Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет инженерно-экономический

Кафедра экономической информатики

Дисциплина «Объектно – ориентированное проектирование и программирование»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  Старший преподаватель кафедры ЭИ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю. Ю. Петрович |
|  | \_\_\_.\_\_\_\_.2024 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИ СКА**

к курсовой работе

на тему:

**«ПРОГРАММНОЕ СРЕДСТВО С АВТОМАТИЗИРОВАННОЙ СИСТЕМОЙ РАСЧЁТА БАНКОВСКИХ УСЛУГ И ТРАНЗАКЦИОННЫХ ПЛАТЕЖЕЙ»**

БГУИР КР 1–40 05 01–02 ПЗ

|  |  |
| --- | --- |
|  | Выполнил студент группы 272303  БАДЕЙ Павел Юрьевич  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовая работа представлена на проверку \_\_\_.\_\_\_\_.2024  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2024

**РЕФЕРАТ**

БГУИР КР 1–40 05 01–02 ПЗ

**Бадей, П.Ю.** Программное средство с автоматизированной системой расчёта банковских услуг и транзакционных платежей/              П.Ю. Бадей. – Минск: БГУИР, 2024. – 39 с.

Пояснительная записка 39 с., 33 рис., 7 источников, 4 приложения

ПРОГРАММНОЕ СРЕДСТВО С АВТОМАТИЗИРОВАННОЙ СИСТЕМОЙ РАСЧЁТА БАНКОВСКИХ УСЛУГ И ТРАНЗАКЦИОННЫХ ПЛАТЕЖЕЙ.

*Цель* *курсовой работы*: систематизация и углубление теоретических знаний, полученных за время обучения, а так же приобретение и закрепление навыков самостоятельной работы.

*Методология проведения работы*: в процессе решения поставленных задач использованы принципы системного подхода и аналитические методы.

*Результаты работы*: выполнена постановка задачи и определены основные методы ее решения; в ходе объектного моделирования системы построен ряд *UML*–диаграмм; разработаны модели бизнес–процессов предметной области на основе нотаций IDEF0 и BPMN; описаны основные алгоритмы работы программного средства; разработано руководство пользователя; выполнено тестирование программного средства, показавшее его соответствие функциональным требованиям, поставленным в задании на разработку.

Программный продукт разработан на языке *JavaScript* с применением веб-фреймворка express.

*Область применения результатов*: c помощью разработанного программного средства можно автоматизировать работу банков по предоставлению и оказанию услуг.

**СОДЕРЖАНИЕ**

**ВВЕДЕНИЕ**

В современном банковском секторе автоматизация играет ключевую роль в обеспечении эффективности и надежности банковских услуг. Программное средство с автоматизированной системой расчёта банковских услуг и транзакционных платежей (АБС) представляет собой комплекс программного и технического обеспечения, разработанный для автоматизации банковской деятельности.

Автоматизированные банковские системы (АБС) имеют долгую и интересную историю развития. Выделяют несколько этапов становления данной системы, начиная с первых шагов в СССР и до современных решений:

1. Первый этапы (1960—1975 гг.): в СССР начали разрабатываться аналогичные системы в отраслевых научно-исследовательских институтах, таких как “Киевский операционный день” и “Тульский операционный день”. На рынке появились первые коммерческие разработчики АБС, такие как компания ПрограмБанк (позднее переименованная в Центавр), Инверсия и АСОФТ. Первые версии АБС работали на операционных системах MS-DOS и Netware, используя текстовый интерфейс.
2. Второй этап (1972—1989 гг.): в этот период развития банковской сферы появились новые технологии и стандарты. Программные продукты начали переходить на графический интерфейс и использовать СУБД, такие как Oracle Database, Sybase и Microsoft SQL Server.
3. Третий этап (с 1989 г. по настоящее время): в 1998 году в России был введен новый план счетов бухгалтерского учета для банков и кредитных организаций. Этот момент стал переломным, заставив банки перейти на промышленные АБС. Специализированные решения, такие как DiasoftBANK, RS-Bank и другие, стали широко распространены.

С развитием технологий, автоматизированные системы расчёта банковских услуг и транзакционных платежей становятся всё более мощными и гибкими. Выделяют несколько направлений, которые могут определить будущее АБС:

* Искусственный интеллект и машинное обучение;
* Блокчейн и криптовалюты;
* Интернет вещей (IoT);
* Кибербезопасность;

Автоматизированные системы расчёта банковских услуг и транзакционных платежей продолжают эволюционировать, обеспечивая банкам и клиентам более удобные и безопасные услуги. Надеюсь, что этот курсовой проект поможет вам лучше понять и оценить значимость АБС в современной банковской индустрии.

## 1 АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОГРАММНОГО СРЕДСТВА

Предметная область данного программного средства - это автоматизированная система расчёта банковских услуг и транзакционных платежей. Оно включает в себя обработку и анализ данных о банковских операциях, расчёт комиссий и процентов, а также управление транзакциями.

В рамках этой предметной области основными сущностями являются:

1. Банковский счёт. Это основная единица для проведения транзакций. Каждый счёт имеет уникальный номер и принадлежит определённому клиенту.
2. Транзакция. Это операция перевода денег с одного счёта на другой. Каждая транзакция имеет уникальный номер, дату и время проведения, а также сумму перевода.
3. Услуга. Это дополнительные услуги, предоставляемые банком, за которые взимается плата. Примеры услуг включают обслуживание счёта, предоставление кредитов и страхование.

Для моделирования предметной области можно использовать различные подходы, такие как ER-диаграммы (Entity-Relationship), UML-диаграммы (Unified Modeling Language) или BPMN-диаграммы (Business Process Model and Notation).

В контексте данного проекта, ER-диаграмма может быть наиболее подходящим инструментом для визуализации структуры данных и отношений между сущностями. Например, сущности «Банковский счёт», «Транзакция» и «Услуга» могут быть связаны следующим образом:

* банковский счёт может иметь множество транзакций;
* каждая транзакция связана с двумя банковскими счетами: счётом отправителя и счётом получателя;
* каждый банковский счёт может быть связан с множеством услуг, которые предоставляются клиенту.

Важно отметить, что модель должна быть достаточно гибкой, чтобы учесть возможные изменения в бизнес-процессах или требованиях к программному средству. Поэтому процесс моделирования предметной области должен включать постоянное взаимодействие с заинтересованными сторонами, такими как бизнес-аналитики, разработчики и конечные пользователи.

**1.1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

В современном мире банковские услуги и транзакционные платежи играют ключевую роль в финансовой сфере. С развитием информационных технологий возникла необходимость в создании программных средств с автоматизированными системами, способными обеспечивать эффективное управление банковскими операциями и платежными процессами. В данной главе производится обзор предметной области, рассматриваются основные аспекты банковских услуг и транзакционных платежей.

Банки предоставляют кредиты и дебетовые карты своим клиентам для осуществления покупок, снятия наличных средств, оплаты услуг и т. д. Кредитные продукты включают в себя кредитные линии, потребительские кредиты, ипотечные кредиты и другие виды кредитования.

Банки предоставляют клиентам возможность инвестировать свои средства в различные финансовые инструменты, такие как акции, облигации, фонды и другие. Инвестиционные услуги могут включать в себя управление портфелем, консультации по инвестициям и т. д.

Транзакционные платежи представляют собой передачу денежных средств между различными участниками финансовых операций. Они могут быть осуществлены как внутри одной страны, так и между различными странами.

Платежи между клиентами и банками – это платежи, осуществляемые между клиентами и их банками. Они могут включать в себя оплату счетов, переводы между счетами, покупку товаров и услуг с использованием платежных карт и т. д. Предметная область программного средства с автоматизированной системой расчета банковских услуг и транзакционных платежей охватывает широкий спектр финансовых операций, осуществляемых в банковской сфере. Для успешной разработки такого программного продукта необходимо глубокое понимание всех аспектов банковских услуг и платежных процессов, что и было представлено в данной главе.

Помимо основных аспектов банковских услуг и транзакционных платежей, важно также уделить внимание технологическим и регуляторным аспектам данной области. Технологические инновации играют существенную роль в развитии банковской отрасли. Современные информационные системы, включая базы данных и прикладные программы, обеспечивают не только эффективное управление банковскими операциями, но и гарантируют высокий уровень безопасности при проведении платежных транзакций. Также стоит отметить внедрение технологий блокчейн и цифровых валют, которые предоставляют новые возможности для оптимизации транзакционных процессов и укрепления доверия между участниками финансовых операций.

Кроме того, регуляторные нормативы играют ключевую роль в функционировании банковской системы. Законодательство и нормативы регулируют широкий спектр банковских операций, включая кредитование, платежные системы, а также обеспечивают стабильность финансовой системы в целом. Регулирование банковской сферы включает в себя нормативные акты центральных банков, регуляторных органов и международных организаций, таких как Банк международных расчетов (БМР) и Международный валютный фонд (МВФ).

**1.2 РАЗРАБОТКА ФУНКЦИОНАЛЬНОЙ МОДЕЛИ ПРЕДМЕТНОЙ ОБЛАСТИ**

IDEF0 (сокращение от "Integrated Definition for Function Modeling 0") – это методология, которая используется для моделирования и анализа функций в системах.

Методология IDEF0 использует графическое представление в виде функциональных блоков и стрелок для обозначения потока данных, управления и данных. Все функциональные блоки связаны взаимосвязями, отображающими логические отношения между функциями.

IDEF0 может использоваться в различных областях, таких как разработка программного обеспечения, производство, бизнес–процессы и т. д. Она помогает улучшить понимание функций системы, идентифицировать проблемы и улучшить процессы.

Ниже представлен нулевой уровень модели «Автоматизировать систему расчета банковских услуг и транзакционных платежей» (рисунок 1.1). Здесь на вход поступают документы клиента и сам клиент, а на выходе получаем оказанную услугу или отказ в услуге.

