Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

Рязанский государственный радиотехнический университет  
имени В.Ф. Уткина

Кафедра АСУ

К защите

Руководитель работы:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ**

по дисциплине

**«Технология программирования»**

Тема:

«Разработка информационной системы банка»

Выполнил студент группы 135

Сахаров Н.М.

дата сдачи на проверку, подпись

Руководитель работы

Ст. преп. каф. АСУ

Аникеев Д.В.

оценка дата защиты, подпись

Федеральное государственное бюджетное образовательное учреждение   
высшего образования

Рязанский государственный радиотехнический университет

им.В.Ф.Уткина

Кафедра Автоматизированных Систем Управления

З А Д А Н И Е

на курсовую работу по дисциплине

"Технологии программирования"

Студент — Сахаров Никита Михайлович группа – 135

1. Тема: Разработка информационно-поисковой системы «Кредит в банке»

2. Срок представления работы к защите - « » 2023г.

3. Исходные данные для проектирования:

* 1. Язык программирования – C#.
  2. СУБД — MS SQL Compact Edition, MS SQL Express Edition
  3. Технология объектного связывания данных – Entity Framework.
  4. Общее количество таблиц - не менее 5.
  5. Обязательные операции с данными — просмотр записей таблиц в виде списка, сортировка, поиск, добавление, изменение, удаление записей.
  6. Все страницы сайт должны быть наполнены осмысленной информацией.

4. Содержание курсовой работы

4.1. Задание.

4.2. Пояснительная записка, оформленная по ГОСТ 19.404-79, содержащая:

4.2.1. Описание предметной области.

4.2.2. Структурную схему модели данных.

4.2.3. Структурную схему программы.

Руководитель работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Аникеев Д.В. /

(подпись)

Задание принял к исполнению\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ « » 2023г.

(подпись)

**Содержание**

[Введение 4](#_Toc136463112)

[1. Описание предметной области 5](#_Toc136463113)

[1.1 Описание процесса получения кредита в банке 7](#_Toc136463114)

[2. Сравнительный анализ 8](#_Toc136463115)

[3.Реализация системы 11](#_Toc136463116)

[Создание Базы данных 11](#_Toc136463117)

[Data Access Layer (DAL) 15](#_Toc136463118)

[Domain 20](#_Toc136463119)

[Пользовательский интерфейс 24](#_Toc136463120)

[4. Тестирование 32](#_Toc136463121)

[Заключение 42](#_Toc136463122)

[Список используемой литературы 43](#_Toc136463123)

### Введение

Банки являются важным звеном в функционировании экономики страны и мировой экономики. Банковская система повышает эффективность всего производства[1].

Основное назначение банка - посредничество в перемещении денежных средств от кредиторов к заёмщикам и от продавцов к покупателям.

Кредит состоит из элементов, находящихся в тесном взаимодействии друг с другом. Такими элементами являются, прежде всего, субъекты его отношений. В кредитной сделке субъекты отношений всегда выступают как кредитор и заемщик. Становление кредитора и заемщика происходит, прежде всего, на базе товарного обращения. Процесс купли-продажи товаров не всегда приводит к немедленному получению продавцом их денежного эквивалента, покупатель не всегда имеет возможность сразу заплатить за товар, оплата производится только по истечении определенного срока. Так, продавец становится кредитором, покупатель - должником. Товарное обращение не представляет собой единственную базу возникновения кредитора и заемщика. Кредитор и заемщик появляются во всех случаях, когда на одном полюсе (у одного субъекта) отсрочено получение эквивалента; на другом - его уплата.

В условиях перехода общества в информационную эпоху, все большую и большую актуальность приобретают автоматизированные системы и бизнес-процессы. Следовательно, целью качественной работы банков является разработка автоматизированной системы управления принятия клиентов на обслуживание.

Информационно-поисковая система должна повысить эффективность работы сотрудников банка, а также помочь осуществить своевременный учет данных о динамике выдачи кредитов.

# **Описание предметной области**

Кредит или кредитные отношения - общественные отношения, возникающие между субъектами экономических отношений по поводу движения стоимости[2]. Кредитные отношения могут выражаться в разных формах кредита (коммерческий кредит, банковский кредит и др.), займе, лизинге. Возникновение кредита как особой формы стоимостных отношений происходит тогда, когда стоимость, высвободившаяся у одного экономического субъекта, какое-то время не вступает в новый воспроизводственный цикл. Благодаря кредиту она переходит от субъекта, не использующего её (кредитор), к другому субъекту, испытывающему потребность в дополнительных средствах (заёмщик).  
Загруженность банков иногда имеет пиковый характер. Эту загруженность можно снизить путём внедрения информационной системы.

Обслуживание клиентов, которые хотят оформить кредит, должно проходить быстро, тем самым и регистрация клиента должна оформляться быстро. Следовательно, есть необходимость в ускорении данного процесса.

Комплексная автоматизация управления предприятия на сегодняшний день - один из самых эффективных и функциональных инструментов систематизации работы ключевых бизнес-процессов. Зачастую управление бизнес-процессами становится трудоемким, а анализ большого потока первичных данных отнимает много сил, времени. В условиях современной жизни требуется ускорение процессов обработки информации. Этот процесс подлежит автоматизации, так как обработка информации очень долгий, кропотливый и требующий больших ресурсов процесc.

### 

Use Case модель

Диаграмма прецедентов, отражающая отношения между акторами и прецедентами [7]. На этой диаграмме можно выделить следующие объекты:

* клиент выбирает вид кредита, подает заявление и взаимодействует с сотрудником банка;
* сотрудник банка вносит информацию о клиенте в базу данных, производит проверку кредитной истории и оформляет договор;

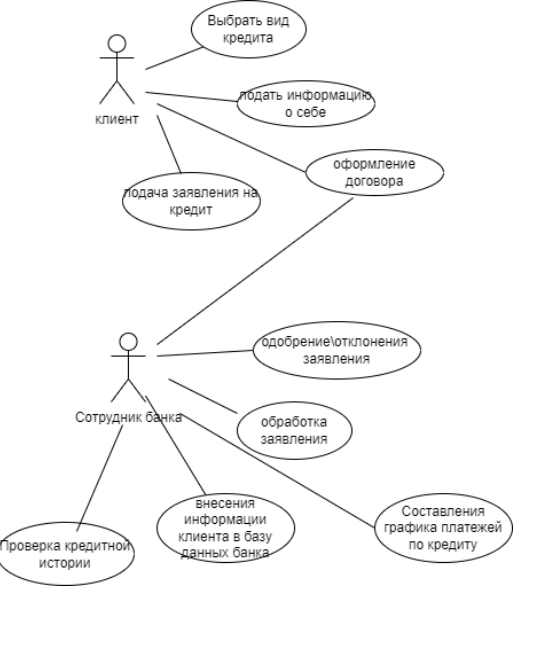


Рисунок 2 - Use Case модель

### 1.1 Описание процесса получения кредита в банке

BPMN[6] — это стандартизированный метод отображения блок-схем, который позволяет создавать и обмениваться простыми для понимания диаграммами. Эти диаграммы могут визуально моделировать этапы бизнес-процесса от начала до конца.

В начале сотрудник вводит данные человека, который подал документы о получении кредита. Дальше сотрудник интересуется целью получения кредита и вводит ее в базу данных. Вскоре консультант узнаёт необходимую сумму для кредита, от которой зависит разные условия кредитования. Потом сотрудник узнаёт финансовые возможности кредитора: место работы, заработную плату, наличие недвижимого имущества. Если все необходимые условия соблюдены, консультант вносит полученные сведения в личное дело. Далее идет проверка на непогашенные задолженности. В случае отсутствия их сотрудник финансового учреждения принимает заявку и отправляет её на рассмотрение в течение 5 рабочих дней. После прохождения определенного срока выносится решение, при котором клиент идет подписывать кредитный договор. В противной случае, банк информирует клиента об отказе. В обоих случаях процесс завершается.

BPMN модель (рисунок 2)

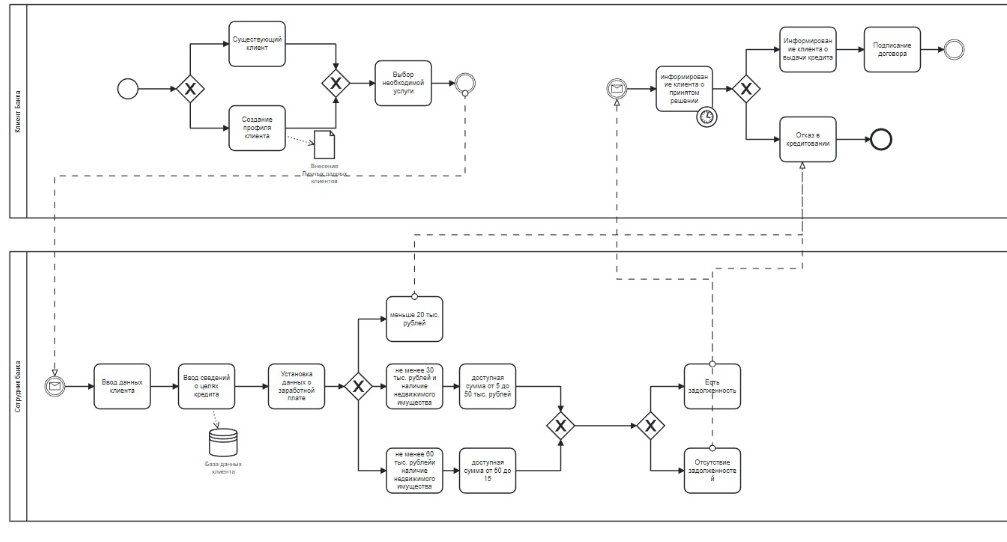
**

Рисунок 2 - BPMN модель

### Сравнительный анализ

Несмотря на то, что большинство людей пользуются услугами Сбербанка, многие отдают предпочтение Тинькофф. Чтобы понять, чем привлекает каждая организация, можно ознакомиться со сравнительным обзором Сбербанк[3] или Тинькофф[4].

