Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Проектирование и разработка информационных систем (ПиРИС)

ОТЧЕТ

По практической работе №3

Тема работы: Этапы разработки ИС. Стадия «Реализация»

Выполнили

студент: гр. 051006 Крыжановский А.В.

Бобрик П.В.

Проверил: Грибович А.А.

Минск 2024

Содержание

[1 Листинг 3](#_Toc159347809)

[1.1 Закрытие банковского дня 3](#_Toc159347810)

[1.2 Создание нового клиента 3](#_Toc159347811)

[1.3 Изменение информации о клиенте 3](#_Toc159347812)

[1.4 Заключение депозитного контракта 4](#_Toc159347813)

[1.5 Заключение кредитного контракта 5](#_Toc159347814)

[1.6 Создание новой кредитной карты 7](#_Toc159347815)

[2 Интерфейс пользователя 8](#_Toc159347816)

[3 Руководство системного программиста 9](#_Toc159347817)

[3.1 Общие сведения о программе 9](#_Toc159347818)

[3.2 Структура программы 9](#_Toc159347819)

[3.3 Настройка программы: 13](#_Toc159347820)

[3.4 Запуск и работа с приложением: 13](#_Toc159347821)

[3.5 Дополнительные возможности: 13](#_Toc159347822)

[3.6 Сообщения системному программисту: 14](#_Toc159347823)

[4 Руководство программиста 15](#_Toc159347824)

[4.1 Назначения и условия применения программы: 15](#_Toc159347825)

[4.2 Характеристики программы 15](#_Toc159347826)

[4.3 Обращение к программе 15](#_Toc159347827)

[4.4 Входные и выходные данные 15](#_Toc159347828)

[4.5 Сообщения программисту 16](#_Toc159347829)

[5 Руководство оператора 16](#_Toc159347830)

[5.1 Назначение программы 16](#_Toc159347831)

[5.2 Условия выполнения программы 16](#_Toc159347832)

[5.3 Выполнения программы 17](#_Toc159347833)

[5.4 Сообщения оператору 17](#_Toc159347834)

[6 Руководство пользователя 17](#_Toc159347835)

[6.1 Как использовать функции системы 17](#_Toc159347836)

[6.1.1 Банкомат 17](#_Toc159347837)

[6.1.2 Заключение договора 18](#_Toc159347838)

[6.2 Возможные проблемы и пути их решения 18](#_Toc159347839)

[6.3 Часто задаваемые вопросы и ответы на них 18](#_Toc159347840)

[6.4 Где найти информацию по предмету, контактная информация 18](#_Toc159347841)

[6.4.1 Банкомат 18](#_Toc159347842)

[6.4.2 Заключение договора 19](#_Toc159347843)

[7 Результаты работы программы 19](#_Toc159347844)

# Листинг

## Закрытие банковского дня

public DateOnly CloseBankDate(int days = 1)

{

if (days < 1) throw new ServiceException($"Сan not turn back time");

DateOnly today = \_settingsService.ReadToday();

DateOnly future = today.AddDays(days);

\_depositService.CloseBankDay(today, future);

\_creditService.CloseBankDay(today, future);

return \_settingsService.UpdateToday(future);

}

## Создание нового клиента

public Client Create(Client data)

{

using var dbContext = new BankContext();

var client = dbContext.Clients.Add(data);

try

{

dbContext.SaveChanges();

}

catch (DbUpdateException ex)

{

throw new ServiceException($"Can not create a client: {ex.InnerException?.Message}");

}

return client.Entity;

}

## Изменение информации о клиенте

public Client Update(Client data)

{

using var dbContext = new BankContext();

try

{

var existingClient = dbContext.Clients.FirstOrDefault(c => c.Id == data.Id);

if (existingClient != null)

{

dbContext.Entry(existingClient).CurrentValues.SetValues(data);

dbContext.SaveChanges();

return existingClient;

}

throw new ServiceException($"Client with id {data.Id} not found");

}

catch (DbUpdateException ex)

{

throw new ServiceException($"Can not update a client: {ex.InnerException?.Message}");

}

}

## Заключение депозитного контракта

public void ContractConclusion(int contractId)

{

using var dbContext = new BankContext();

var contract = dbContext.Contracts.Find(contractId);

if (contract == null) return;

var currentClientAccount = dbContext.ClientAccounts.Add(new ClientAccount

{

AccountNumber = GenerateAccountNumber(contract.ClientId, contract.AccountTypeId),

AccountTypeId = contract.AccountTypeId,

Debit = 0,

Credit = 0,

CurrencyId = contract.CurrencyId,

ClientId = contract.ClientId,

ContractId = contract.Id,

AccountClientType = AccountClientType.Current

}).Entity;

dbContext.SaveChanges();

var interestClientAccount = dbContext.ClientAccounts.Add(new ClientAccount

{

AccountNumber = GenerateAccountNumber(contract.ClientId, contract.AccountTypeId),

AccountTypeId = contract.AccountTypeId,

Debit = 0,

Credit = 0,

CurrencyId = contract.CurrencyId,

ClientId = contract.ClientId,

ContractId = contract.Id,

AccountClientType = AccountClientType.Interest

}).Entity;

dbContext.SaveChanges();

using var transaction = dbContext.Database.BeginTransaction();

try

{

var activeBankAccount = dbContext.Accounts.FirstOrDefault(account =>

account.CurrencyId == contract.CurrencyId && account.AccountType.Code == BankContext.BANK\_ACTIVE\_CODE);

var passiveBankAccount = dbContext.Accounts.FirstOrDefault(account =>

account.CurrencyId == contract.CurrencyId && account.AccountType.Code == BankContext.BANK\_PASSIVE\_CODE);

if (activeBankAccount == null || passiveBankAccount == null)

throw new ServiceException($"There is no bank account with {contract.Currency.Code} currency");

var depositAmount = contract.Amount;

activeBankAccount.Debit += depositAmount;

activeBankAccount.Credit += depositAmount;

currentClientAccount.Credit += depositAmount;

currentClientAccount.Debit += depositAmount;

passiveBankAccount.Credit += depositAmount;

dbContext.SaveChanges();

transaction.Commit();

