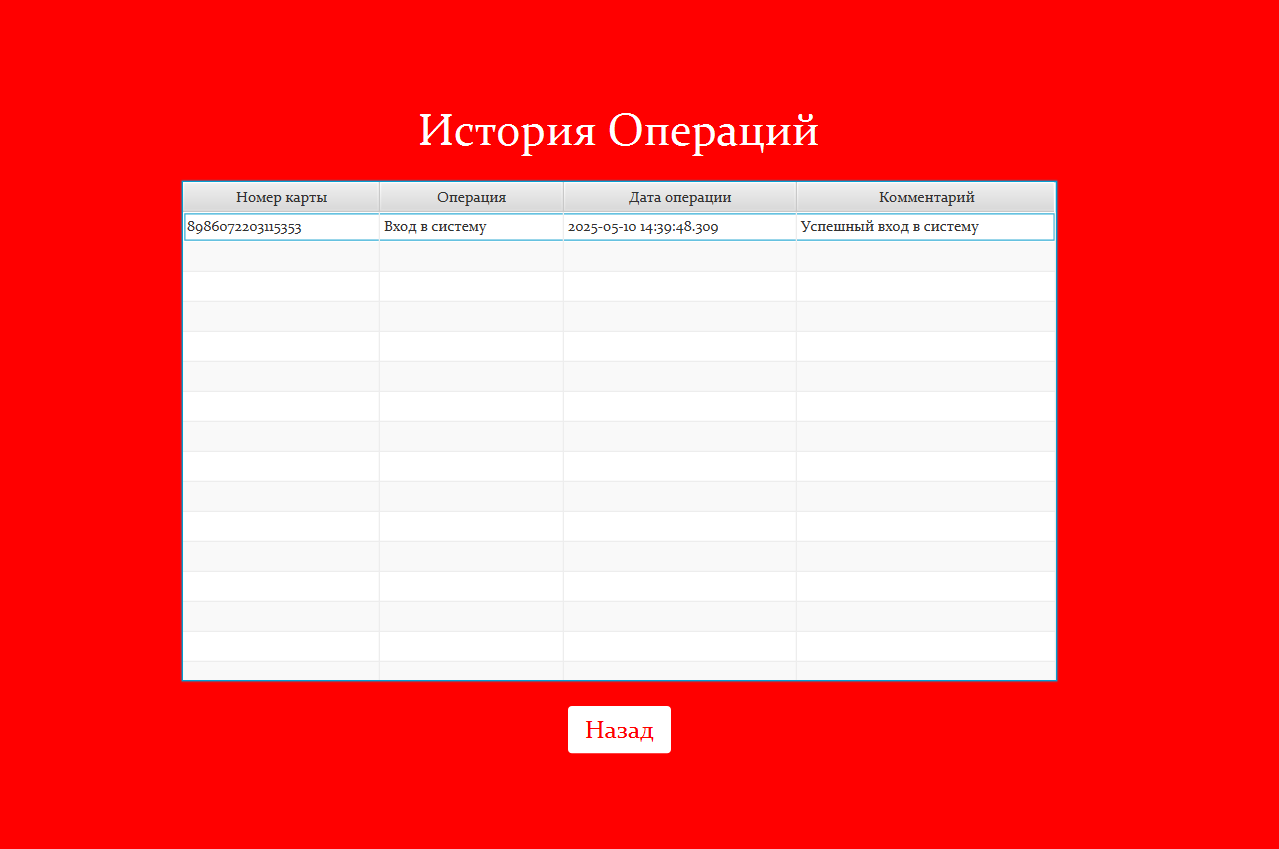
Рисунок 41 — Сцена с выводом информации об операции по оплате услуги

### 3.2.12 Разработка сцены настроек и модуля смены пин-кода

  
Рисунок 42 — Сцена с настройками аккаунта клиента

На сцене отображена вся информация из таблицы CLIENT\_INFO в котором указаны все данные клиента данного счета. Справа есть кнопки управления, отображающие различные действия.

При нажатии на кнопку «История операций» мы отправляем клиента на сцену с таблицей в которой выводится все операции из логов по номеру счета клиента. На этой сцене может познакомится со своими действиями.

Рисунок 43 — Сцена с выводом инфомации действия пользователя.

Когда клиент нажимает на кнопку «Смена пин-кода» он преходит на сцену по смене пин-кода. От клиента требуется ввести текущий пин-код, новый пин-код и подтвердить новый пин-код. Сцена проверяет на совпадение всех полей, и если все верно , то обрабатывает операцию.

