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

Практическая работа №3

Этапы проектирования и разработки ИС при структурном подходе к программированию. Стадия ”Реализация”

Выполнили: Шуляк А. В.
Студенты группы 051006 Дранкевич А. А.

Проверил: Грибович А. А

Минск 2024

CONTENTS

1 ЛИСТИНГ

1.1 Закрытие банковского дня

```
function processBankingDay(): void {
    console.log(`processBankingDay date ${date}`)
    clients.forEach(client => {
        // process client deposits
        client.deposits.forEach(deposit => {
            if (!deposit.active) return
            deposit.active = isLessThanOrEqualDay(date
              , deposit.endDate)
            if (deposit.active) {
                // deposit is still active
                const value = deposit.interestRate /
                  365 * deposit.value
                console.log(`deposit payments ${value}
                   at ${date}`)
                // Interest accrual
                deposit.accInterest.credit += value
                accountCashHolder.debit += value
                if (deposit.revocable) {
                    // Interest withdrawal (part)
                    deposit.accInterest.debit += value
                    accountCashRegister.debit += value
                    accountCashRegister.credit +=
                      value
                }
            } else {
                // finish of contract?
                console.log(`deposit withdrawal at ${
                  date}`)
                // Interest withdrawal (full)
                if (!deposit.revocable) {
```

```
                deposit.accInterest.debit +=
                  deposit.accInterest.credit
                accountCashRegister.debit +=
                  deposit.accInterest.credit
                accountCashRegister.credit +=
                  deposit.accInterest.credit
            }
            // Deposit withdrawal
            accountCashHolder.debit += deposit.
              value
            deposit.accCurrent.credit += deposit.
              value
            deposit.accCurrent.debit += deposit.
              value
            accountCashRegister.debit += deposit.
              value
            accountCashRegister.credit += deposit.
              value
            //
            deposit.accCurrent.info += ” Закрыт()”
            deposit.accInterest.info += ” Закрыт()”
        }
    })
    // process client credits
    client.credits.forEach(credit => {
        if (!credit.active) return
        //
        credit.active = isLessThanOrEqualDay(date,
            credit.endDate)
        // finish of contract?
        if (!credit.active) {
            // here credit must be 0.0 ???
            console.log(‘credit withdrawal at ${
              date}‘)
            //
```

```
    credit.accCurrent.info += ” Закрыт()”
    credit.accInterest.info += ” Закрыт()”
    return
}
// credit is still active
const days = (credit.endDate.getTime() -
  credit.startDate.getTime()) / (1000 *
  3600 * 24)
console.log(‘days ${days}‘)

let valueA = 0
let valueB = 0
if (credit.revocable) { // annuity
    const interestDaily = credit.
      interestRate / 365
    const daily = credit.value *
      interestDaily * Math.pow(1 +
      interestDaily, days) / (Math.pow(1 +
       interestDaily, days) - 1)
    console.log(‘daily ${daily}‘)

    valueA = daily - credit.value / days
    valueB = credit.value / days
} else {
    const interestDaily = credit.
      interestRate / 365
    const daysLeft = (credit.endDate.
      getTime() - date.getTime()) / (1000
      * 3600 * 24)

    valueA = (credit.value / days * (
      daysLeft + 1)) * interestDaily
    valueB = credit.value / days
}
```

```
            console.log(`credit payments ${valueA +
              valueB} at ${date}`)
            // Interest accrual
            credit.accInterest.credit += valueA
            accountCashHolder.credit += valueA
            // receive payments for credit interest
            accountCashRegister.debit += valueA
            accountCashRegister.credit += valueA
            credit.accInterest.debit += valueA
            // receive payments for credit value
            accountCashRegister.debit += valueB
            accountCashRegister.credit += valueB
            credit.accCurrent.debit += valueB
            // end of credit (money back)
            credit.accCurrent.credit += valueB
            accountCashHolder.credit += valueB
        })
    })
    //
    fixBalance(accounts)
}
```

1.2 Создание нового клиента

```
async function create_client(): void {
    const formData = Object.assign({}, req.body);
    const index = clients.findIndex(client =>
        (client.passportSeries === formData.
          passportSeries && client.passportNumber ===
          formData.passportNumber) ||
        client.identificationNumber === formData.
          identificationNumber
    );
    if (index !== -1) {
```

```
            await get_client_in_db(formData);
            return res.render('add_client.hbs', {layout :
              'index', error: true, client: formData});
        }
        formData.id = (clients.length > 0 ? clients[
          clients.length - 1].id + 1 : 1);
        clients.push(formData);
        await add_or_update_client_in_db(formData);
        res.render('add_client.hbs', {layout : 'index',
          error: false, client: formData});
}
```

1.3 Редактирование клиента

```
async function(): void {
    const id = parseInt(req.params.id);
    console.log(`client edit request ${id}`)
    var client = clients[id - 1]
    if (!client) {
        return res.status(404).render('404.hbs', {
          layout : 'index'});
    }
    await get_client_from_db(client);
    res.render('add_client.hbs', {
        layout : 'index',
        client: client,
        edit: true
    });
}
```