}

catch (ServiceException ex)

{

transaction.Rollback();

throw new ServiceException($"Contract failure: {ex.Message}");

}

catch (Exception ex)

{

transaction.Rollback();

throw new ServiceException($"Contract failure: {ex.InnerException?.Message}");

}

}

## Заключение кредитного контракта

public void ContractConclusion(int contractId)

{

using var dbContext = new BankContext();

var contract = dbContext.Contracts.Find(contractId);

if (contract == null) return;

var currentClientAccount = dbContext.ClientAccounts.Add(new ClientAccount

{

AccountNumber = GenerateAccountNumber(contract.ClientId, contract.AccountTypeId),

AccountTypeId = contract.AccountTypeId,

Debit = 0,

Credit = 0,

CurrencyId = contract.CurrencyId,

ClientId = contract.ClientId,

ContractId = contract.Id,

AccountClientType = AccountClientType.Current

}).Entity;

dbContext.SaveChanges();

if (contract.UseCard)

{

var cardService = new CardService();

cardService.Create(contractId);

}

var percentClientAccount = dbContext.ClientAccounts.Add(new ClientAccount

{

AccountNumber = GenerateAccountNumber(contract.ClientId, contract.AccountTypeId),

AccountTypeId = contract.AccountTypeId,

Debit = 0,

Credit = 0,

CurrencyId = contract.CurrencyId,

ClientId = contract.ClientId,

ContractId = contract.Id,

AccountClientType = AccountClientType.Interest

}).Entity;

dbContext.SaveChanges();

using var transaction = dbContext.Database.BeginTransaction();

try

{

var activeBankAccount = dbContext.Accounts.FirstOrDefault(account =>

account.CurrencyId == contract.CurrencyId && account.AccountType.Code == BankContext.BANK\_ACTIVE\_CODE);

var passiveBankAccount = dbContext.Accounts.FirstOrDefault(account =>

account.CurrencyId == contract.CurrencyId && account.AccountType.Code == BankContext.BANK\_PASSIVE\_CODE);

if (activeBankAccount == null || passiveBankAccount == null)

throw new ServiceException($"There is no bank account with {contract.Currency.Code} currency");

var creditAmount = contract.Amount;

passiveBankAccount.Debit += creditAmount;

currentClientAccount.Debit += creditAmount;

// Cash out

if (!contract.UseCard)

{

currentClientAccount.Credit += creditAmount;

activeBankAccount.Debit += creditAmount;

activeBankAccount.Credit += creditAmount;

}

dbContext.SaveChanges();

transaction.Commit();

}

catch (Exception ex)

{

transaction.Rollback();

throw new ServiceException($"Contract failure: {ex.InnerException?.Message}");

}

}

## Создание новой кредитной карты

public Card Create(int contractId)

{

using var dbContext = new BankContext();

var contract = dbContext.Contracts.Find(contractId);

if (contract == null)

throw new ServiceException($"Contract with id {contractId} does not exists");

if (contract.ProgrammeType != ProgrammeType.Annuity && contract.ProgrammeType != ProgrammeType.Differential)

throw new ServiceException($"Сan not create a credit card for a non credit contract");

try

{

var card = dbContext.Cards.Add(new Card

{

Number = GenerateCardNumber(),

ContractId = contractId

});

dbContext.SaveChanges();

return card.Entity;

}

catch (Exception ex)

{

throw new ServiceException($"Can not create a card: {ex.InnerException?.Message}");