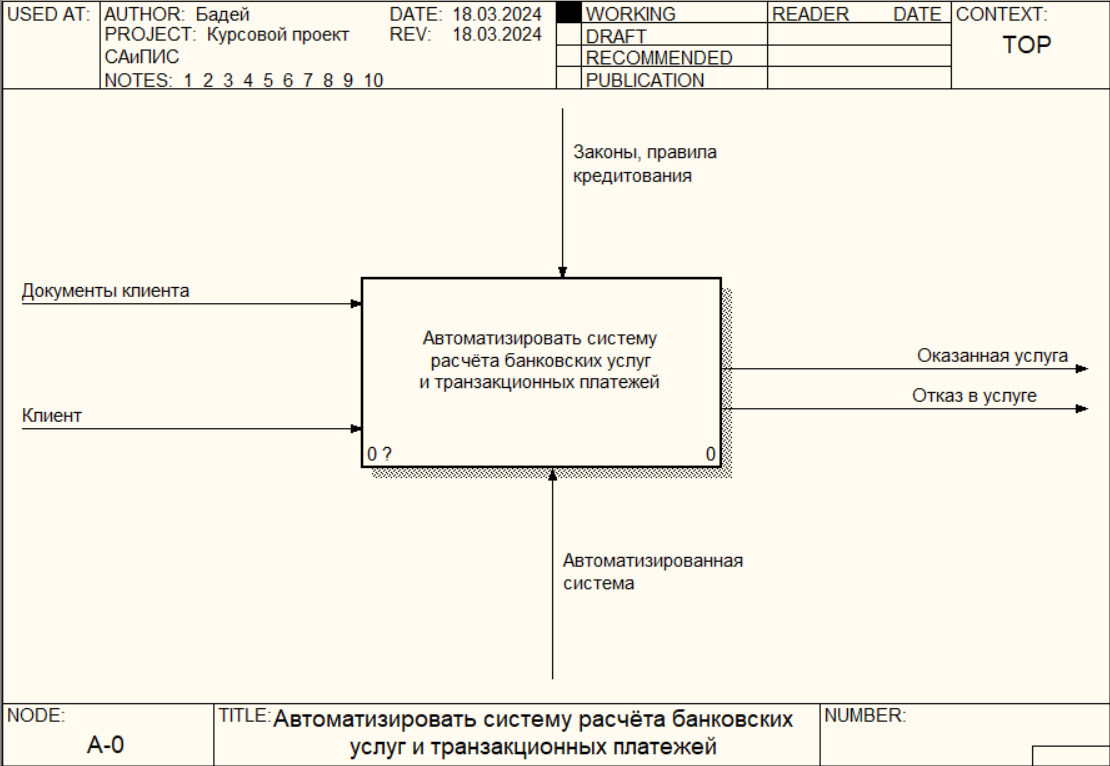


Рисунок 1.1 – Главный блок

В данном случае (рисунок 1.2) на вход поступают все те же документы клиента и сам клиент в блок «Собрать информацию о клиенте». На выходе получаем удовлетворяющую информацию или отказ в оказании услуги по причине ненадежности клиента, которая переходит в следующий блок «Выбрать предоставляемую услугу». Далее получаем выбранную услугу, которая входит в блок «Оказать услугу» и на выходе из этого блока получаем оказанную услугу.

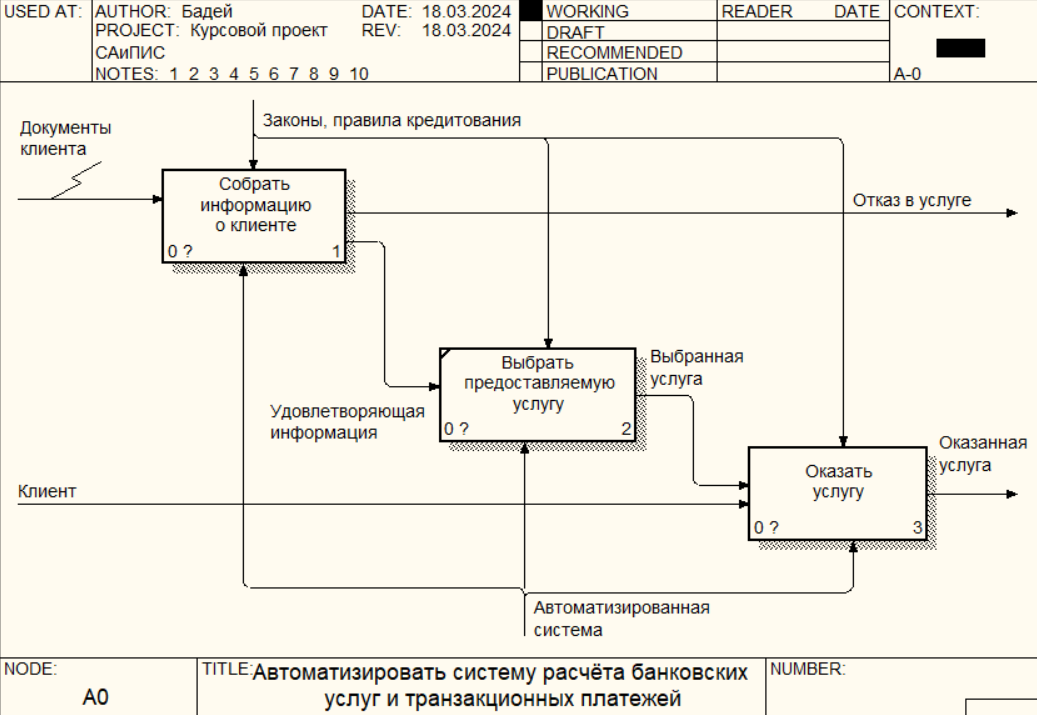


Рисунок 1.2 – Декомпозиция главного блока моделирования

В данном случае на вход поступают все те же документы клиента и сам клиент в блок «Собрать информацию о клиенте». На выходе получаем удовлетворяющую информацию или отказ в оказании услуги по причине ненадежности клиента, которая переходит в следующий блок «Выбрать предоставляемую услугу». Далее получаем выбранную услугу, которая входит в блок «Оказать услугу» и на выходе из этого блока получаем оказанную услугу.

Далее (рисунок 1.3) на вход поступают документы клиента в блок «Обработать документы». На выходе получаем обработанные документы, которые входят в следующий блок – «Проверить надежность клиента в соответствии с оказанием определенных услуг». На выходе получаем надежность клиента, который входит в блок «Занести клиента в соответствующий список». В итоге на выходе из блока получаем отказ в услуге или удовлетворяющую информацию.

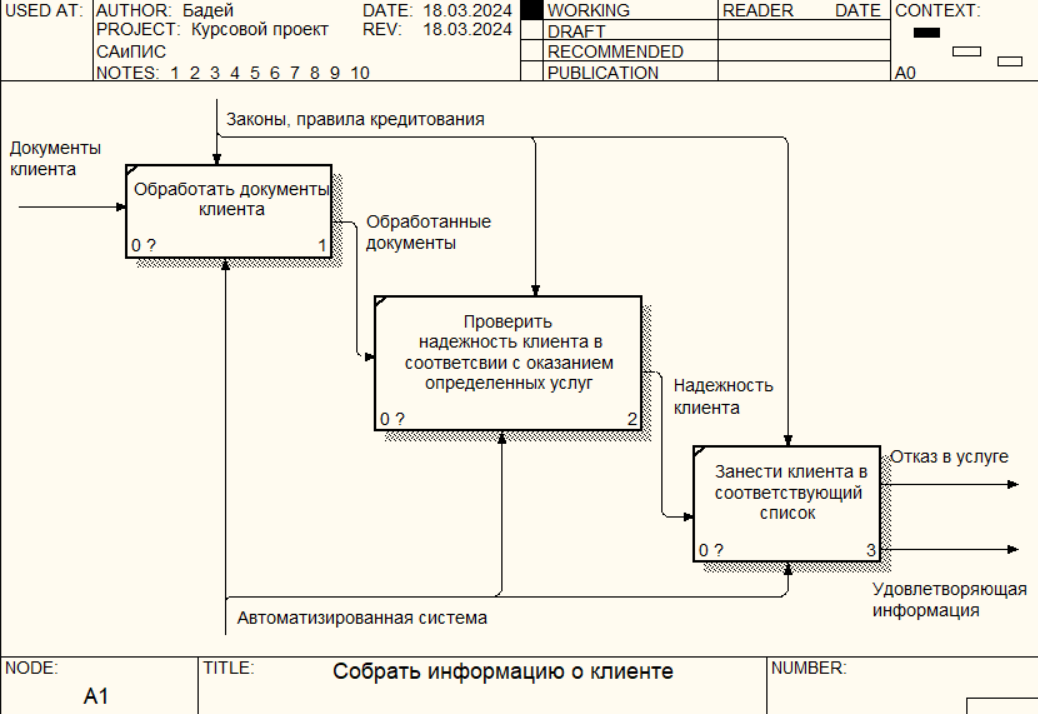


Рисунок 1.3 – Декомпозиция блока «Собрать информацию о клиенте»

Здесь (рисунок 1.4) на вход поступают клиент в блок «Повторно убедиться в надежности клиента» и выбранная услуга в блок «Обратиться в головной офис банка для оказания услуги».

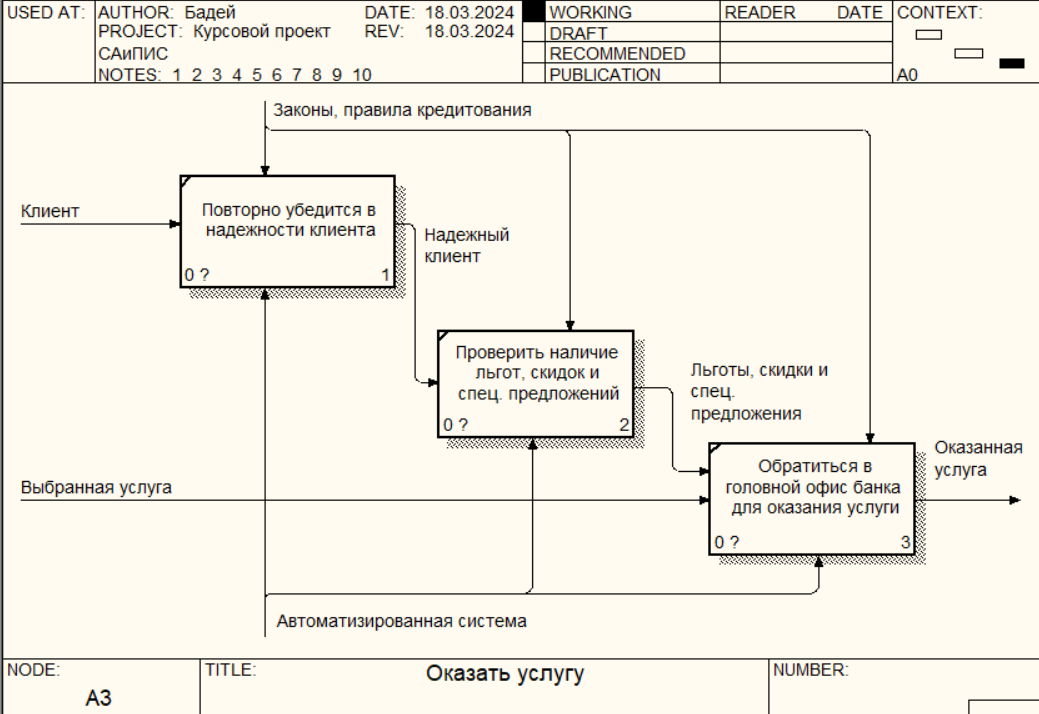


Рисунок 1.4 – Декомпозиция блока «Оказать услугу»

На выходе получаем надежного клиента, который входит в блок «Проверить наличие льгот, скидок и спец. предложений». На выходе из этого блока получаем льготы, скидки и спец. предложения и входим в блок «Обратиться в головной офис банка для оказания услуги». Далее на выходе получаем оказанную услугу.