1.4 Заключение депозитного контракта

```
async function create_deposit(): void {
const formData = req.body;
```

```
const clientId = parseInt(formData.contractNumber)
console.log(formData);
const index = await get_client_index();
if (index === -1) {
    return res.render('deposit.hbs', {layout : '
      index', error: true});
}
let depositType = безотз""
if (formData.depositType.toLowerCase() === "
  revocable") {
    depositType = отзывн""
}
const client = clients[formData.contractNumber -
  1];
const value = parseInt(formData.depositAmount)
const accCurrent =  addAccountForClient(client, "
  Passive",
    formData.currencyType, 0, Текущий‘ счет ${
      depositType} депозита‘);
const accInterest = addAccountForClient(client, "
  Passive",
    formData.currencyType, 0, Процентный‘ счет ${
      depositType} депозита‘);
accounts.push(accCurrent)
accounts.push(accInterest)
const card: Card = {number: "0000-0000-0000-0000",
   pin: "0000", contract: null, attempts: 3, otp:
  ''};
const contract: Contract = {
    accCurrent: accCurrent,
    accInterest: accInterest,
    interestRate: parseFloat(formData.interestRate
      ) * 0.01, // TODO
    revocable: formData.depositType === "revocable
      ",
```

```
            endDate: new Date(date.getFullYear(), date.
              getMonth(), date.getDate() + parseInt(
              formData.interestTime)),
            startDate: new Date(date),
            value: value,
            active: true,
            card: card
        };
        card.contract = contract
        client.deposits.push(contract)
        accountCashRegister.debit += value
        accountCashRegister.credit += value
        accCurrent.credit += value
        accCurrent.debit += value
        accountCashHolder.credit += value
        await update_accs_in_db(client);
        fixBalance([accountCashRegister, accountCashHolder
          , accCurrent, accInterest])
        console.log("deposit contract signed")
        console.log(client)
        res.render('deposit.hbs', {layout : 'index', error
          : false});
}
```

1.5 Заключение кредитного контракта

```
async function(): void {
    const formData = req.body;
    const clientId = parseInt(formData.contractNumber)
    console.log(formData);
    const index = clients.findIndex(client =>
        client.id === clientId
    );
    if (index === -1) {
```

```
        return res.render('credit.hbs', {layout : '
            index', error: true});
    }
    let creditType = дифф""
    if (formData.creditType === "a") {
        creditType = аннуитетн""
    }
    const client = clients[formData.contractNumber -
        1];
    const value = parseInt(formData.creditAmount)
    const accCurrent =  addAccountForClient(client, "
        Active",
        "BYN", 0, Текущий‘ счет ${creditType} кредита‘);
    const accInterest = addAccountForClient(client, "
        Active",
        "BYN", 0, Процентный‘ счет ${creditType}
            кредита‘);
    accounts.push(accCurrent)
    accounts.push(accInterest)
    const card: Card = createNewCard(client.id, client
        .deposits.length);
    const contract: Contract = {
        accCurrent: accCurrent,
        accInterest: accInterest,
        interestRate: 0.14, // TODO fixed interest
            rate
        revocable: formData.creditType === "a", //
            here it means annuity
        endDate: new Date(date.getFullYear(), date.
            getMonth(), date.getDate() + parseInt(
            formData.interestTime)),
        startDate: new Date(date),
        value: value,
        active: true,
        card: card
```

```
};
card.contract = contract;
client.credits.push(contract);
cards.push(card)
accountCashHolder.debit += value
accCurrent.debit += value
fixBalance([accountCashRegister, accountCashHolder
  , accCurrent, accInterest])
const plan: {date: Date, payment: number, rest:
  number, sum: number}[] = []
if (formData.creditType === "a") { // annuity
    const m = parseInt(formData.interestTime)
    const s = value
    const interestDaily = 0.14 / 365
    const daily = s*interestDaily * Math.pow(1+
      interestDaily, m) / (Math.pow(1+
      interestDaily, m) - 1)
    let rest = s
    let sum = 0
    for (let i = 0; i < m; ++i) {
        rest -= s / m
        sum += daily
        plan.push({
            date: new Date(date.getFullYear(),
              date.getMonth(), date.getDate() + i
              + 1),
            payment: daily,
            rest: Math.abs(rest),
            sum: Math.abs(sum)
        })
    }
} else {
    const m = parseInt(formData.interestTime)
    const s = value
    const interestDaily = 0.14 / 365
```

```
            let dailyFirst = s / m + s * interestDaily
            let rest = s
            let sum = 0
            for (let i = 0; i < m; i++) {
                dailyFirst = s / m + rest * interestDaily
                sum += dailyFirst
                rest -= s / m
                plan.push({
                    date: new Date(date.getFullYear(),
                      date.getMonth(), date.getDate() + i
                      + 1),
                    payment: dailyFirst,
                    rest: Math.abs(rest),
                    sum: Math.abs(sum)
                })
            }
        }
        await update_credit_in_db(client);
        console.log("credit contract signed")
        console.log(client)
        res.render('credit.hbs', {layout : 'index', error:
            false, plan: plan});
}
```

1.6 Создание новой кредитной карты

```
function createNewCard(client_id: number, credit_id:
  number): Card {
    let randomNumber = Math.floor(Math.random() *
      10000);
    let numberString = randomNumber.toString().
      padStart(4, '0');
    let number = "3456-"
```

```
        + ((client_id * 17 + 6789) % 10000).toString().
          padStart(4, '0') + "-"
        + ((credit_id * 31 + 4321) % 10000).toString().
          padStart(4, '0') + "-"
        + Math.floor(Math.random() * 10000).toString().
          padStart(4, '0')
        return {
            number: number,
            pin: numberString,
            contract: null,
            attempts: 3,
            otp: ''
        }
    }
```