}

}

# Интерфейс пользователя

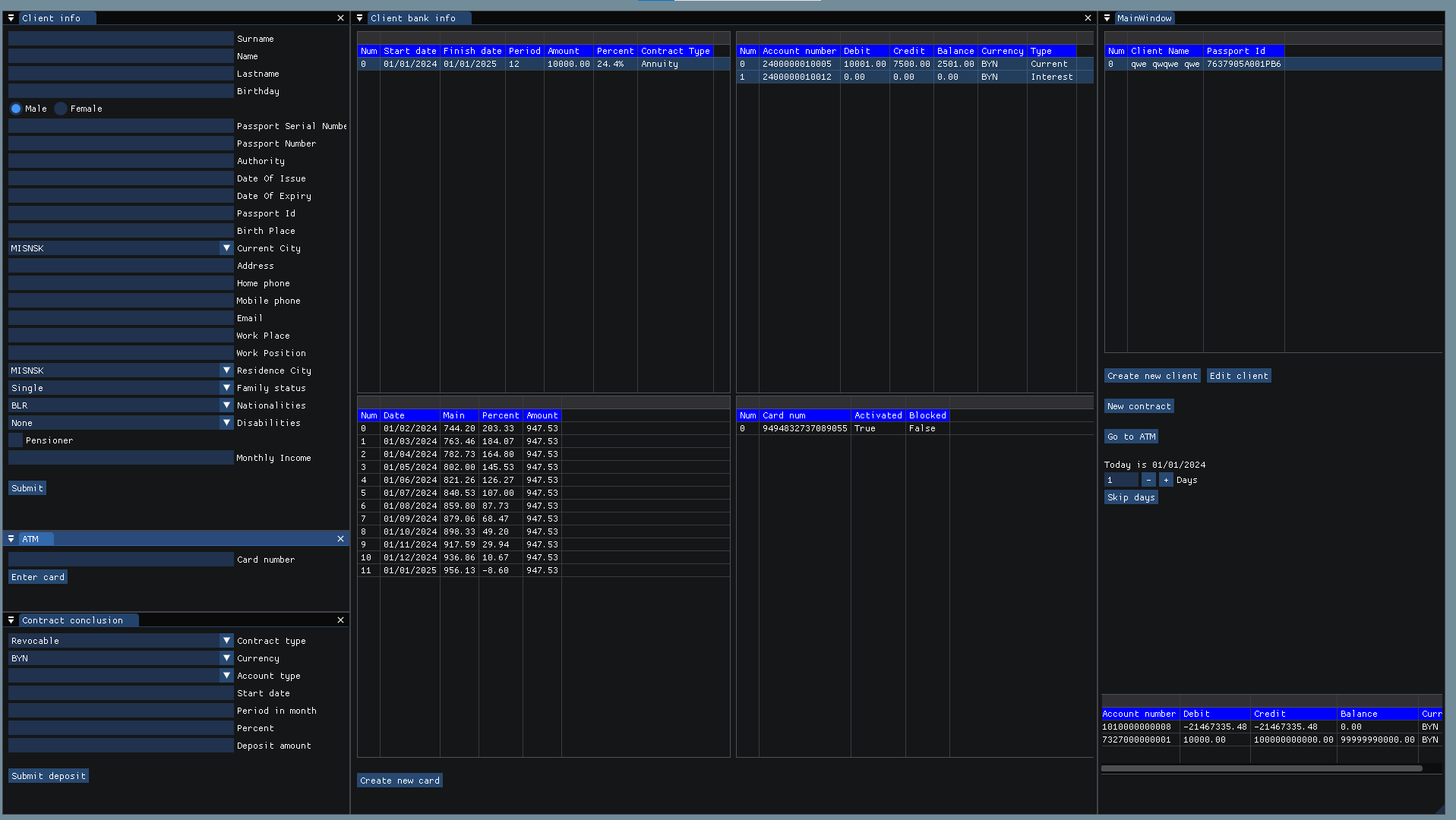


Рисунок 1 – Пользовательский интерфейс

Пользовательский интерфейс состоит из следующих окон:

* Main window – содержит информацию о клиентах банка, о сегодняшней дате и о состоянии счёта банка. Данное окно не может быть закрыто, из него вызываются остальные окна в приложении;
* Client info – окно, которые содержит всю информацию о выбранном клиенте, позволяет создавать новых клиентов и редактировать существующих клиентов. Вызывается через кнопки «Create new client» и «Edit client»;
* Contract conclusion – окно для заключения различных договоров с клиентами банка. Вызывается через кнопку «New contract»;
* Client bank info – окно с информацией о заключённых контрактах клиента, о всех его соответствующих счетах, информацией о необходимых кредитных выплатах, кредитных карт, привязанных к счетам, и возможностью создавать новые кредитные карты. Вызывается двойным нажатием на клиента из списка в Main window;
* ATM – окно, которое эмулирует работу банкомата. Необходимо для тестирования. Вызывается через кнопку «Go to ATM».

# Руководство системного программиста

## Общие сведения о программе

Десктопное приложение работает на платформе .NET 8.0.0. Требуется предварительная установка данной платформы. Требуется предварительная настройка информационной системы: указание режима подключения к базе данных. База данных работает под управлением СУБД PostgreSQL. Требуется предварительная установка и конфигурация данной СУБД.

## Структура программы

Приложение состоит из 2 частей – десктопного приложения, написанного на языке программирования С# и базы данных PostgreSQLю

Десктопное приложение содержит следующие NuGet пакеты: Npgsql, ImGui.Net, Veldrid, Veldrid.SDL2, Veldrid.StartapUtiities.

Файловая структура десктопного приложения:

* папка Context содержит классы для объектно-реляционного отображения объектов баз данных в объекты языка С# и для получения контекста базы данных;
* папка Converters содержит классы для конвертации данных из представления клиентской части в представление данных в базе данных на сервере;
* папка DTO – классы, агрегирующие данные для передачи;
* папка Entities – классы, представляющие объекты в БД;
* папка Enums – перечисления, которые представляют поля в БД;
* папка Exceptions – исключения, возникающие в программе;
* папка Migrations – миграции для инициализации базы данных;
* папка Services – сервисы, через которые UI связывается с бэк-логикой приложения;
* папка Shaders – шейдеры для работы ImGui;
* файл EntryPoint – входная точка в приложение содержит инициализацию UI и работы с БД;
* файл Helper – весь UI приложения;
* файл ImGuiController – контроллер для работы ImGui.

Структура базы данных:

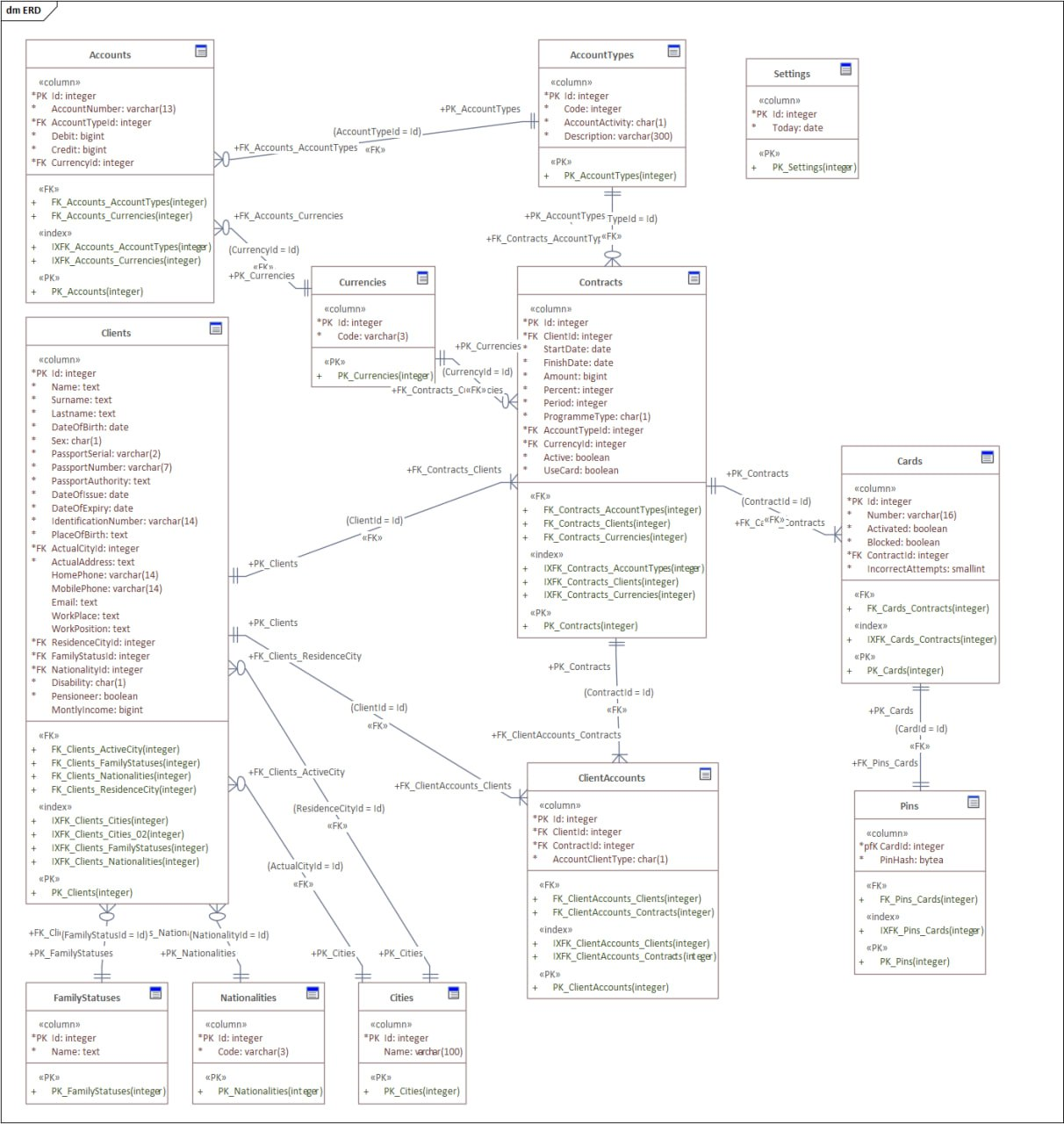


Рисунок 2 – Структура базы данных

AccountTypes – Типы счетов

* Id – уникальный идентификатор типа счёта;
* Code – уникальный код типа счёта на основании «Плана счетов бухгалтерского учета в банках и небанковских кредитно-финансовых организациях»;
* AccountActivity – тип счёта: активный или пассивный;
* Description – описание типа счёта.

Accounts – Счета

* Id – уникальный идентификатор счёта;
* AccountNumber – уникальный 13-ти значный номер счёта;
* AccountTypeId – идентификатор типа счёта;
* Debit – сумма начислений на счёт;
* Credit – сумма списаний со счёта;
* CurrencyId – идентификатор валюты счёта.

Cards – кредитные карты

* Id – уникальный идентификатор кредитной карты;
* Number – уникальный 16-ти значный номер кредитной карты;
* Activated – флаг активации кредитной карты;
* Blocked – флаг блокировки кредитной карты;
* ContractId – идентификатор договора кредитного счёта;
* IncorrectAttempts – количество неверных попыток ввода PIN-кода кредитной карты.

Cities – города

* Id – уникальный идентификатор города;
* Name – название города;

ClientAccounts – счета клиентов

* Id – уникальный идентификатор счёта;
* ClientId – идентификатор клиента;
* ContractId – идентификатор договора;
* AccountClientType – тип счёта клиента (текущий, процентный).

Clients – клиенты

* Id – уникальный идентификатор клиента;
* Name – имя клиента;
* Surname – фамилия клиента;
* Lastname – отчество клиента;
* DateOfBirth – дата рождения клиента;
* Sex – пол клиента (Male, Female);
* PassportSerial – серия паспорта клиента;
* PassportNumber – номер паспорта клиента;
* PassportAuthority – орган, который выдал паспорт клиенту;
* DateOfIssue – дата выдачи паспорта клиента;
* DateOfExpiry – дата истечения срока действия паспорта клиента;
* IdentificationNumber – идентификационный номер паспорта клиента;
* PlaceOfBirth – место рождения клиента;
* ActualCityId – идентификатор города фактического проживания клиента;
* ActualAddress – фактический адрес клиента;
* HomePhone – номер домашнего телефона клиента;
* MobilePhone – номер мобильного телефона клиента;
* Email – email клиента;
* WorkPlace – место работы клиента;
* WorkPosition – занимая должность клиента;
* ResidenceCityId – идентификатор города прописки клиента;
* FamilyStatusId – идентификатор семейного положения клиента;
* NationalityId – идентификатор национальности клиента;
* Disability – степень инвалидности клиента;
* Pensioner – отношение клиента к пенсионерам;
* MonthlyIncome – ежемесячный доход.

Contracts – договоры

* Id – уникальный идентификатор договора;
* ClientId – уникальный идентификатор клиента;
* StartDate – дата начала действия договора;
* FinishDate – дата окончания действия договора;
* Amount – сумма договора;
* Percent – процент договора;
* Period – срок действия договора в месяцах;
* ProgrammeType – тип депозитной (отзывная, безотзывная) или кредитной (аннуитетная, дифференциальная) программы;
* AccountTypeId – идентификатор типа открываемого счёта;
* CurrencyId – идентификатор валюты открываемого счёта;
* Active – флаг активности счёта;
* UseCard – флаг пользования кредитной картой.