**1.3 АНАЛИЗ ТРЕБОВАНИЙ К РАЗРАБАТЫВАЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ. СПЕЦИФИКАЦИЯ ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

Программное средство должно быть выполнено как функционирующее клиент-серверное веб-приложение с самостоятельно установленным веб-сервером в архитектуре Model-View-Controller или Model-View-Presenter с использованием технологии Ajax и CommonJS /UMD/ ESmodules модульных концепций. Веб-страницы, объединенные общей темой и дизайном, должны быть созданы с использованием языков HTML, DHTML, JavaScript, XML и содержать гиперссылки для быстрого перехода на другие страницы.

Ко всему веб-приложению для форматирования текста/объектов должна применяться одна каскадная таблица стилей. Данные (логин, пароль), а также ресурсные данные (видео-, звуковые файлы, изображения и т.д.) и иная информация должны храниться в документоориентированной или реляционной базе данных. Конкретные версии фреймворков и технологий, применяемых для реализации программного средства, должны быть актуальными на январь 2023 года.

Серверная часть проекта реализуется на сервере Node.js с использованием npm и веб-фреймворка express. Бизнес-логика приложения должна быть реализована на основе языка JavaScript. Бизнес-логика реализует основные алгоритмы обработки данных. Клиентский интерфейс реализуется с использованием технологий HTML5, CSS3, DHTML, Ajax, актуальных JavaScript библиотек и других методов разработки веб-интерфейса (React/Angular - по согласованию с руководителем). СУБД для создания базы данных (реляционной/нереляционной) выбирается студентом самостоятельно. Схема базы данных должна обеспечивать ссылочную целостность данных.

Пользовательский интерфейс должен обеспечивать доступ ко всем функциям. Система должна реализовать не менее 10 бизнес-операций, не считая простых операций с базой данных (чтение, запись, изменение данных).

Необходимо предусмотреть механизм авторизации всех пользователей. Количество ролей – не менее 2-3 в зависимости от специфики предметной области (администратор, сотрудник компании, гость и т.п.).

**1.4 РАЗРАБОТКА ИНФОРМАЦИОННОЙ МОДЕЛИ ПРЕДМЕТНОЙ ОБЛАСТИ**

В данной главе будет представлена информационная модель предметной области разрабатываемого программного средства с автоматизированной системой расчёта банковских услуг и транзакционных платежей. Эта модель позволяет структурировать и описать основные сущности, их атрибуты и взаимосвязи в рамках системы (рисунок 1.5).

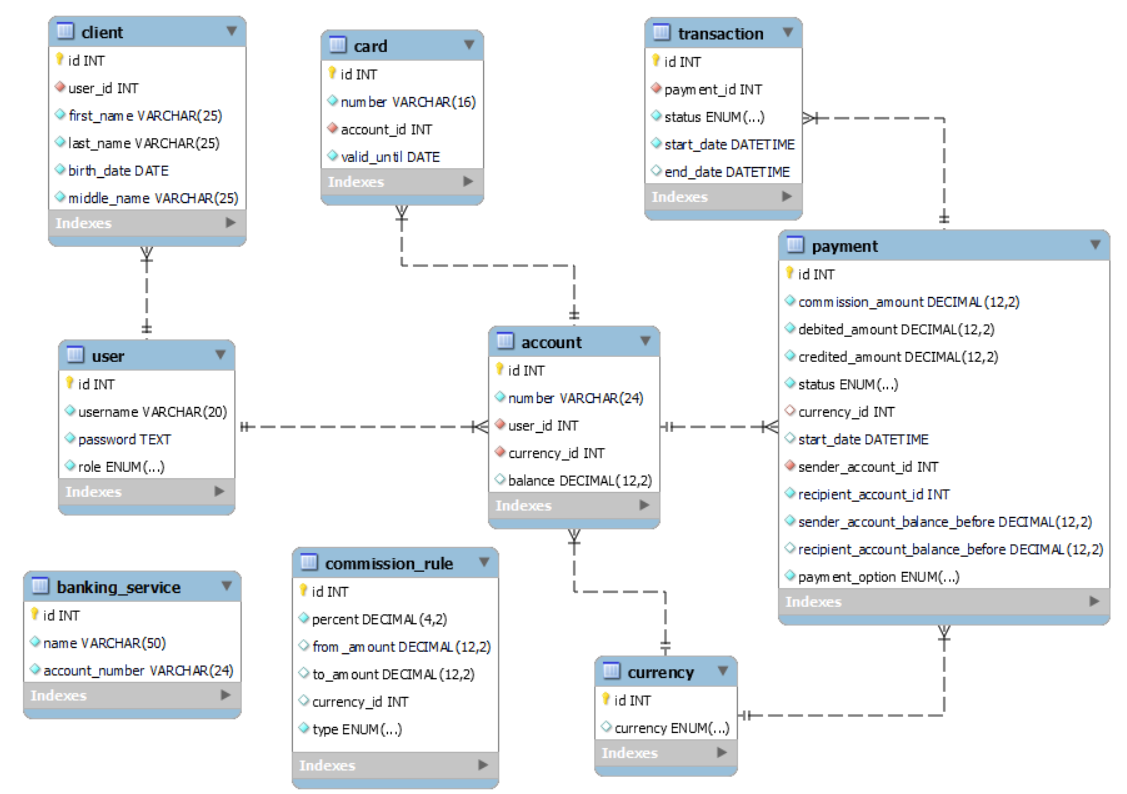


Рисунок 1.5 – Информационная модель предметной области

Таблица client содержит информацию о клиенте, такую как: идентификатор, идентификатор пользователя, имя, фамилию, отчество и дату рождения.

Таблица card содержит информацию о карте: идентификатор, номер карты, идентификатор счета, к которому привязана карта, и дата валидности карты.

Таблица transaction содержит информацию о транзакции: идентификатор, идентификатор оплаты, статус, начало транзакции и конец транзакции.

Таблица payment содержит информацию об оплатах: идентификатор, сумма комиссии, списываемая сумма, начисляемая сумма, статус, идентификатор валюты, начало оплаты, идентификатор отправителя, идентификатор получателя, баланс отправителя до, баланс получателя до и вид оплаты.

Таблица account содержит информацию о счете: идентификатор, номер счета, идентификатор пользователя, идентификатор валюты и баланс.

Таблица currency содержит информацию о валюте: идентификатор и название валюты.

Таблица commission\_rule содержит информацию о правилах начисления комиссии: идентификатор, процент, с какого значения, по какое значение, идентификатор валюты и вид оплаты.

Таблица banking\_service содержит информацию и банкинге (услуге): идентификатор, название услуги, номер счета для услуги.

Таблица user содержит информацию о пользователе: идентификатор, имя пользователя, пароль и роль.

**1.5 UML-МОДЕЛИ ПРЕДСТАВЛЕНИЯ ПРОГРАММНОГО СРЕДСТВА И ИХ ОПИСАНИЕ**

Диаграмма вариантов использования (Use Case) используется для описания поведения системы с точки зрения ее пользователей или внешних акторов. Она помогает идентифицировать и описать различные сценарии использования системы, ее возможности и функциональность.

Главная цель диаграммы вариантов использования заключается в понимании бизнес – требований к системе со стороны конечных пользователей.

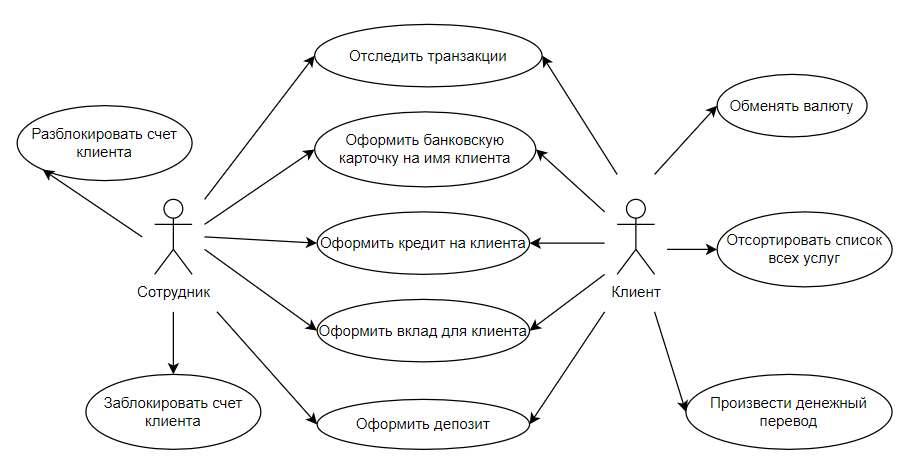


Рисунок 1.6 – Диаграмма вариантов использования

На рисунке 1.6 представлены 10 основных бизнес-функций программы, которые она должна выполнять. Так же на рисунке присутствуют два актера (роли): сотрудник и пользователь.

**2 Проектирование и конструирование программного средства**

## **2.1 Постановка задачи**

Основная задача курсового проекта – спроектировать и разработать программное средство с автоматизированной системой расчёта банковских услуг и транзакционных платежей.

Программное средство должно обладать следующими основными функциями, предоставляемые клиенту:

* регистрация;
* авторизация;
* возможность производить различные транзакционные переводы;
* возможность обмена валют;
* возможность просмотра истории платежей и переводов;
* возможность открытия счета.

