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)

ОТЧЁТ

по практической работе №3

на тему

Этапы разработки ИС. Стадия «Реализация»

Выполнил: П. Ю. Рафеев

гр. 951007

Проверила: Г.В. Данилова

Минск 2023

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc126747141)

[1 Листинг программы……………………………………………………………..4](#_Toc126747142)

[2 Интерфейс пользователя………………………………………………………..5](#_Toc126747145)

[2.1 Главная страница……………………………………………………………5](#_Toc126747146)

[2.2 Страница ввода и редактирования персональных данных………………..5](#_Toc126747147)

[2.3 Страница вариантов депозитных или кредитных договоров 6](#_Toc126747148)

2.4 Страница описания договора для депозита или кредита ………...…........6

2.5 Страница заключения договора для кредита или депозита………….......7

2.6 Страница с информацией по счетам…………..…………………………...7

[3 Руководство пользователя 9](#_Toc126747149)

[4 Результат работы программы 12](#_Toc126747150)

Приложение А. Код программы………………………………………………...13

# **ВВЕДЕНИЕ**

Автоматизированная банковская система (сокр. АБС) – комплекс программного и технического обеспечения, направленный на автоматизацию банковской деятельности.

Американская исследовательская и консалтинговая компания Gartner определяет основную банковскую систему как внутреннюю автоматизированную систему, которая обрабатывает ежедневные банковские операции и производит обновление счетов и других финансовых записей. Основные банковские системы обычно включают в себя возможности:

* открытия и ведения банковских счетов физических и (или) юридических лиц;
* обработки депозитов, займов и кредитов;
* подключения к стандартным системам бухгалтерского учета и инструментам отчетности.

Целью данной практической работы является реализация программного средства для автоматизации деятельности условного коммерческого учреждения (банка).

# **ЛИСТИНГ ПРОГРАММЫ**

Код состоит из 2 основных частей: клиентской и серверной.

Клиентская часть включает в себя страницы (папка Pages), компоненты (папка Components), вспомогательные элементы (папки Elements и Helpers), запросы к серверу (файл agent.js) и главный компонент (файл App.js).

Страницы представлены следующими файлами:

* Страница вариантов договоров ContractOptionsPage.jsx;
* Страница заключенных договоров ContractsPage.jsx;
* Страница создания пользователя CreateUserPage.jsx;
* Страница входа LoginPage.jsx;
* Страница заключения договора SignContractPage.jsx;
* Главная страница UserMainPage.jsx;
* Страница статистики по счетам TransactionsPage.jsx;
* Страница со списком пользователей UsersListPage.jsx.

Серверная часть включает в себя файл запуска (Program.cs), файл контекста базы данных (ApplicationDbContext.cs), контроллеры (папка Controllers), модели базы данных (папка Models), модели для взаимодействия с клиентской частью (папка ViewModels), сервисы (папка Services), файлы предварительной проверки запроса (папка Middleware), статические данные (папка Enums) и миграции базы данных (папка Migrations).

Контроллеры представлены следующими файлами:

* Контроллер авторизации AuthController.cs;
* Контроллер пользователей UsersController.cs;
* Контроллер дебиторских договоров DebetController.cs;
* Контроллер кредиторских договоров CreditController.cs;
* Контроллер неизменяемых данных StaticDataController.cs;
* Контроллер транзакций TransactionsController.cs;
* Контроллер банкомата BankomatController.cs.

# **ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ**

## **2.1 Главная страница**

На главной странице находится навигационное меню, позволяющее перейти на основные разделы сайта, а также введенные пользователем персональные данные и кнопки для их изменения и удаления. Пример главной страницы представлен на рисунке 2.1:

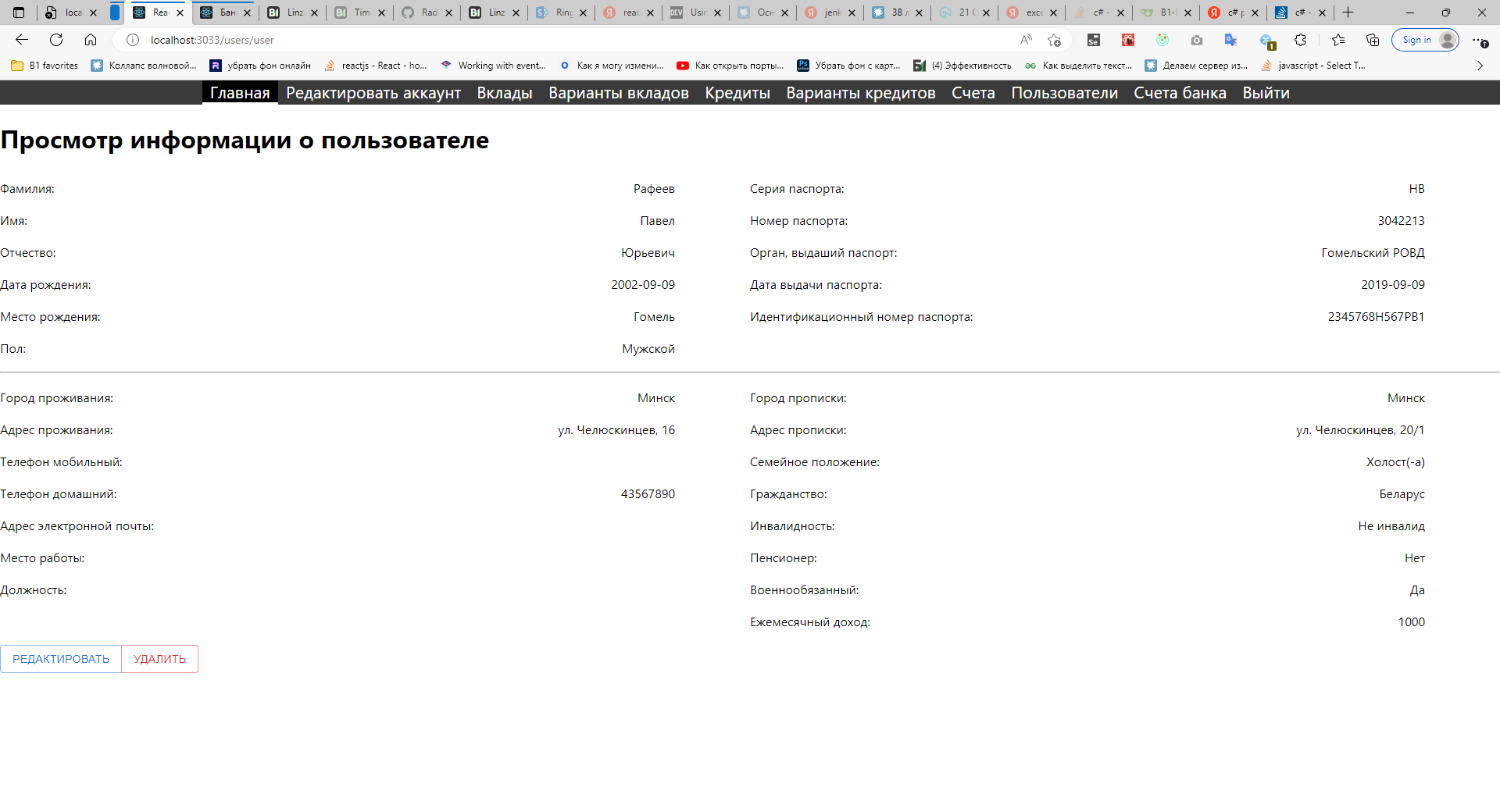


Рисунок 2.1 – Пример главной страницы

Навигационное меню обеспечивает навигацию на следующие страницы:

− главная страница;

− страница редактирования персональных данных;

− страница заключенных депозитных договоров;

− страница вариантов депозитных договоров;

− страница заключенных кредитных договоров;

− страница вариантов кредитных договоров;

− страница с информацией по счетам пользователя;

− страница со списком пользователей (доступна только сотрудникам банка);

− страница с информацией по банковским счетам (доступна только сотрудникам банка);

− страница авторизации в системе.

* 1. **Страница ввода и редактирования персональных данных**

На странице ввода и редактирования персональных данных находятся текстовые поля для ввода персональных данных, а также кнопка подтверждения ввода, позволяющая создать аккаунт нового пользователя либо изменить данные действующего. На каждое поле есть проверка в виде регулярного выражения, а также есть списки с выбором ответа. Пример данной страницы представлен на рисунке 2.2:

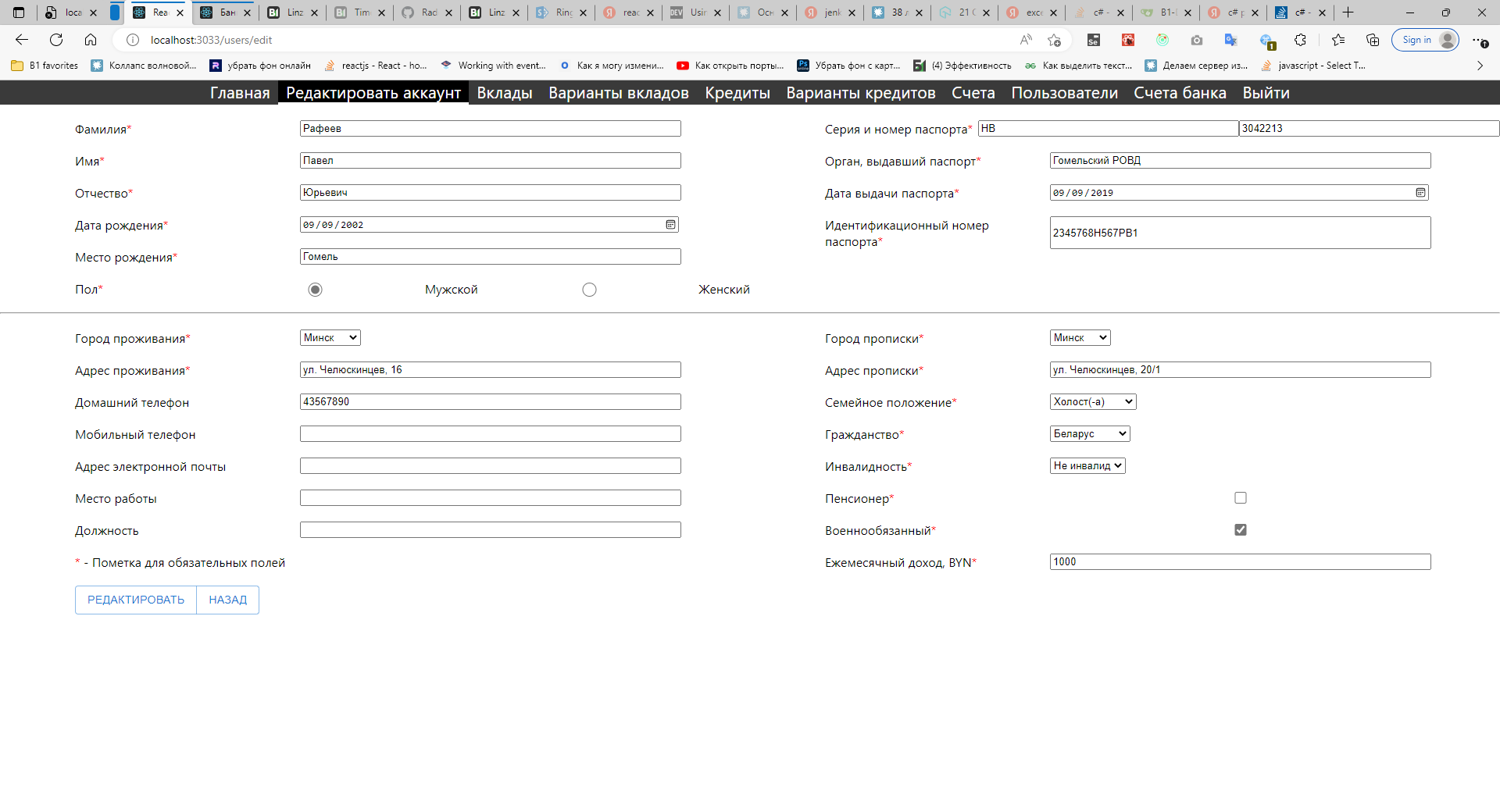


Рисунок 2.2 – Пример страницы ввода и редактирования персональных данных

## **Страница вариантов депозитных или кредитных договоров**

На странице вариантов депозитных или кредитных договоров располагается таблица со списком договоров, доступных к заключению. На рисунке 2.3 представлен пример данной страницы:

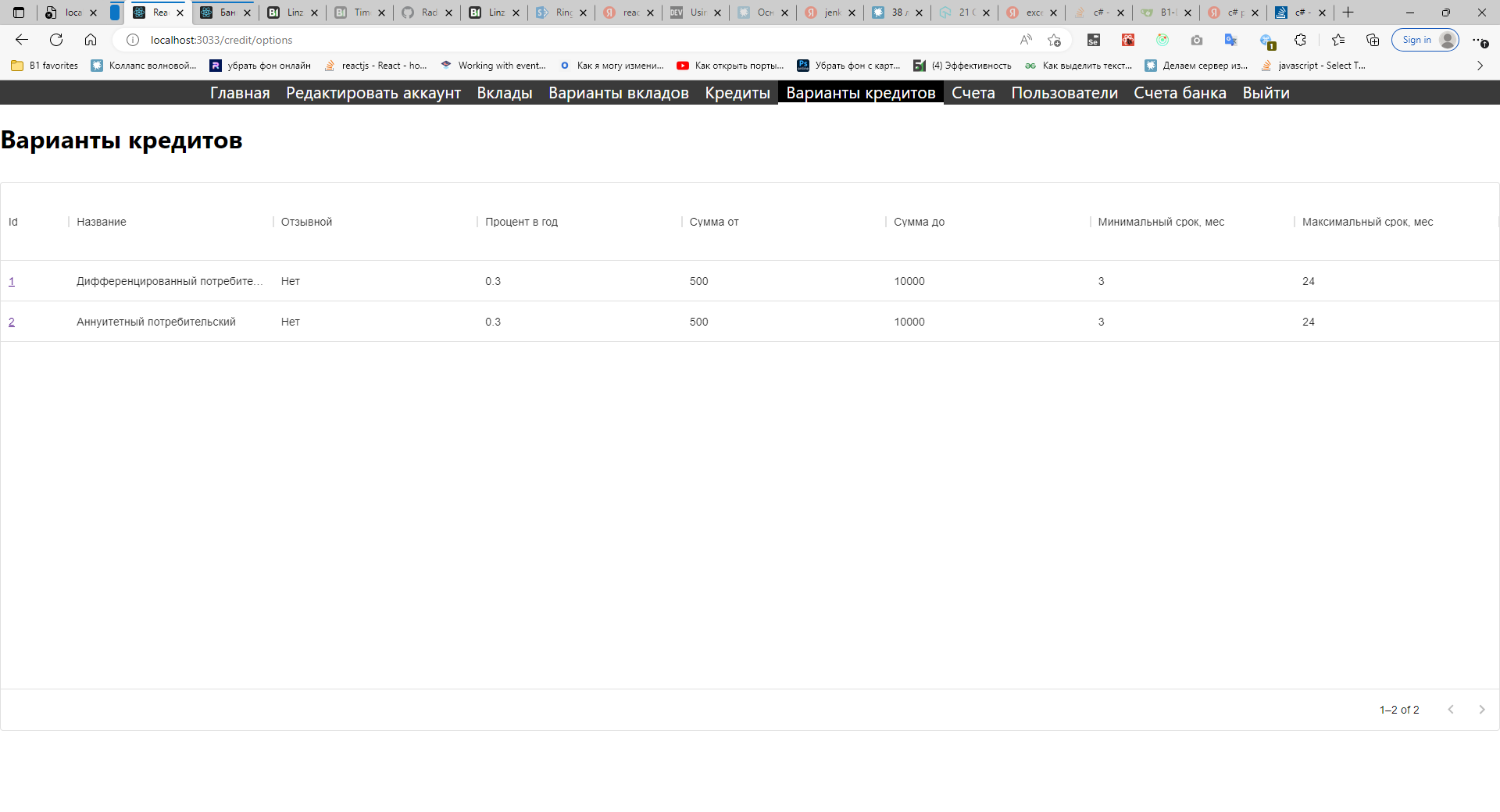


Рисунок 2.3 – Пример страницы вариантов кредитных договоров

* 1. **Страница описания договора для депозита или кредита**

На странице описания договора отображается основная информация по договору, включающая название, описание, диапазон допустимых сумм, диапазон допустимых периодов заключения и процент в год. Также внизу страницы имеются кнопки перехода на страницу заключения договора и возврата на предыдущую страницу. На рисунке 2.4 представлен пример данной страницы:

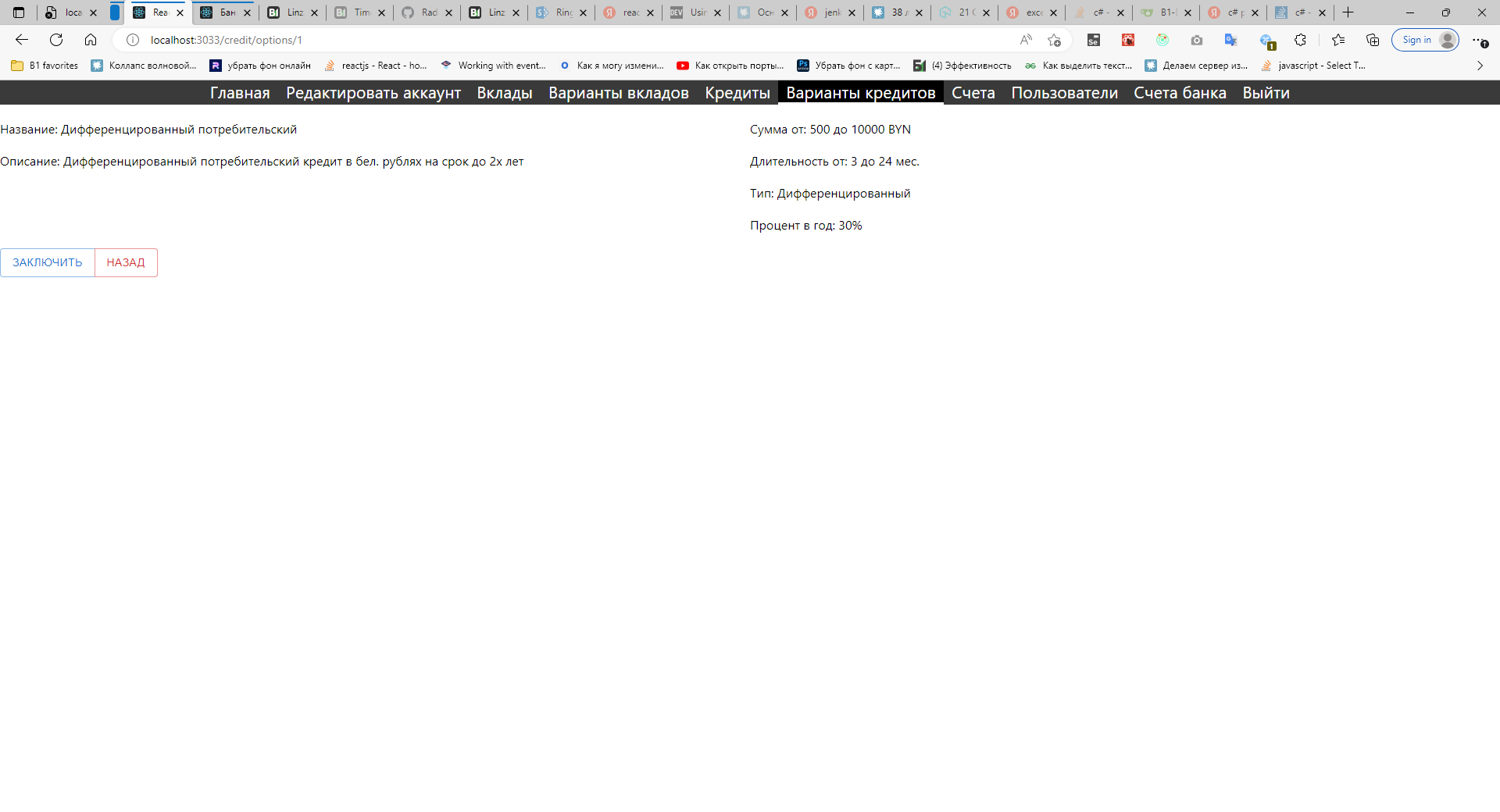


Рисунок 2.4 – Пример страницы описания договора

**2.5** **Страница заключения договора для кредита или депозита**

На странице заключения договора отображается форма с текстом договора и полями для ввода суммы договора, периода заключения и валюты (если договор допускает несколько вариантов). Внизу страницы расположены кнопки заключения договора и возврата на предыдущую страницу. На рисунке 2.5 представлен пример данного окна:

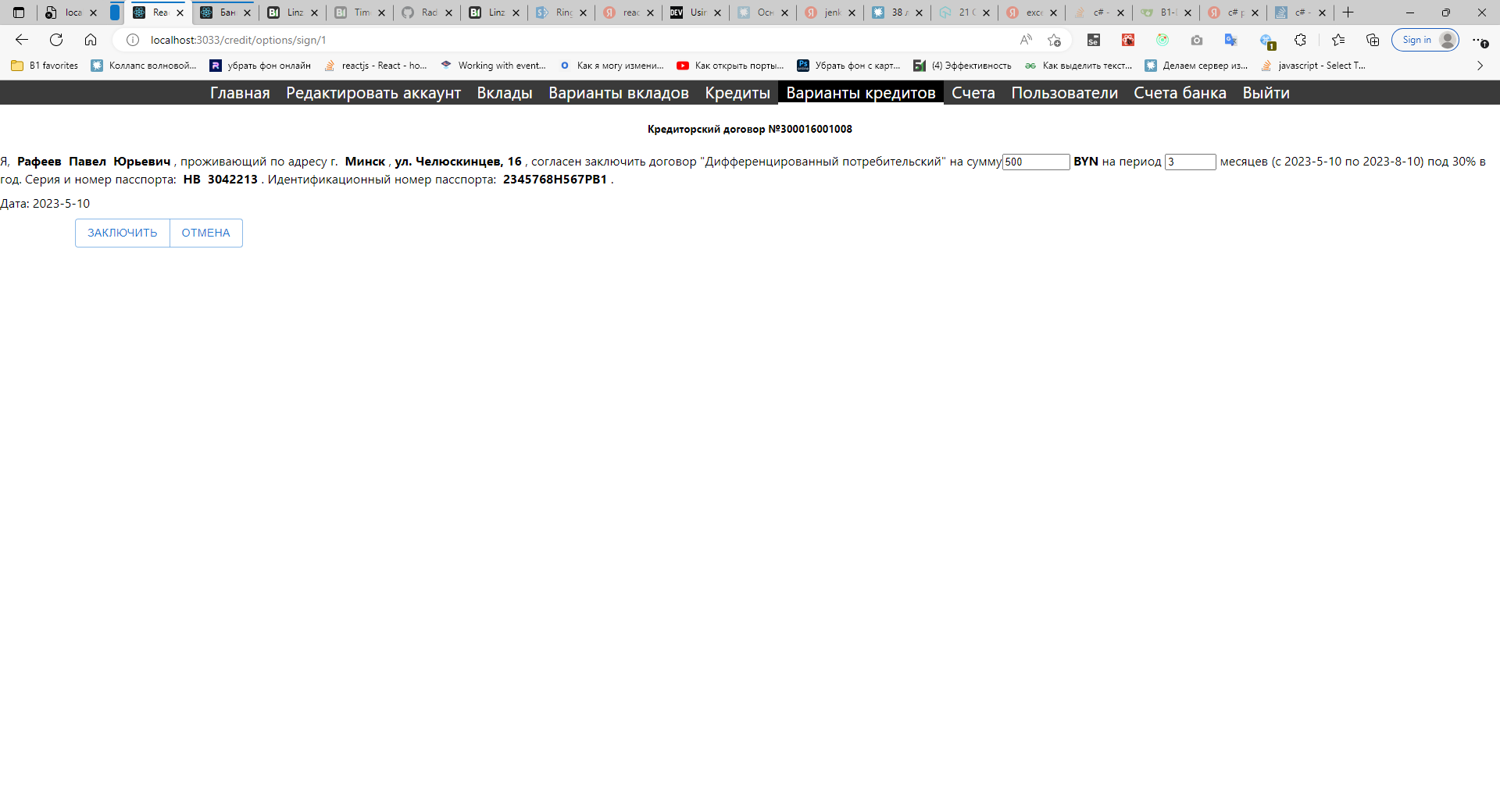


Рисунок 2.5 – Пример страницы заключения договора

**2.6 Страница с информацией по счетам**

На странице с информацией по счетам есть располагаются таблицы транзакций по каждому из счетов пользователя (банка) и график зависимости баланса на счетах от времени. Снизу есть зафиксированная полоска с кнопками закрытия банковского дня, закрытия месяца и текущей системной датой. Пример страницы с информацией по счетам представлен на рисунке 2.6:

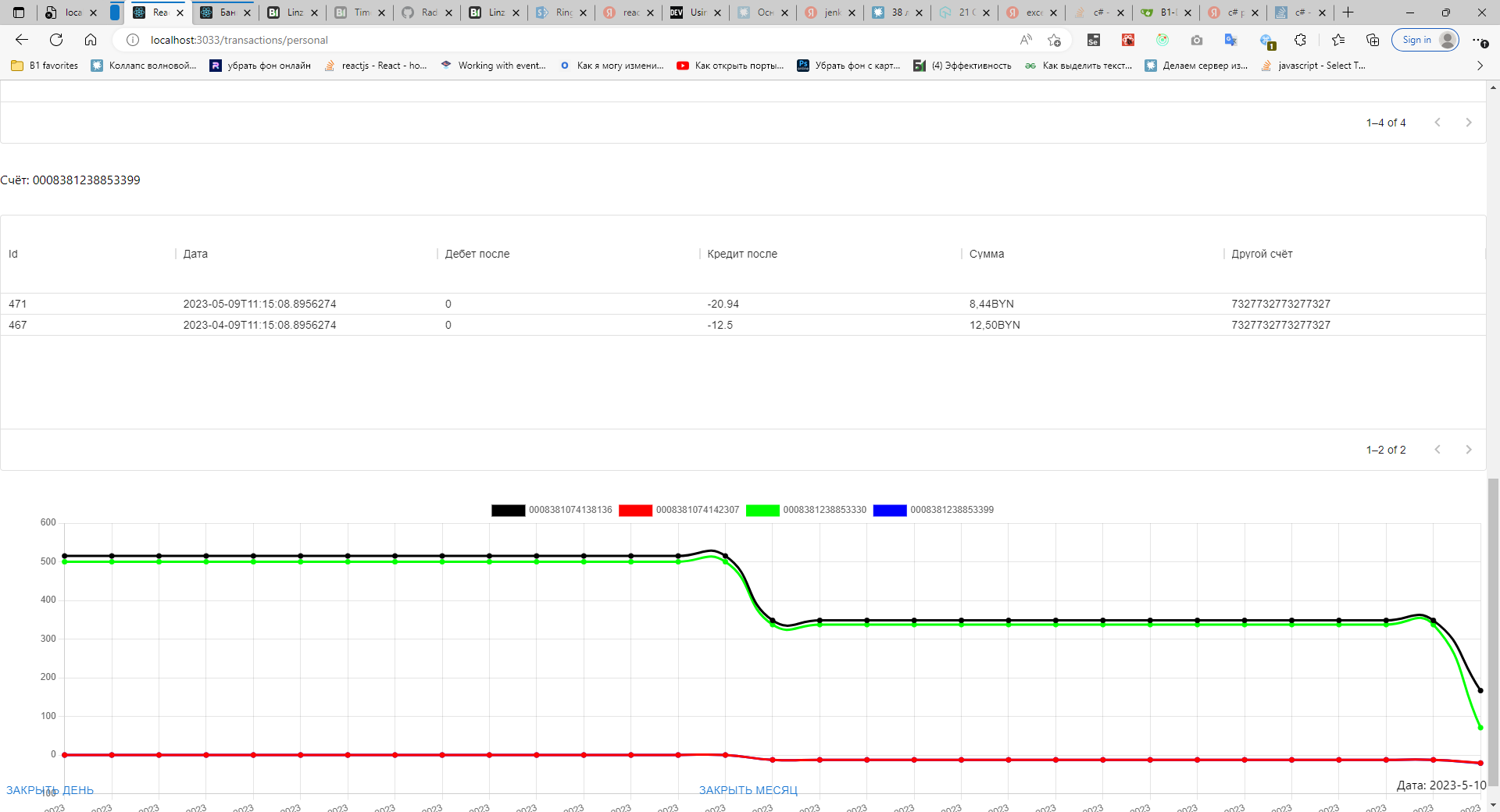


Рисунок 2.6 – Пример страницы с информацией по счетам

# **3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

При открытии приложения пользователь видит перед собой страницу входа, представленную на рисунке 3.1:

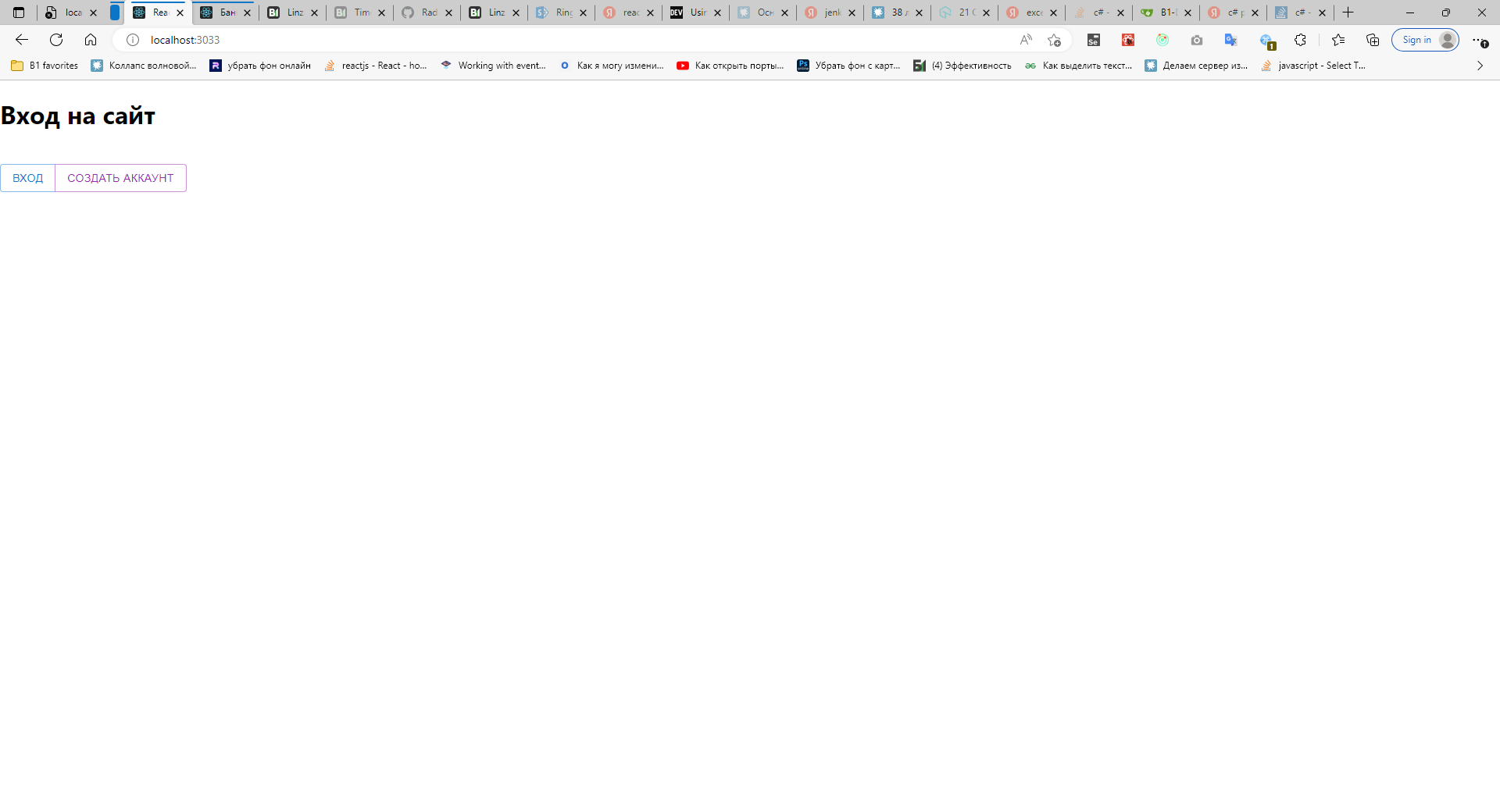


Рисунок 3.1 – Страница входа

Для начала необходимо создать аккаунт, нажав на соответствующую кнопку и введя персональные данные. На рисунке 3.2 представлена страница создания аккаунта пользователя:

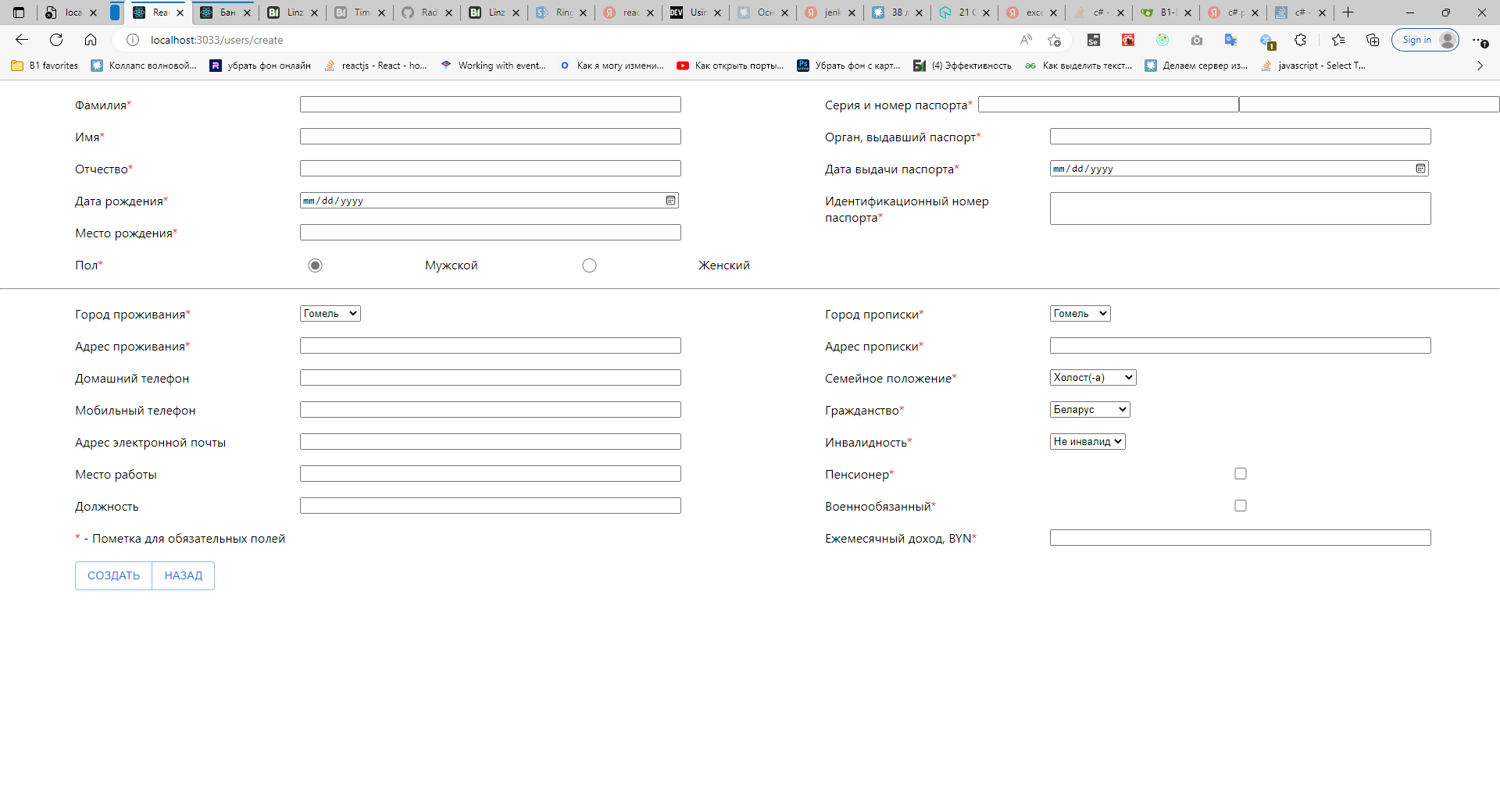


Рисунок 3.2 – Страница создания аккаунта пользователя

В случае корректного заполнения всех требуемых полей, в системе будет создан аккаунта, а пользователь увидит соответствующее сообщение. После этого можно будет вернуться на страницу входа и, по нажатию на соответствующую кнопку, войти в систему и увидеть главную страницу, представленную на рисунке 3.3:

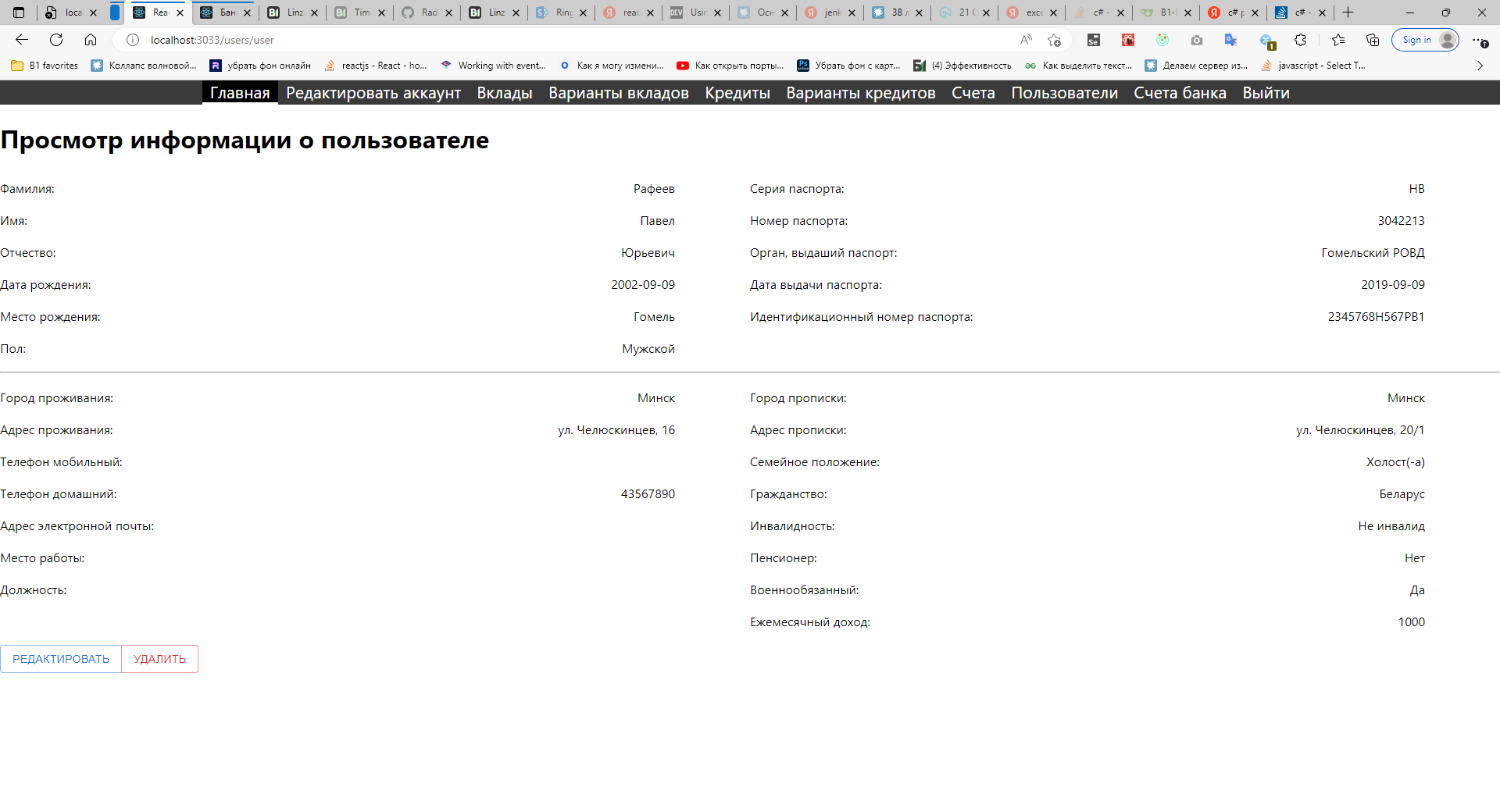


Рисунок 3.3 – Главная страница

Затем можно перейти на страницу со списком вариантов кредитов или депозитов, представленную на рисунке 3.4:

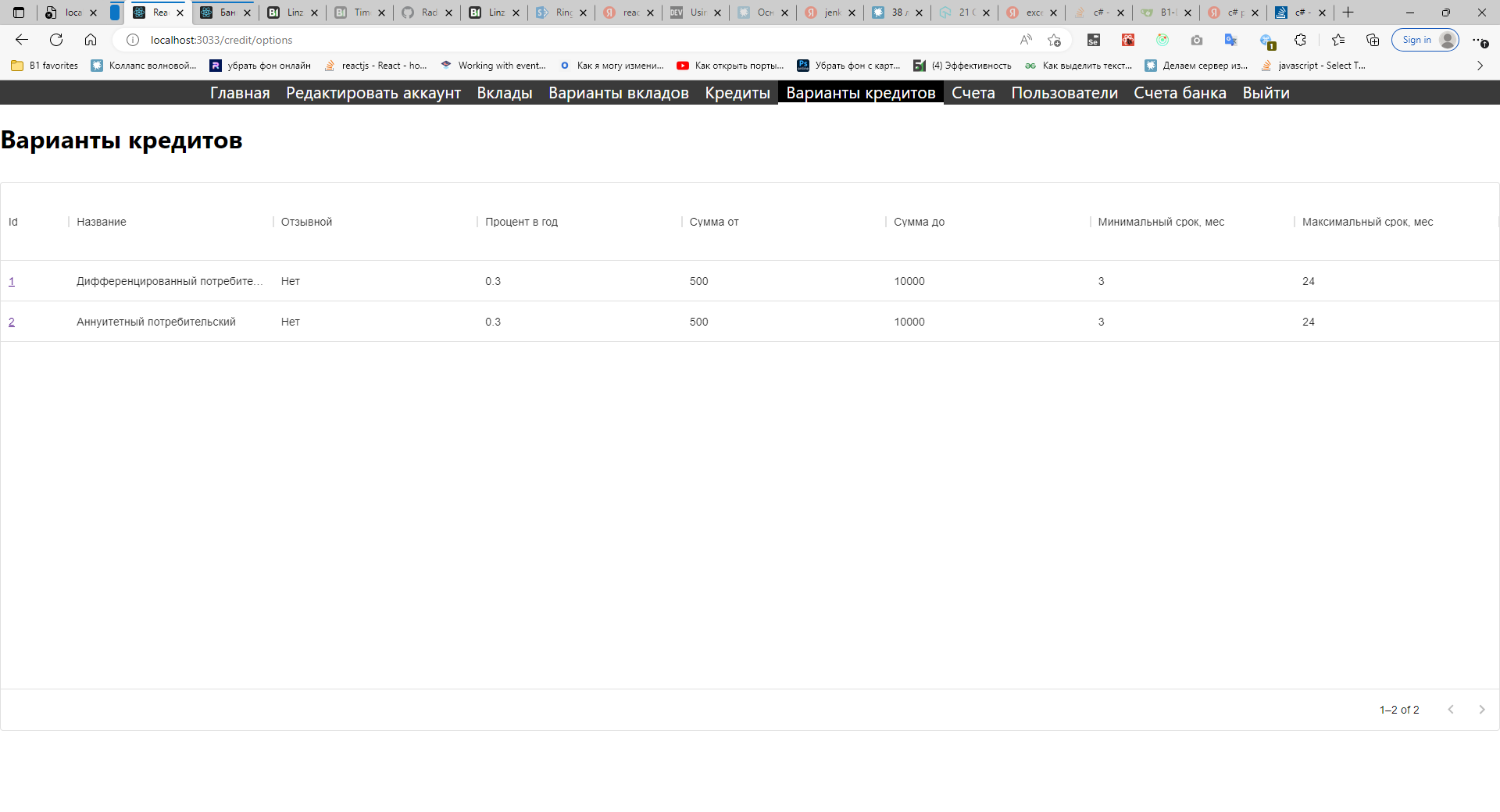


Рисунок 3.4 – Страница со списком вариантов кредитов

Для перехода на страницу описания интересующего кредита или депозита достаточно нажать на его Id в списке. На рисунке 3.5 показана страница с описанием договора:

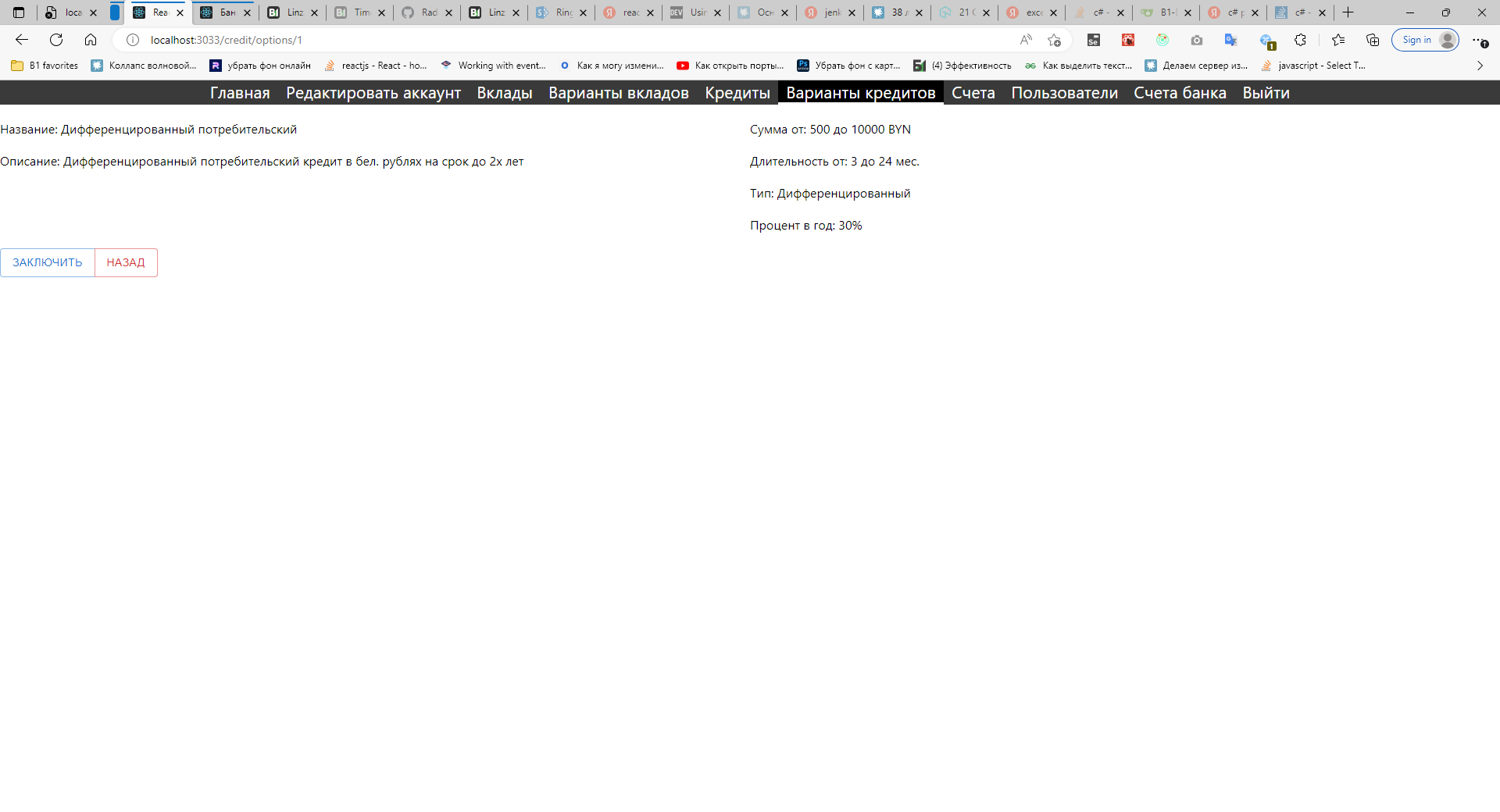


Рисунок 3.5 – Страница описания кредитного договора

Для заключения договора необходимо нажать кнопку «Заключить» и ввести необходимые данные. На рисунке 3.6 показана страница заключения договора:

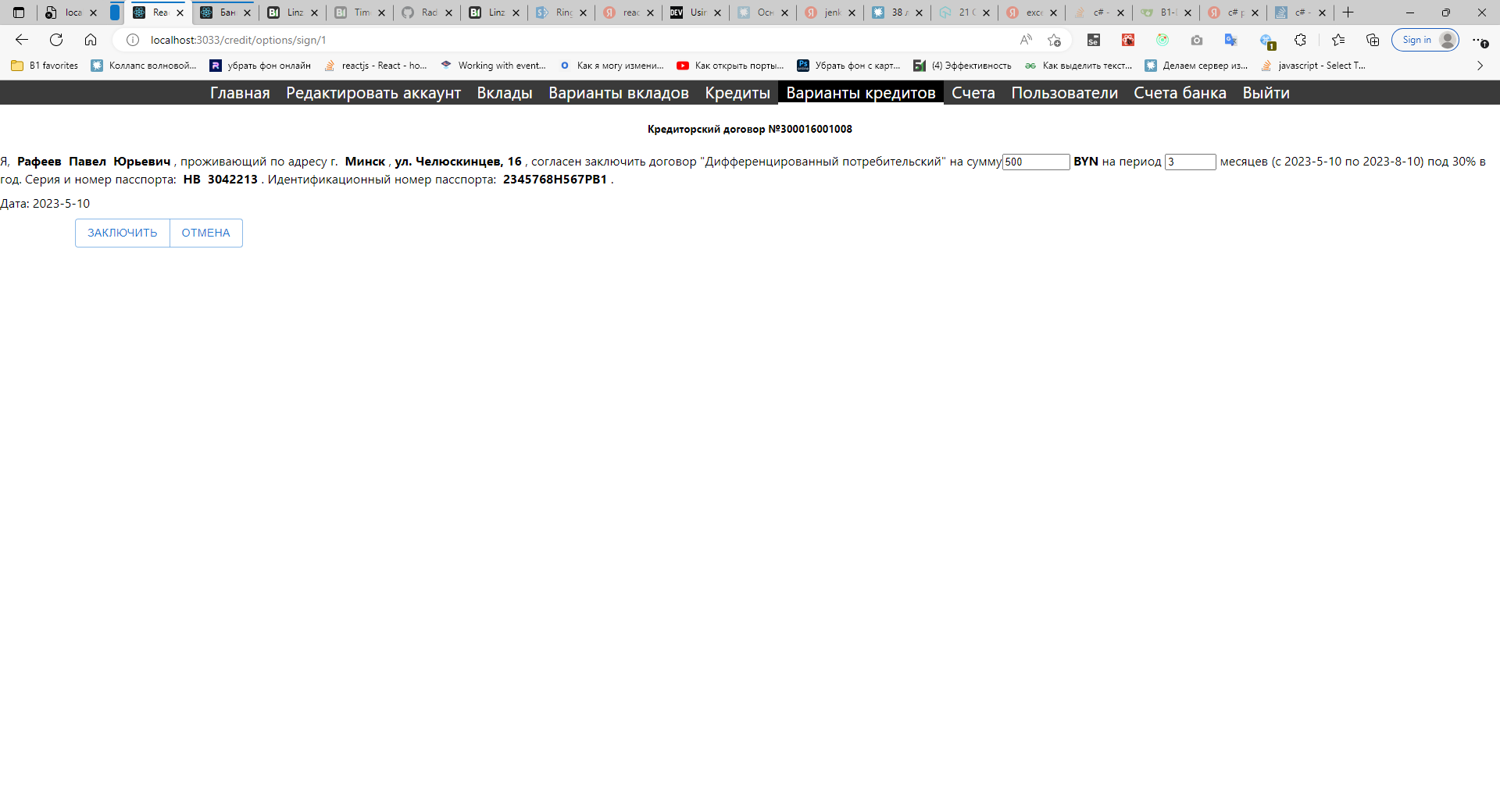


Рисунок 2.5 – Страница заключения договора

После заключения договора создаются 2 счета: основной и процентный. Информация по ним доступна в разделе «Счета» и включает в себя таблицы транзакций по каждому счету и график зависимости суммы на счете от веремени.

# **4 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

На рисунке 4.1 представлен результат симуляции с одним из клиентов. У пользователя есть 2 кредита: дифференцированный и аннуитетный, заключенные на 3 месяца на сумму в 500 BYN под 30% годовых. По каждому из них прошло 2 месяца со дня заключения:



Рисунок 4.1 – Страница симуляции с 2 кредитами

**ПРИЛОЖЕНИЕ А.**

**Код программы**

файл **program.cs**

using Microsoft.AspNetCore.Identity;

using Microsoft.Extensions.Configuration;

using PiRiS\_back.Models;

using PiRiS\_back;

using PiRiS\_back.Middleware;

using Microsoft.EntityFrameworkCore;

using PiRiS\_back.Services;

using PiRiS\_back.Enums;

using Microsoft.AspNetCore.Authentication.Negotiate;

using Microsoft.AspNetCore.Server.IISIntegration;

var builder = WebApplication.CreateBuilder(args);

var services = builder.Services;

var configuration = builder.Configuration;

services.Configure<BankAccConfig>(configuration.GetSection("Accounts"));

services.AddTransient<UserInfoFillerService>();

services.AddTransient<AccountsService>();

services.AddSingleton<ContractsServiceSingletone>();

services.AddScoped<IdentityNameFilter>();

services.AddScoped<AuthFilter>();

services.AddControllers();

services.AddSession();

services.AddEndpointsApiExplorer();

services.AddSwaggerGen();

services.AddCors(options =>

{

options.AddPolicy("AllowAllHeaders",

builder =>

{

builder.WithOrigins("http://localhost:3033")

.AllowCredentials()

.AllowAnyHeader()

.AllowAnyMethod();

});

});

services.AddAuthentication(NegotiateDefaults.AuthenticationScheme)

.AddNegotiate();

services.AddAuthorization(options =>

{

// By default, all incoming requests will be authorized according to the default policy.

options.FallbackPolicy = options.DefaultPolicy;

});

var accountsConfiguration = configuration.GetRequiredSection("Accounts");

services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(

configuration.GetConnectionString("DefaultConnection")));

services.AddDatabaseDeveloperPageExceptionFilter();

services.AddDefaultIdentity<User>(options => options.SignIn.RequireConfirmedAccount = true).AddRoles<IdentityRole<int>>()

.AddEntityFrameworkStores<ApplicationDbContext>().AddDefaultTokenProviders();

var app = builder.Build();

//app.UseCors("AllowAllHeaders");

//app.UseCors(policy => policy.AllowCredentials().AllowAnyHeader().AllowAnyMethod().WithOrigins("http://localhost:3033"));

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseAuthentication();

app.UseAuthorization();

app.UseSession();

app.MapControllers();

app.Run();

файл **usersController.cs**

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using PiRiS\_back.Enums;

using PiRiS\_back.Middleware;

using PiRiS\_back.Models;

using PiRiS\_back.Services;

using PiRiS\_back.ViewModels;

using static PiRiS\_back.Services.UserInfoFillerService;

namespace PiRiS\_back.Controllers

{

[ApiController]

[Route("/api/[controller]")]

public class UsersController : ControllerBase

{

ApplicationDbContext \_context;

UserInfoFillerService \_userInfoFillerService;

public UsersController(ApplicationDbContext context, UserInfoFillerService userInfoFillerService)

{

\_context = context;

\_userInfoFillerService = userInfoFillerService;

}

[HttpGet]

[AuthFilter] //Admin only

public async Task<IActionResult> GetUsersList()

{

return new OkObjectResult(\_context.Users.ToList());

}

[HttpGet("user")]

[AuthFilter]

public async Task<IActionResult> GetCurrentUserInfo()

{

var userName = HttpContext.User.Identity.Name;

return new OkObjectResult(new UserViewModel(await \_context.Users.FirstAsync(u => u.UserName == userName), \_context));

}

[HttpGet("user/{id}")]

[AuthFilter] //Admin only

public async Task<IActionResult> GetUserInfo(int? id)