2 ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ

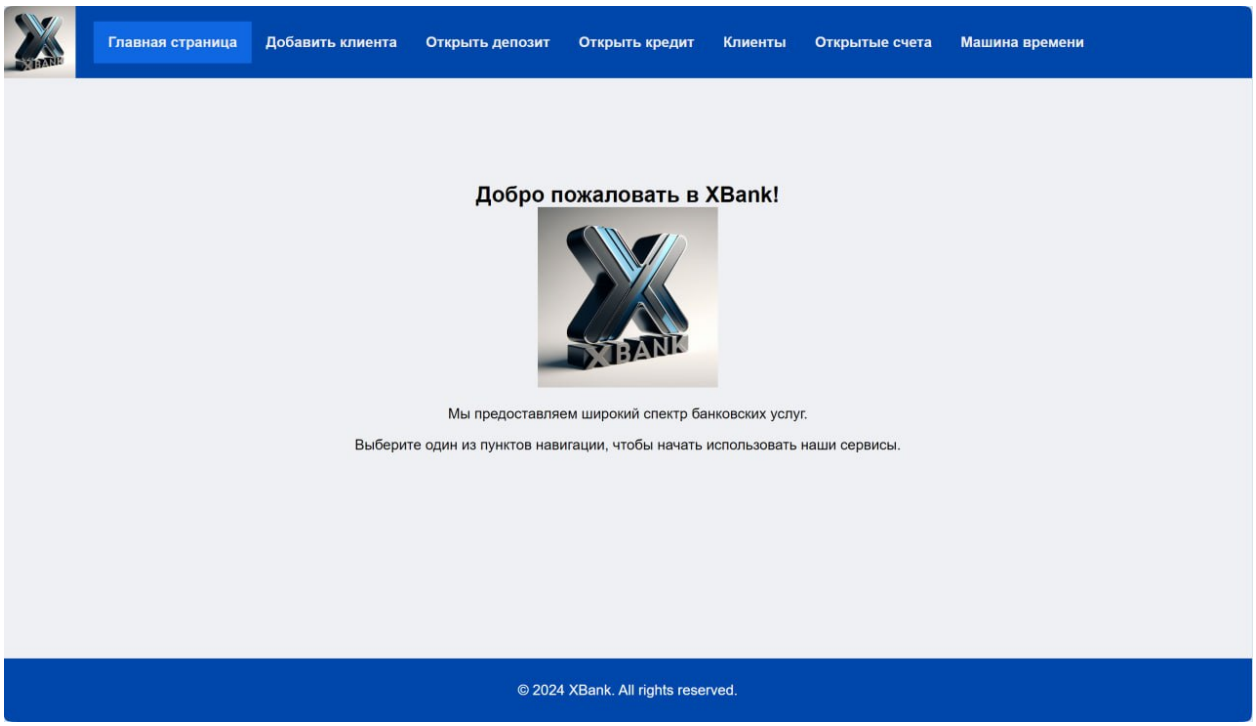

Рисунок 1 — Пользовательский интерфейс, главная страница

Пользовательский интерфейс состоит из следующих страниц:

– ”Добавить клиента” - меню добавления новых клиентов в систему
– ”Открыть депозит” - меню открытия депозита для выбранного пользователя
– ”Открыть кредит” - меню открытия кредита для выбранного пользователя
– ”Клиенты” - список клиентов с информацией по каждому клиенту
– ”Открытые счёта” - просмотр открытых счетов (пластиковых банковских карт)
– ”Машина времени” - демонстрация осуществления закрытия банковского дня
– ”Банкомат” - симуляция работа банкомата

3 РУКОВОДСТВО СИСТЕМНОГО ПРОГРАММИСТА

3.1 Общие сведения о программе

Веб-приложение работает на платформе NodeJS. Требуется предварительная установка данной платформы. Требуется предварительная настройка информационной системы: указание режима подключения к базе данных. База данных работает под управлением СУБД MariaDB. Требуется предварительная установка и конфигурация данной СУБД.

3.2 Структура программы

Приложение состоит из 2 частей - клиент-серверного приложения, написанного на языке программирования TypeScript и базы данных MariaDB. Веб-приложение содержит следующие npm-пакеты: nodemon, handlebars, mariadb. Файловая структура веб-приложения:

- Директория app содержит корневой файл веб-приложения, в котором осуществляется запуск и настройка веб-сервера информационного средства;
- Директория views содержит графический интерфейс пользователя информационной системы;
- Директория helpers содержит вспомогательные средства для работы информационного средства;
- Директория public содержит публичные таблицы каскадных стилей.

Структура базы данных:

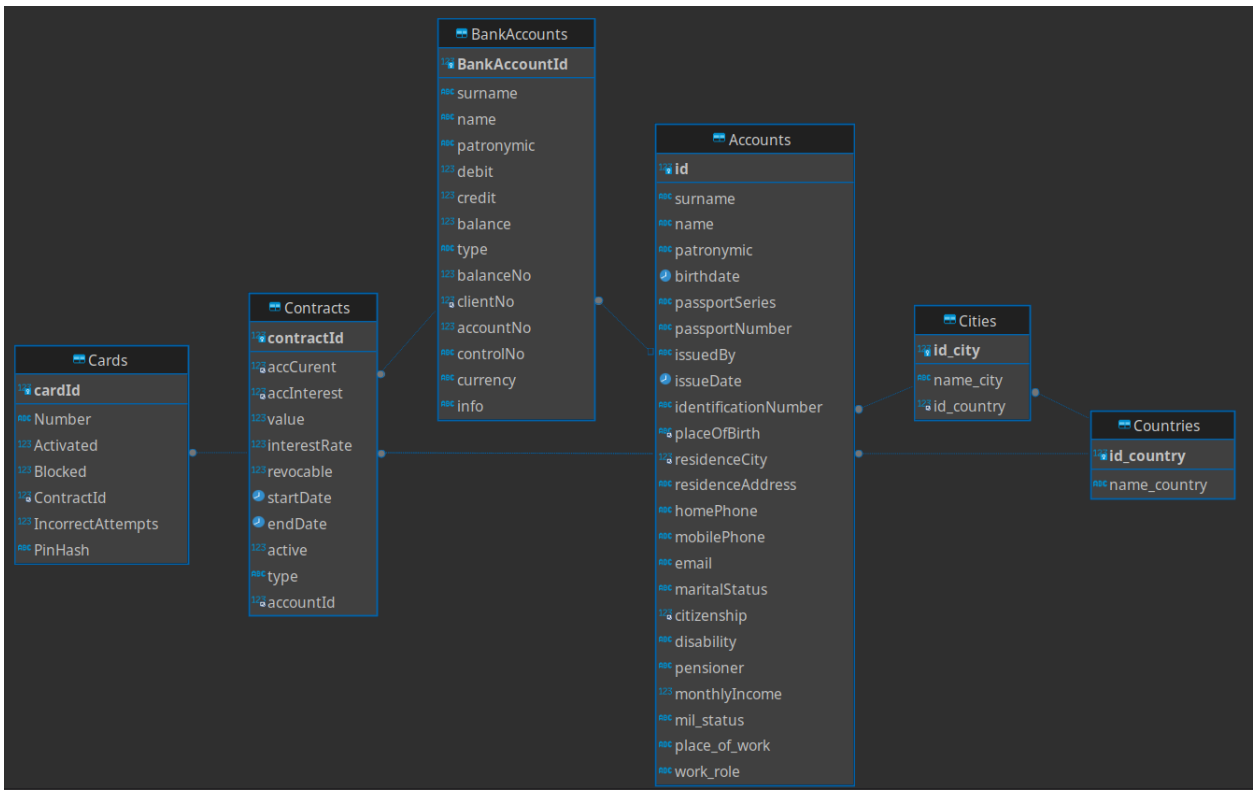

Рисунок 2 — Структура базы данных

Cards - Пластиковые карты:

– cardId - уникальный идентификатор;
– Number - номер карты;
– Activated - флаг активности карты;
– Blocked - флаг блокировки карты;
– ContractId - идентификатор привязанного контракта;
– IncorrectAttempts - число неверных попыток ввода пин-кода;
– PinHash - хеш пин-кода карты.

Contracts - Контракты:

– contractId - уникальный идентификатор;
– accCurent - идентификатор привязанного активного счёта;
– accInterest - идентификатор привязанного пассивного счёта;
– value - сумма, на которую заключён контракт;
– interestRate - процент, под который заключён контракт;
– revocable - флаг отзываемости;
– startDate - начало срока действия контракта;
– endDate - окончание срока действия контракта;

– active - флаг активности контракта;
– type - тип контракта;
– accountId - уникальный идентификатор привязанного аккаунта.

Bank Accounts - Банковские счета:

– BankAccoundId - уникальный илентификатор;
– surname - фамилия клиента;
– name - имя клиента;
– patronymic - отчество клиента;
– debit - дебет данного счёта;
– credit - кредит данного счёта;
– balance - сальдо данного счёта;
– type - тип счёта;
– balanceNo - номер счёта;
– clientNo - идентификатор клиента;
– accountNo - номер клиента;
– controlNo - контрольный символ;
– currency - валюта счёта;
– info - информация о счёте.

Accounts - аккаунты пользователей:

– id - уникальный идентификатор;
– surname - фамилия пользователя;
– name - имя пользователя;
– patronymic - отчество пользователя;
– birthdate - дата рождения пользователя;
– passportSeries - серия паспорта пользователя;
– passportNumber - номер паспорта пользователя;
– issuedBy - орган выдачи паспорта;
– issueDate - срок действия паспорта;
– identificationNumber - идентификационный номер;
– placeOfBirth - место рождения пользователя;
– residenceCity - город проживание пользователя;
– residenceAddress - адрес проживания пользователя;
– homePhone - домашний телефон пользователя;

– mobilePhone - мобильный телефон пользователя;
– email - почта пользователя;
– maritalStatus - семейный статус пользователя;
– citizenship - гражданство пользователя;
– disability - флаг инвалидности пользователя;
– pensioner - флаг наличия пенсии у пользователя;
– monthlyIncome - доход пользователя за месяц;
– mil_status - флаг военнообязанности пользователя;
– place_of_work - место работы пользователя;
– work_role - должность пользователя.

Cities - Города:

– id_city - уникальынй идентификатор пользователя;
– name_city - название города;
– id_country - идентификатор страны города.

Countries - страны:

– id_country - уникальный идентификатор страны;
– name_country - название страны.

3.3 Настройка программы

– Установите NodeJS на сервер, если его ещё нет;
– Установите MariaDB на сервер. Используйте официально поддерживаемый операционной системой сервера метод установки.
– Создайте новую базу данных в MariaDB для приложения ”Банк”
– Отредактируйте реквизиты доступа к базе данных в /app/index.ts, необходимо указать имя пользователя, пароль и название базы данных

3.4 Запуск и работа с приложением

– Запустите системный сервис базы данных;
– Запустите веб-приложение;

– Проверьте работоспособность веб-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.

3.5 Дополнительные рекомендации

– Регулярно проверяйте создание резервных копии базы данных приложения для обеспечения безопасности данных;
– Создайте учетные записи для системных администраторов и других пользователей, которым необходим доступ к приложению;
– Назначьте соответствующие права доступа каждому пользователю в базе данных MariaDB. Ограничьте доступ только к необходимым таблицам и функциям, чтобы обеспечить безопасность данных;
– Регулярно проверяйте журналы доступа и мониторинга для выявления подозрительной активности и потенциальных угроз безопасности.

3.6 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 решени;
– 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.

4 РУКОВОДСТВО ПРОГРАММИСТА

4.1 Назначения и условия применения программы

Приложение разработано как клиент-серверное решение на базе Node.js, обеспечивающее взаимодействие между клиентскими и серверными компонентами. Основные модули и компоненты приложения включают:

- Клиентская часть
  - Реализована с использованием TypeScript и Handlebars
  - Взаимодействие с серверной частью через REST API
- Серверная часть
  - Разработана на платформе NodeJs с использованием TypeScript для повышения читаемости и безопасности кода.
  - Основные модули включают в себя API-маршруты, контроллеры для обработки запросов, а также слой доступа к данным для взаимодействия с базой данных MariaDB.

Используемые технологии и фреймворки:

- NodeJs
  - Используется для создания серверной части приложения и обеспечения его работы в асинхронной среде
  - Позволяет эффективно обрабатывать запросы от клиентов и взаимодействовать с базой данных.
- TypeScript
  - Применяется как основной язык программирования как для клиентской, так и для серверной части приложения
  - Позволяет разработчикам писать более структурированный и поддерживаемый код благодаря статической типизации и другим преимуществам.
- MariaDB
  - Выбрана в качестве реляционной базы данных для хранения и управления данными приложения.