Currencies – валюты

* Id – уникальный идентификатор валюты;
* Code – уникальный код валюты.

FamilyStatuses – семейные положения

* Id – уникальный идентификатор семейного положения;
* Name – название семейного положения.

Nationalities – национальности

* Id – уникальный идентификатор национальности;
* Code – уникальный код национальности.

Pins – PIN-коды

* Id – идентификатор кредитной карты;
* PinHash – хэш код PIN-кода кредитной карты.

## Настройка программы:

* установите .NET на клиент, если его еще нет. Поддерживаемая версия – 8.0 и выше.
* установите PostgreSQL на сервер. Скачайте установочный файл с официального сайта PostgreSQL и следуйте инструкциям по установке.
* создайте новую базу данных в PostgreSQL для приложения "Банк".
* загрузите миграции базы данных, выполнив следующую команду:

dotnet ef database update

* отредактируйте файл конфигурации приложения "Банк" (обычно называется appsettings.json), чтобы указать правильные параметры подключения к базе данных PostgreSQL. Необходимо указать имя сервера, имя базы данных, имя пользователя и пароль.

Пример файла конфигурации:

{

"ConnectionStrings": {

"DefaultConnection": "Server=hostname;Database=bank;User Id=username;Password=password;"

}

}

Замените «hostname», «username» и «password» на соответствующие значения.

## Запуск и работа с приложением:

* запустите базу данных и запустите приложение, после успешного запуска приложение готово к работе.
* проверьте работоспособность приложения, проведя серию тестов пользовательского интерфейса.
* убедитесь, что все функции приложения работают корректно, включая регистрацию новых пользователей, совершение транзакций, заключение договоров, расчёт кредитных выплат и фиксирование потока денежных средств по дебет-кредитовой системе.

## Дополнительные возможности:

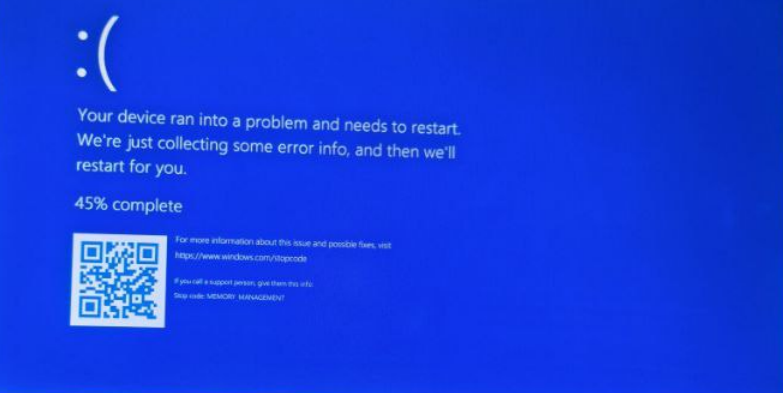
* регулярно проверяйте создание резервных копии базы данных приложения для обеспечения безопасности данных.
* резервные копии по умолчанию создаются в папке рядом с самой базой данных. Изменить это можно в конфигурационном файле базы данных, или через командную строку;
* для восстановления базы данных из резервной копии выполните следующую команду: psql -U username -d bank -f path/to/backup.sql

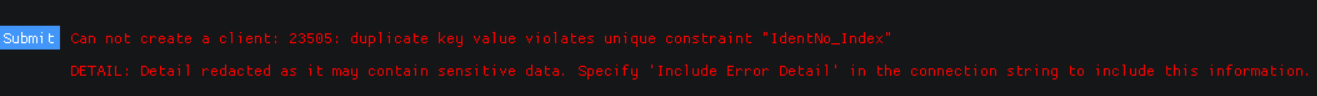
Замените "username" на имя пользователя, от имени которого осуществляется доступ к базе данных, и "path/to/backup.sql" на путь к файлу резервной копии;

* создайте учетные записи для системных администраторов и других пользователей, которым необходим доступ к приложению;
* назначьте соответствующие права доступа каждому пользователю в базе данных PostgreSQL. Ограничьте доступ только к необходимым таблицам и функциям, чтобы обеспечить безопасность данных;
* управляйте учетными записями пользователей, включая создание, блокировку, удаление и изменение паролей;
* регулярно проверяйте журналы доступа и мониторинга для выявления подозрительной активности и потенциальных угроз безопасности.

## Сообщения системному программисту:

* в случае возникновения проблем с приложением обратитесь к документации, разработчикам или сообществу пользователей для получения помощи.
* записывайте и анализируйте ошибки и проблемы, чтобы определить их причины и найти решения.
* поддерживайте регулярные резервные копии данных, чтобы восстановиться в случае сбоев или потери данных.
* участвуйте в обновлениях и исправлениях безопасности, рекомендованных разработчиками, чтобы обеспечить безопасность приложения и данных.





# Руководство программиста

## Назначения и условия применения программы:

Общая структура приложения, включая основные модули, компоненты и их взаимодействие.

Используемые технологии и фреймворки, включая ASP.NET Core для веб-интерфейса, C# для бизнес-логики и взаимодействия с базой данных PostgreSQL.

## Характеристики программы

Программное средство работает в режиме приближённом к режиму реального времени для предоставления всегда актуальных данных. UI программы построен по методу immediate mode для упрощения программного продукта. Программное средство содержит обширное множество способов обработки и уведомления об ошибках, а также различного рода проверки входных данных на стороне клиента и БД. Также система обладает множеством

## Обращение к программе

Для связи UI и бизнес-логики программы предоставляется множество сервисов, которые удобным образом позволяют получать и предоставлять актуальную информацию.

UI сделан по принципу immediate mode, поэтому кэширование всех данных для отображения происходит в одном месте и нет необходимости синхронизировать несколько мест.

Для работы с сервисом его необходимо создать и в дальнейшем обращаться. Все сервисы предоставляют одинаковый по структуре CRUD интерфейс, общение с которым производится по одному принципу: указать что сделать и с кем сделать.