{

var res = await \_context.Users.FirstOrDefaultAsync(u => u.Id == id);

if (res == null)

{

return new BadRequestObjectResult($"Invalid id: {id}. User not found!");

}

return new OkObjectResult(new UserViewModel(res, \_context));

}

[HttpPost("create")]

[IdentityNameFilter]

public async Task<object> CreateUser(UserViewModel userViewModel)

{

try

{

User newUser = new();

\_userInfoFillerService.FillUserInfo(userViewModel, \_context, newUser);

var currUserName = HttpContext.User.Identity?.Name;

if (!(currUserName is not null && \_context.Users.Any(u => u.UserName == currUserName)) && HttpContext.Session.GetString(currUserName ?? "") != Constants.AUTHORIZED) //if user not authorized then it's registration

{

newUser.UserName = currUserName;

}

await \_context.AddAsync(newUser);

await \_context.SaveChangesAsync();

HttpContext.Response.StatusCode = 201;

return newUser;

}

catch (UserValidationException ex)

{

return new BadRequestObjectResult(ex.Message);

}

}

[HttpPut("edit")]

[AuthFilter]

public async Task<IActionResult> EditCurrentUser([FromForm] UserViewModel userViewModel)

{

try

{

var currUser = \_context.Users.First(u => u.UserName == HttpContext.User.Identity.Name);

\_userInfoFillerService.FillUserInfo(userViewModel, \_context, currUser, false);

\_context.Update(currUser);

await \_context.SaveChangesAsync();

return new OkObjectResult(currUser);

}

catch (UserValidationException ex)

{

return new BadRequestObjectResult(ex.Message);

}

}

[HttpPut("edit/{id}")]

[AuthFilter] //Admin only

public async Task<IActionResult> EditUser(int? id, [FromForm] UserViewModel userViewModel)

{

try

{

var editedUser = await \_context.Users.FirstOrDefaultAsync(u => u.Id == id);

if (editedUser == null)

{

return new BadRequestObjectResult($"Invalid id: {id}. User not found!");

}

\_userInfoFillerService.FillUserInfo(userViewModel, \_context, editedUser, false);

\_context.Update(editedUser);

await \_context.SaveChangesAsync();

return new OkObjectResult(editedUser);

}

catch (UserValidationException ex)

{

return new BadRequestObjectResult(ex.Message);

}

}

[HttpDelete("delete")]

[AuthFilter]

public async Task<IActionResult> DeleteCurrentUser()

{

var currUser = await \_context.Users.FirstAsync(u => u.UserName == HttpContext.User.Identity.Name);

HttpContext.Session.Remove(currUser.UserName ?? ""); //if user tries to delete himself then he will be logged out

\_context.Remove(currUser);

await \_context.SaveChangesAsync();

return new NoContentResult();

}

[HttpDelete("delete/{id}")]

[AuthFilter] //Admin only

public async Task<IActionResult> DeleteUser(int? id)

{

var userToDelete = await \_context.Users.FirstOrDefaultAsync(u => u.Id == id);

if (userToDelete == null)

{

return new NoContentResult();

}

var currUserName = HttpContext.User.Identity?.Name;

if (userToDelete.UserName == currUserName) //if user tries to delete himself then he will be logged out

{

HttpContext.Session.Remove(currUserName ?? "");

}

\_context.Remove(userToDelete);

await \_context.SaveChangesAsync();

return new NoContentResult();

}

}

}

файл **contractsServiceSingletone.cs**

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Storage.ValueConversion.Internal;

using Microsoft.Extensions.Options;

using Microsoft.Identity.Client;

using PiRiS\_back.Enums;

using PiRiS\_back.Models;

using PiRiS\_back.ViewModels;

using System.ComponentModel.DataAnnotations;

using static PiRiS\_back.Services.UserInfoFillerService;

using System.Text.RegularExpressions;

using System.Diagnostics.Contracts;

using System.Security.Principal;

namespace PiRiS\_back.Services

{

public class ContractsServiceSingletone

{

private AccountsService \_accountsService;

public DateTime AppDateTime { get; set; }

readonly IOptions<BankAccConfig> \_config;

public ContractsServiceSingletone(AccountsService accountsService, IOptions<BankAccConfig> config)

{

AppDateTime = DateTime.Now;

\_accountsService = accountsService;

\_config = config;

}

public async Task<List<AccountStatesViewModel>> GetUserTransactionsAsync(string userName, ApplicationDbContext context)

{

var user = await context.Users.FirstAsync(u => u.UserName == userName);

try

{

return context.Accounts.Where(acc => acc.UserId == user.Id).Select(acc => new AccountStatesViewModel()

{

Account = acc,

TransactionList = context.Transactions.Include(tr => tr.Currency).Where(tr => tr.NumberFrom == acc.Number || tr.NumberTo == acc.Number)

.OrderByDescending(tr => tr.Id).Select(tr => new TransactionViewModel(tr, acc, tr.Currency)).ToList(),

}

).ToList();

} catch (NullReferenceException)

{

throw new TransactionValidationException($"No transactions found for user: {userName}!");

}

}

public async Task<List<AccountStatesViewModel>> GetBankTransactionsAsync(ApplicationDbContext context)

{

try

{

return await context.Accounts.Where(acc => acc.Code == \_config.Value.BankAccountPassive || acc.Code == \_config.Value.BankAccountActive)

.Select(acc => new AccountStatesViewModel()

{

Account = acc,

TransactionList = context.Transactions.Include(tr => tr.Currency).Where(tr => tr.NumberFrom == acc.Number || tr.NumberTo == acc.Number)

.OrderByDescending(tr => tr.Id).Select(tr => new TransactionViewModel(tr, acc, tr.Currency)).ToList(),

}).ToListAsync();

}

catch (NullReferenceException)

{

throw new TransactionValidationException($"No transactions found for bank!");

}

}

public async Task SignDebetContractAsync(ContractViewModel contract, string userName, ApplicationDbContext context, AccountsService accountsService)

{

await validateContractAsync(contract, context);

var user = await context.Users.FirstAsync(u => u.UserName == userName);

var bankAcountActive = await context.Accounts.FirstAsync(ac => ac.Code == \_config.Value.BankAccountActive);

var bankAccountPassive = await context.Accounts.FirstAsync(ac => ac.Code == \_config.Value.BankAccountPassive);

var contractCurrency = await context.Currencies.FirstOrDefaultAsync(cur => cur.Name == contract.Currency);

if (contractCurrency is null) throw new ContractValidationException($"Неизвестная валюта: {contract.Currency}!");

var code = await generateAccountCodeNumberAsync(user.Id, true, context);

var newAccountForMainSum = new Account()

{

UserId = user.Id,

Code = code.ToString(),

Pin = "0000",

Number = generateAccountNumber(user.Id),

Debet = 0,

Credit = 0,

Currency = contractCurrency,

IsActive = false

};

var newAccountForPercents = new Account()

{

UserId = user.Id,

Code = (code+1).ToString(),

Pin = "0000",

Number = generateAccountNumber(user.Id),

Debet = 0,

Credit = 0,

Currency = contractCurrency,

IsActive = false

};

await accountsService.CreateTransactionAsync("", false, newAccountForMainSum.Number, true, contract.Sum, contractCurrency, context, AppDateTime); //To cassa debet

await accountsService.CreateTransactionAsync(bankAcountActive, false, newAccountForMainSum, false, contract.Sum, contractCurrency, context, AppDateTime); //From cassa credit to main credit

if(contract.IsRequestable == false)

await accountsService.CreateTransactionAsync(newAccountForMainSum, true, bankAccountPassive, false, contract.Sum, contractCurrency, context, AppDateTime); //From main debet to bank credit

var newContract = new DebetContract()

{

DebetContractOptionId = contract.OptionId ?? -1,

Number = contract.Number,

Account1 = newAccountForMainSum,

Account2 = newAccountForPercents,

PersonSurname = contract.PersonSurname,

PersonName = contract.PersonName,

PersonMiddlename = contract.PersonMiddlename,

PassportSeria = contract.PassportSeria,

PassportNumber = contract.PassportNumber,

PassportIdentityNumber = contract.PassportIdentityNumber,

CityLivingId = context.Cities.First(cit => cit.Name == contract.CityLiving).Id,

AddressLiving = contract.AddressLiving,

Sum = contract.Sum,

CurrencyId = context.Currencies.First(cur => cur.Name == contract.Currency).Id,

PercentPerYear = contract.PercentPerYear,

StartDate = AppDateTime,

EndDate = AppDateTime.AddMonths(contract.DurationMonth)

};

await context.Accounts.AddAsync(newAccountForMainSum);

await context.Accounts.AddAsync(newAccountForPercents);

await context.DebetContracts.AddAsync(newContract);

await context.SaveChangesAsync();

}

public async Task SignCreditContractAsync(ContractViewModel contract, string userName, ApplicationDbContext context, AccountsService accountsService)

{

await validateContractAsync(contract, context);

var user = await context.Users.FirstAsync(u => u.UserName == userName);

var bankAcountActive = await context.Accounts.FirstAsync(ac => ac.Code == \_config.Value.BankAccountActive);

var bankAccountPassive = await context.Accounts.FirstAsync(ac => ac.Code == \_config.Value.BankAccountPassive);

var contractCurrency = await context.Currencies.FirstOrDefaultAsync(cur => cur.Name == contract.Currency);

if (contractCurrency is null) throw new ContractValidationException($"Неизвестная валюта: {contract.Currency}!");

var code = await generateAccountCodeNumberAsync(user.Id, false, context);

var newAccountForMainSum = new Account()

{

UserId = user.Id,

Code = code.ToString(),

Pin = "0000",

Number = generateAccountNumber(user.Id),

Debet = 0,

Credit = 0,

Currency = contractCurrency,

IsActive = true

};

var newAccountForPercents = new Account()

{

UserId = user.Id,

Number = generateAccountNumber(user.Id),

Code = (code+1).ToString(),

Pin = "0000",

Debet = 0,

Credit = 0,

Currency = contractCurrency,

IsActive = true

};

await accountsService.CreateTransactionAsync(bankAccountPassive, true, newAccountForMainSum, true, contract.Sum, contractCurrency, context, AppDateTime); //From bank debet to main debet

//await accountsService.CreateTransactionAsync(newAccountForMainSum, false, bankAcountActive, true, contract.Sum, contractCurrency, context, AppDateTime); //From main credit to cassa debet

//await accountsService.CreateTransactionAsync(newAccountForMainSum.Number, false, "", false, contract.Sum, contractCurrency, context, AppDateTime); //From cassa credit

var newContract = new CreditContract()

{

CreditContractOptionId = contract.OptionId ?? -1,

Number = contract.Number,

Account1 = newAccountForMainSum,

Account2 = newAccountForPercents,

PersonSurname = contract.PersonSurname,

PersonName = contract.PersonName,

PersonMiddlename = contract.PersonMiddlename,

PassportSeria = contract.PassportSeria,

PassportNumber = contract.PassportNumber,

PassportIdentityNumber = contract.PassportIdentityNumber,

CityLivingId = context.Cities.First(cit => cit.Name == contract.CityLiving).Id,

AddressLiving = contract.AddressLiving,

Sum = contract.Sum,

CurrencyId = context.Currencies.First(cur => cur.Name == contract.Currency).Id,

PercentPerYear = contract.PercentPerYear,

StartDate = AppDateTime,

EndDate = AppDateTime.AddMonths(contract.DurationMonth)

};

await context.Accounts.AddAsync(newAccountForMainSum);

await context.Accounts.AddAsync(newAccountForPercents);

await context.CreditContracts.AddAsync(newContract);

await context.SaveChangesAsync();