Банки активно конкурируют друг с другом в плане предоставления новых возможностей и удобства использования своих продуктов[5]. В связи с этим возникает вопрос – услугами какого банка лучше пользоваться: Сбербанка или Тинькофф? Каждый из них достаточно технологичен и привлекателен.

Несмотря на то, что по объему активов Тинькофф не может сравниться с крупнейшим банком страны – Сбербанком, он предлагает немало эксклюзивных возможностей.

Например, взаимоотношение с клиентами Тинькофф Банк практически полностью строит на цифровых технологиях. Связь с держателями карт осуществляется через мобильное приложение, чат на сайте и электронную почту. Разработан интернет-банк, в котором собраны все имеющиеся продукты – от депозитов до брокерского счета и инвестиционного страхования жизни.

В плане предоставления дополнительных услуг и Сбербанк, Тинькофф стараются отвечать нуждам клиентов. Оба банка предлагают открыть брокерский счет, оформить страховку, положить деньги на вклады, заказать кредит и проверить кредитную историю. Но если у Тинькофф всё это собрано в рамках одного кабинета, то у Сбербанка «разнесено» по разным платформам.

Рассуждая о том, какой банк выбрать – Сбербанк или Тинькофф – стоит рассмотреть и сравнить предлагаемые каждым из них продукты. К числу наиболее активно используемых частными клиентами относятся дебетовые и кредитные карты, депозиты и мобильные приложения.

Сбербанк и Тинькофф - это две из крупнейших финансовых компаний в России, которые активно используют современные технологии для разработки веб-приложений.

Сравнительный анализ используемых технологий в Сбербанке и Тинькофф позволяет выявить следующие особенности:

1. Языки программирования.

В обеих компаниях широко используются языки программирования Java и JavaScript. Однако, Тинькофф активно развивает стек технологий, используя Rust и Kotlin, которые обеспечивают высокую производительность и безопасность при разработке приложений.

2. Фреймворки.

Сбербанк использует широкий спектр фреймворков, таких как Spring (для Java), React и Vue (для JavaScript). Тинькофф, в свою очередь, активно использует фреймворк Vue.js и его расширения, такие как Nuxt.js, Vuex, Vuetify и др.

3. Контейнеризация.

Обе компании активно используют технологию контейнеризации, такую как Docker, для создания и запуска однородных приложений.

4. Облачные сервисы.

Тинькофф использует облачные сервисы Яндекса, такие как Яндекс.Облако и Яндекс.Клауд, для хранения данных и обработки информации о клиентах. В свою очередь, Сбербанк использует свою собственную облачную платформу, называемую "Сбербанк-Облако". Она предоставляет возможность размещения виртуальных машин, хранения данных и использования различных инструментов для анализа данных.

5. Мониторинг и анализ производительности.

Сбербанк использует системы мониторинга, такие как Prometheus, Graphana, ELK-stack, для анализа производительности своих приложений. Тинькофф также активно использует ELK-stack, а также системы мониторинга Loki и Jaeger для анализа работы приложений.

Таким образом, обе компании привлекают высококвалифицированных специалистов и инвестируют в новейшие технологии, чтобы обеспечить успешную работу своих веб-приложений на рынке финансовых услуг.

### Реализация системы

Как правило, в приложении определяются слои пользовательского интерфейса, бизнес-логики и доступа к данным[9]. В рамках такой архитектуры пользователи выполняют запросы через слой пользовательского интерфейса, который взаимодействует только со слоем бизнес-логики. Слой бизнес-логики, в свою очередь, может вызывать слой доступа к данным для обработки запросов. Слой пользовательского интерфейса не должен выполнять запросы напрямую к слою доступа к данным и какими-либо другими способами напрямую взаимодействовать с функциями сохраняемости. Аналогичным образом, слой бизнес-логики должен взаимодействовать с функциями сохраняемости только через слой доступа к данным. Таким образом, для каждого слоя четко определена своя обязанность.

Общепринятая организация логики приложения по слоям показана рисунке 3.

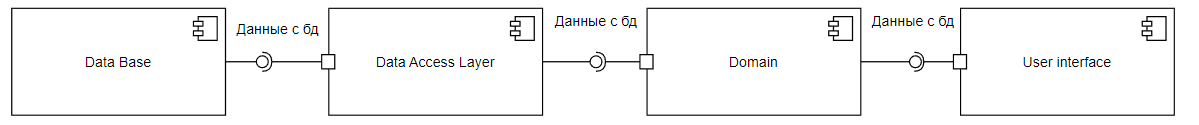


Рисунок 3 - Общепринятая организация логики приложения

### Создание Базы данных

База данных[10] — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. База данных обычно управляется системой управления базами данных (СУБД). Данные вместе с СУБД, а также приложения, которые с ними связаны, называются системой баз данных, или, для краткости, просто базой данных

На основании проведенного анализа предметной области «Кредит в банке» можно выделить следующие сущности: клиент, вид кредита, заявление, график платежей, кредитный договор, должность, сотрудники.

Таблица 1 – Список сущностей

|  |  |  |
| --- | --- | --- |
| № | Название | Назначение |
| 1 | Client | Перечень людей, подающие заявку на кредит. |
| 2 | View | Представляет различные виды кредитов, которые можно получить. |
| 3 | Statement | Характеризуетвыбранный вид кредита , сумму и процентную ставку. |
| 4 | ChartNumber | Содержит график дней выплат по кредиту. |
| 5 | Post | Описывает названия должностей. |
| 6 | Employee | Перечень сотрудников банка. |
| 7 | NumberOfContract | Хранит информацию о заключении кредитного договора |

На основе анализа требований и use case диаграммы, можно определить основные сущности и связи между ними, которые будут представлены в ER-модели (Рисунок 4).



Рисунок 4 – ER – диаграмма

ER-диаграмма качество отношений между сущностями, но не показывает, какие типы должны быть поля, поэтому изготовлена ​​модель производителя данных (рисунок 5).

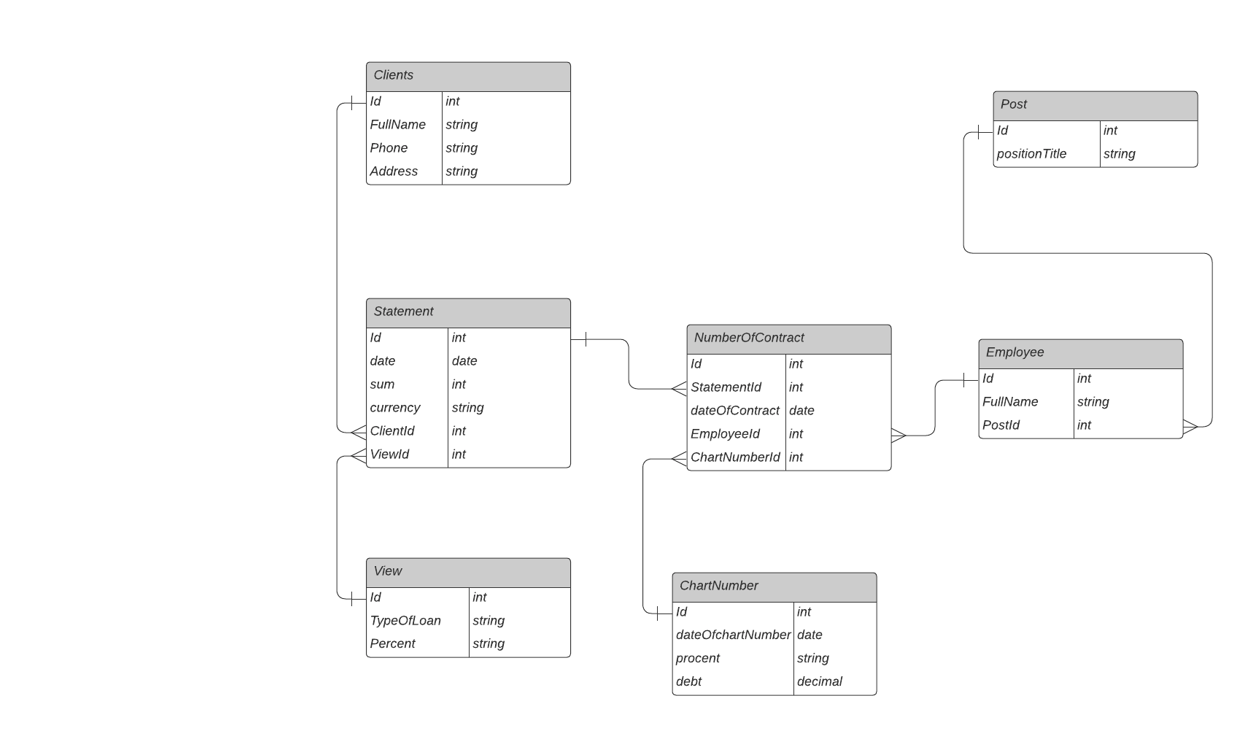


Рисунок 5 - Физическая модель данных

Для создания и управление базой данных в работе использовалась СУБД SQL Server Management Studio[11] (SSMS) от Microsoft. Таблицы в базе данных разрабатывались на основании диаграммы сущностей (рисунок 4).