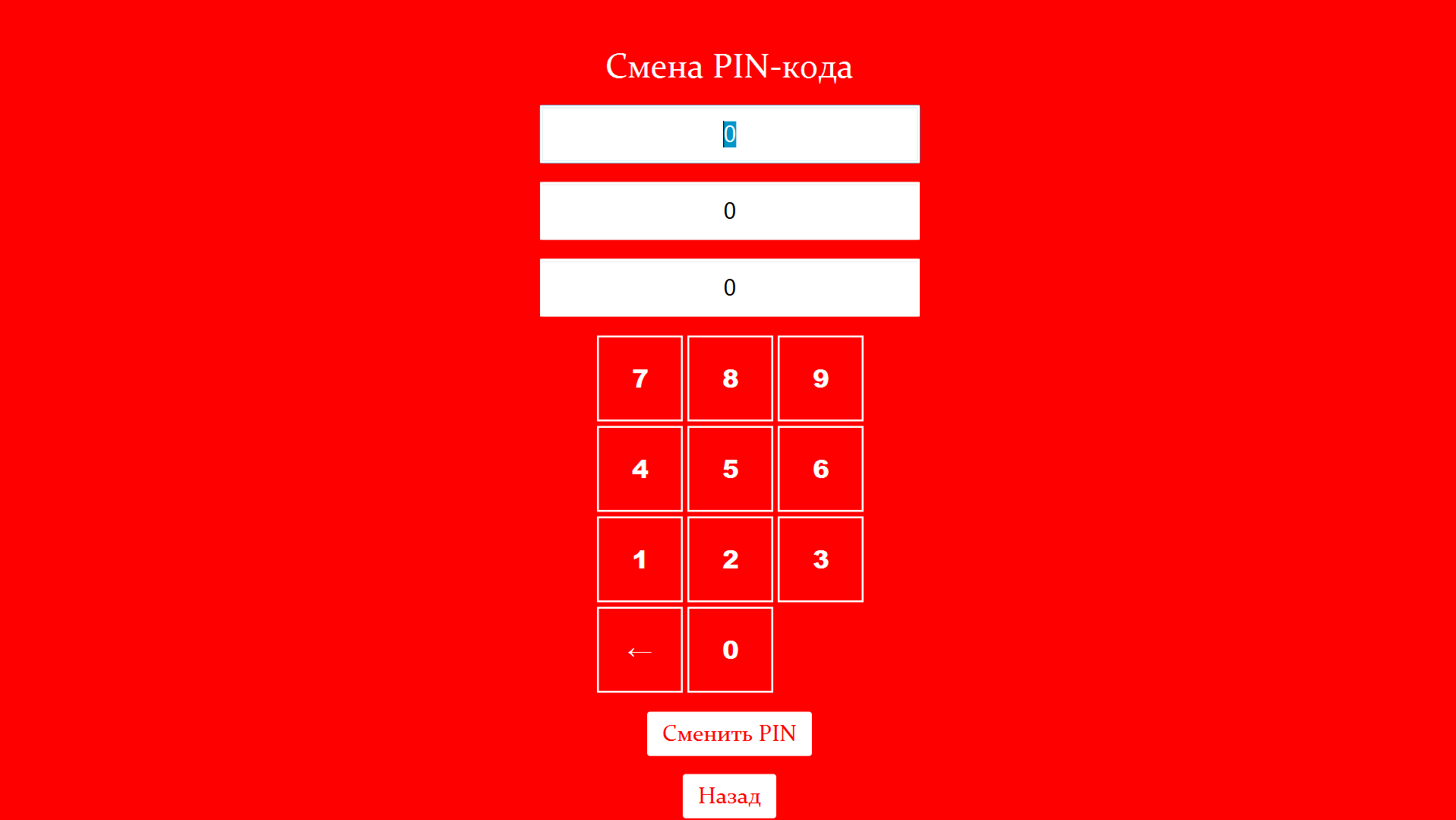
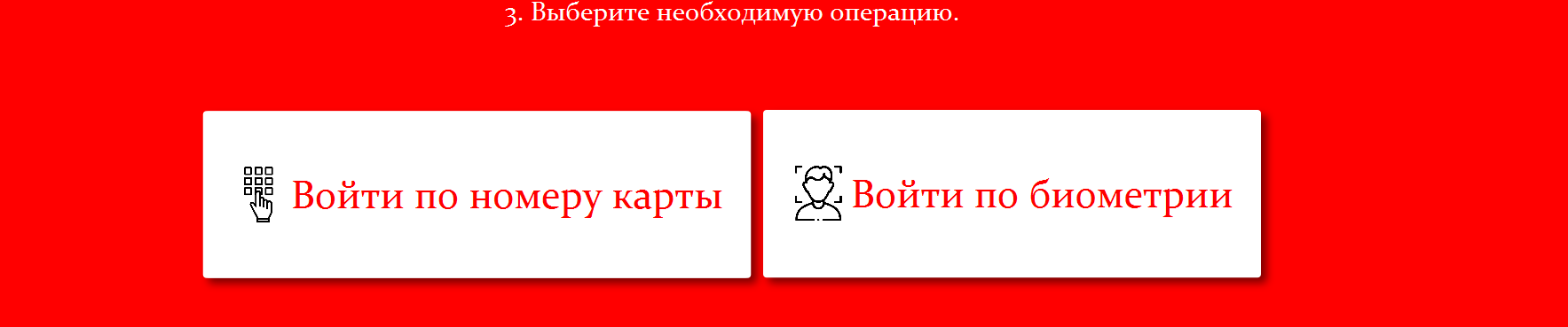
Рисунок 44 — Сцена по смене пин-кода.

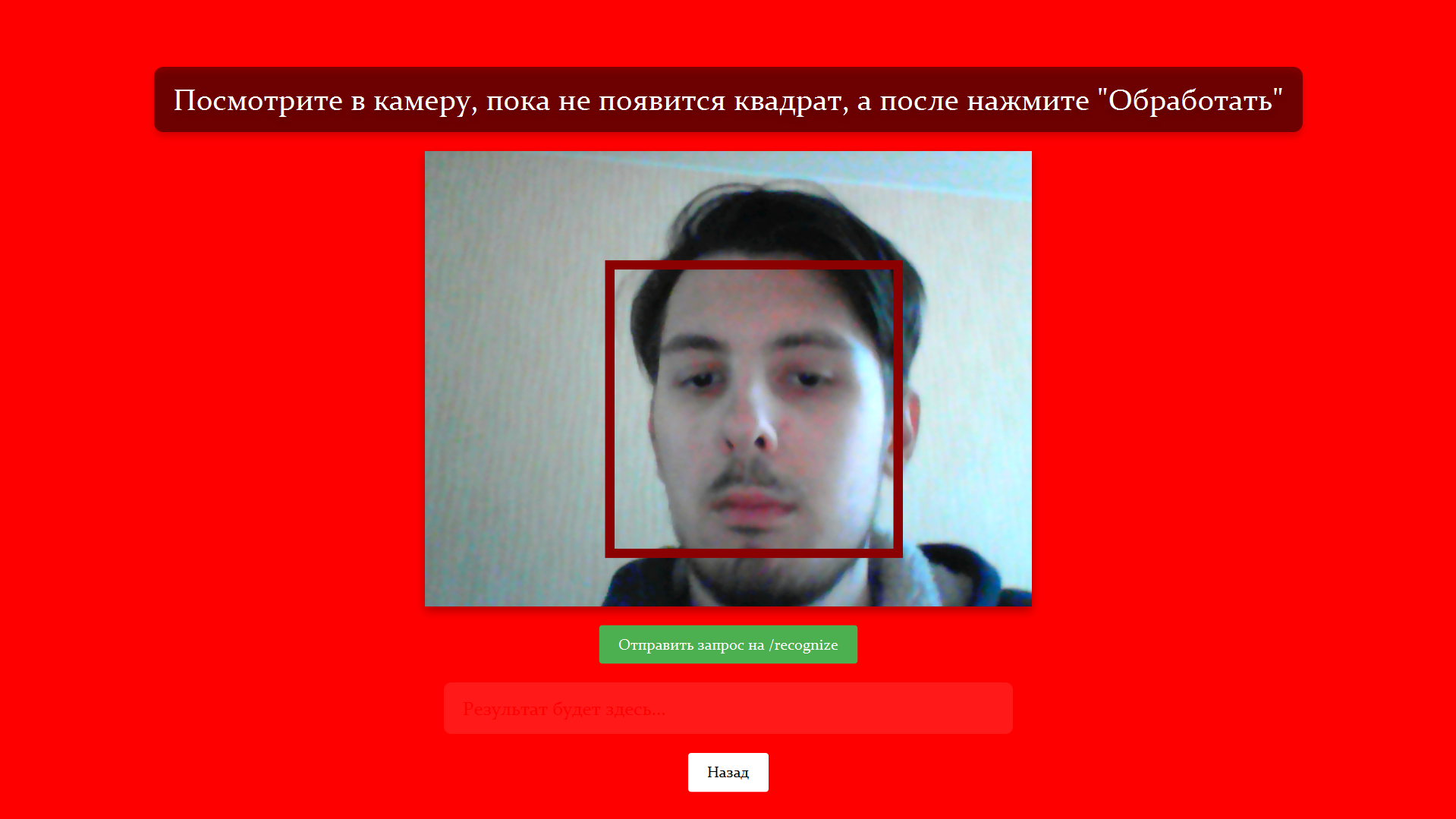
Рисунок 45 — Следующая сцена с уведомлением пользователя о том что его пин-код сменен.

