

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

Кафедра Автоматизированных Систем Управления

К защите

Руководитель работы:

дата, подпись

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

по дисциплине

«Технологии программирования»

Тема:

«Разработка информационно-поисковой системы «Вклад в банке»»

Выполнил студент группы 135

Бобычев А.Г.

дата сдачи на проверку, подпись

Руководитель работы

Преподаватель каф. АСУ

Аникеев Д.В.

оценка

дата защиты, подпись

Рязань 2023

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 — Бобычев Александр Геннадьевич группа – 135

1. Тема: Разработка информационно-поисковой системы «Вклад в банке»
2. Срок представления работы к защите - «_» _____ 2023г.
3. Исходные данные для проектирования:
 - 3.1. Язык программирования – C#.
 - 3.2. СУБД — MS SQL Compact Edition, MS SQL Express Edition
 - 3.3. Технология объектного связывания данных – Entity Framework.
 - 3.4. Общее количество таблиц - не менее 5.
 - 3.5. Обязательные операции с данными — просмотр записей таблиц в виде списка, сортировка, поиск, добавление, изменение, удаление записей.
 - 3.6. 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. /

Задание принял к исполнению _____ « 14 » февраля 2023г

Содержание

Введение	4
1 Описание предметной области	5
1.1 Семантическое описание предметной области.....	5
1.2 Описание BPMN модели.....	6
1.3 Use Case диаграмма	8
2 Сравнительный анализ существующих решений.....	11
3 Разработка системы	15
3.1 Архитектура информационной системы	15
3.2 Диаграмма сущностей	16
3.2 База данных	18
3.3 Data Access Layer	20
3.4 Слой бизнес-логики	27
3.5 Пользовательский интерфейс	29
4 Тестирование системы.....	38
Заключение	54
Список используемых источников.....	55

Введение

В настоящее время, происходит активная цифровизация банковского сектора в России [1]. Это обусловлено необходимостью решать определённые задачи, например: оптимизация удаленной работы банковских сотрудников, рост операций, совершаемых в онлайн, стремление увеличить прибыль, упрощение доступа к услугам банка, развитие каналов дистанционных продаж, борьба с мошенниками, расширение применения технологий искусственного интеллекта, переход к управлению на основе данных и т.д. Особенно процесс цифровизации подтолкнула пандемия Covid-19 [2]. Безусловно, это коснулось и рассматриваемой темы – оформление вклада в банке.

Обслуживание клиентов, желающих оформить вклад, должно проходить быстро, поэтому и регистрация клиента, как отдельный его этап, не должна занимать много времени. Следовательно, существует необходимость в ускорении данного процесса. С учётом вышеупомянутых тенденций, на подобную роль хорошо подходят информационные системы [3].

Таким образом, очевидным для качественной работы банков является разработка и внедрение автоматизированной системы управления принятия клиентов на обслуживание. Комплексная автоматизация управления посредством информационных систем на сегодняшний день - один из самых эффективных и функциональных инструментов систематизации работы ключевых бизнес-процессов [4].

1 Описание предметной области

1.1 Семантическое описание предметной области

Банковский вклад - денежные средства в валюте Российской Федерации или иностранной валюте, размещаемые вкладчиками или в их пользу в банке на территории Российской Федерации на основании договора банковского вклада или договора банковского счета, включая капитализированные (причисленные) проценты на сумму вклада [5].

Иными словами, банковский вклад или депозит — это сумма денежных средств, которую человек на определенное время отдает на хранение в банк, а затем забирает обратно. Пока эти деньги находятся у банка, он может распоряжаться ими в своих целях. Например, выдавать клиентам кредиты, торговать на фондовых рынках и валютных биржах, что принесет банку коммерческий доход. За эту возможность использовать вложенные средства банки готовы платить, поэтому по окончании срока действия депозита вкладчик получит свои деньги с процентами.