Таким образом, получилась следующая база данных (рисунок 6-8)

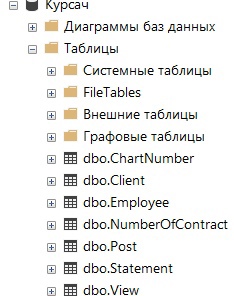


Рисунок 6 – Общий вид таблиц в базе данных

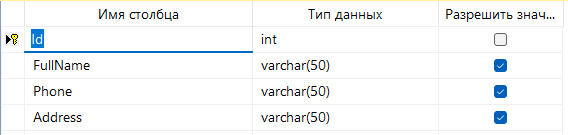


Рисунок 7 – Устройство таблицы в базе данных на примере таблицы клиентов



Рисунок 8 – Пример записей в таблице клиентов

### Data Access Layer (DAL)

Первым уровнем, который мы реализуем, будет Data Access Layer или уровень доступа к данным. Этот уровень обычно содержит все модели данных, хранящихся в БД, а также классы, через которые идет взаимодействие с БД.

Слой доступа к данным (DAL) — это компонент архитектуры программного обеспечения, отвечающий за управление хранением и извлечением данных приложения. Он находится между уровнем бизнес-логики и системой хранения данных и обеспечивает уровень абстракции, который позволяет уровню бизнес-логики взаимодействовать с системой хранения данных, не зная о ее конкретной реализации[12] (рисунок 8).



Рисунок 8 - Data Access Layer в общей архитектуре

DAL отвечает за выполнение таких задач, как:

* Подключение к системе хранения данных и управление подключением.
* Генерация и выполнение SQL-запросов или других команд доступа к данным для извлечения и хранения данных.
* Отображение данных из системы хранения данных в объекты данных приложения и наоборот.
* Обработка ошибок и исключений, связанных с доступом к данным.
* Обеспечение поддержки транзакций и других функций доступа к данным.

DAL предназначен для многократного использования и не зависит от бизнес-логики и реализации хранилища данных. Это позволяет легко модифицировать или расширять приложение, не затрагивая базовый код доступа к данным.

Таким образом, уровень доступа к данным — это компонент архитектуры программного обеспечения, который отвечает за управление хранением и извлечением данных приложения. Он находится между уровнем бизнес-логики и системой хранения данных и обеспечивает уровень абстракции, который позволяет уровню бизнес-логики взаимодействовать с системой хранения данных, не зная о ее конкретной реализации. DAL.

DAL составлен из моделей, контроллеров, Application context и связи с БД (рисунок 9).

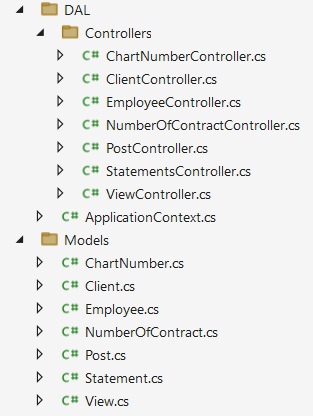


Рисунок 9 – Общий вид слоя доступа к данным

Для функционирования слоя DAL были созданы классы, соответствующие таблицам в БД. Названия и типы данных свойств созданных классов соответствуют столбцам соответствующих таблиц. Как пример рассмотрим класс Client (листинг 1) и таблицу Client (рисунок 10).

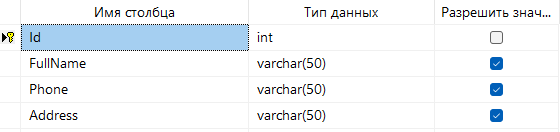


Рисунок 10 – Таблица Client

Листинг 1 – Класс Clients

namespace Bank.Models

{

public class Client

{

public int Id { get; set; }

public string? FullName { get; set; }

public string? Phone { get; set; }

public string? Address { get; set; }

}

}

Для остальных таблиц были также созданы соответствующие классы.

Для взаимодействия с базой данных через Entity Framework Core необходим контекст данных - класс, унаследованный от класса Microsoft.EntityFrameworkCore.DbContext. Поэтому добавим в проект новый класс, который назовем ApplicationContext (листинг 2):

Листинг 2- класс ApplicationContext

public class ApplicationContext : DbContext

{

public DbSet<Client> Client { get; set; }

public DbSet<View> View { get; set; }

public DbSet<Statement> Statement { get; set; }

public DbSet<ChartNumber> ChartNumber { get; set; }

public DbSet<NumberOfContract> NumberOfContract { get; set; }

public DbSet<Employee> Employee { get; set; }

public DbSet<Post> Post { get; set; }

public ApplicationContext(DbContextOptions<ApplicationContext> options)

: base(options)

{

Database.EnsureCreated();

}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

var builder = new ConfigurationBuilder();

builder.SetBasePath(Directory.GetCurrentDirectory());

builder.AddJsonFile("appsettings.json");

var config = builder.Build();

string connectionString = config.GetConnectionString("DefaultConnection");

optionsBuilder.UseSqlServer(connectionString);

}

}

}

Свойство DbSet представляет собой коллекцию объектов, которая сопоставляется с определенной таблицей в базе данных. В конструкторе класса ApplicationContext через параметр options будут передаваться настройки контекста данных. А в самом конструкторе с помощью вызова Database.EnsureCreated() по определению модели будет создаваться база данных (если она отсутствует). Чтобы подключаться к базе данных, необходимо задать параметры подключения. Это можно сделать в файле конфигурации. Для этого изменили файл appsettings.json, добавив в него строку для подключения к базе данных (листинг 3).

Листинг 3- Конфигурационный файл appsettings.json

{

"ConnectionStrings": {

"DefaultConnection": "Server=LAPTOP-4Q96PGU9\\SQLEXPRESS;DataBase=Курсач;Trusted\_Connection=True;MultipleActiveResultSets=true;TrustServerCertificate=True"

}

}

В слое DAL так же были реализованы CRUD операции, позволяющие создавать, редактировать, считывать и удалять записи из соответствующих таблиц в базе данных.

CRUD (создание, чтение, обновление, удаление) — это аббревиатура, обозначающая четыре функции, которые мы используем для реализации приложений постоянного хранения и приложений реляционных баз данных, включая Oracle Database, Microsoft SQL Server и MySQL.

CRUD постоянно используется для всего, что связано с базами данных и проектированием баз данных. Разработчики программного обеспечения ничего не могут сделать без операций CRUD. Например, при разработке веб-сайтов используется REST (передача репрезентативного состояния), который является надмножеством CRUD, используемого для ресурсов HTTP[13].

С другой стороны, CRUD не менее важен для конечных пользователей. Без него такие вещи, как регистрация на веб-сайтах, создание блогов или закладок, были бы невозможны. Большинство приложений, которые мы используем, позволяют нам добавлять или создавать новые записи, искать существующие, вносить в них изменения или удалять их.

***CRUD предлагает множество преимуществ, в том числе:***

1. Это облегчает контроль безопасности, удовлетворяя различные требования доступа.
2. Он упрощает и упрощает разработку приложений, делая их более масштабируемыми.
3. Он имеет лучшую производительность по сравнению со специальными операторами SQL.

В качестве примера ниже показан класс, реализующий CRUD операции (листинг 4):

Листинг 4 – Класс ClientController

namespace bank\_v.\_1.DAL.Controllers

{

[ApiController]

[Route("/Client")]

public class ClientController : ControllerBase

{

ApplicationContext db;

public ClientController(ApplicationContext context)

{

db = context;

}

[HttpGet]

public async Task<ActionResult<IEnumerable<Client>>> Get()

{

return await db.Client.ToListAsync();

}

[HttpGet("{id}")]

public async Task<ActionResult<LinkedList<Client>>> GetClient(int id)

{

Client client = await db.Client.FirstOrDefaultAsync(x => x.Id == id);

if (client == null)

return NotFound();

return new ObjectResult(client);

}

[HttpPost]

public async Task<ActionResult<Client>> Post(Client client)

{

if (client == null)

{

return BadRequest();

}

db.Client.Add(client);

await db.SaveChangesAsync();

return Ok(client);

}

[HttpPut]

public async Task<ActionResult<Client>> Put(Client client)

{

if (client == null)

{

return BadRequest();

}

if (!db.Client.Any(x => x.Id == client.Id))

{

return NotFound();

}

db.Update(client);

await db.SaveChangesAsync();

return Ok(client);

}

[HttpDelete("{Id}")]

public async Task<ActionResult<Client>> Delete(int Id)

{

var client = db.Client.FirstOrDefault(x => x.Id == Id);

if (client == null)

{

return NotFound();

}

db.Client.Remove(client);

await db.SaveChangesAsync();

return Ok(client);

}

}

### Domain

Слой Domain[14] в архитектуре web-приложений, также известный как слой бизнес-логики, представляет собой часть приложения, которая отвечает за обработку бизнес-логики и хранение данных. Он является промежуточным слоем между слоем представления (UI) и слоем доступа к данным (Data Access Layer). Содержит классы, которые представляют бизнес-объекты и определяют логику, связанную с этими объектами, также включает в себя классы для выполнения операций с базой данных и другими источниками данных, обеспечивает связь между слоем представления и слоем доступа к данным, преобразуя данные, полученные от пользовательского интерфейса в объекты, которые могут быть сохранены в базе данных.