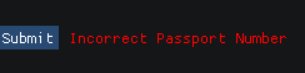
## Входные и выходные данные

Входная и выходная информация проверяется программно при создании и изменении пользователей, заключении договором, создании и работе с кредитными картами клиентов. К особенностям организации входной информации можно добавить, что взаимодействие с БД происходит по ORM-модели. Все данные представлены в кодировке UTF-8.

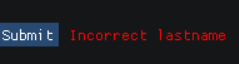
## Сообщения программисту

* в случае возникновения и проблем или ошибок следует обратиться к документации;
* все крупные изменения, дополнения, удаления программного кода следует документировать;

Примеры ошибок:







# Руководство оператора

## Назначение программы

Функциональным назначением программы является предоставлению сотрудникам профильных подразделений Заказчика удобного автоматизированного интерфейса для:

* создания, чтения, редактирования и удаления клиентов банка;
* открытия и закрытия кредитного клиентского счёта;
* открытия и закрытия депозитного клиентского счёта;
* автоматизированного составления договора на открытие счёта;
* автоматического фиксирования потока денежных средств по дебет-кредитовой системе;
* автоматического расчёта кредитных выплат;
* системы управления кредитными картами клиентов.

## Условия выполнения программы

Для корректного функционирования программы требуется персональный компьютер, включающий в себя:

* минимальная конфигурация ЦПУ: Intel Core I5 5500K;
* минимальный объем ОЗУ: 16 ГБ;
* минимальный объём дискового пространства: 512 ГБ;
* ОС Windows 10 22H2;
* минимальное разрешение дисплея: 720х640;

Для запуска программы на ПК должен быть .NET не ниже версии 8.0 и сам исполняемый файл с необходимыми для его работы файлами

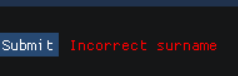
## Выполнения программы

* для запуска приложения достаточно запустить соответствующий .exe файл;
* при первом запуске приложения на экране будет находится окно, содержащее пользователей системы;
* приложение позволяет добавлять, удалять, изменять информацию о пользователях, их заключённых договорах, счетах, и кредит
* приложение закрывается комбинацией кнопок Alt+F4;
* для более подробной информации о интерфейсе приложения читайте раздел «интерфейс пользователя».

## Сообщения оператору

* в случае некорректного ввода данных приложение уведомит соответствующим сообщением об ошибке, если ошибка вам не знакома, обратитесь в техподдержку и выполните их указания;
* при аварийном завершении приложения, или если приложение не запускается уведомите техподдержку о проишествии.

Примеры ошибок:







# Руководство пользователя

## Как использовать функции системы

### Банкомат

* Вставьте банковскую карту в устройство чтения карты с чипом или магнитной полосой.
* Введите PIN-код для аутентификации.
* Выберите желаемую операцию (снятие/пополнение наличных, просмотр баланса и т.д.).
* Следуйте инструкциям на экране и выполните необходимые действия (выбор счета, ввод суммы и т.д.).
* После завершения операции возьмите карту, деньги и полученный чек.

### Заключение договора

* Обратитесь в ближайшее отделение банка.
* Предъявите документы, удостоверяющие личность.
* Обсудите условия и требования договора с банковским менеджером.
* Подпишите необходимые документы.
* Получите копии подписанных договоров и уточните детали сотрудником банка, если это необходимо.

## Возможные проблемы и пути их решения

* Проблема: Карта застряла в устройстве.
  + Решение: Немедленно свяжитесь с банковским оператором по указанному на банкомате номеру или обратитесь в ближайшее отделение банка для помощи.
* Проблема: Необходимо дополнительное время для понимания условий договора.
  + Решение: Попросите банковского менеджера прояснить все непонятные моменты и задайте все интересующие вас вопросы.

## Часто задаваемые вопросы и ответы на них

Могу ли я использовать банкомат другого банка?

* Да, но будет взиматься комиссия за обслуживание

Какие документы необходимо предоставить для заключения договора?

* Паспорт или другой документ, удостоверяющий личность

## Где найти информацию по предмету, контактная информация

### Банкомат

Для получения подробной информации о работе банкомата и возможных комиссиях необходимо обратиться в банк или посетить веб-сайт банка.

### Заключение договора

Для получения дополнительной информации о заключении договоров и условиях обслуживания обратитесь в ближайшее отделение банка или посетите его веб-сайт.