Для вкладчиков банковский депозит — это возможность обезопасить свои сбережения, сохранить их и даже увеличить, а для кредитных организаций — способ получить свободные средства, которые будут работать и приносить доход.

Вклады различаются по следующим параметрам.

1. **Срок.** Вклад открывается на оговоренный срок или на неопределенное время. В первом случае — это срочный вклад, во втором — до востребования или бессрочный. Наиболее доходными являются срочные вклады с длительным сроком размещения — за них банки предлагают наиболее высокую процентную ставку.
2. **Валюта.** Открыть депозит можно в рублях, в одной иностранной валюте или сразу в нескольких, однако последнюю опцию предоставляют не все кредитные организации. Валютные вклады

обычно имеют меньшую доходность, чем рублевые, поскольку из-за колебаний курсов они несут больше рисков для банка. Возможность пополнения или снятия.

Открытие вкладов в банках на протяжении многих столетий представляло собой визит в финансовую организацию с целью решения данной задачи. Этот процесс можно хорошо описать, воспользовавшись BPMN диаграммой.

Данная диаграмма отображает бизнес-процессы с помощью блок-схем. Она показывает в какой последовательности совершаются рабочие действия и перемещаются потоки информации. Диаграмма показана на рисунке 1.

1.2 Описание BPMN модели

В начале сотрудник вводит данные человека, который подал документы о получении вклада. **Клиент** приходит в банк и выбирает через терминал нужную ему услугу-оформление вклада.

После выбора услуги ему выдаётся талон, с номером, получив талон, **клиент** ожидает своей очереди на приём к **консультанту**.

Дождавшись своей очереди, **клиент** подходит к нужному окну и начинает разговор с **консультантом**.

В конечном итоге, **клиент** и **консультант** или приходят к согласию в отношении того какой вклад оформить, или **клиент** уходит, не получив устроившего его предложения.

В случае, если решение оформить конкретный вклад принято, то начинается процедура верификации личности **сотрудником банка**, определения допустимо ли выполнять финансовые операции с этим физическим лицом. Это определяется посредством проверки паспорта, как на достоверность (с помощью ультрафиолетового излучения), так и на предмет истечения срока годности и/или механических повреждений.

После проверки, при отсутствии серьёзных замечаний, паспорт сканируется и вносится в базу, где проходит последний этап промежуточной верификации-сопоставление клиента с базой лиц, подпадающих под действие Федерального закона №115 "О противодействии легализации (отмыванию)

доходов, полученных преступным путем, и финансированию терроризма" [6], который регулирует финансовый оборот в отношении лиц, замешанных в мошеннических или террористических активностях. В случае, если человек находится в этом списке, банк имеет право отказать в предоставлении услуг без объяснения причин.

Если **клиент** прошёл данный этап, то данные о нём и открытом на его имя вкладе, передаются в **информационную систему банка**, которая снова выдаётся талон **клиенту**, на этот раз, связанный с переадресацией в кассу.

После повторного ожидания, **работник кассы** принимается за окончательную проверку документов, а вместе с ним и денег, которые физ. лицо решило внести на счёт. В случае обнаружения несоответствия в документах **клиенту** отказывают в предоставлении услуг, а процесс создания вклада обнуляется. Однако, если окажется, что человек принёс фальшивые купюры, то банк имеет право вызвать полицию и задержать человека до выяснения обстоятельств.

Если всё проходит успешно, то **клиенту** выдают чек, договор и его документы обратно, а в **информационной системе** на его имя создаётся счёт, на который вносятся деньги.