Основные преимущества:

1. Гибкость: Слой Domain позволяет изменять бизнес-логику и работу с данными, не затрагивая другие слои приложения. Это делает приложение более гибким и легко поддерживаемым.

2. Независимость от технологий: Слой Domain не зависит от конкретных технологий, таких как базы данных или пользовательские интерфейсы. Это позволяет использовать различные технологии для разных частей приложения, не затрагивая бизнес-логику[15].

3. Улучшенная безопасность: Слой Domain может использоваться для реализации правил безопасности, таких как проверка прав доступа и аутентификация пользователей.

4. Повторное использование: Слой Domain может использоваться в различных проектах и приложениях. Это позволяет повторно использовать код и ускоряет процесс разработки.

5. Упрощение тестирования: Слой Domain обычно имеет меньшую зависимость от внешних систем, что упрощает тестирование и отладку приложения.

6. Легкая масштабируемость: Слой Domain может использоваться для управления бизнес-логикой приложения любого размера, что позволяет легко масштабировать приложение в случае необходимости

Стоит отдельно выделить термин НТТР, поскольку именно благодаря HTTP запросам происходит взаимодействие внутри системы.

HTTP (HyperText Transfer Protocol)[19] - это протокол прикладного уровня, который используется для передачи гипертекстовых документов в сети Интернет. Он определяет способ передачи данных между клиентом и сервером и позволяет клиенту отправлять запросы на сервер, а серверу отправлять ответы на эти запросы. HTTP работает по модели запрос-ответ, где клиент отправляет запрос на сервер, а сервер отвечает на этот запрос.

HTTP использует TCP/IP для передачи данных между клиентом и сервером. Клиент отправляет запрос на сервер, указывая метод запроса (GET, POST, PUT, DELETE и т.д.), URL, HTTP версию и другие параметры. Сервер обрабатывает этот запрос и отправляет ответ клиенту, указывая HTTP версию, статус ответа (например, 200 OK или 404 Not Found) и тело ответа, которое может содержать данные или ошибки.

**GET Метод**

GET запрашивает представление ресурса. Запросы с использованием этого метода могут только извлекать данные.

**POST Метод**

POST используется для отправки сущностей к определённому ресурсу. Часто вызывает изменение состояния или какие-то побочные эффекты на сервере.

В данном слое используются такие же классы, как и в слое DAL. Пример класса показан в листинге 5:

Листинг 5 – Класс Client

namespace Domain.Controllers

{

[ApiController]

[Microsoft.AspNetCore.Mvc.Route("api/Domain/Client")]

public class ClientsController : ControllerBase

{

private readonly string? \_dalUrl;

private readonly HttpClient \_client;

public ClientsController(IConfiguration conf)

{

\_dalUrl = conf.GetValue<string>("DalUrl");

\_client = new HttpClient();

}

[HttpGet]

public async Task<ActionResult<Client[]>> GetClients()

{

var response = await \_client.GetAsync($"{\_dalUrl}/Client");

response.EnsureSuccessStatusCode();

var content = await response.Content.ReadAsStringAsync();

if (content == null) return NotFound();

return JsonSerializer.Deserialize<Client[]>(content, new JsonSerializerOptions

{

PropertyNameCaseInsensitive = true

}) ?? Array.Empty<Client>();

}

[HttpPost]

public async Task<ActionResult<Client>> PostClient(Client client)

{

JsonContent content = JsonContent.Create(client);

using var result = await \_client.PostAsync($"{\_dalUrl}/Client", content);

var dalClient = await result.Content.ReadFromJsonAsync<Client>();

Console.WriteLine($"{dalClient?.Id}");

if (dalClient == null)

return BadRequest();

else

return dalClient;

}

}

}

### Пользовательский интерфейс

После разработки предыдущих двух слоёв и базы данных, можно приступить к созданию самого сайта.

Основой нашего веб-сайта являются HTML, CSS, JavaScript.

HTML (HyperText Markup Language)[16] - это язык разметки, используемый для создания и форматирования содержимого веб-страниц. HTML позволяет определять структуру документа, включая заголовки, абзацы, списки, таблицы, ссылки и многое другое. Он также позволяет добавлять изображения, аудио и видео на страницы. HTML-код может быть написан в любом текстовом редакторе или использованы специальные визуальные редакторы, такие как Adobe Dreamweaver[17]. и Microsoft Expression Web.

HTML использует теги и атрибуты для определения структуры и содержания документа. Теги заключают содержимое внутри них и могут быть вложенными друг в друга. Атрибуты предоставляют дополнительную информацию о теге и используются для настройки внешнего вида и поведения элементов на странице.

CSS — Cascading Style Sheets — это каскадные таблицы стилей. По сути — язык, который отвечает за описание внешнего вида HTML-документа. Подавляющее большинство современных веб-сайтов работают на основе связки HTML+CSS.

Если HTML структурирует контент на странице, то CSS позволяет отформатировать его, сделать более привлекательным для читателя. Изначально веб-разработчики использовали исключительно HTML — так было на заре развития интернет-технологий. С помощью разметки можно было выделить параграф, заголовок, изменить начертание текста. А большего и не требовалось.

Со временем этих возможностей стало мало — и появилась технология форматирования без изменения самого содержимого и структуры документа. CSS решил проблему «зоопарка» тегов форматирования, когда разные браузеры поддерживали разные теги. Их унификация и единая база упростили работу с веб-документами и облегчили жизнь веб-мастерам.

JavaScript[20] — это еще один интерфейсный язык программирования, который является одной из основных технологий Всемирной паутины, наряду с HTML и CSS. JavaScript используют на стороне клиента для поведения веб-страниц, часто включая сторонние библиотеки. В то время как HTML и CSS используются для управления представлением, форматированием и макетом, JavaScript используется для управления поведением различных веб-элементов.

Несмотря на то, что эти технологии легли в основу нашего сайта, реализован он был в основном через библиотеку вышеупомянутого JavaScript – React.

React[21] — это библиотека JavaScript с открытым кодом для создания внешних пользовательских интерфейсов. В отличие от других библиотек JavaScript, предоставляющих полноценную платформу приложений, React ориентируется исключительно на создание представлений приложений через инкапсулированные единицы (называются компонентами), которые сохраняют состояние и генерируют элементы пользовательского интерфейса. Вы можете разместить отдельный компонент на веб-странице или вложить иерархии компонентов для создания сложного пользовательского интерфейса.

Компоненты React часто пишутся на JavaScript и JSX[22] (JavaScript XML), который является расширением JavaScript, очень похожим на HTML и включающим некоторые функции синтаксиса для оптимизированного выполнения распространенных задач, например регистрации обработчиков событий для элементов пользовательского интерфейса. Компонент React реализует метод отрисовки, который возвращает код JSX, представляющий пользовательский интерфейс компонента. В веб-приложении код JSX, возвращаемый компонентом, преобразуется в поддерживаемый браузером код HTML, который затем обрабатывается для отображения браузером.

В React доступны различные способы использования REST API в приложениях, в том числе использование встроенного JavaScript- метода fetch() и Axios, который является HTTP-клиентом.

API fetch() является встроенным методом JavaScript, предназначенным для получения ресурсов от сервера или конечной точки API. Он определяет такие понятия, как CORS и семантика заголовка HTTP Origin, перенося их отдельные определения в другие места.

Метод fetch() API[23] всегда принимает обязательный аргумент, представляющий собой путь или URL-адрес ресурса, который вы хотите получить. Он возвращает метод, который указывает на ответ от запроса, независимо от того, был ли запрос успешным или нет. При желании вы также можете передать объект параметров инициализации в качестве второго аргумента.

После получения ответа доступно несколько встроенных методов, позволяющих определить, каково содержимое тела и как его следует обрабатывать.

Axios[24] — это библиотека JavaScript, используемая для выполнения HTTP-запросов с помощью XMLHttpRequests или HTTP клиентов в браузере и на сервере Node.js. Она предоставляет простой и удобный интерфейс для отправки HTTP-запросов и получения ответов, поддерживая такие функции, как интерцепторы, автоматическое преобразование JSON-данных и установку общих настроек для запросов. Axios также обладает многими другими возможностями, которые делают его инструментом первого выбора для многих разработчиков.

В качестве примера ниже будут приведены компоненты пользовательского интерфейса.