}

private async Task validateContractAsync(ContractViewModel contract, ApplicationDbContext context)

{

if (contract.StartDate is null) contract.StartDate = AppDateTime;

if (contract.EndDate is null) contract.EndDate = AppDateTime.AddMonths(contract.DurationMonth);

if (contract.IsDifferentive is not null && !context.CreditContractOptions.Any(co => co.Id == contract.OptionId && co.Name == contract.Name)) throw new ContractValidationException("Не найден выбраный кредитный план (либо его имя не совпадает с ситемным)");

if (contract.IsRequestable is not null && !context.DebetContractOptions.Any(co => co.Id == contract.OptionId && co.Name == contract.Name)) throw new ContractValidationException("Не найден выбраный дебеторский план (либо его имя не совпадает с ситемным)");

if(!Regex.IsMatch(contract.Number, @"^\d{12}$")) throw new ContractValidationException("Неверный номер договора (должен состоять из 12 цифр)");

if ((await context.DebetContracts.AnyAsync(dc => dc.Number == contract.Number)) || (await context.CreditContracts.AnyAsync(cc => cc.Number == contract.Number)))

throw new ContractValidationException("Договор с таким номером уже существует!");

if (contract.StartDate?.AddMonths(contract.DurationMonth) != contract.EndDate) throw new ContractValidationException("Дата начала + Длительность в месяцах не равняется дате окончания");

if (String.IsNullOrEmpty(contract.PersonName)) throw new ContractValidationException("Имя не должно быть пустым!");

if (!Regex.IsMatch(contract.PersonName, @"^\w.+?$")) throw new ContractValidationException("Неверное имя!");

if (String.IsNullOrEmpty(contract.PersonSurname)) throw new ContractValidationException("Фамилия не должна быть пустой!");

if (!Regex.IsMatch(contract.PersonSurname, @"^\w.+?$")) throw new ContractValidationException("Неверная фамилия!");

if (String.IsNullOrEmpty(contract.PersonMiddlename)) throw new ContractValidationException("Отчество не должно быть пустым!");

if (!Regex.IsMatch(contract.PersonMiddlename, @"^\w.+?$")) throw new ContractValidationException("Неверное отчество!");

if (String.IsNullOrEmpty(contract.PassportSeria)) throw new ContractValidationException("Серия паспорта не должна быть пустой!");

if (!Regex.IsMatch(contract.PassportSeria, @"^\w{2}$")) throw new ContractValidationException("Неверная серия пасспорта!");

if (String.IsNullOrEmpty(contract.PassportNumber)) throw new ContractValidationException("Номер паспорта не должен быть пустым!");

if (!Regex.IsMatch(contract.PassportNumber, @"^\d{7}$")) throw new ContractValidationException("Неверный номер паспорта!");

if (String.IsNullOrEmpty(contract.PassportIdentityNumber)) throw new ContractValidationException("Идентификационный номер паспорта не должен быть пустым!");

if (!Regex.IsMatch(contract.PassportIdentityNumber, @"^\d{7}\w\d{3}\w{2}\d$")) throw new ContractValidationException("Неверный идентификационный номер паспорта!");

}

private async Task<long> generateAccountCodeNumberAsync(int userId, bool isDebet, ApplicationDbContext context)

{

long maxNumber = 0;

if (isDebet && await context.DebetContracts.AnyAsync())

{

maxNumber = await context.DebetContracts.MaxAsync(dc => System.Convert.ToInt64(dc.Number));

}

if(!isDebet && await context.CreditContracts.AnyAsync())

{

maxNumber = await context.CreditContracts.MaxAsync(cc => System.Convert.ToInt64(cc.Number));

}

if (maxNumber == 0) return isDebet ? 3000 : 2000;

return maxNumber + 1;

}

private string generateAccountNumber(int userId)

{

long now = (DateTime.Now.Ticks) & 0xFF\_FF\_FF\_FF\_FF; //BY06MMBN{0}

return String.Format("{0:D4}{1:D12}", userId, now);

}

public async Task<string> GetContractNumberAsync(ApplicationDbContext context, int optionId, int userId, bool isDebet = true)

{

var contractsLastId = 0;

if(isDebet ? await context.DebetContracts.AnyAsync() : await context.CreditContracts.AnyAsync())

contractsLastId = isDebet? await context.DebetContracts.MaxAsync(op => op.Id) : await context.CreditContracts.MaxAsync(op => op.Id);

string res = String.Format("{0:D1}{1:D5}{2:D3}{3:D3}", isDebet ? 2 : 3, contractsLastId, optionId, userId);

return res;

}

public async Task SkipDayAsync(ApplicationDbContext context, AccountsService accountsService)

{

AppDateTime += TimeSpan.FromDays(1);

await closeBankDay(context, accountsService);

}

public async Task SkipMonthAsync(ApplicationDbContext context, AccountsService accountsService)

{

var finishDate = AppDateTime.AddMonths(1);

while(AppDateTime < finishDate) await SkipDayAsync(context, accountsService);

}

private async Task closeBankDay(ApplicationDbContext context, AccountsService accountsService)

{

var bankAccountPassive = await context.Accounts.FirstAsync(ac => ac.Code == \_config.Value.BankAccountPassive);

foreach (var dc in await context.DebetContracts.Include(dc => dc.Account1).Include(dc => dc.Account2).Include(dc => dc.Currency).ToListAsync())

{

if (dc.StartDate >= AppDateTime) continue;

var dco = await context.DebetContractOptions.FirstAsync(dco => dco.Id == dc.DebetContractOptionId);

var monthGone = (AppDateTime - dc.StartDate).TotalDays / 30d;

if (dc.EndDate >= AppDateTime && isInteger(monthGone)) //fullfilling percents

{

if(dco.IsRequestable)

{

var mainAcc = await context.Accounts.FirstAsync(acc => acc.Id == dc.Account1Id);

var percentsSum = mainAcc.Credit \* (dc.PercentPerYear / 12);

await accountsService.CreateTransactionAsync(bankAccountPassive, true, dc.Account2, false, percentsSum, dc.Currency, context, AppDateTime);

}

}

if((dc.EndDate.Day == AppDateTime.Day && dc.EndDate.Month == AppDateTime.Month && dc.EndDate.Year == AppDateTime.Year) && !dco.IsRequestable) //deposit end

{

var duration = (int)Math.Round((dc.EndDate - dc.StartDate).TotalDays / 30d);

await accountsService.CreateTransactionAsync(bankAccountPassive, true, dc.Account2, false, dc.Sum \* (dc.PercentPerYear \* duration / 12), dc.Currency, context, AppDateTime); //percents

await accountsService.CreateTransactionAsync(bankAccountPassive, true, dc.Account1, false, dc.Sum, dc.Currency, context, AppDateTime); //body

}

}

foreach (var cc in await context.CreditContracts.Include(cc => cc.Account1).Include(cc => cc.Account2).Include(dc => dc.Currency).ToListAsync())

{

if (cc.StartDate >= AppDateTime) continue;

var cco = await context.CreditContractOptions.FirstAsync(cco => cco.Id == cc.CreditContractOptionId);

var monthGone = (AppDateTime - cc.StartDate).TotalDays / 30d;

double monthDuration = (cc.EndDate.Month - cc.StartDate.Month) + 12 \*(cc.EndDate.Year - cc.StartDate.Year);

if (cc.EndDate >= AppDateTime && isInteger(monthGone)) //paying percents

{

if (cco.IsDifferentive)

{

var percentsToPay = (cc.PercentPerYear / 12) \* (cc.Sum \* (decimal)((monthDuration - monthGone + 1) / monthDuration));

var bodyToPay = cc.Sum \* (decimal)(1 / monthDuration);

await accountsService.CreateTransactionAsync(cc.Account2, false, bankAccountPassive, false, percentsToPay, cc.Currency, context, AppDateTime);

await accountsService.CreateTransactionAsync(cc.Account1, false, bankAccountPassive, false, bodyToPay, cc.Currency, context, AppDateTime);

} else

{

var percMonth = (cc.PercentPerYear / 12);

var divider = (decimal)Math.Pow(1 + (double)percMonth, monthDuration) - 1;

var firstBody = cc.Sum \* percMonth / divider;

var curBody = firstBody \* (decimal)Math.Pow((double)(1+percMonth), monthGone-1);

var curPerc = cc.Sum \* percMonth - (firstBody \* (decimal)(Math.Pow(1+(double)percMonth, monthGone - 1) -1));

await accountsService.CreateTransactionAsync(cc.Account2, false, bankAccountPassive, false, curPerc, cc.Currency, context, AppDateTime);

await accountsService.CreateTransactionAsync(cc.Account1, false, bankAccountPassive, false, curBody, cc.Currency, context, AppDateTime);

}

}

if (cc.EndDate.Day == AppDateTime.Day && cc.EndDate.Month == AppDateTime.Month && cc.EndDate.Year == AppDateTime.Year) //credit end

{

//await accountsService.CreateTransactionAsync(cc.Account1, false, bankAccountPassive, false, cc.Sum, cc.Currency, context, AppDateTime);

}

}

await context.SaveChangesAsync();

}

private bool isInteger(double num)

{

int doubleToInt = (int)num;

return Math.Abs(num - doubleToInt) < 0.02;

}

}

public class ContractValidationException : ValidationException

{

public ContractValidationException(string message) : base(message) { }

public ContractValidationException() : base() { }

}

public class TransactionValidationException : ValidationException

{

public TransactionValidationException(string message) : base(message) { }

public TransactionValidationException() : base() { }

}

}

файл **App.js**

import './App.css';

import { Routes, Route, BrowserRouter, Navigate } from 'react-router-dom';

import requests from './agent';

import { useState, useEffect } from 'react';

import toast, { Toaster } from 'react-hot-toast';

import { ROLE } from "./Helpers/Constants";

import LoginPage from './Pages/LoginPage';

import CreateUserPage from './Pages/CreateUserPage';

import UserMainPage from './Pages/UserMainPage';

import { Provider, useSelector } from 'react-redux';

import EditUserPage from './Pages/EditUserPage';

import store from './Redux/Store/Store';

import UsersListPage from './Pages/UsersListPage';

import Header from './Components/Header';

import DebetContractsPage from './Pages/DebetContractsPage';

import DebetOptionsPage from './Pages/DebetOptionsPage';

import CreditContractsPage from './Pages/CreditContractsPage';

import CreditOptionsPage from './Pages/CreditOptionsPage';

import TransactionsPage from './Pages/TransactionsPage';

import SignContractPage from './Pages/SignContractPage';

import ContractOptionPage from './Pages/ContractOptionPage';

import ContractPage from './Pages/ContractPage';

function App() {

const isAdmin = useSelector(state => state.user.info.isAdmin);

const pathes = [

{ path: '/users/user', name: 'Главная', element: < UserMainPage />, authorizationLevel: 1 },

{ path: '/users/edit', name: 'Редактировать аккаунт', element: < EditUserPage />, authorizationLevel: 1 },

{ path: '/users/user/:id', name: undefined, element: < UserMainPage />, authorizationLevel: 2 },

{ path: '/users/edit/:id', name: undefined, element: < EditUserPage />, authorizationLevel: 2 },

{path: '/debet/contracts', name: 'Вклады', element: <DebetContractsPage/>, authorizationLevel: 1},

{path: '/debet/contracts/:contractId', name: undefined, element: <ContractPage isDebet={true}/>, authorizationLevel: 1},

{path: '/debet/options', name: 'Варианты вкладов', element: <DebetOptionsPage />, authorizationLevel: 1},