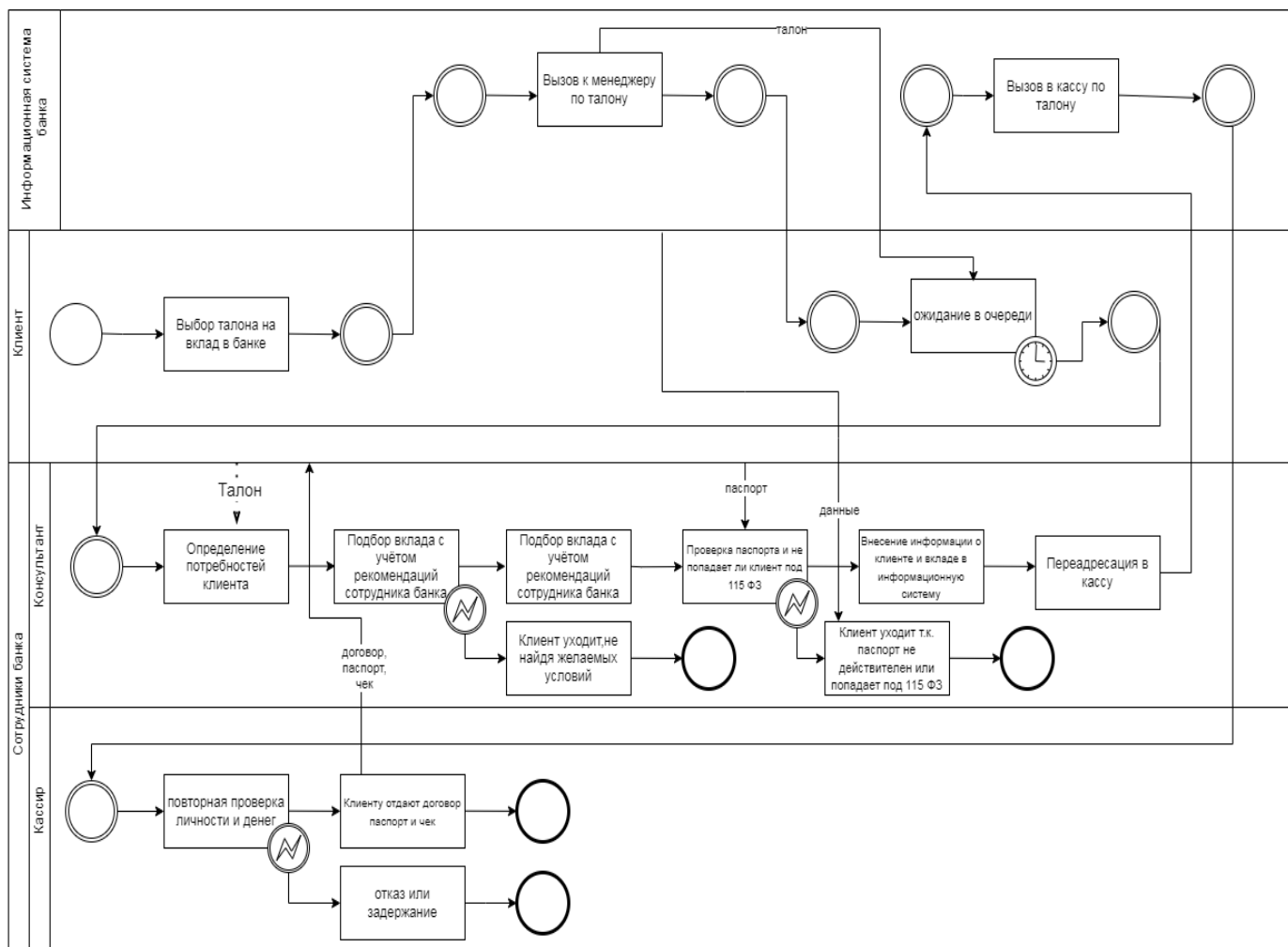


Рисунок 1 - BPMN модель

1.3 Use Case диаграмма

Иным образом данный процесс можно представить с помощью Use Case диаграммы. Use case диаграммы являются графическим представлением функциональных требований к системе. Они используются для описания сценариев взаимодействия между актерами и системой. Основной целью данных диаграмм является улучшение понимания целей, потребностей и ожиданий пользователей, а также спецификации и документирования функционала, который должен быть реализован в системе.

Диаграмма показана на рисунке 2.