Компонент App.js является "главным компонентом", который содержит логику всего приложения. Для возможности перехода между страницами был настроен роутинг. Данный компонент показан в листинге 6:

Листинг 6 – Компонент App.js

function App() {

return (

<>

<BrowserRouter>

<NaviBar/>

<Routes>

<Route path="/" element={<Home/>} />

<Route path="/Company" element={<Company/>} />

<Route path="/Credits" element={<Credits/>} />

<Route path ="/Form" element = {<Form/>} />

</Routes>

</BrowserRouter>

<Footer />

</>

);

}

export default App;

Для реализации начальной страницы был создан компонент Home (листинг 7):

Листинг 7 – Компонент Home.js

const ColoredLine = ({ color }) => (

<hr

style={{

color: color,

backgroundColor: color,

height: 5

}}

/>)

export const Home = () => (

<>

<Slider />

<ColoredLine color="yellow" />

<Container style={{ paddingTop: '2rem', paddingBottom: '2rem' }} >

<Row>

<Col>

<Card style={{ width: '19rem' }}>

<Card.Img variant="top" src={credit2.substring(1)}

<Card.Body>

<Card.Title> Микрозайм </Card.Title>

<Card.Text>

Беспроцентный период до 120 дней

</Card.Text>

<Link to="/Form">

<Button variant="warning">

Подать

</Button>

</Link>

</Card.Body>

</Card>

</Col>

<Col>

<Card style={{ width: '19rem' }}>

<Card.Img variant="top" src='https://moneyman.ru/wp-content/uploads/2020/07/Kreditnaya-ekspress-karta.jpg' />

<Card.Body>

<Card.Title> Потребительский кредит </Card.Title>

<Card.Text>

Беспроцентный период до 90 дней

</Card.Text>

<Link to="/Form">

<Button variant="warning">

Подать заявку

</Button>

</Link>

</Card.Body>

</Card>

</Col>

<Col>

<Card style={{ width: '19rem' }}>

<Card.Img variant="top" src={ipoteka} />

<Card.Body>

<Card.Title> Ипотечное кредитование </Card.Title>

<Card.Text>

Беспроцентный период до 120 дней

</Card.Text>

<Link to="/Form">

<Button variant="warning">

Подать заявку

</Button>

</Link>

</Card.Body>

</Card>

</Col>

</Row>

</Container>

<ColoredLine color="yellow" />

<Container style={{ marginBottom: '100px' }}>

<Row>

<Col mr={7}>

<img src={people} height={400} />

</Col>

<Col mr={7}>

<div>

На сегодняшний день Тинькок является абсолютным лидером российской банковской системы.

</div>

</Col>

</Row>

</Container>

</>

В компоненте Form1 используется метод fetch, с помощью которого данные, заполненные пользователем в форму, передаются на слой Domain. Данный компонент приведен в листинге 8:

Листинг 8 – Компонент Form1.js

export class Form1 extends Component {

static displayName = Form1.name;

constructor(props){

super(props);

this.state = {

fio :'',

phone : '',

adress:''

}

this.onInputChange = this.onInputChange.bind(this);

this.sendClientData = this.sendClientData.bind(this);

}

onInputChange(event) {

this.setState({

[event.target.name]: event.target.value

});

}

render() {

return (

<div>

<div class="form-row">

<div class="form-group col-md-6">

<label for="inputName">ФИО</label>

<input class="form-control" type="text" id="inputfio" placeholder="" value={this.state.fio} name="fio" onChange={this.onInputChange} />

</div>

<div class="form-group col-md-6">

<label for="inputPassword4">Телефон</label>

<input class="form-control" id="inputPhone" type="text" value={this.state.phone} name="phone" onChange={this.onInputChange} />

</div>

</div>

<div class="form-group col-md-6">

<label for="inputAddress">Адрес</label>

<input class="form-control" id="inputAddress" type="text" placeholder="Например : ул.Интернациональная д.14 кв.71" value={this.state.adress} name="adress" onChange={this.onInputChange} />

</div>

<div>

<div class="form-group col-md-4">

<label for="inputProduct">Продукт</label>

<select id="inputState" class="form-control">

<option selected>Выбрать</option>

<option>Микрозайм</option>

<option>Потребительский кредит</option>

<option>Ипотечное кредитование</option>

</select>

</div>

</div>

<button class="f-button" onClick={this.sendClientData}> Продолжить</button>

</div>

);

}

async sendClientData() {

let client = {

"FullName": this.state.fio,

"Phone": this.state.phone,

"Address": this.state.adress

};

console.log(client);

const reponse = await fetch('/api/Domain/Client', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'Authorization': 'Basic dGVzdDp0ZXN0Mg=='

},

body: JSON.stringify(client)

});

const data = await reponse.json();

console.log(data);

}

}

После выполнения всех ранее названных действий, информационно-справочная система является готовой.

### Тестирование

Поскольку, говоря о текущем проекте, мы имеем дело с многоуровневым приложением, тестировать его работу стоит соответствующе - отдельно проверяя работу каждого слоя, с последующей проверкой совместной работы всех компонентов вместе. Хорошим инструментом для тестирования слоя данных и бизнес-логики является веб-Фреймворк для интерактивной документации - Swagger UI[25].

Для начала оценим корректность функционирования DAL на примере таблицы Client (рисунок 11):

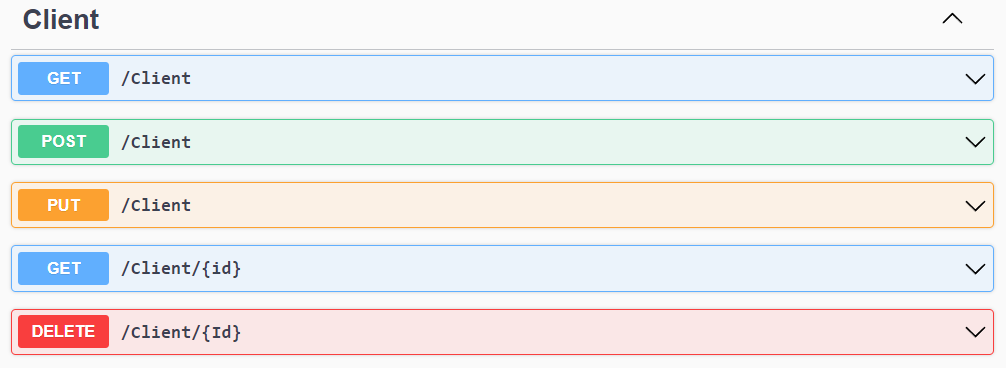
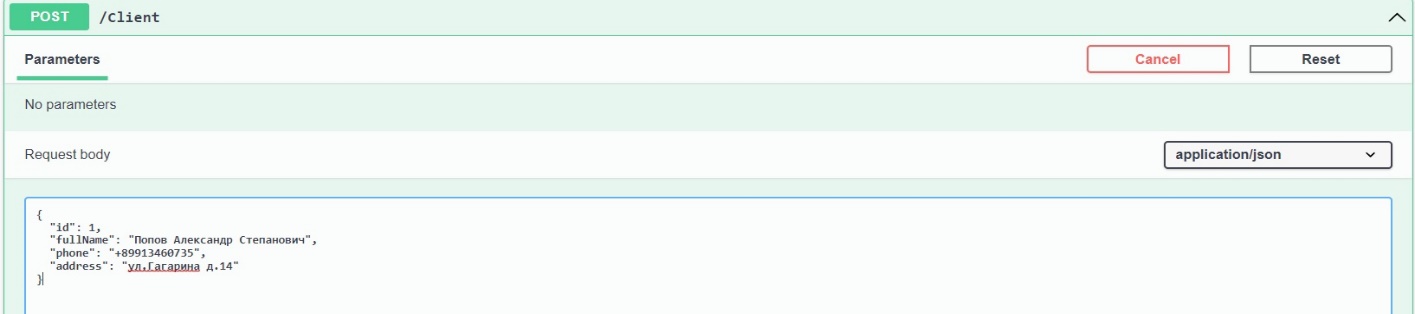


Рисунок 11 – Таблица Client в Swagger UI

Для этого пройдёмся по всем возможным операциям, начиная с POST. Нажав кнопку «Try it out», создадим нового клиента:

* id: «1»;
* fullName: «Попов Александр Степанович»;
* phone: «+89913460735»;
* address:«ул.Гагарина д.14»

Выполнение операции POST (рисунок 12).

Рисунок 12 – Выполнение операции POST

Проверим операцию создания нового клиента с помощью метода GET (рисунок 13).