### 3.2.13 Разработка сцены авторизации через биометрию

Если сервер по биометрии доступен по адресу http://127.0.0.1:8080, то мы делаем доступным кнопку «Вход по биометрии». Также мы можем проверять доступность сайта каждые 3 секунды, чтобы не требовалось перезагружать приложение.

Рисунок 46 — Активна кнпока входа по биометрии

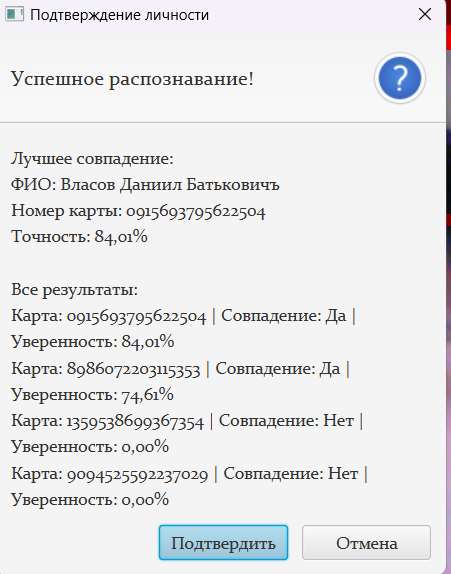
После того как клиент нажмет на кнопку входа по биометрии, его перекинет на сцену с авторизацией по лицу. Сцена представлет окн ос выводом веб-камерой и кнопками по авторизацией или же отмены.

Рисунок 47. Сцена с биометрией лица для авторизации на счет.

Лицо мы обрабатываем через Open-CV в котором через каскад для лиц, мы определяем границу лица. Далеее мы уже обрабатываем запрос на сервер. Порядок такой:

* 1. Захватываем лицо с вебкамеры (при условии что есть красный квадратик).
  2. Отправляем лицо через массив битов на сервер по запросу POST /recognize
  3. Сервер обрабатывает лицо (делает векторизацию и находит соотнощение)
  4. Сервер позвращает запросу ответ в виде: Номер карты, соотношение и прочее.
  5. Класс с биометрией получает ответ и выводит его в модальном окне.