Описание объектов и их взаимодействия в USE CASE диаграмме:

— «Клиент» обращается в терминал в здании банка совершая, тем самым, факт обращения. Дожидаясь своей очереди, он указывает желаемые условия вклада, и в ходе взаимодействия с клиентским менеджером выбирает подходящую опцию. При условии прохождения проверки личности, клиент может внести средства на счёт, чтобы после проверки уже самих средств, получить договор о оформлении вклада и чек.

— Клиентский менеджер помогает клиенту с выбором вклада, проводит первичную проверку клиента, в случае успеха запускает процедуру оформления вклада, внося запись о вкладе в реестр. В противном случае, клиентский менеджер имеет право отказать в выдаче вклада.

— Проверяющий менеджер проверяет корректность выполняемой операции, документы клиента и действия других сотрудников, например, клиентского менеджера или кассира. В случае обнаружения нарушений в ходе операции со стороны сотрудника, или некорректности документов клиента, проверяющий менеджер имеет право прервать операцию и отказать в выдаче вклада.

— Кассир проверяет деньги, которые клиент вносит на счёт, в случае успеха, кассир отвечает за внесение денег на счёт и оформление договора с чеком. В противном случае, кассир имеет право отказать в выдаче вклада.



Рисунок 2 - Use Case модель

Как показывают вышеописанные схемы и модели, несмотря на формальное наличие информационно-поисковой системы, во время процесса по оформлению вклада основную роль играют люди, причём, как правило, не менее трёх человек со стороны банковского персонала. Это всё делает оформление вклада, непосредственно в банке громоздким процессом, создавая параллельно необходимость в поиске других способов проведения этой операции.

2 Сравнительный анализ существующих решений

В настоящее время, несмотря на невозможность полностью исключить необходимость непосредственного присутствия в банке, многие банковские операции возможно осуществить дистанционно, с помощью соответствующего приложения или веб-сайта.

Как было сказано ранее, это является частью общего стремления крупных банков в России в освоении новых цифровых технологий, в первую очередь, для увеличения прибыли. Проведём сравнение существующих информационных систем банков России на примере трёх крупнейших представителей отрасли – Сбербанк, Тинькофф и Альфа-Банк [7]. Для этого выделим 4 основных признака:

- 1) наличие API;
- 2) масштабируемость и гибкость;
- 3) надежность и безопасность;
- 4) облачные технологии.

Наличие API [8].

Все три банка предоставляют API для удобства взаимодействия с клиентами и сторонними разработчиками. Однако, различия между ними заключаются в способе предоставления API и доступность инструментов. Например, в Тинькофф и Сбербанке API представлены в виде открытого доступа, в то время как Альфа-Банк направляет запрос на получение доступа к API на рассмотрение комиссии.

Масштабируемость и гибкость.

Тинькофф на сегодняшний день является самой гибкой и масштабируемой финтех-системой, которая позволяет внедрять новые продукты и услуги быстро и без задержек [9]. Сбербанк также показал себя в этом направлении, благодаря своим мощным ресурсам. В то же время, Альфа-

Банк отставал от них до недавнего времени своей менее гибкой и неспособной к масштабированию системой.

Надежность и безопасность.

С точки зрения надежности и безопасности, все три банка находятся на высоком уровне [10]. В Тинькофф и Сбербанке применяются высокие стандарты безопасности в сочетании с использованием современных технологий, таких как blockchain. Альфа-Банк также инвестировал средства для улучшения безопасности и создания бесконтактных технологий, таких как NFC-технологии [11].

Облачные технологии.

С точки зрения облачных технологий, все три банка являются инновационными и используют облачные платформы для управления своими проектами. В частности, Тинькофф использует облачную платформу AWS, а Сбербанк и Альфа-Банк используют свои собственные облачные платформы [12].

После общего анализа информационных технологий в названных банках, перейдём к описанию их подходов в вопросе выдачи вкладов.