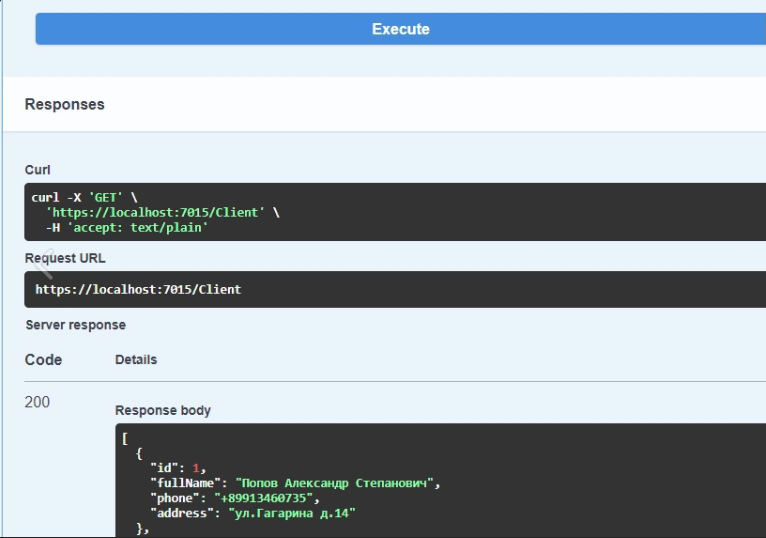


Рисунок 13 – Проверка

Проверим выполнения операции GET (рисунок 14).

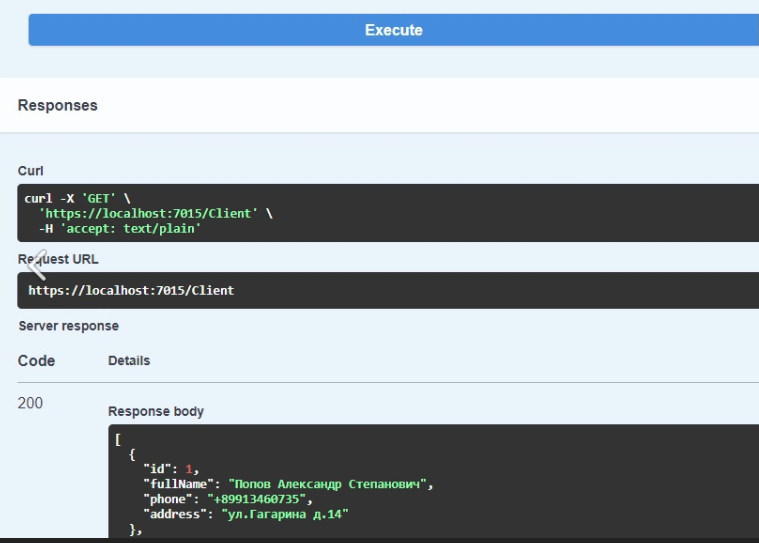


Рисунок 14 – выполнение операции GET

Далее проверим выполнение операции PUT, изменив адрес с «ул. Гагарина д.14» на «ул. Интернациональная д.14» (рисунок 15).



Рисунок 15 – Выполнение операции PUT

Проверим выполнение операции PUT с помощью GET метода (рисунок 16).

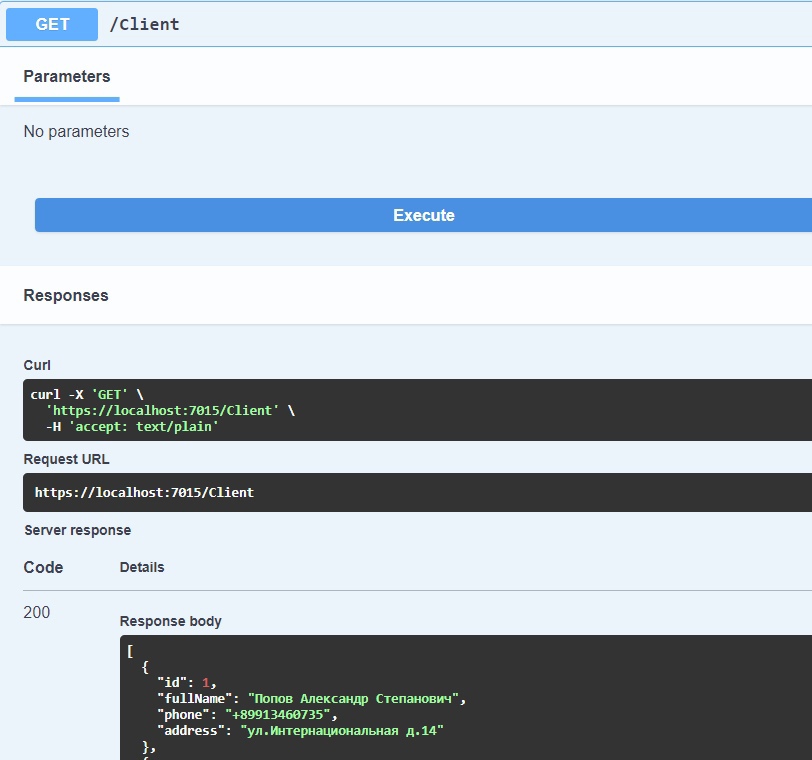


Рисунок 16 – Проверка

И последним этапом проверим метод DELETE, удалив пользователя с идентификатором «1» (рисунок 17).

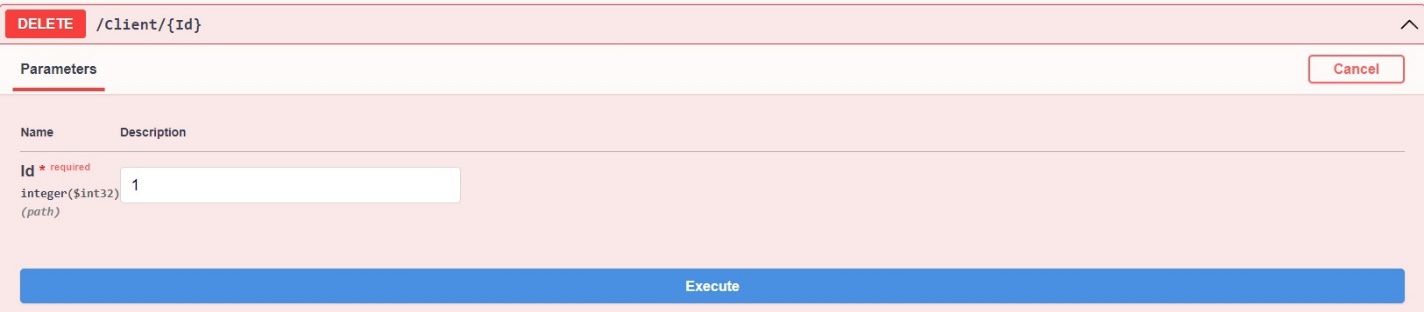


Рисунок 17 – Выполнение операции DELETE

Проверим выполнение операции DELETE с помощью GET метода (рисунок 18).



Рисунок 18 – Проверка

Выполнение вышеуказанных действий позволяет сделать вывод, что слой доступа данных работает корректно.

Проверка слоя бизнес-логики будет выполняться схожим образом, через веб-Фреймворк Swagger UI на примере таблицы Clients (Рисунок 19):

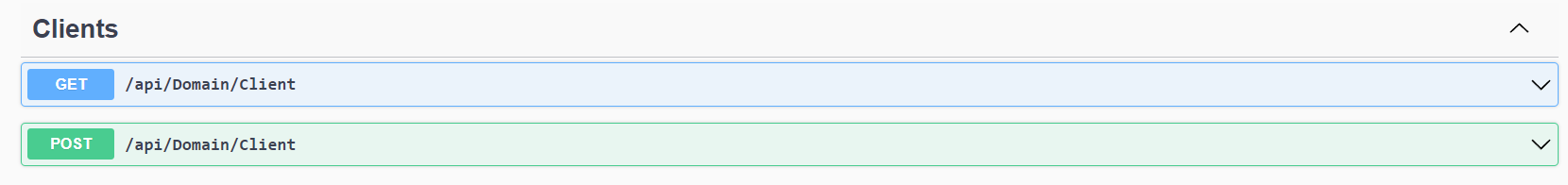


Рисунок 19 – Таблица Client в Swagger UI

Для этого пройдёмся по всем возможным операциям, начиная с POST. Нажав кнопку «Try it out», создадим нового клиента:

* id: «2»;
* fullName: «Сахаров Никита Михайлович»;
* phone: «+7891654534»;
* address:«ул.Полетаева д.30к1»

Выполнение операции POST (рисунок 20).

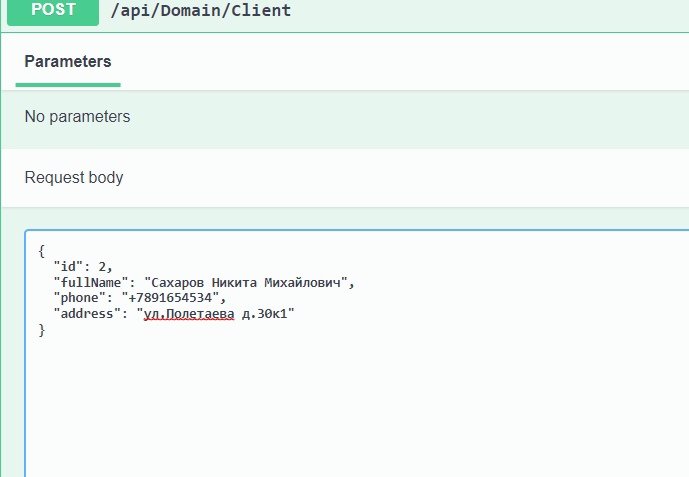


Рисунок 20 – Выполнение операции POST

Проверим операцию создания нового клиента с помощью метода GET (рисунок 21).