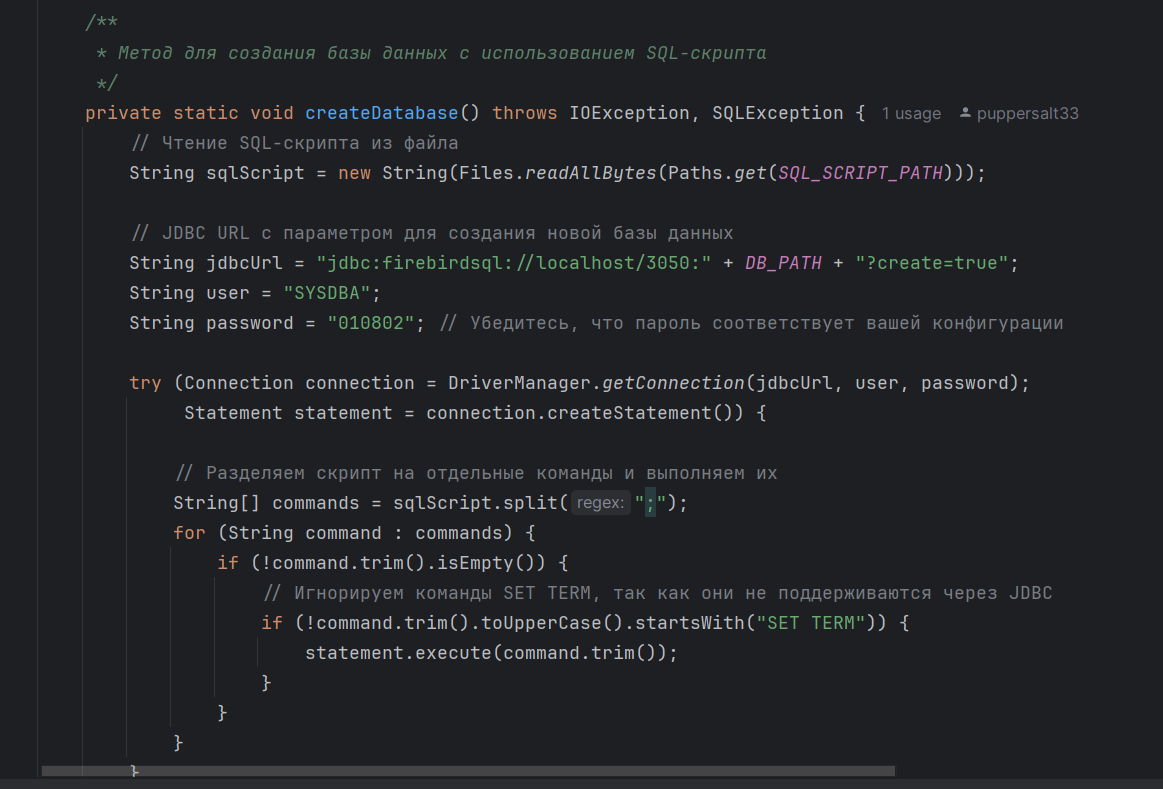
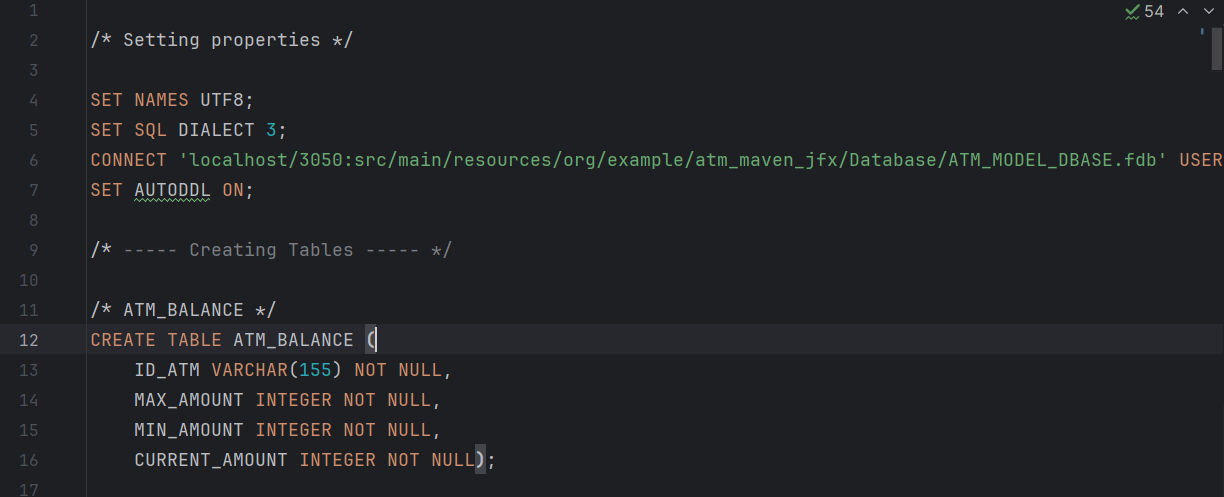
В предложенном окне, выдается информация для клиента, и кнопки действия. Помимо информации о найденом человеке, выводим информацию о схожим данным и % соотношения. Итогом будет выбор из двух кнопок «ОК» или «Отмена», где ок перекинет на сцену ввода пин-кода по номеру карты, полученный из запроса.

Рисунок 48 — Окно с подверждением личности после биометрии.

## 3.3 Разработка прочих модулей для разработки приложения.

### 3.3.1 Разработка модуля автосоздания базы данных.

Для того чтобы приложение можно было развернуть без проблем на любом устройстве, требуется автоматизировать ее развертование. Учтем то, что для работы приложения необходима работающая база данных. Для этого в приложении предусмотрен скрипт по созданию базы данных из заготовленного SQL файла, который запускается перед запуском JavaFX сцены. Этот файл содержит настройки базы, таблиц, процедур, функций и также имеет несколько строк INNER JOIN для важных таблиц в которой заполнены необходимые данные. Файл конфигурациии базы данных должен находиться там же где указан путь к ней. Тем самым мы проверяем о существовании базы данных, если файл базы данных есть, то создание не требуется, иначе создаем ее программно.

Рисунок 49-50 — Скрипт создания базы даных из приложения.

### 3.3.2 Разработка модуля сессии клиента

Для того чтобы обезопасить пользователя, и моделировать работу защиты банкомата, был реализован модуль сессии для клиента. Алгоритм построен на функции таймера. Когда таймер запускается, есть 10 секунд на действия пользователей. Что будет дальше уже было описано выше. Другие функции данного модуля влкючают: остановка таймер — полное прекращение работы на сцене и сброс таймера — обновление времени таймера на 10 секунд.

На особых страницах, сначала запускается таймер через функцию, для кнопок действия включена функция сброса таймера, если клиент пользователь переходит на другую сцену, таймер останавливаем.

## 3.4 Результаты разработки приложения

В ходе разработки приложения для моделирования работы банкомата была создана стабильная версия программного обеспечения, соответствующая современным требованиям безопасности, функциональности и удобства. Основные задачи, связанные с реализацией клиентских и административных функций, успешно выполнены. Приложение включает авторизацию технического персонала с расширенным администраторским меню для управления состоянием банкомата, инкассации, блокировки устройства и настройки параметров. Также реализованы модуль управления услугами (оплата коммунальных платежей, мобильной связи и других сервисов), система блокировки устройства при сбоях или несанкционированном доступе, а также демо-режим для ознакомления пользователей с интерфейсом без финансовых операций.

Для клиентов разработано начальное окно банкомата с выбором языка и приветственным сообщением. Авторизация пользователей осуществляется через ввод пин-кода и биометрическую аутентификацию по лицу, что обеспечивает высокий уровень защиты данных. В приложении интегрированы модули снятия и пополнения купюр с поддержкой различных номиналов, проверкой корректности сумм и наличия средств на счете. Добавлен функционал оплаты услуг и настройки параметров счета клиента, включающий установку лимитов, привязку карт и изменение контактных данных. Все модули взаимосвязаны и обеспечивают комплексное выполнение операций, необходимых для повседневной работы банкомата.