Открытие вкладов онлайн и в приложении в Сбербанке, Тинькофф и Альфа-Банке использует схожие технические решения, но есть и отличия.

Сбербанк предоставляет возможность открытия вкладов через сайт и мобильное приложение. В обоих случаях клиент проходит процедуру идентификации, после чего подтверждает выбранные параметры вклада и производит перевод денежных средств на вклад. Сайт и приложение Сбербанка имеют высокую степень защиты от взлома и шифрование передаваемых данных.

Тинькофф также предлагает возможность открытия вкладов через сайт и мобильное приложение. Однако в отличие от Сбербанка, у Тинькоффа клиенты могут выбирать несколько вариантов вклада с различной процентной ставкой и сроком. Подобно Сбербанку, сайт и приложение Тинькоффа используют шифрование для передачи информации.

Альфа-Банк также предоставляет возможность открытия вкладов через сайт и мобильное приложение, но на сайте предлагается больше вариантов вкладов с различными условиями. Вклады от Альфа-Банка имеют возможность выбрать, как процентную ставку, так и срок хранения. Приложение и сайт Альфа-Банка также используют шифрование для защиты данных.

Итоговое сравнение решений Сбербанка, Тинькофф и Альфа-Банка показывает, что все банки предлагают возможность открытия вкладов онлайн и в приложении с высоким уровнем безопасности и довольно похожей функциональностью. Основное различие заключается в количестве доступных вариантов вкладов и для каких зарегистрированных в банке устройств доступно открытие вклада через мобильное приложение.

Таким образом, при всех отличиях, все крупные банки России используют онлайн и мобильный банкинг. Оба варианта имеют схожий функционал - они позволяют клиентам совершать различные операции с имеющимися банковскими картами, получать достоверную информацию обо всех операциях, а также совершать платежи, переводы, оплату различных сервисов и погашать свои кредиты и займы посредством приложения или веб-сайта. Безусловно мобильные приложения являются более популярными, чем сайты, так, например мобильным приложением Сбербанк пользуется 60 миллионов пользователей, а сайтом всего лишь 10 миллионов [13], это связано с большей надёжностью мобильных приложений, и их доступностью [14]. Однако разработка целого кроссплатформенного приложения представляется гораздо более трудоёмкой и, следовательно, дорогой задачей. Очевидно, что по схожему принципу не стоит брать в учет и разработку полноценной системы управления финансовыми услугами (финтех), поскольку, при избыточном, для нашего случая, функционале, на её создание и поддержку уйдут несопоставимо большие ресурсы.

Целью курсовой работы является создание системы для взаимодействия с конечным потребителем, обеспечить для него возможность оформить вклад в банке, в связи с чем, можно заключить, что оптимальным вариантом является именно разработка информационно-поисковой системы онлайн-банкинга.

3 Разработка системы

3.1 Архитектура информационной системы

В разработке системы в соответствии с предыдущими схемами и диаграммами будет использоваться трехзвенная архитектура приложения, состоящая из четырех слоев.

Первый слой находится на уровне клиента и является слоем представления, в котором реализуется пользовательский интерфейс для взаимодействия с пользователем и отображение данных.

Второй слой находится на уровне сервера приложений и представляет собой слой бизнес-логики, отвечающий за выполнение операций над данными и обеспечивающий их правильность и целостность. Он должен быть написан таким образом, чтобы его можно было легко использовать другими частями системы.

Третий слой находится на уровне сервера приложений и является слоем доступа к данным, который управляет доступом к данным в приложении и обеспечивает их взаимодействие с системой хранения данных. Он предоставляет интерфейс для чтения, записи, изменения и удаления данных и использует различные компоненты, такие как модели данных и объекты доступа к данным.

Четвертый слой находится на уровне сервера БД и составляет слой хранения данных, который отвечает за сохранение и управление данными в постоянном хранилище. Он используется вместе с другими компонентами приложения, чтобы обеспечить его правильное функционирование и учитывает требования приложения при выборе типа хранилища данных, такого как реляционная база данных, NoSQL-хранилище или файловое хранилище.