# Результаты работы программы

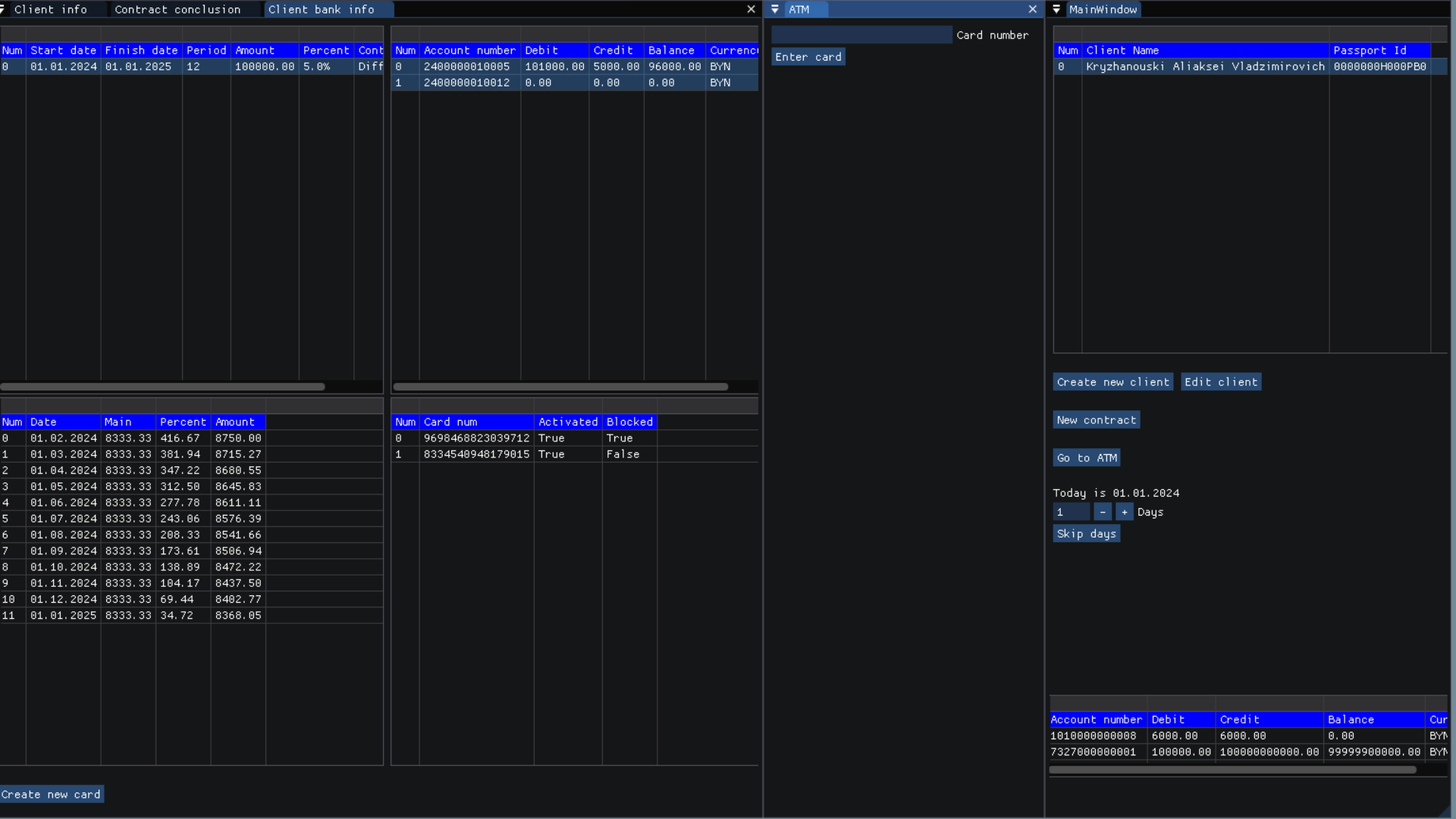


Рисунок 3 – Главное окно программы

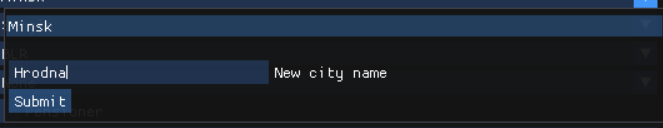


Рисунок 4 – Окно добавления нового города

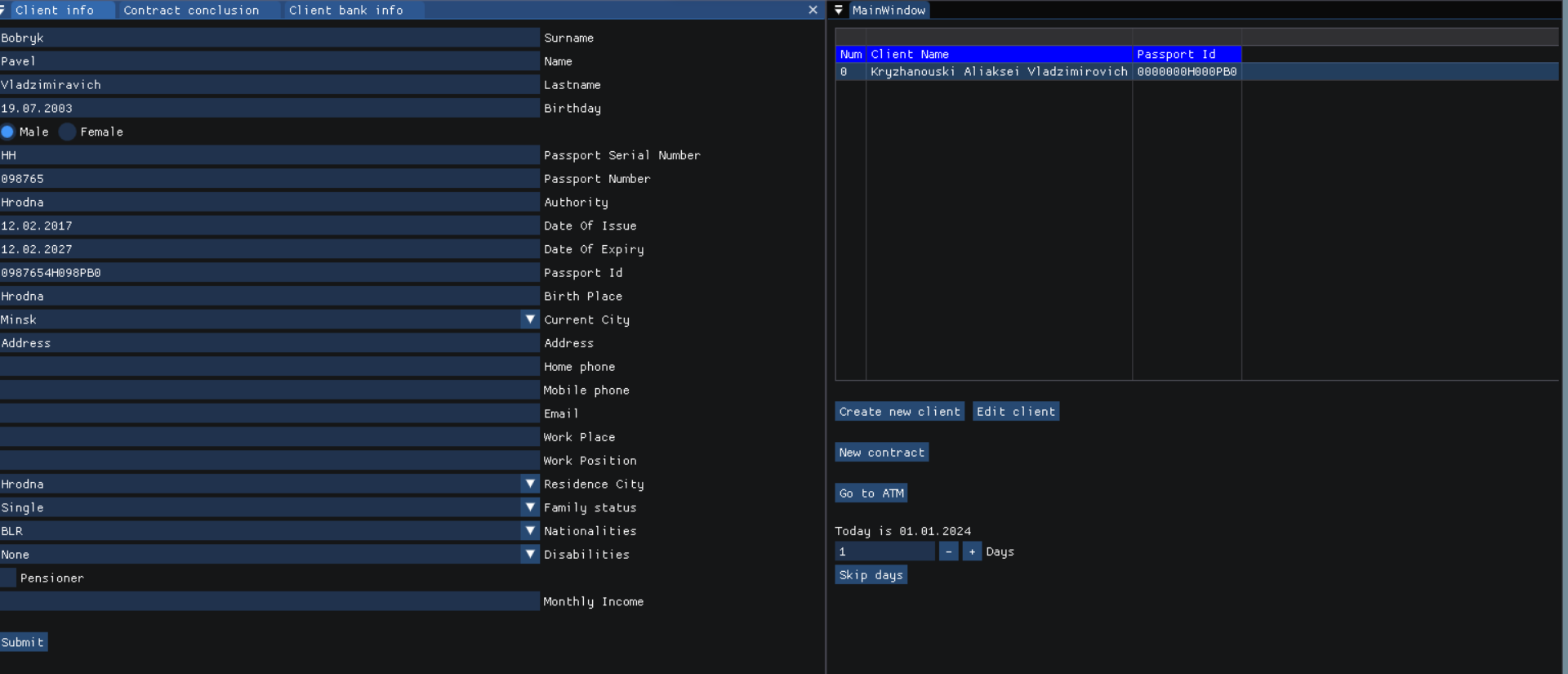


Рисунок 5 – Окно добавления нового пользователя

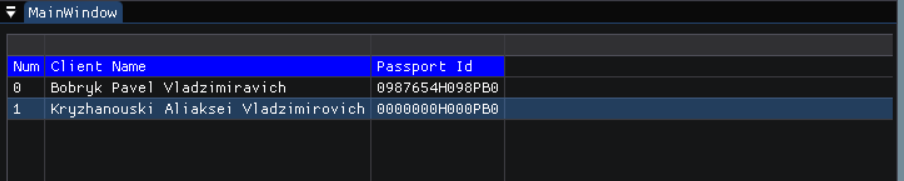