Полученный продукт представляет собой полностью функциональное приложение, моделирующее ключевые аспекты взаимодействия с банкоматом. Реализованные функции охватывают как клиентские задачи, так и административные процессы, обеспечивая надежность и гибкость системы. Несмотря на выявленные недочеты, такие как необходимость оптимизации производительности и расширения возможностей биометрической аутентификации, приложение может служить основой для дальнейшего развития, включая интеграцию новых сервисов и улучшение пользовательского интерфейса.

# **4. Тестирование**

Тестирование приложения стало ключевым этапом разработки, поскольку позволило выявить и устранить недочеты, обеспечить стабильную работу всех функциональных возможностей и подтвердить соответствие приложения заявленным требованиям. Основное внимание при тестировании было уделено проверке пользовательского интерфейса, логики приложения и взаимодействия с базой данных

Перед началом тестирования был составлен план, который включал:

* Цели тестирования: проверка корректной работы всех функций, обеспечение безопасности данных, проверка удобства интерфейса.
* Объекты тестирования: пользовательский интерфейс, логика приложения, взаимодействие с базой данных.
* Сценарии тестирования: создание тестовых данных, проверка различных сценариев использования приложения.

## 4.1 Тестирование пользовательского интерфейса

Тестирование пользовательского интерфейса было направлено на проверку корректности отображения элементов, их функциональности и удобства использования.

### 4.1.1 Окно авторизации для технического специалиста.

* Сценарий 1: Ввод корректных данных для входа.
* Действия: Ввести логин и пароль существующего пользователя. Нажать кнопку "Войти".
* Ожидаемый результат: Пользователь успешно авторизуется и перенаправляется на соответствующую панель (меню пользователя, меню администратора)
* Сценарий 2: Ввод некорректных данных для входа.
* Действия: Заведомо ввести неверные данные пользователя в виде логина и пароля. Нажать кнопку «Войти»
* Ожидаемый результат: Приложение выдаст ошибку, выведит ее пользователю. Приложение будет ожидать ввода новых данных.
* Сценарий 3: Проверка подключения к базе данных. Условие: Успешное подключение.
* Действия: Запуск приложения.
* Ожидаемый результат: надпись соеденения выдает результат «Соеденинение установлено»
* Сценарий 4: Проверка подключения к базе данных. Условие: Провал подключения.
* Действия: Запуск приложения.
* Ожидаемый результат: надпись соединения выдает результат ошибки подключения.
* Сценарий 5: Закрытие приложения.
* Действия: Нажатие кнопки закрытия, закрытие приложения вручную
* Ожидаемый результат: Приложение закрывается.

### 4.1.2 Сцена администратора и его меню

* Сценарий 1: Просмотр сцены с администраторскиими кнопками
* Действия: Вход в меню.
* Ожидаемый результат: Пользователь видит кнопки.
* Сценарий 2: Переход в сцену по кнопке.
* Действия: Нажатие на кнопку в панеле
* Ожидаемый результат: Открытие новой сцены

### 4.1.3 Сцена инкассации

* Сценарий 1: Просмотр сцены
* Действия: Вход на сцену
* Ожидаемый результат: Пользователь видит все текстовые элементы, заполную таблицу, кнопки управления.
* Сценарий 2: Добавление купюр
* Действия: Нажатие на кнопку добавить
* Ожидаемый результат: Открытие меню с добавлением
* Сценарий 3: Ввод корректных значений для добавления
* Действия: Ввод в поля корректных значений для добавления
* Ожидаемый результат: Открытие сцены загрузки купюр, отсуствие ошибок при добавлении
* Сценарий 4: Ввод некорректных значений для добавления
* Действия: Ввод в поля некорректных значений для добавления
* Ожидаемый результат: Вывод ошибки на экран сцены.
* Сценарий 5: Удаление купюр
* Действия: Нажатие на кнопку «Удалить»
* Ожидаемый результат: Вывод модального окна с полями номиналов.
* Сценарий 6. Ввод корректных значений для удаления
* Действия: Ввод в поля корректных значений для удаления
* Ожидаемый результат: Удаление купюр, переход на сцену загрузки.
* Сценарий 7: Ввод некорректных значений для удаления
* Действия: Ввод в поля некорректных значений для удаления
* Ожидаемый результат: Вывод ошибки на сцену
* Сценарий 8: Обновление данных через кнопку «Обновить»
* Действия: Нажатия на кнопку «Обновить»
* Ожидаемый результат: Сцена обновит все значения в обьектах.

### 4.1.4 Сцена управления услугами

* Сценарий 1: Открытие сцены
* Действия: Переход на сцену
* Ожидаемый результат: Загрузка всех обьектов и заполнение их
* Сценарий 2: Добавление новой услуги и ввод корректного значения
* Действия: Нажатие на кнопку, Написание название услуги уникальной и правильного типа
* Ожидаемый результат: Услуга добавляется
* Сценарий 3: Добавление новой услуги и ввод некорректного значения
* Действия: Нажатие на кнопку, Написание название услуги с ошибками или не уникальной
* Ожидаемый результат: Вывод ошибки, ожидание ввода верного названия.
* Сценарий 4: Выбор услуги и удаление ее
* Действия: Выбрать услуги через галочку в первом столбце, нажатие на кнопку «Удалить»
* Ожидаемый результат: Вылет подтверждения, удаление услуг из базы данных
* Сценарий 5: Изменение статуса услугаи (Активен/Неактивен)
* Действия: Найти услугу, выбрать режим активности (Смена галочки), нажатие на кнопку «Сохранить Изменения»
* Ожидаемый результат: Вылет подтверждения, обновление статуса услуг в базе данных.

### 4.1.5 Сцена блоикровки банкомата

* Сценарий 1: Переход в режим блокировки
* Действия: Перейход по кнопке в админитраторском меню
* Ожидаемый результат: Открытие сцены без кнопок управления, отключение от действий клиента.