{path: '/debet/options/:contractId', name: undefined, element: < ContractOptionPage isDebet={true}/>, authorizationLevel: 1},

{path: '/debet/options/sign/:id', name: undefined, element: < SignContractPage isDebet={true}/>, authorizationLevel: 1},

{path: '/credit/contracts', name: 'Кредиты', element: <CreditContractsPage/>, authorizationLevel: 1},

{path: '/credit/contracts/:contractId', name: undefined, element: <ContractPage isDebet={false}/>, authorizationLevel: 1},

{path: '/credit/options/:contractId', name: undefined, element: < ContractOptionPage isDebet={false}/>, authorizationLevel: 1},

{path: '/credit/options', name: 'Варианты кредитов', element: <CreditOptionsPage />, authorizationLevel: 1},

{path: '/credit/options/sign/:id', name: undefined, element: < SignContractPage isDebet={false}/>, authorizationLevel: 1},

{path: '/transactions/personal', name: 'Счета', element: < TransactionsPage showBank={false}/>, authorizationLevel: 1},

// {path: '', name: '', element: < />, authorizationLevel: 1},

{ path: '/users/all', name: 'Пользователи', element: <UsersListPage />, authorizationLevel: 2 },

//{path: '/debet/contracts/all', name: 'Все вклады пользователей', element: <DebetContractsPage />, authorizationLevel: 2},

{path: '/transactions/bank', name: 'Счета банка', element: < TransactionsPage showBank={true}/>, authorizationLevel: 2},

// {path: '', name: '', element: < />, authorizationLevel: 2},

{ path: '/users/create', name: undefined, element: <CreateUserPage />, authorizationLevel: 0 },

{ path: '/', name: 'Выйти', element: <LoginPage />, authorizationLevel: 0 },

];

const authLevel = isAdmin == null ? 0 : isAdmin ? 2 : 1;

const routing = pathes.filter(p => p.authorizationLevel <= authLevel);

console.log(routing);

return (

<div>

<Toaster

position='top-center'

reverseOrder={false}

toastOptions={{

duration: 3000

}}

/>

<BrowserRouter forceRefresh={false}>

{authLevel > 0 ? <Header routes={routing} /> : undefined}

<Routes>

{routing.map(r => <Route key={r.path} path={r.path} exact element={r.element} />)}

<Route path="\*" element={<Navigate to={authLevel > 0 ? '/users/user' : "/"} />} />

</Routes>

</BrowserRouter>

</div>

);

}

export default App;

файл **Transaction.jsx**

import { useEffect, useState } from "react";

import { toast } from "react-hot-toast";

import requests from '../agent';

import TransactionsListTable from "../Components/TransactionsListTable";

import AccountsStatisticsGraph from "../Components/AccountsStatisticsGraph";

import CloseBankDayLine from "../Components/ClosaBankDayLine";

const TransactionsPage = ({showBank = false}) => {

const [loading, setLoading] = useState(true);

const [accStates, setAccStates] = useState([]);

useEffect(() => {

if(loading){

setLoading(false);

const request = showBank ? requests.transactions.bank() : requests.transactions.current();

request.then(res => {

console.log(res);

setAccStates(res.data);

}).catch(err => console.error(err));

}

}, [loading]);

return (<>

{accStates.map(as => <TransactionsListTable accState={as} />)}

<AccountsStatisticsGraph data={accStates} />

<CloseBankDayLine setLoading={setLoading} data={accStates}/>

</>);

}

export default TransactionsPage;

файл **TransactionsListTable.jsx**

import { DataGrid } from "@mui/x-data-grid";

import { useNavigate } from "react-router";

const TransactionsListTable = ({accState}) => {

const columns = [

{

field: 'id',

headerName: 'Id',

flex: 0.1,

headerClassName: 'subtext',

headerAlign: 'left',

align: 'left'

},

{

field: 'date',

headerName: 'Дата',

flex: 0.15,

headerClassName: 'subtext',

headerAlign: 'left',

align: 'left'

},

{

field: 'debet',

headerName: 'Дебет после',

flex: 0.15,

headerClassName: 'subtext',

headerAlign: 'left',

align: 'left'

},

{

field: 'credit',

headerName: 'Кредит после',

flex: 0.15,

headerClassName: 'subtext',

headerAlign: 'left',

align: 'left'

},

{

field: 'sum',

headerName: 'Сумма',

flex: 0.15,

headerClassName: 'subtext',

headerAlign: 'left',

align: 'left'

},

{

field: 'account',

headerName: 'Другой счёт',

flex: 0.15,

headerClassName: 'subtext',

headerAlign: 'left',

align: 'left'

},

];

return <>

Счёт: {accState.account.number}

<DataGrid

headerHeight={100}

rowHeight={27}

rows={accState.transactionList}

columns={columns}

getRowId={(x) => x.id}

disableSelectionOnClick

pageSize={10}

pagination

sx={{

height: '35vh',

my: '35px',

'& .MuiDataGrid-columnHeaderTitle': {

textOverflow: "clip",

whiteSpace: "break-spaces",

lineHeight: 1,

textAlign: "left"

},

}}

/>

</>

}

export default TransactionsListTable;

файл **TransactionsListTable.jsx**

import { useEffect } from "react";

import { useState } from "react";

import { Line } from "react-chartjs-2";

import {

Chart as ChartJS,

CategoryScale,

LinearScale,

PointElement,

LineElement,

Title,

Tooltip,

Legend,

} from "chart.js";

ChartJS.register(CategoryScale, LinearScale, PointElement, LineElement, Title, Tooltip, Legend);

const AccountsStatisticsGraph = ({ data }) => {

const [diagrammData, setDiagrammData] = useState({ datasets: [] });

useEffect(() => {

const colors = ["#000000", "#ff0000", "#00ff00", "#0000ff", "#ffaa00", "#777777", "#00aaff", "#aa33aa", "#7777ff", "#ff7777", "#33ff77", "#ff00ff"];

const minDate = getMinCompared(data).minTransaction?.date;

const maxDate = getMinCompared(data, (date1, date2) => ( date1 > date2 )).minTransaction?.date;

const times = getTimes(minDate, maxDate);

setDiagrammData({

labels: times.map(t => (new Date(t))?.toDateString()) ?? [],

datasets: data.map((dat, i) => {

return {

visible: true,

data: extendDataOnAllDays(addZeroTransaction(dat.transactionList, dat.account, minDate), times),

label: dat.account.number,

borderColor: colors[i],

backgroundColor: colors[i],

fill: true,

lineTension: 0.5

};

})

});

}, [data]);

const changeVisibleActionCreator = (i) => () => {

const newDiagrammData = { ...diagrammData };

newDiagrammData.datasets[i].visible = !newDiagrammData.datasets[i].visible;

setDiagrammData(newDiagrammData);

}

return <>

<Line

type="line"

width={130}

height={70}

options={{

title: {

display: true,

text: 'Разность дебета и кредита на счетах',

fontSize: 15

},

legend: {

display: true, //Is the legend shown?

position: "top" //Position of the legend.

}

}}

data={diagrammData}

/>

</>

}

const getMinCompared = (dat, compare = (date1, date2) => ( date1 < date2 )) => dat?.reduce((modelAccum, modelVal) => {

var modelAccumReduce = modelAccum?.minTransaction ?? modelAccum?.transactionList.reduce((acc, val) => compare(acc.date, val.date) ? acc : val);

var modelValReduce = modelVal.transactionList.reduce((acc, val) => compare(acc.date, val.date) ? acc : val, {date: new Date()});

let resModel;

if (compare(modelAccumReduce.date, modelValReduce.date)) {

resModel = modelAccum;

resModel.minTransaction = modelAccumReduce;

} else {

resModel = modelVal;

resModel.minTransaction = modelValReduce;

}

return resModel;

}, {minTransaction: {date: new Date()}});

const addZeroTransaction = (transactions, account, minDate) => {

const res = [{

date: minDate,

debet: account.debet,

credit: account.credit

}];

transactions.slice().reverse().map(tr => res.push(tr));

return res;

}

const getTimes = (minDate = new Date(), maxDate = new Date(), intervals = 30) => {

const minTime = (new Date(minDate)).getTime();

const maxTime = (new Date(maxDate)).getTime();

const delta = (maxTime - minTime) / intervals;

return Array(intervals+1).fill(0).map((el, i) => {

const res = (minTime + i \* delta);

return res;

});

}

const extendDataOnAllDays = (transactions, times = []) => {

console.log('Tra', transactions);

let curIndex = 0;

return times.map(time => {

console.log(transactions[curIndex], new Date(time));

while (curIndex < transactions.length - 1 && (new Date(transactions[curIndex+1]?.date)).getTime() <= time) {

curIndex++;

}

return transactions[curIndex]?.debet + transactions[curIndex]?.credit;

})

}

export default AccountsStatisticsGraph;

файл **agent.jsx**

import axios from "axios";

axios.defaults.withCredentials = true;

const server = 'http://localhost:62424/api'; ////023872

const requests = {

auth: {

index: () => axios.post(server+'/auth'),

logout: () => axios.post(server+'/auth/logout')

},

users: {

index: () => axios.get(server+'/users'),

currentUser: () => axios.get(server+'/users/user'),

user: (id) => axios.get(server+`/users/user/${id}`),

create: (userViewModel) => axios.post(server+'/users/create', userViewModel),

editCurrent: (userViewModel) => axios.put(server+'/users/edit', userViewModel, { headers: { 'Content-Type': 'multipart/form-data' } }),

edit: (id, userViewModel) => axios.put(server+`/users/edit/${id}`, userViewModel, { headers: { 'Content-Type': 'multipart/form-data' } }),

deleteCurrent: () => axios.delete(server+'/users/delete'),

delete: (id) => axios.delete(server+`/users/delete/${id}`),

},

staticData: {

roles: () => axios.get(server+'/staticData/roles'),

citizenships: () => axios.get(server+'/staticData/citizenships'),

disabilities: () => axios.get(server+'/staticData/disabilities'),

familyStatuses: () => axios.get(server+'/staticData/familyStatuses'),

cities: () => axios.get(server+'/staticData/cities'),

currencies: () => axios.get(server + '/staticData/currencies'),

appDate: () => axios.get(server + '/staticData/app-date')

},

transactions: {

current: () => axios.get(server+'/transactions'),

bank: () => axios.get(server+'/transactions/bank'),

closeDay: () => axios.post(server+'/transactions/closeDay'),

closeMonth: () => axios.post(server+'/transactions/closeMonth')

},

debet: {

currentContracts: () => axios.get(server+'/debet'),

contract: (id) => axios.get(server+`/debet/${id}`),

allContracts: () => axios.get(server+'/debet/all'),

allOptions: () => axios.get(server+'/debet/options'),

option: (id) => axios.get(server+`/debet/options/${id}`),

contractNumber: (id) => axios.get(server+`/debet/contract-number/${id}`),

sign: (id, contractViewModel) => axios.post(server+`/debet/options/${id}`, contractViewModel, { headers: { 'Content-Type': 'multipart/form-data' } })

},

credit: {

currentContracts: () => axios.get(server+'/credit'),

contract: (id) => axios.get(server+`/credit/${id}`),

allContracts: () => axios.get(server+'/credit/all'),

allOptions: () => axios.get(server+'/credit/options'),

option: (id) => axios.get(server+`/credit/options/${id}`),

contractNumber: (id) => axios.get(server+`/credit/contract-number/${id}`),

sign: (id, contractViewModel) => axios.post(server+`/credit/options/${id}`, contractViewModel, { headers: { 'Content-Type': 'multipart/form-data' } })

}

};

export default requests;