Рисунок 6 – Новый пользователь в системе

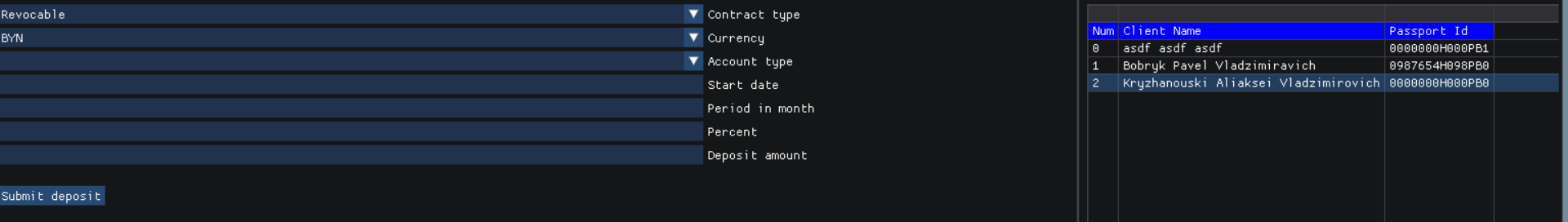


Рисунок 7 – Окно заключения договора

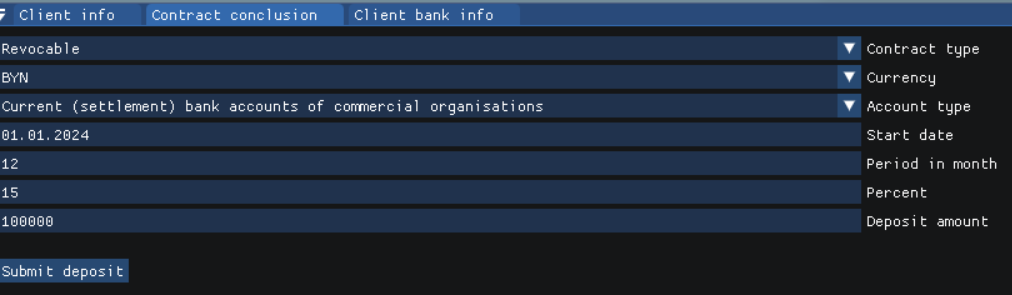


Рисунок 8 – Окно заполненного договора

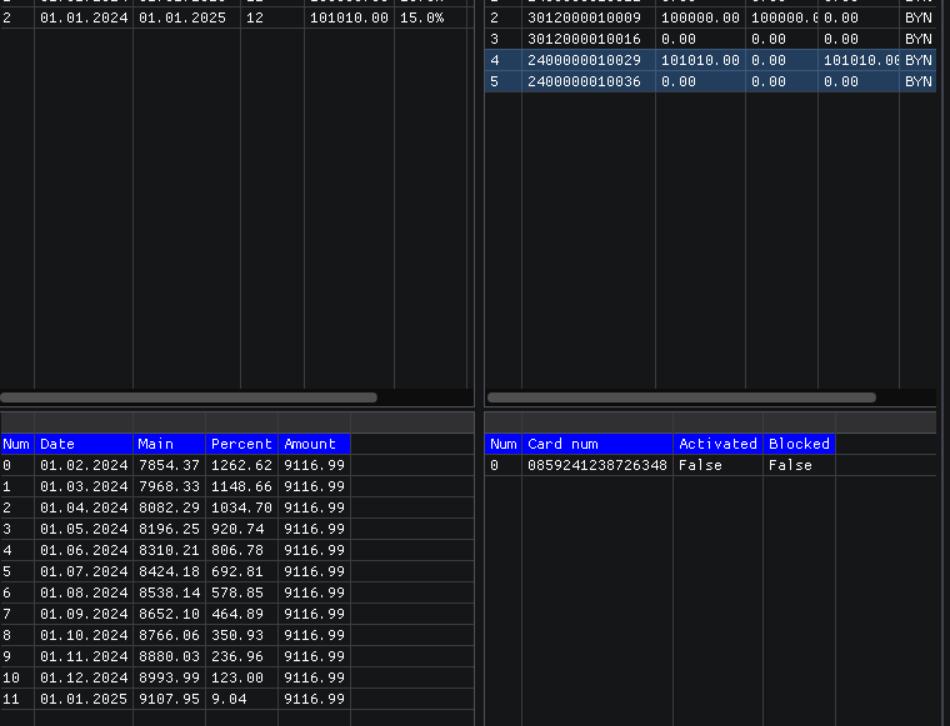


Рисунок 9 – Окно графика расчёта кредитных выплат