  - Обеспечивает надежное хранение информации, эффективные механизмы обработки запросов и поддержку транзакций.
- Express
  - Используется в качестве веб-фреймворка для Node.js, облегчая создание маршрутов API и обработку HTTP-запросов.
  - Обладает гибкой архитектурой и множеством расширений для разработки разнообразных веб-приложений.

4.2 Характеристики программы

Программное средство работает в режиме приближённом к режиму реального времени для предоставления актуальных данных в любой момент времени. Графический интерфейс пользователя 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.

4.3 Обращение к программе

Для связи графического интерфейса пользователя и бизнес-логики программы используется REST API, что позволяет удобным образом получать и предоставлять пользователю актуальную информацию. Доступ к данным производится через CRUD-операции, взаимодействие с которым производится следующим образом: необходимо указать желаемую операцию и над какими данными её произвести.

4.4 Входные и выходные данные

Входная и выходная информация проверяется программно при создании и изменении пользователей, заключении договором, создании и работе с кредитными картами клиентов. Все данные представлены в кодировке UTF-8.

4.5 Сообщения программисту

- В случае возникновения и проблем или ошибок следует обратиться к документации;
- Все крупные изменения, дополнения, удаления программного кода следует документировать;

5 РУКОВОДСТВО ОПЕРАТОРА

5.1 Назначение программы

Функциональным назначением программы является предоставлению сотрудникам профильных подразделений Заказчика удобного автоматизированного интерфейса посредством веб-браузера для:

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

5.2 Условия выполнения программы

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

- минимальная конфигурация ЦПУ: Intel Core I5 5500K;
- минимальный объем ОЗУ: 16 ГБ;
- минимальный объём дискового пространства: 512 ГБ;
- Linux-подобная операционная система.

Для корректного доступа к функционирующей программе требуется персональный компьютер, включающий в себя:

- ОС: Windows XP или новее, MacOS, Linux-подобная система
- Веб-браузер Firefox, Google Chrome или Opera
- Минимальная конфигурация ПК: соответствует используемой операционной системе и веб-браузеру
- Наличие монитора, клавиатуры и мыши

5.3 Выполнение программы

- Для запуска приложения необходимо запустить скрипт start.sh;
- Приложение позволяет добавлять, удалять, изменять информацию о пользователях, их заключённых договорах, счетах, и кредит
- Для остановки приложения необходимо остановить на сервере выполнение скрипта start.sh
- Для более подробной информации о интерфейсе приложения читайте раздел «интерфейс пользователя»

5.4 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.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

6.1 Использование функций системы

6.1.1 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.