Функции, доступные сотруднику:

* авторизация;
* открытие счета клиенту;
* оформление карты клиенту;
* пополнение счета клиента;
* просмотр истории платежей и переводов клиента.

## **2.2 Обоснование выбора компонентов и технологий для реализации программного средства**

Для реализации программного средства с автоматизированной системой расчёта банковских услуг и транзакционных платежей необходимо выбрать компоненты и технологии, которые обеспечат эффективную и надежную работу системы. Ниже приведены некоторые компоненты и технологии, которые могут быть использованы при разработке данного программного средства:

1. Язык программирования: для разработки программного средства можно использовать различные языки программирования, такие как Java, Python, C++, C# и др. Выбор языка программирования зависит от требований к системе, опыта команды разработчиков и других факторов. В нашем случае это express js, т.к. в требованиях необходимо использовать язык java script.

2. База данных: для хранения и управления данными системы можно использовать различные базы данных, такие как MySQL, PostgreSQL, Oracle или MongoDB. В нашем случае была выбрана база данных MySQL, т.к. работа с ней весьма легка и лаконична. Для работы с самой базой данных была использована графическая оболочка Heidi для лучшей доступности к данным, хранящимся в базе и лучшей их визуализации.

4. Среда разработки: для разработки программного средства можно использовать различные интегрированные среды разработки (IDE), такие как Eclipse, IntelliJ IDEA, Visual Studio и др. Среда разработки облегчает процесс разработки и улучшает качество кода. В нашем случае был выбран один из самых лучших и легковесных редакторов Visual Studio Code.

5. Система контроля версий: для управления исходным кодом программного средства можно использовать различные системы контроля версий, такие как Git, SVN и др. Система контроля версий позволяет отслеживать изменения в коде, управлять версиями и восстанавливать предыдущие версии.

7. Библиотеки: для хэширования паролей была использована библиотека crypto, для преобразования строки в строку запроса БД была использована библиотека querystring.

## **2.3. Архитектурные решения**

Архитектурное решение MVC (Model-View-Controller) является распространенным подходом к проектированию и разработке веб-приложений. Оно позволяет разделить приложение на три основных компонента: модель, представление и контроллер.

* Модели: счет, банкинг, карта, клиент, правила\_комиссии, валюта, плата, транзакция, пользователь;
* База данных: MySQL;
* Контроллер: Node.js;
* Представление: HTML, CSS.

Модель выполняет роль абстракции, инкапсулирующей данные о необходимом разработчику предмете. Можно провести параллель с классами в программировании. Благодаря моделям между View и Controller налажена связь.

Контроллер - это компонент, который управляет потоком данных между моделями и представлением. Он получает запросы от пользователей, обрабатывает их и вызывает соответствующие методы моделей для выполнения необходимых операций.

Представление - это компонент, который отображает данные пользователю. В нашем случае, представление будет реализовано с помощью и HTML.

Приложение на Node.js с использованием MySQL в качестве базы данных позволит создать эффективную и масштабируемую систему, которая сможет обрабатывать большой объем данных.

## **2.4 Описание алгоритмов, реализующих ключевую бизнес-логику разрабатываемого программного средства**

Для описания алгоритмов, реализующих бизнес-логику выбран алгоритм перевода средств пользователем.

Схема представлена на рисунке 2.4.1.

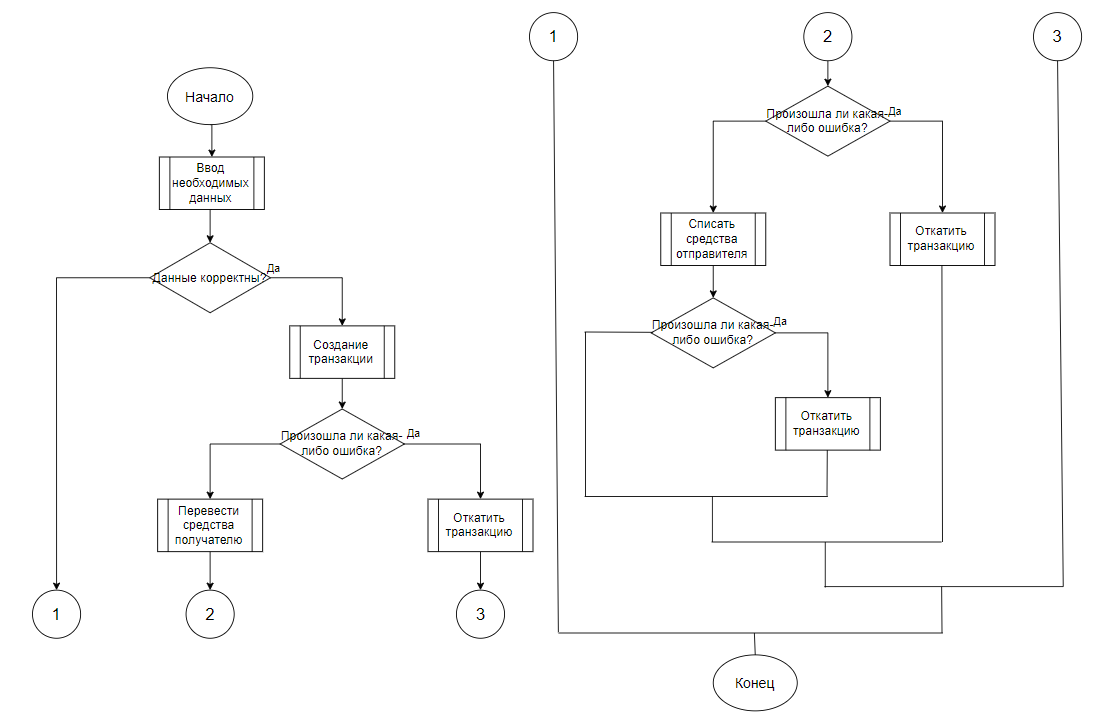


Рисунок 2.4.1 – Алгоритм перевода средств по номеру счета

Из процесса основного алгоритма, реализующего ключевую бизнес-логику приложения, видно, как подробно мы стараемся обработать малейшую ошибку при ходе выполнения кода, чтобы предотвратить утрату денежных средств наших клиентов.

Таким образом мы обеспечиваем надежность и безопасность выполнения любых механизмов обработки и отправки денежных средств, будь то обмен валют, перевод клиенту банка или по номеру счета. Все это обеспечивается надежностью транзакций.

##### **2.5 Проектирование пользовательского интерфейса**

Пользовательский интерфейс (UI) - это важный аспект любого программного продукта, в том числе и веб-приложения. Он обеспечивает взаимодействие пользователя с системой и позволяет ему выполнять необходимые действия.

Для разработки пользовательского интерфейса цвета фона и его тоны были взаимствованы у сайта GitHub, оставшиеся основные цвета были выбраны по собственному желанию главного разработчика, дизайнера и проектировщика, далее разработчика.

Структура расположения элементов была относительно полностью придумана разработчиком, а именно: заголовок страницы с логотипом вымышленного банка, правая форма (с расположением активных действий пользователя, над которыми располагаются ссылка на домашнюю страницу и логин пользователя). Отображение доступных карт пользователя было взаимствовано у существующего банка «Alfa-Bank».

Пользовательский интерфейс должен быть понятен, ясен, привлекателен и прост. Все эти атрибуты присущи реализованному приложению в полной мере, так как не имея в своем арсенале реализацию вышеперечисленных атрибутов, пользователь будет с нежеланием пользоваться интерфейсом и скорее всего предпочтет более удобный интерфейс не нашей разработки, что приведет к потере потенциального пользователя, а может и инвестора.

Как итог, можно подметить что, разработка пользовательского интерфейса является важной частью процесса разработки любого веб-приложения и требует внимательного проектирования и реализации. Описывая процесс разработки пользовательского интерфейса в курсовой работе, мы продемонстрировали понимание важности этого аспекта при создании программного продукта.

###### **2.6. Методы и средства, используемые для обеспечения безопасности данных**

В разрабатываемом нами приложении безопасность является важным аспектом, т.к. мы храним не только учетные данные пользователей, но еще и их денежные средства.

Хоть безопасность и не является основной темой курсовой работы, но безопасность играет ключевую роль при обработке исключительных ситуаций и ошибок при обработке значений переменных при осуществлении транзакций.

Для надежности хранения паролей от учетных записей наших пользователей мы их хэшируем, используя встроенную библиотеку crypto и храним в базе данных, а для авторизации пользователя, заносим его в сессию. В отличие от других работ, обеспечение безопасности данных достигает своего апогея, когда мы начинаем речь не об использованных библиотек, но о транзакциях, создающихся в ходе переводов с одного счета на другой. Так мы перестраховываемся от случаев, когда клиент может безвозвратно утратить свои средства.

Для осуществления такой надежности у нас имеются две таблицы: payments и transaction. Когда клиент вводит необходимые реквизиты и подтверждает их корректность, создается запись в таблицу payments, которая обладает следующими полями: id, размер комиссии, списываемая сумма, начисляемая сумма, статус('started','successful','failed'),id валюты, начало перевода, id счета отправителя, id счета получателя, баланс отправителя до, баланс получателя до, тип перевода ('banking', 'bank\_client', 'account\_number', 'change\_currency'). После создания данной записи, создается запись в таблицу transaction, хранящую поля id, payment\_id, статус('started','successful','failed'), начало транзакции, конец транзакции.

Далее по ходу выполнения кода, если возникнет какая-либо ошибка, то балансы пользователей исправляются на те, которые занесены в таблицу payment в полях ‘баланс отправителя до’ и ‘баланс получателя до’, в полях статус устанавливается значение ‘failed’, функция завершается.

Таким образом осуществляется обеспечение безопасности данных, являющееся необходимым в рамках данного курсового проекта.

# **3 Тестирование и проверка работоспособности программного средства**

При разработке программного кода необходимо учесть возможные исключительные ситуации и предусмотреть их обработку. Ниже приведены некоторые примеры таких ситуаций.

При неверных данных при входе появляется соответствующая надпись (рисунок 3.1).