### 4.1.6 Сцена запуска привественного окна банкомата

* Сценарий 1: Запуск банкомата для клиента с выключеным сервером
* Действия: Нажатие на кнопку «Запустить»
* Ожидаемый результат: Окно с админ меню закрывается, Окно с клиентской части открывается в новом окне. Доступные кнопки: Авторизация по номеру карты.
* Сценарий 2: Запуск банкомата для клиента с включеным сервером
* Действия: Нажатие на кнопку «Запустить»
* Ожидаемый результат: Окно с админ меню закрывается, Окно с клиентской части открывается в новом окне. Доступные кнопки: Авторизация по номеру карты. Авторизация по биометрии

### 4.1.7 Сцена авторизации по номеру карты

* Сценарий 1: Ввод существующей карты
* Действия: Ввод 16-ти цифр существующей карты
* Ожидаемый результат: Переход на сцену для ввода пин-кода
* Сценарий 2: Ввод несуществующей карты
* Действия: Ввод 16-ти цифр несуществующей карты
* Ожидаемый результат: Вывод ошибки о том что такой карты нету
* Сценарий 3: Ввод номера карты меньше 16-ти цифр
* Действия: Ввод номера карты до 16 цифр
* Ожидаемый результат: Вывод ошибки о том что номер карты должен быть не менее 16 цифр
* Сценарий 4: Попытка ввода буквеных симоволов и спецсимволов
* Действия: Ввод в поле номера карты нецифренные символы
* Ожидаемый результат: Вывод ошибки о том что номер карты должен состоять из цифр.

### 4.1.8 Сцена биометрии для авторизации

* Сценарий 1: Корректная отправка лица на сервер
* Действия: Правильное расположение лица на вебкамере
* Ожидаемый результат: Запрос пройдет успешно, выведет окно с подтверждением
* Сценарий 2: Неправильное расположение лица на вебкамере
* Действия: Сделать так чтобы лицо не было видно на вебкамере
* Ожидаемый результат: Запрос вернет значения Null, тем самым останавливая авторизацию

### 4.1.9 Сцена сцены ввода пин-кода

* Сценарий 1: Ввод правильного пин-кода
* Действия: Ввод 4 цифры верного пин-кода
* Ожидаемый результат: Переход на сцену главного меню
* Сценарий 2: Ввод неправильного пин-кода
* Действия: Ввод 4 цифры неверного пин-кода
* Ожидаемый результат: Отказ от авторизации.
* Сценарий 3: Ввод пин-кода менее 4 цифр.
* Действия: Ввод номера карты до 16 цифр
* Ожидаемый результат: Вывод ошибки о том что пин-код должен быть не менее 4 цифр
* Сценарий 4: Попытка ввода буквеных симоволов и спецсимволов
* Действия: Ввод в поле номера карты нецифренные символы
* Ожидаемый результат: Вывод ошибки о том что пин-код должен состоять из цифр.

### 4.1.10 Сцена сцены ввода значений для вывода среств

* Сценарий 1: Ввод циферного значения суммы снятия
* Действия: Ввод цифр для снятия
* Ожидаемый результат: Переход на сцену с загрузкой
* Сценарий 2: Ввод символьного значения
* Действия: Ввод символьных значений для суммы снятия
* Ожидаемый результат: Вывод ошибки о том что должны быть цифры в поле
* Сценарий 3: Ввод суммы больше суммы баланса счета
* Действия: Ввод суммы превышающая баланс счета
* Ожидаемый результат: Вывод ошибки о недостаточности денег на счету
* Сценарий 4: Ввод суммы более 15.000 или меньше 1
* Действия: Ввод в поле суммы превышающее 15.000 или меньше 1.
* Ожидаемый результат: Вывод ошибки о том сумма должна быть в лимите

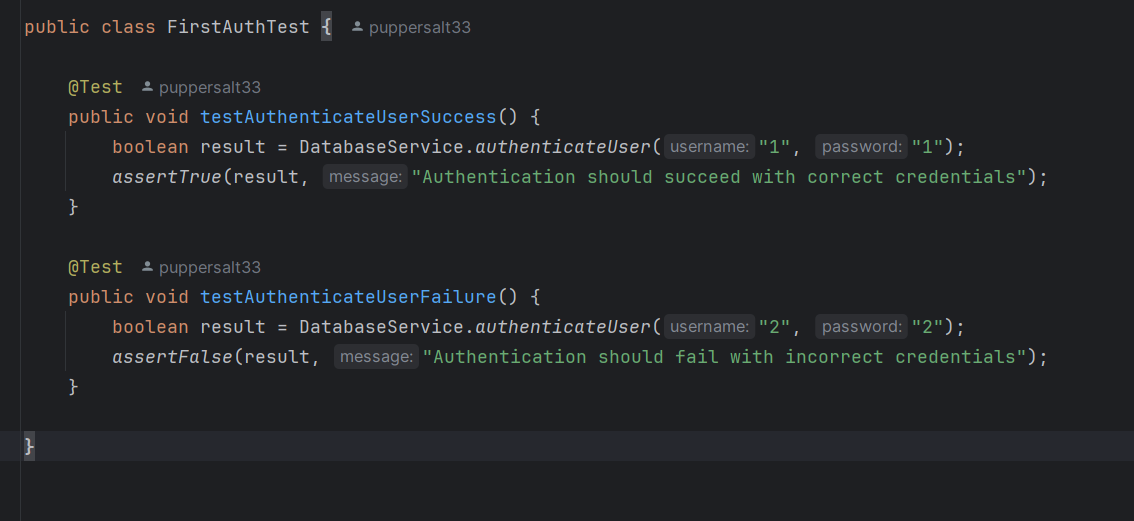
### 4.1.10 Сцена сцены ввода значений для счета и оплаты услуг

* Сценарий 1: Ввод циферного значения суммы оплаты
* Действия: Ввод цифр для оплаты
* Ожидаемый результат: Переход на сцену с загрузкой
* Сценарий 2: Ввод символьного значения
* Действия: Ввод символьных значений для суммы оплаты
* Ожидаемый результат: Вывод ошибки о том что должны быть цифры в поле
* Сценарий 3: Ввод суммы больше суммы баланса счета
* Действия: Ввод суммы превышающая баланс счета
* Ожидаемый результат: Вывод ошибки о недостаточности денег на счету
* Сценарий 4: Ввод суммы более 15.000 или меньше 1
* Действия: Ввод в поле суммы превышающее 15.000 или меньше 1.
* Ожидаемый результат: Вывод ошибки о том сумма должна быть в лимите

## 4.2 Тестирование взаимодействия с базой данных

Тестирование взаимодействия с базой данных работает на основе автотестов перед компиляцией проекта. Так как большенство опреаций в приложении работают через процедуры, мы можем их моделировать на этапе сборки проекта.