6.1.2 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

6.2 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

6.3 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

6.4 Где найти информацию по предмету, контактная информация

6.4.1 Банкомат

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

6.4.2 Заключение договора

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

7 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

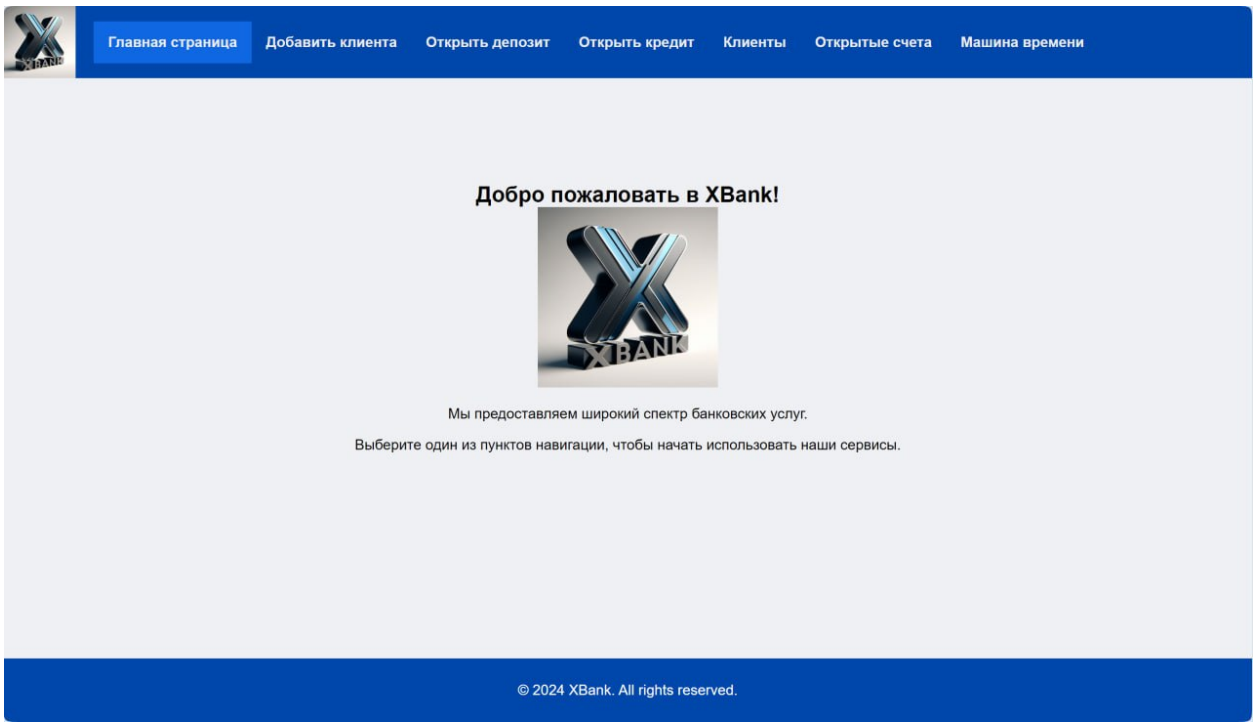

Рисунок 3 — Главная страница программного средства