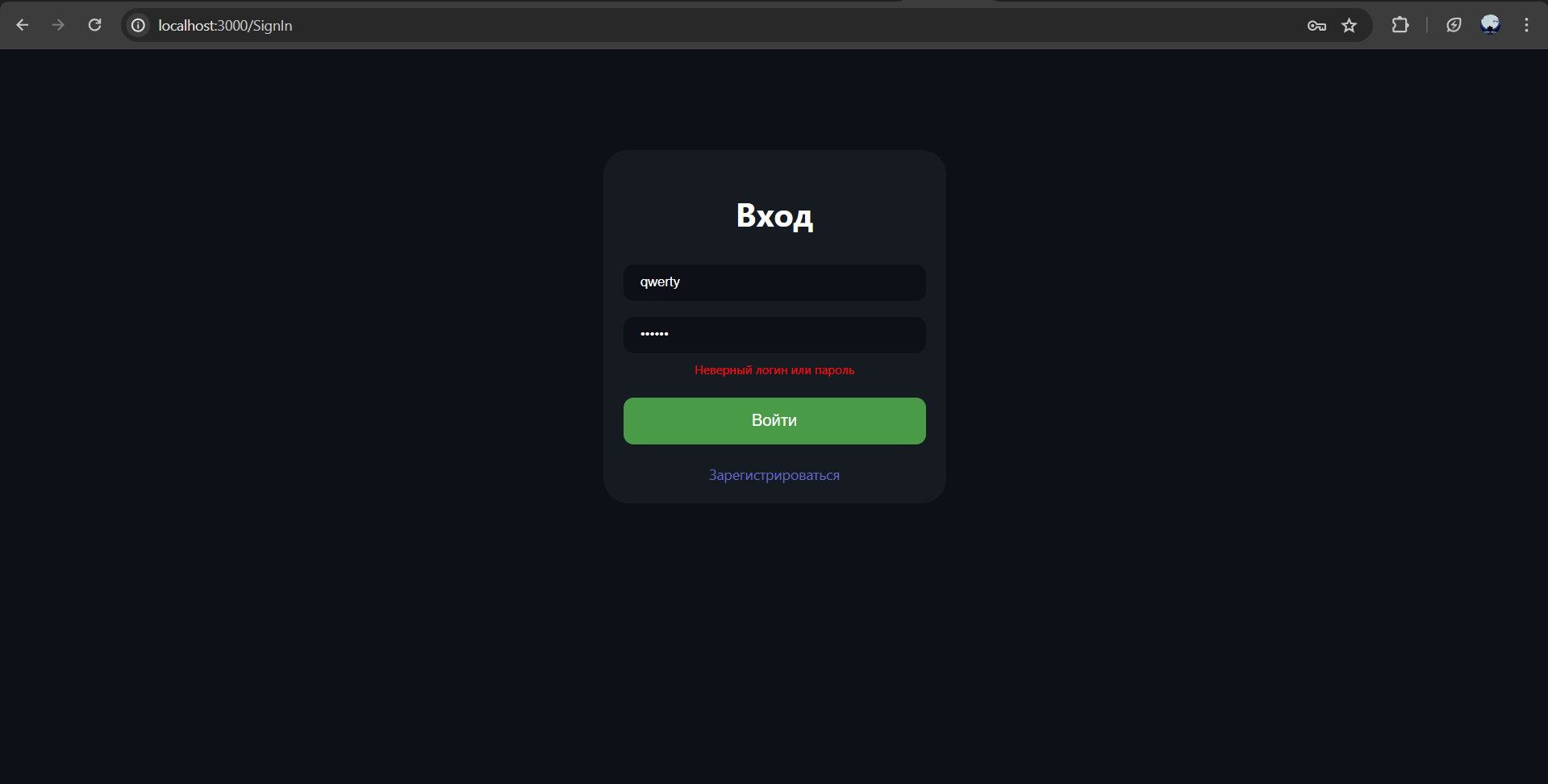


Рисунок 3.1 – Сообщение о неверно введенных данных

При регистрации незаполненные поля подсвечиваются (рисунок 3.2).

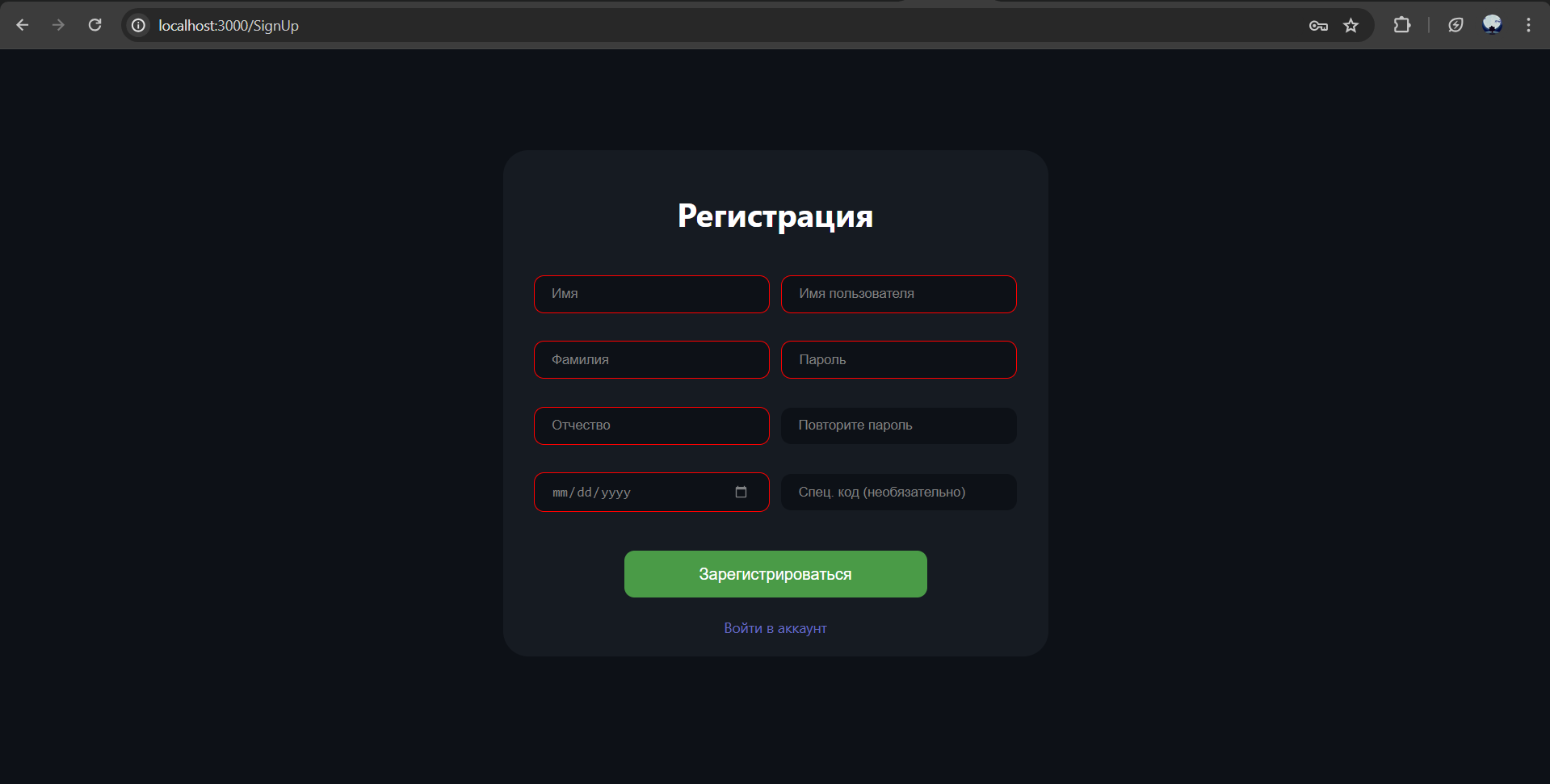


Рисунок 3.2 – Подсвечивание незаполненных полей

При совершении операций клиентом, также будут возникать предупреждения (рисунок 3.3, рисунок 3.4).