Например мы можем протестировать вход специалиста в админменю через окно авторизации.



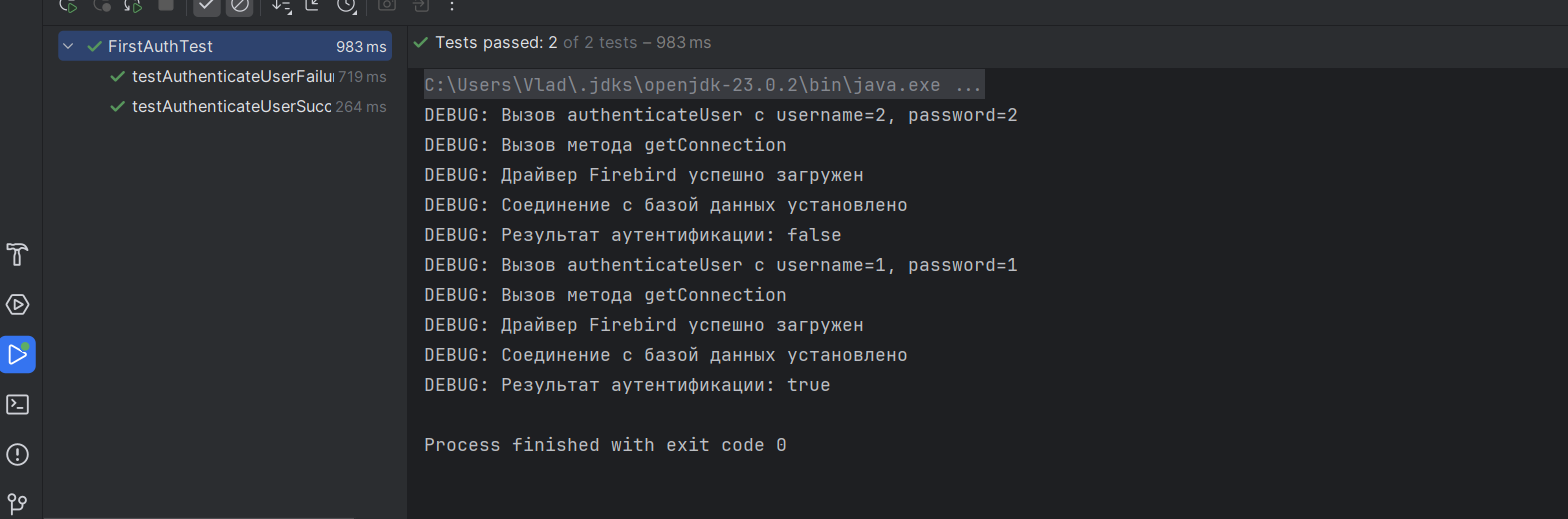
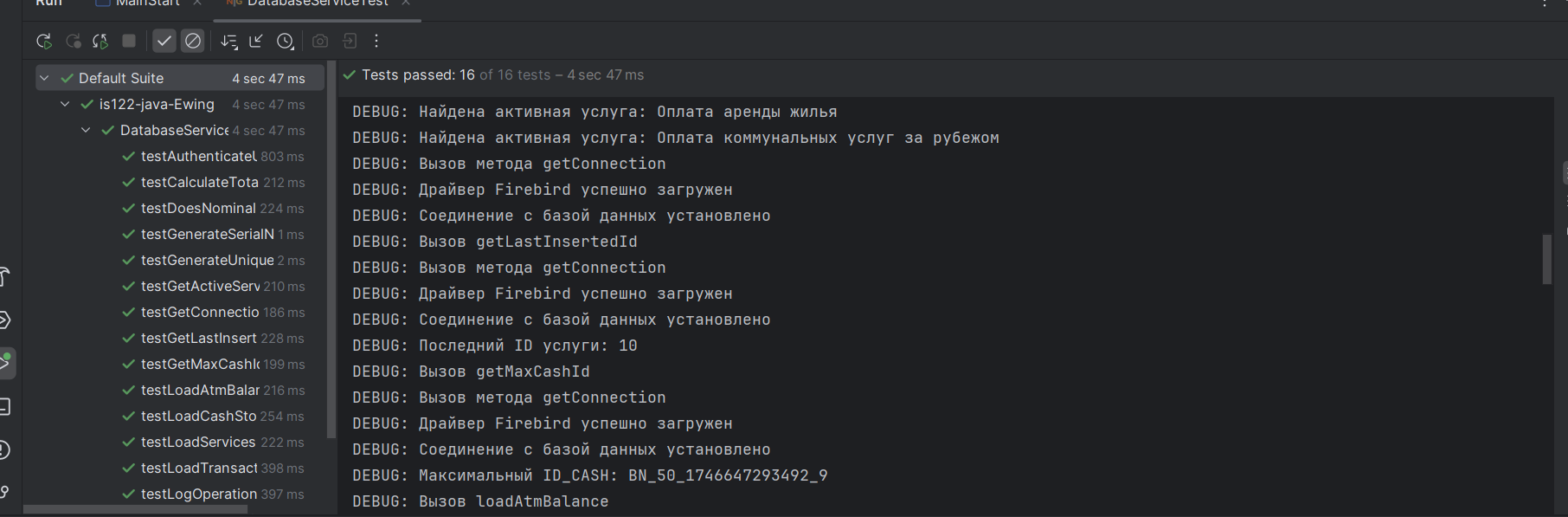


Рисунок 51-52 — Автотест на проверку входа специалиста

Также есть автотесты на основные процедуры для различных операций. Например как вывод баланса, проверки существования карты и т. д.