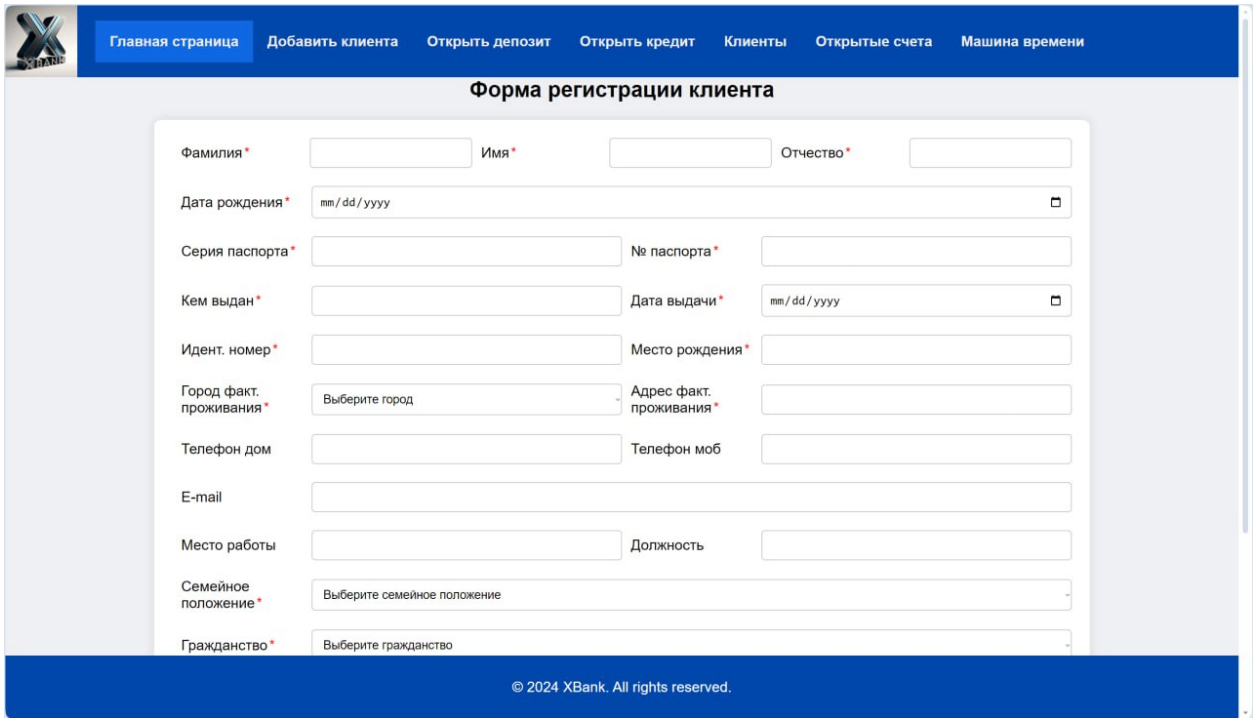

Рисунок 4 — Страница меню добавления нового клиента в систему

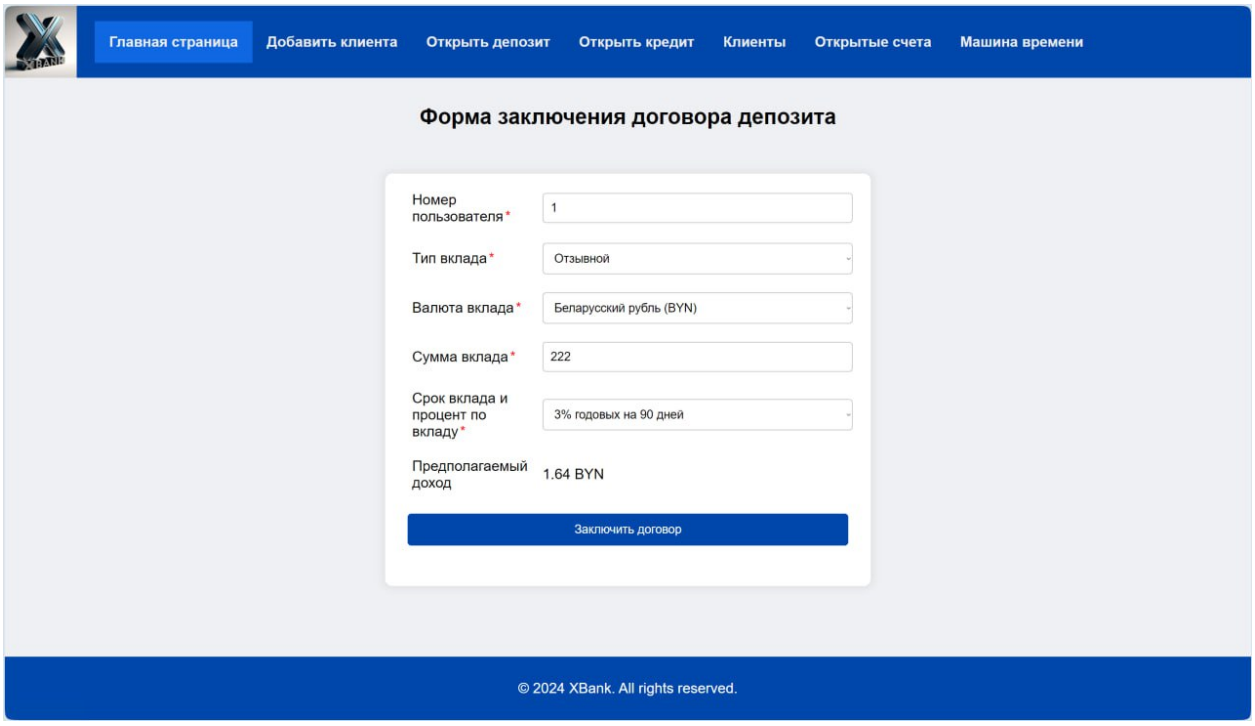

Рисунок 5 — Страница добавления в систему нового депозита

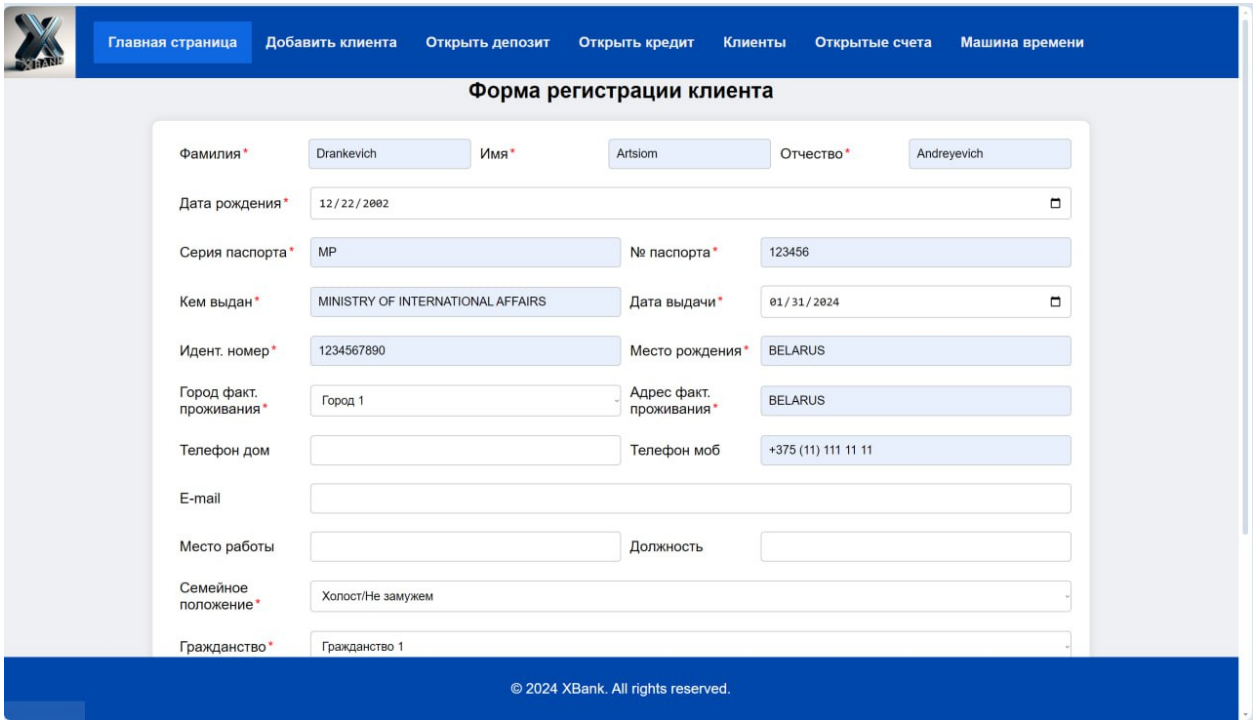

Рисунок 6 — Пример заполненной формы регистрации нового клиента

Рисунок 7 — Пример содержимого страницы ”Клиенты”

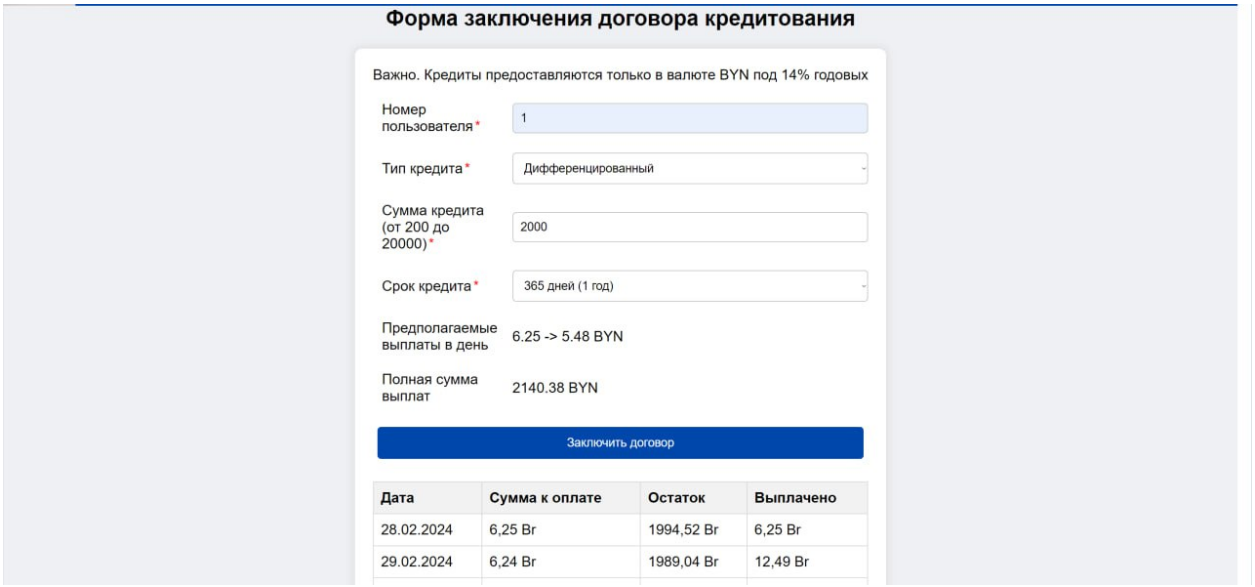

Рисунок 8 — Заполненная форма заключения договора кредитования

Рисунок 9 — Пример содержимого страницы ”Открытые счета”