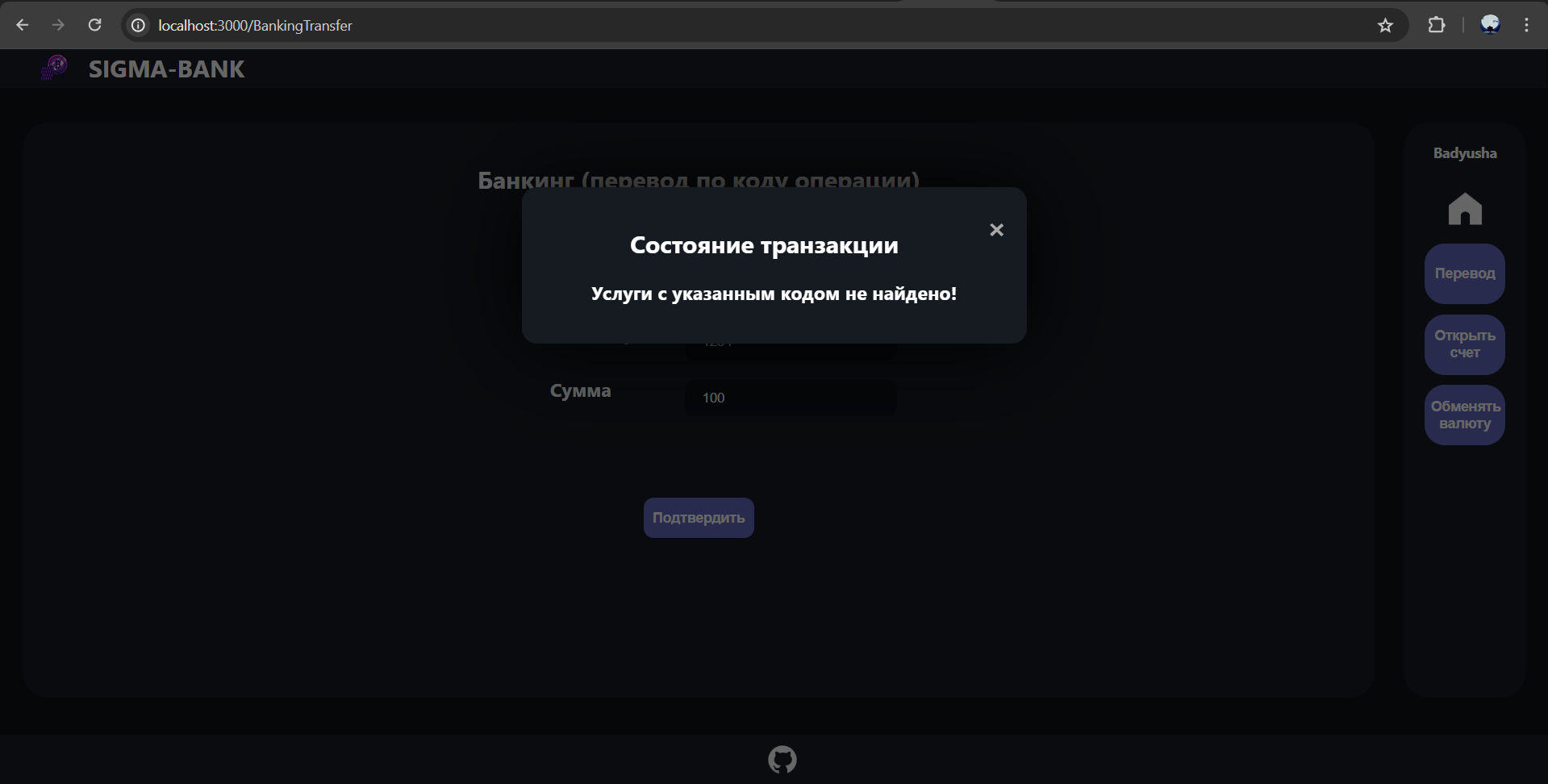


Рисунок 3.3 - Модальное окно об ошибке ввода кода услуги

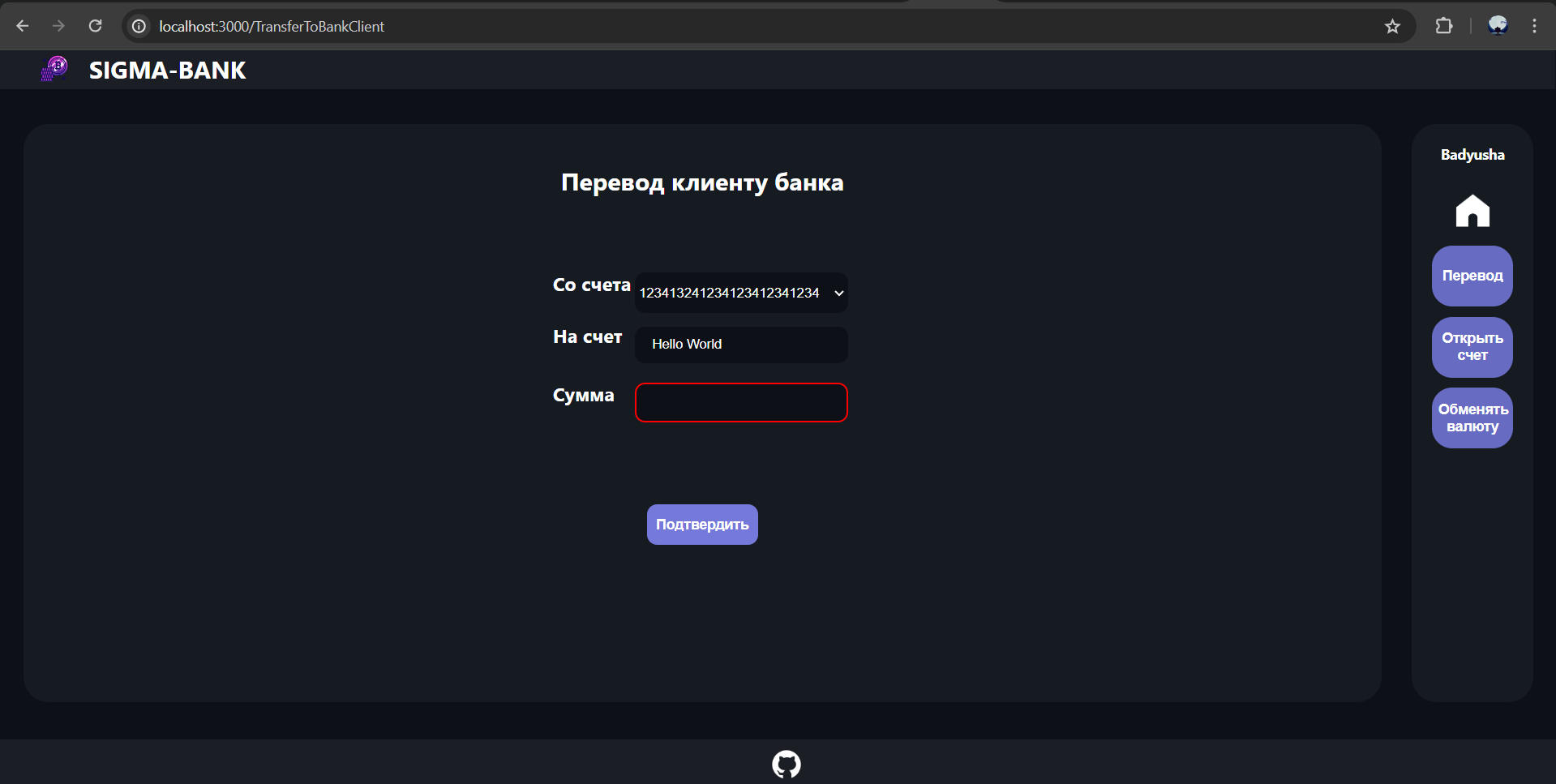


Рисунок 3.4 – Выделение незаполненного поля

Это лишь малая часть от всего количества похожих ситуаций, когда пользователь ввел некорректные данные.

Тестирование и проверка работоспособности программного средства – существенно важный этап в процессе разработки и проектирования данного программного средства.

Тестирование призвано свести на нет непредвиденные ошибки и недочеты, способные вызвать нарушение стандартного выполнения работы программного средства. Это дает нам уверенность, что пользование нашим приложением не прервется неожиданной, необработанной технической ошибкой.

**4 Руководство по развертыванию и использованию программного средства**

## 4.1 Руководство по развертыванию программного средства

Для развертывания программного средства необходимо иметь следующие компоненты:

* Node.js;
* СУБД MySQL.

В файле сервер (рисунок 4.1.1) необходимо поменять поля host, user, password, database, если необходимо.

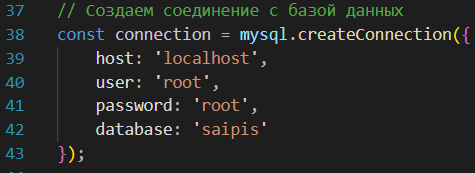


Рисунок 4.1.1 – Создание соединения с базой данных

Далее нужно создать саму базу данных под именем, указанном выше в поле database.

**4.2 Руководство пользователя**

Перед тем, как ощутить весь пользовательский функционал, необходимо зарегистрироваться (рисунок 4.2.1).

Специальный код необходимо вводить, если необходимо зарегистрировать сотрудника, поэтому данное поле является необязательным для заполнения.

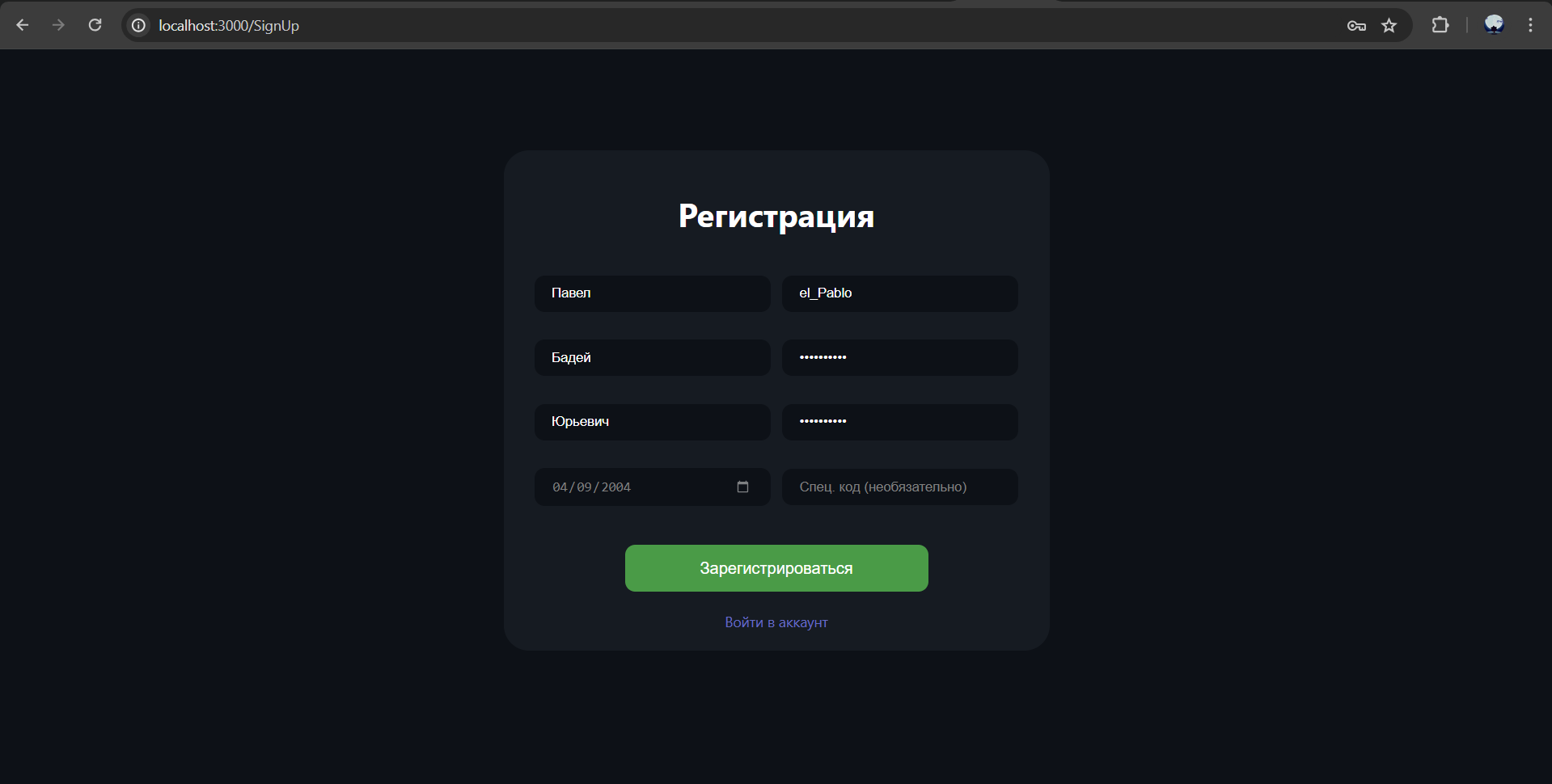


Рисунок 4.2.1 – Форма регистрации

Так как мы только зарегистрировались, то у нас нет, ни счетов, ни карт. Проделаем манипуляции и у перед нами появится следующая картина (рисунок 4.2.2).