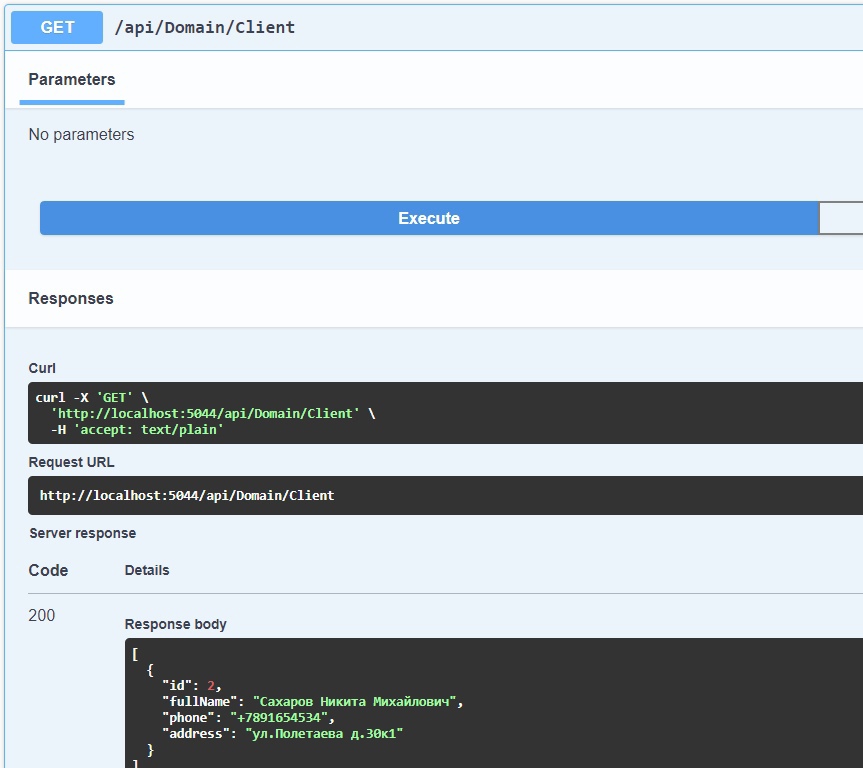


Рисунок 21 – Проверка

Следующим этапом тестирования является просмотр веб-сайта (Рисунки 22-25):



Рисунок 22 – Главная страница



Рисунок 23 – Главная страница

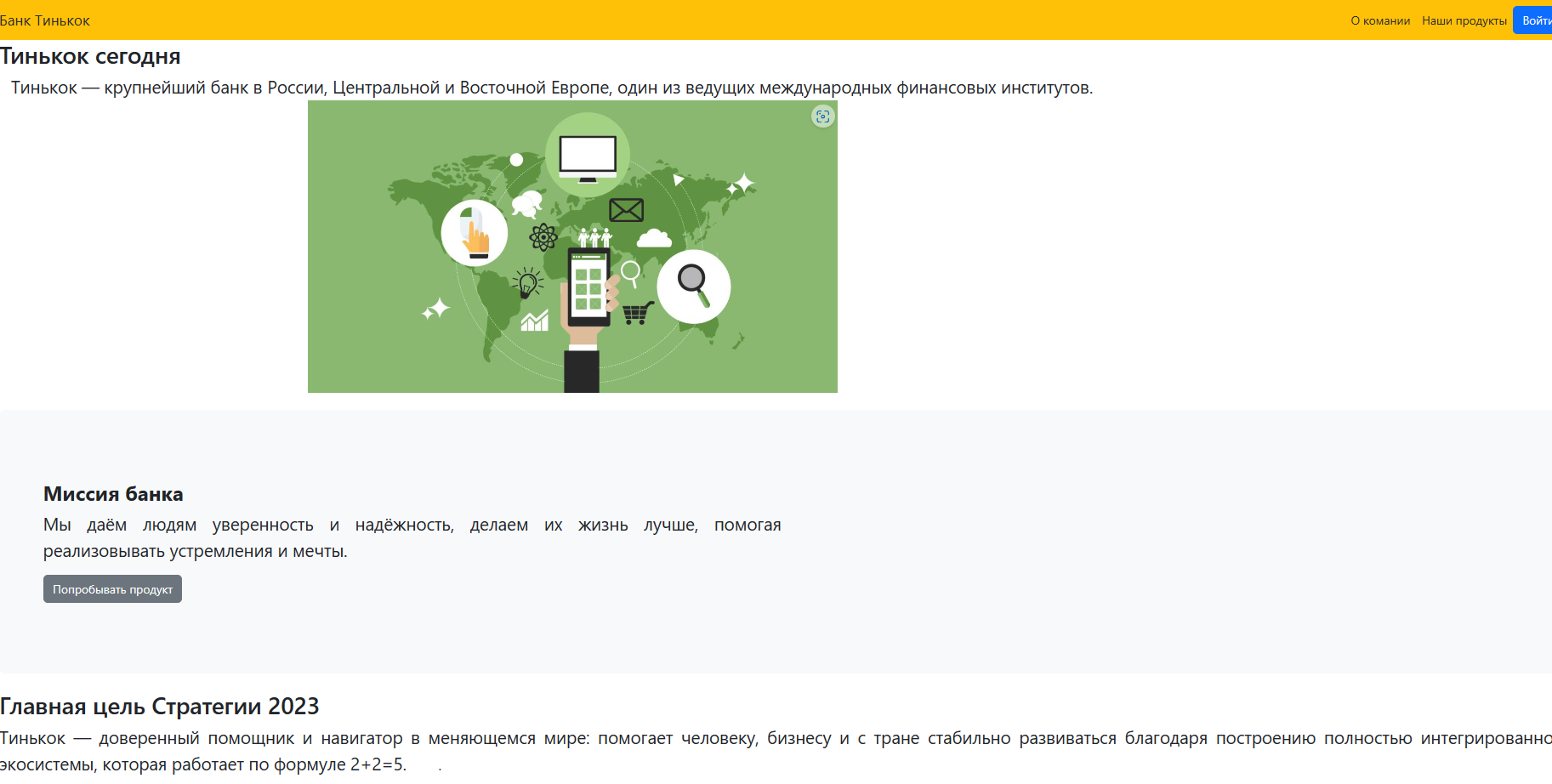


Рисунок 24 – Вкладка «О компании»

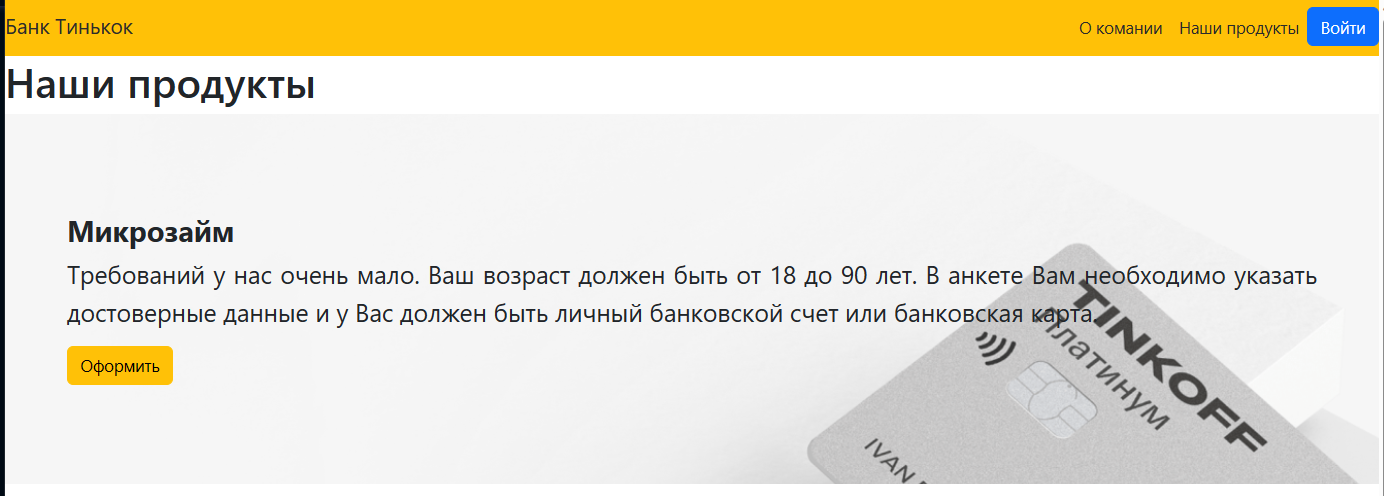


Рисунок 25 – Вкладка «Наши продукты»

Для заказа товара следует заполнить форму, в которой требуется указать свои данные. После ввода данных следует нажать на кнопку «Отправить». После этого данные перенесутся в базу данных (рисунок 26-28).