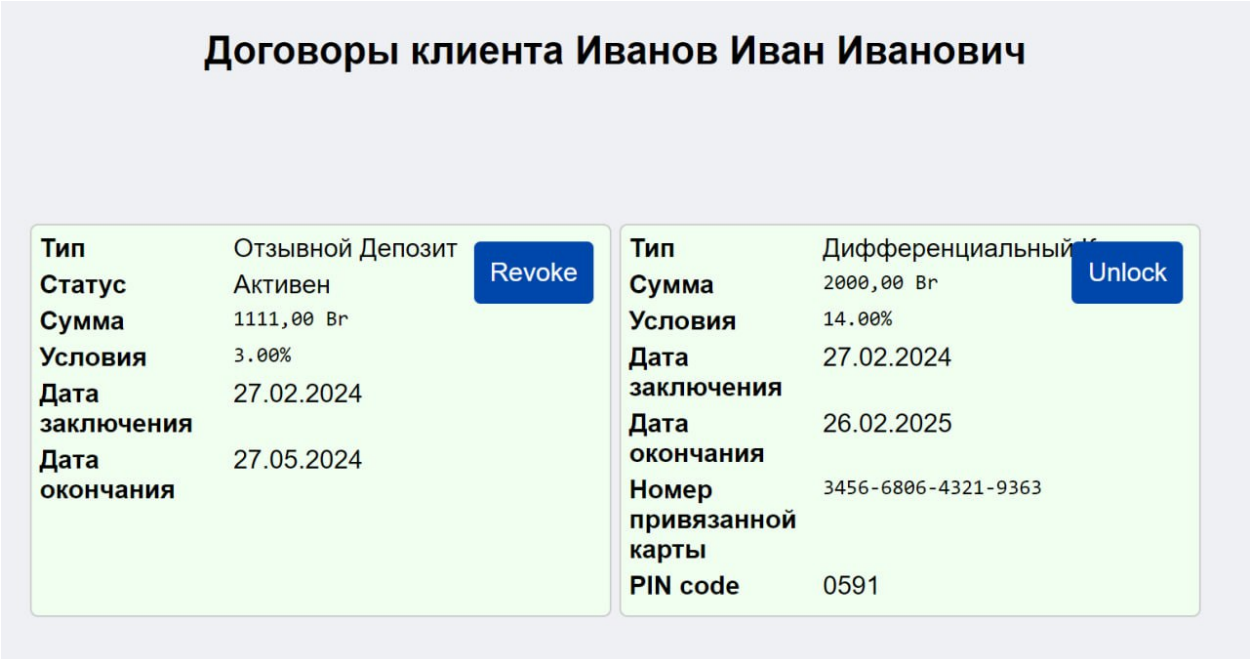

Рисунок 10 — Пример содержимого страницы просмотра договоров клиента, договоры активны

Рисунок 11 — Пример содержимого страницы просмотра договоров клиента, закрытый договор

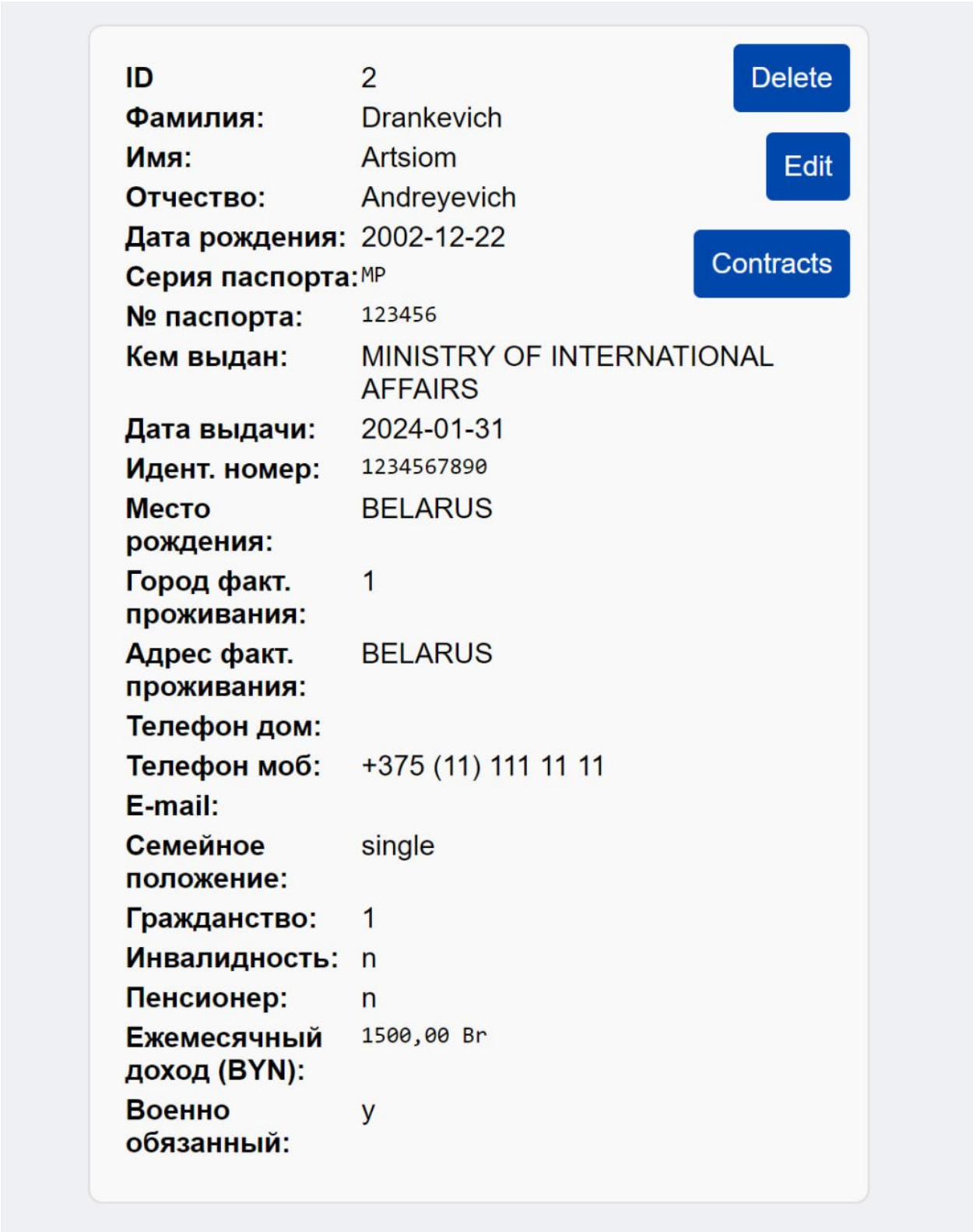

Рисунок 12 — Информация о клиенте после его добавления