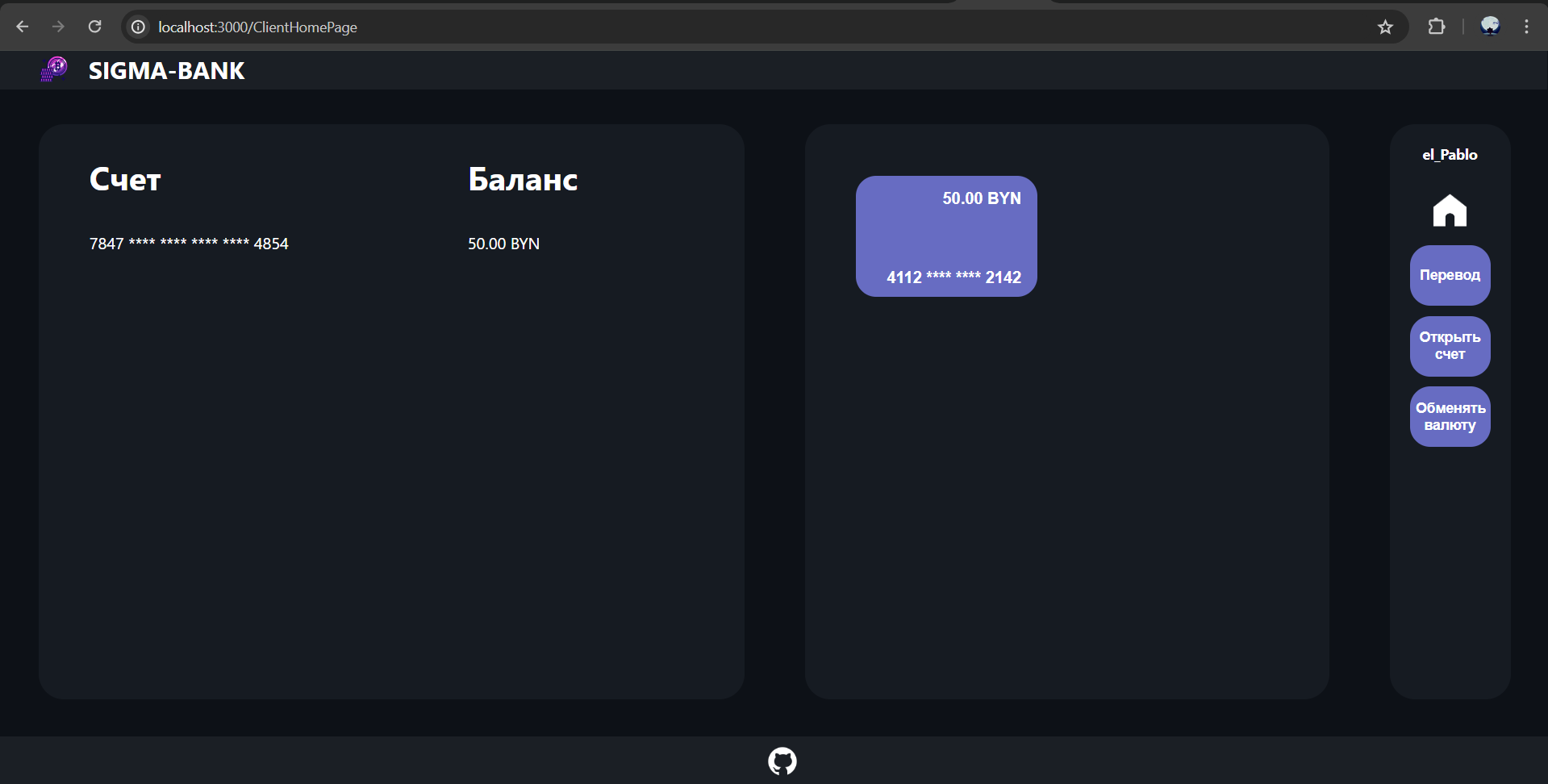


Рисунок 4.2.2 – Домашняя страница клиента

Далее клиент может совершать перевод средств, открывать счета и обменивать валюту (правая форма). Нажав на карту (центральная форма) клиент может просматривать все транзакции, совершенные по этой карте.

**ЗАКЛЮЧЕНИЕ**

Было разработано веб-приложение, предназначенное для организации работы школы иностранных языков. Из использованных средств можно выделить Node.js и Express на стороне бэкенда, а также HTML, CSS и JavaScript на стороне фронтенда.

Создание подобного приложения может помочь понять структуру организации потока данных и их обработку в сфере банковских услуг. Это помогает понять какие функции можно реализовать и какие для этого нужны технологии, понять что для этого нужно и как этого добиться.

Данная тема крайне актуальна в современном мире, т.к. мы живем во время цифровых технологий и деньги также становятся цифровыми. Все наибольшее распространение получает криптовалюта, ядром которой является технология блокчейна. В данной курсовой работе можно было бы реализовать некоторые взаимодействия с криптовалютой, но тема курсового проекта была отлична от тематики криптовалюты.

При разработке данного средства были уяснены основные принципы работы банковских услуг и механизм транзакций при совершении переводов и других банковских услуг.

Программное средство представляет собой комплексное решение, которое позволяет банкам и финансовым учреждениям существенно упростить и автоматизировать процессы обслуживания клиентов, расчёта банковских операций и проведения транзакционных платежей. Благодаря использованию современных технологий и методов разработки, система обладает высокой надёжностью, безопасностью и эффективностью работы.

Как итог, разработанное программное средство представляет собой эффективный инструмент для автоматизации и оптимизации банковских процессов, что способствует повышению конкурентоспособности банка и улучшению качества обслуживания клиентов.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Node.js. Документация. [Электронный ресурс]. – Режим доступа: https://nodejs.org/en/docs/.

[2] Express.js. Документация. [Электронный ресурс]. – Режим доступа: https://expressjs.com/ru/guide/routing.html.

[3] Habr [Электронный ресурс]. – Режим доступа: <http://habr.com/ru/post/508710/>

[4] Дубинина С.К. Методическое пособие по курсу "Финансовые рынки и институты": учебно–методическое пособие: Экономический факультет МГУ имени М. В. Ломоносова, 2023 – 32 с.

[5] Esphere [Электронный ресурс]. – Режим доступа: <https://www.esphere.ru/blog/kak-avtomatizirovat-magazin-roznichnoj-torgovli>

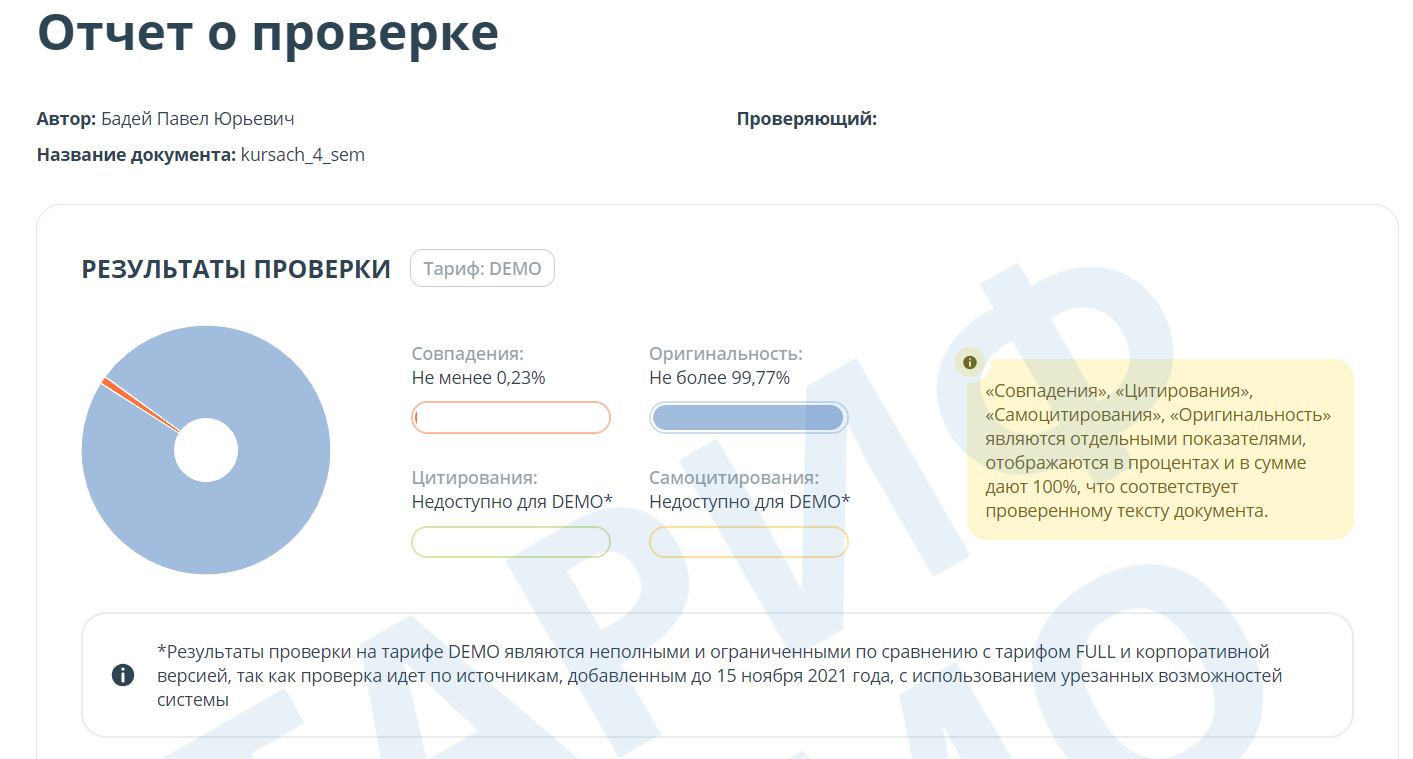
[6] Цуканова О.А. Методология и инструментарий моделирования бизнес–процессов: учебное пособие–СПб.: Университет ИТМО, 2015. – 100с.

[7] Браун И., Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript / И Браун. - СПб : Питер, 2017. - 336 с. - ISBN 978-5-496-02156-2.