Рисунок 53 — Автотесты процедур в приложении.

## 4.4. Результаты тестирования

В ходе тестирования разработанного приложения для моделирования работы банкомата была проверена корректность выполнения всех реализованных функций, устойчивость системы к внештатным ситуациям и соответствие требованиям безопасности. Основное внимание уделялось проверке взаимодействия модулей, правильности обработки финансовых операций, а также надежности авторизационных механизмов. Тестирование проводилось в несколько этапов, включая модульное тестирование отдельных компонентов, интеграционное тестирование для проверки взаимодействия между модулями и нагрузочное тестирование для оценки производительности системы в условиях высокой интенсивности использования. В результате выявлено и устранено несколько критических ошибок, связанных с некорректной обработкой ввода данных и временным сбоем в работе биометрической аутентификации. Основные функции, такие как снятие и пополнение купюр, оплата услуг и блокировка банкомата, продемонстрировали стабильную работу, что подтверждает готовность приложения к практическому применению.

Ключевые модули приложения прошли всестороннюю проверку. Модуль авторизации технического персонала и клиентов показал высокую степень защиты от несанкционированного доступа, включая корректную работу биометрической аутентификации по лицу. Интеграционное тестирование модуля инкассации подтвердило точность учета наличности и формирования отчетов для администраторов, а также стабильность работы при изменении параметров кассет с купюрами. Тестирование модулей управления услугами и настройки параметров счета выявило единичные ошибки в обработке прерванных транзакций, которые были оперативно исправлены. Нагрузочные испытания показали, что система сохраняет работоспособность при одновременном выполнении до 50 операций, однако для повышения производительности в дальнейших версиях рекомендуется оптимизировать алгоритмы обработки данных.

Итоговые результаты тестирования подтверждают, что разработанное приложение соответствует заявленным целям и готово к использованию в условиях эксплуатации. Выявленные недочеты, такие как временные задержки при обработке сложных запросов и необходимость улучшения интерфейса административного меню, не влияют на основную функциональность и могут быть доработаны в рамках последующих обновлений. Эффективность тестирования позволила обеспечить высокую надежность системы, а также выработать рекомендации по дальнейшему улучшению пользовательского опыта и расширению возможностей банкомата. Полученные данные свидетельствуют о том, что приложение может служить надежной основой для внедрения в реальные банковские системы с последующей адаптацией под специфические требования.

Приложение тестировалось на следующих системах:

Сиcтема 1:

* + - * Процессор: Intel Core 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz.
      * Оперативная память: 8 гб DDR4.
      * Операционная система: 11 Windows Home 23H2 64x

Система 2:

* + - * Процессор: Intel Core 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz.
      * Оперативная память: 8 гб DDR4.
      * Операционная система: Linux Kubuntu 9

Система 3:

* + - * Процессор: Intel Core 5th Gen Intel(R) Core(TM) i5-4460
      * Оперативная память: 32 гб DDR3
      * Операционная система: Linux Kubuntu 9 / Linux Mint / Linux Majaro / Red OS 8 / Windows 10 Pro 24H4

# **Заключение**

В данной курсовой работе была разработана клиент-серверная система моделирования работы банкомата с целью автоматизации ключевых операций, связанных с обслуживанием клиентов и администрированием устройства. Программное обеспечение реализует функции авторизации пользователей и технического персонала, управления наличностью, выполнения финансовых операций. Разработанное приложение позволяет хранить, обрабатывать и выводить данные, необходимые для корректной работы банкомата. В рамках проекта были достигнуты основные цели: создана архитектура базы данных, реализован интерфейс пользователя, интегрированы модули снятия и пополнения купюр, оплаты услуг, биометрической аутентификации, а также обеспечены механизмы контроля доступа и защиты данных.

В процессе исследования были проанализированы существующие аналоги банкоматных систем, выбраны подходящие инструменты для реализации (языки программирования, СУБД, библиотеки для распознавания лиц), разработана структура базы данных и алгоритмы для взаимодействия с ней.

Созданное программное обеспечение отличается интуитивно понятным интерфейсом, что делает его доступным для пользователей с разным уровнем подготовки. Для клиентов реализованы удобные механизмы авторизации (включая биометрию по лицу), быстрое выполнение операций снятия наличных, пополнения счета и оплаты услуг. Для технического персонала разработаны инструменты управления устройством: инкассация, блокировка банкомата, настройка параметров. Итоговый продукт демонстрирует высокую эффективность в моделировании работы банкомата, снижает вероятность ошибок в финансовых операциях и повышает прозрачность административных процессов. Полученные результаты подтверждают возможность использования разработанного приложения в реальных банковских системах с последующей адаптацией под специфические требования.

# Список литературы

1. Хорстманн К., Корнелл Г. Java. Базовый курс . — Москва: Вильямс, 2017.
2. Шилдт Г. Java 8. Полное руководство . — Москва: Вильямс, 2015.
3. Фрисби Д. JavaFX 8. Создание графических приложений . — Москва: ДМК Пресс, 2015.
4. Горнер К. JavaFX 8. Введение в создание графических приложений . — Москва: ДМК Пресс, 2016.
5. Сухин И. JavaFX для начинающих . — Санкт-Петербург: БХВ-Петербург, 2018.
6. Васильев А. Разработка приложений на Java с использованием JavaFX . — Санкт-Петербург: Питер, 2020.
7. Абрамова Н. Основы программирования на Java . — Москва: Кудиц-Образ, 2019.
8. Oracle. Документация JavaFX [Электронный ресурс]. — Режим доступа: https://openjfx.io/ .
9. Habr. Руководства и статьи по Java и JavaFX [Электронный ресурс]. — Режим доступа: https://habr.com/ru/post/442796/ .
10. JavaRush. Образовательный портал по Java [Электронный ресурс]. — Режим доступа: https://javarush.ru/ .

# ПРИЛОЖЕНИЕ А — Источник исходного кода приложения

https://github.com/MemorialVietnama/is122-java-Ewing