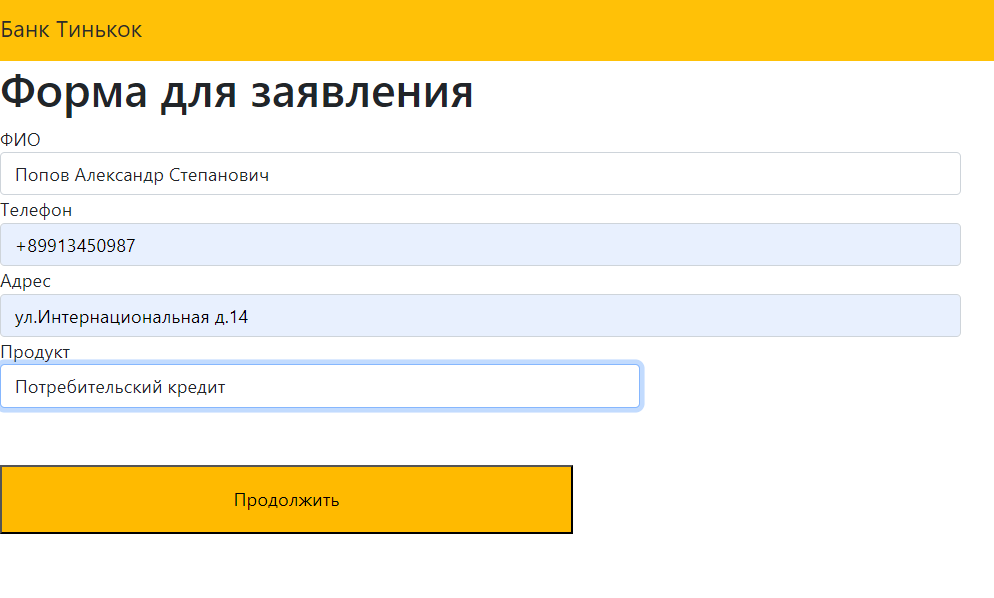


Рисунок 26 – Форма для заявления

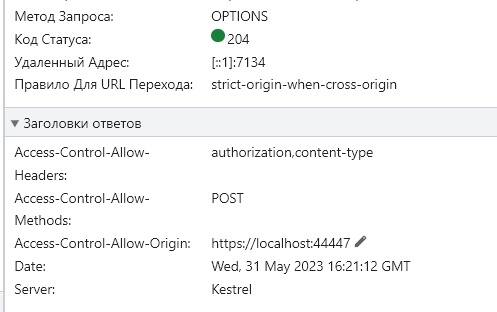


Рисунок 27 – Отправка

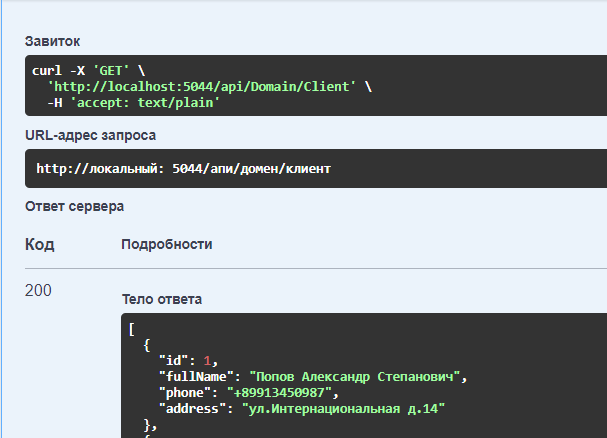


Рисунок 28 – Проверка

После проверки работы всех модулей можно заключить, что система работает корректно в соответствие с поставленными задачами.

# **Заключение**

Создание Информационно-справочной системы представляет собой сложный процесс проектирования, разработки и тестирования. Целью проектирования являются обозначение контуров и деталей будущей системы и создание соответствующей документации. В процессе разработки выстраивается сам объект, устанавливается его архитектура, связи между компонентами, характерные особенности. Конечным этапом является тестирование системы, в ходе которого выявляются все возможные недостатки и ошибки, с последующим их устранением.

В данной курсовой работе был разработан проект информационной системы «Кредит в банке»

В ходе курсовой работы были решены следующие поставленные задачи:

- выполнен анализ предметной области;

- смоделирована предметная область;

- разработаны все слои системы;

-протестирован функционал конечного продукта.

### Список используемой литературы

1. Управление ликвидностью коммерческого банка // Учебные материалы [электронный ресурс] / URL: https://works.doklad.ru/view/rA5lnI\_5cM4.html (дата обращения: 24.11.2022).
2. Сущность, функции и формы кредита / [studfile.net](https://studfile.net) [электронный ресурс] / URL: <https://studfile.net/preview/7377817/page:2/>
3. Сбербанк // Сбербанк [электронный ресурс] / URL: https://online.sberbank.ru/CSAFront/index.do (дата обращения: 24.11.2022).
4. Тинькофф // [электронный ресурс] / Тинькофф URL: https://www.tinkoff.ru/cards/debit-cards/tinkoff-black/ (дата обращения: 26.11.2022).
5. Рейтинг товаров [электронный ресурс] / URL: https://everychild.ru/reyting/chto-luchshe-tinkoff-ili-sberbank-sravnenie/ (дата обращения: 26.05.2023).
6. Что такое BPMN-диаграмма и зачем она нужна в разработке // URL: https://vc.ru/dev/273096-chto-takoe-bpmn-diagramma-i-zachem-ona-nuzhna-v-razrabotke (дата обращения: 26.05.2023).
7. Диаграмма прецедентов [электронный ресурс] // URL: https://kolei.ru/ekolesnikov/PiRIS/src/master/articles/5\_1\_1\_10\_uml\_use\_case.md (дата обращения: 26.11.2022).
8. Краткий обзор языка C# // Microsoft [электронный ресурс] / URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/ (дата обращения: 26.05.2023).
9. Введение в ASP.NET Core MVC // Metanit [электронный ресурс] / URL: https://metanit.com/sharp/aspnetmvc/1.1.php (дата обращения: 26.05.2023).
10. Как используют базы данных // Yandex [электронный ресурс] / URL:https://yandex.ru/q/question/kak\_ispolzuiut\_bazy\_dannykh\_f0bdf7a7/?answer\_id=7286c250-961b-48a8-be63-d8731cc30aad&w\_origin=question\_control#:~:text=%D0%91%D0%B0%D0%B7%D0%B0%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D83% (дата обращения: 26.05.2023).
11. Что такое SQL Server Management Studio (SSMS)? // Microsoft [электронный ресурс] / URL: earn.microsoft.com/RU-RU/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15&viewFallbackFrom=sql-server-linux-ver15 (дата обращения: 23.05.2023).
12. Уровень доступа к данным // GeeksforGeeks [электронный ресурс] / URL: https://www.geeksforgeeks.org/data-access-layer/ (дата обращения: (02.02.2023).
13. Что такое CRUD-операции // Best Programmer [электронный ресурс] / URL: https://bestprogrammer.ru/izuchenie/chto-takoe-crud-operatsii (дата обращения: (05.02.2023).
14. Многоуровневая архитектура // Metanit [электронный ресурс] / URL: https://metanit.com/sharp/mvc5/23.5.php (дата обращения: (06.04.2023).
15. Applying Domain-Driven Design and Patterns: With Examples in C# and .NET // Oreilly [электронный ресурс] / URL: https://www.oreilly.com/library/view/applying-domain-driven-design/0321268202/ (дата обращения: (06.04.2023).
16. Основы HTML // MDN Web docks URL: https://developer.mozilla.org/ru/docs/Learn/Getting\_started\_with\_the\_web/HTML\_basics (дата обращения: (02.04.2023).
17. Adobe [электронный ресурс] / URL: https://www.adobe.com/ru/products/dreamweaver.htm (дата обращения: (05.04.2023).
18. Что такое CSS: объясняем простыми словами // GeekBrains [электронный ресурс] / URL: https://gb.ru/posts/chto-takoe-css-obyasnyaem-prostymi-slovami (дата обращения: (05.05.2023)).
19. Обзор протокола HTTP // MDN Web Docs [электронный ресурс]

/ URL: https://developer.mozilla.org/ru/docs/Web/HTTP/Overview (дата обращения: (12.05.2023)).

1. Для чего используется JavaScript? // Best Programmaer [электронный ресурс] / URL: https://bestprogrammer.ru/programmirovanie-i-razrabotka/dlya-chego-ispolzuetsya-javascript (дата обращения: (13.05.2023)).
2. Общие сведения о React // Microsoft [электронный ресурс] / URL: https://learn.microsoft.com/ru-ru/windows/dev-environment/javascript/react-overview (дата обращения: (14.05.2023)).
3. Учебник. Создание приложения Node.js и React в Visual Studio // Microsoft [электронный ресурс] / URL: learn.microsoft.com/ru-ru/visualstudio/javascript/tutorial-nodejs-with-react-and-jsx?view=vs-2022 (дата обращения: (14.05.2023)).
4. Применение в React REST API с помощью Fetch и Axios // Интернет Технологии [электронный ресурс] / URL: https://www.internet-technologies.ru/articles/primenenie-rest-api-s-pom-fetch-i-axios.html (дата обращения: (20.05.2023)).
5. Axios [электронный ресурс] / URL: https://axios-http.com/ru/ (дата обращения: (20.05.2023)).
6. Документация по веб-API ASP.NET Core с использованием Swagger (OpenAPI) // Microsoft [электронный ресурс] /URL: https://learn.microsoft.com/ru-ru/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-5.0 (дата обращения: (06.09.2022)).