# **Приложение А**

**(обязательное)**

**Отчет о заимствовании в системе «Антиплагиат»**



# **Приложение Б**

**(обязательное)**

**Листинг кода алгоритмов, реализующих бизнес-логику**

// функция перевода средств клиенту банка

app.post('/TransferToBankClient', async (req, res) => {

    try {

        let commission = parseFloat(req.body.commission);

        let transfer\_amount = parseFloat(req.body.transfer\_amount);

        let sender\_account\_number = req.body.sender\_account\_number;

        let currency\_id = await getCurrencyIdByCurrencyName(req.body.currency);

        if (currency\_id === null) {

            res.send("Ошибка при обращении к базе данных!");

            return;

        }

        let sender\_account\_id = await getUserAccountIdByAccountNumber(sender\_account\_number);

if (sender\_account\_id === null) {

            res.send("Ошибка при обращении к базе данных!");

            return;

        }

        let fullname = req.body.reciever\_name.split(' ');

        let last\_name = fullname[0];

        let first\_name = fullname[1];

        let middle\_name = fullname[2];

        if(await clientExistsByFullName(last\_name, first\_name, middle\_name) === false) {

            res.send("Пользователь с таким именем не найден!");

            return;

        };

        let recipient\_account\_id = await getUserAccountIdByFullNameCurrencyId(last\_name, first\_name, middle\_name, currency\_id);

        // check if recipient has an account with sender's currency

        if (recipient\_account\_id === null) {

            res.send("У получателя нет счета для отправляемой валюты!");

            return;

        }

        let sender\_account\_balance = parseFloat(await getUserBalanceByUserAccountIdCurrencyId(sender\_account\_id));

        if (sender\_account\_balance === null) {

            res.send("Ошибка при обращении к базе данных!");

            return;

        }

        // check if there are insufficient funds in the sender's account

        if (sender\_account\_balance < (transfer\_amount + commission)) {

            res.send('На вашем балансе недостаточно средств');

            return;

        }

        let recipient\_account\_balance = parseFloat(await getUserBalanceByAccountIdCurrencyId(recipient\_account\_id, currency\_id));

        if (recipient\_account\_balance === null) {

            res.send("Ошибка при обращении к базе данных!");

            return;

        }

        let payment\_id = await createPayment(commission, transfer\_amount + commission, transfer\_amount,

                                            'started', currency\_id, sender\_account\_id, recipient\_account\_id,

                                            sender\_account\_balance, recipient\_account\_balance, 'bank\_client');

        if (payment\_id === null) {

            res.send("Ошибка при обращении к базе данных!");

            await setPaymentStatus(payment\_id, 'failed');

            return;

        }

        let transaction\_id = await createTransaction(payment\_id, 'started');

        if (transaction\_id === null) {

            res.send("Ошибка при обращении к базе данных!");

            await setPaymentStatus(payment\_id, 'failed');

            await setTransactionStatus(transaction\_id, 'failed');

            return;

        }

        if(await transferMoney(transfer\_amount, commission, recipient\_account\_id, sender\_account\_id, recipient\_account\_balance, sender\_account\_balance) !== null) {

            await setPaymentStatus(payment\_id, 'successful');

            await setTransactionStatus(transaction\_id, 'successful');

            res.send("Средства успешно отправлены!");

            return;

        }

        let sender\_account\_balance\_before = parseFloat(await getSenderAccountBalanceBeforeFailByPaymentId(payment\_id));

        let recipient\_account\_balance\_before = parseFloat(await getRecipientAccountBalanceBeforeFailByAccountId(payment\_id));

        await setClientAccountBalanceByAccountId(sender\_account\_balance\_before, sender\_account\_id);

        await setClientAccountBalanceByAccountId(recipient\_account\_balance\_before, recipient\_account\_id);

        await setPaymentStatus(payment\_id, 'failed');

        await setTransactionStatus(transaction\_id, 'failed');

        res.send("Не получилось перевести средства");

    } catch (error) {

        let sender\_account\_balance\_before = parseFloat(await getSenderAccountBalanceBeforeFailByPaymentId(payment\_id));

        let recipient\_account\_balance\_before = parseFloat(await getRecipientAccountBalanceBeforeFailByAccountId(payment\_id));

        await setClientAccountBalanceByAccountId(sender\_account\_balance\_before, sender\_account\_id);

        await setClientAccountBalanceByAccountId(recipient\_account\_balance\_before, recipient\_account\_id);

        await setPaymentStatus(payment\_id, 'failed');

        await setTransactionStatus(transaction\_id, 'failed');

        console.error('Ошибка:', error);

        res.send("Произошла ошибка при выполнении операции!");

    }

});

# **Приложение В**

**(обязательное)**

**Листинг скрипта генерации базы данных**

-- -----------------------------------------------------

-- Schema saipis

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `saipis` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci ;

USE `saipis` ;

-- -----------------------------------------------------

-- Table `saipis`.`user`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `saipis`.`user` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`username` VARCHAR(20) NOT NULL,

`password` TEXT NOT NULL,

`role` ENUM('employee', 'client') NOT NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `username` (`username` ASC) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 101

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `saipis`.`currency`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `saipis`.`currency` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`currency` ENUM('BYN', 'USD', 'RUB') NULL DEFAULT NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB

AUTO\_INCREMENT = 4

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `saipis`.`account`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `saipis`.`account` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`number` VARCHAR(24) NOT NULL,

`user\_id` INT UNSIGNED NOT NULL,

`currency\_id` INT UNSIGNED NOT NULL,

`balance` DECIMAL(12,2) UNSIGNED NULL DEFAULT NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `number` (`number` ASC) VISIBLE,

INDEX `user\_id` (`user\_id` ASC) VISIBLE,

INDEX `currency\_id` (`currency\_id` ASC) VISIBLE,

CONSTRAINT `account\_ibfk\_1`

FOREIGN KEY (`user\_id`)

REFERENCES `saipis`.`user` (`id`)

ON DELETE RESTRICT

ON UPDATE RESTRICT,

CONSTRAINT `account\_ibfk\_2`

FOREIGN KEY (`currency\_id`)

REFERENCES `saipis`.`currency` (`id`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB

AUTO\_INCREMENT = 17

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `saipis`.`banking\_service`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `saipis`.`banking\_service` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(50) NOT NULL,

`account\_number` VARCHAR(24) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4\_0900\_ai\_ci' NOT NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `account\_number` (`account\_number` ASC) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 790

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `saipis`.`card`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `saipis`.`card` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`number` VARCHAR(16) NOT NULL,

`account\_id` INT UNSIGNED NOT NULL,

`valid\_until` DATE NOT NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `number` (`number` ASC) VISIBLE,

INDEX `account\_id` (`account\_id` ASC) VISIBLE,

CONSTRAINT `card\_ibfk\_1`

FOREIGN KEY (`account\_id`)

REFERENCES `saipis`.`account` (`id`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB

AUTO\_INCREMENT = 6

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `saipis`.`client`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `saipis`.`client` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`user\_id` INT UNSIGNED NOT NULL,

`first\_name` VARCHAR(25) NOT NULL,

`last\_name` VARCHAR(25) NOT NULL,

`birth\_date` DATE NOT NULL,

`middle\_name` VARCHAR(25) CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4\_0900\_ai\_ci' NOT NULL,

PRIMARY KEY (`id`),

INDEX `user\_id` (`user\_id` ASC) VISIBLE,

CONSTRAINT `client\_ibfk\_1`

FOREIGN KEY (`user\_id`)

REFERENCES `saipis`.`user` (`id`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB

AUTO\_INCREMENT = 27

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `saipis`.`commission\_rule`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `saipis`.`commission\_rule` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`percent` DECIMAL(4,2) NOT NULL,

`from\_amount` DECIMAL(12,2) NULL DEFAULT NULL,

`to\_amount` DECIMAL(12,2) NULL DEFAULT NULL,

`currency\_id` INT NULL DEFAULT NULL,

`type` ENUM('bank\_client\_transfer', 'account\_number\_transfer', 'banking', 'change\_currency') CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4\_0900\_ai\_ci' NOT NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `percent` USING BTREE (`percent`, `from\_amount`, `to\_amount`, `currency\_id`) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 35

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `saipis`.`payment`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `saipis`.`payment` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`commission\_amount` DECIMAL(12,2) NOT NULL,

`debited\_amount` DECIMAL(12,2) NOT NULL,

`credited\_amount` DECIMAL(12,2) NOT NULL,

`status` ENUM('started', 'successful', 'failed') NOT NULL,

`currency\_id` INT UNSIGNED NULL DEFAULT NULL,

`start\_date` DATETIME NULL DEFAULT NULL,

`sender\_account\_id` INT UNSIGNED NOT NULL,

`recipient\_account\_id` INT UNSIGNED NOT NULL,

`sender\_account\_balance\_before` DECIMAL(12,2) NOT NULL,

`recipient\_account\_balance\_before` DECIMAL(12,2) NULL DEFAULT NULL,

`payment\_option` ENUM('banking', 'bank\_client', 'account\_number', 'change\_currency') CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4\_0900\_ai\_ci' NOT NULL,

PRIMARY KEY (`id`),

INDEX `sender\_account\_id` (`sender\_account\_id` ASC) VISIBLE,

INDEX `recipient\_account\_id` (`recipient\_account\_id` ASC) VISIBLE,

INDEX `payment\_ibfk\_1` (`currency\_id` ASC) VISIBLE,

CONSTRAINT `payment\_ibfk\_1`

FOREIGN KEY (`currency\_id`)

REFERENCES `saipis`.`currency` (`id`),

CONSTRAINT `payment\_ibfk\_3`

FOREIGN KEY (`sender\_account\_id`)

REFERENCES `saipis`.`account` (`id`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB

AUTO\_INCREMENT = 65

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `saipis`.`transaction`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `saipis`.`transaction` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`payment\_id` INT UNSIGNED NOT NULL,

`status` ENUM('started', 'successful', 'failed') NOT NULL,

`start\_date` DATETIME NOT NULL,

`end\_date` DATETIME NULL DEFAULT NULL,

PRIMARY KEY (`id`),

INDEX `payment\_id` (`payment\_id` ASC) VISIBLE,

CONSTRAINT `transaction\_ibfk\_1`

FOREIGN KEY (`payment\_id`)

REFERENCES `saipis`.`payment` (`id`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB

AUTO\_INCREMENT = 58

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;