Получившуюся архитектуру можно изобразить в виде диаграммы компонентов с помощью языка UML [15] (рисунок 3)

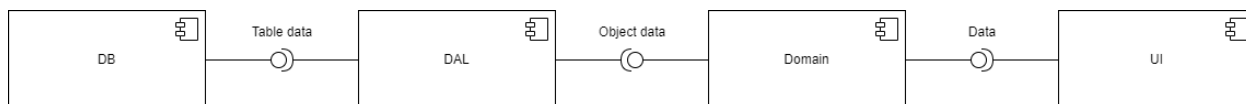


Рисунок 3 – Диаграмма компонентов

После описания базовых понятий становится возможным перейти к первому этапу непосредственного создания системы – проектированию и оформлению Базы данных [16]. Для этого воспользуемся диаграммой сущностей [17].

3.2 Диаграмма сущностей

Диаграмма сущностей (или ER-диаграмма) в базах данных используется для визуализации структуры данных, которые используются в системе. Она позволяет описать сущности в системе и отношения между ними, а также описать атрибуты каждой сущности. Главная цель диаграммы сущностей — это создание единого языка для коммуникации между разработчиками и заказчиками в процессе разработки системы.

Для разработки информационной системы вкладов в банке была создана ER диаграмма, которая показывает предполагаемую для программной реализации структуру базы данных. Всего получилось шесть сущностей.

Сущность – Клиент. В данной таблице хранится информация о клиентах банка. Каждый клиент имеет свой уникальный идентификатор. Для каждого клиента должны быть указаны его паспорт и талон.

Сущность – Факт обращения. В данной таблице хранится информация об фактах обращения в банк с целью оформить кредит. У каждого факта обращения есть свой уникальный идентификатор. Для каждого факта обращения должны быть указаны: паспорт обращающегося, талон с номером обращения, идентификатор клиента, совершающего его и идентификатор сотрудника, работающего с обращением.

Сущность – Вклады. В данной таблице хранится вся информация об опциях вкладов в банке которые можно оформить. Каждый вклад имеет свой

уникальный идентификатор. Для каждого вклада должны быть указаны: название, длительность, размер, тип начисления процентной ставки и валюта.

Сущность – Существующие вклады. В данной таблице хранится информация о уже оформленных вкладах в банке. Каждый существующий вклад имеет свой уникальный идентификатор. Для каждого существующего вклада должны быть указаны: идентификатор вклада в списке опций вкладов, дата оформления, место оформления, идентификатор клиента, который оформил вклад и идентификатор сотрудника с ним работавшего.

Сущность – Сотрудник. В данной таблице хранится информация о сотрудниках, работающих в банке. Каждый сотрудник имеет свой уникальный идентификатор. Для каждого сотрудника должны быть указаны его имя и идентификатор должности им занимаемой.

Сущность – Должность. В данной таблице хранится информация о должностях сотрудников в банке. Каждая должность имеет свой уникальный идентификатор. Для каждой должности должно быть указано её наименование.

Для того чтобы связать эти сущности необходимо установить отношения между ними. Один клиент может оформить как один, так и много вкладов, поэтому отношение клиент-существующие вклады равно один ко многим. Так же один клиент может совершить много обращений, поэтому отношение клиент-факт обращения равно один ко многим. Один вклад может быть оформлен несколько раз, поэтому отношение вклад-существующий вклад равно один ко многим, при этом он также может быть не оформлен ни разу вовсе. Один сотрудник может числиться сразу в нескольких записях об оформленных вкладах и закрытых фактах обращения, поэтому связь один ко многим, при этом, сотрудник может ещё не успеть оформить не оформить ни один вклад, не закрыть ни одно обращение. Одной должности может соответствовать много сотрудников, соответственно связь будет – один ко многим.

Получившаяся диаграмма сущностей представлена на рисунке 4.

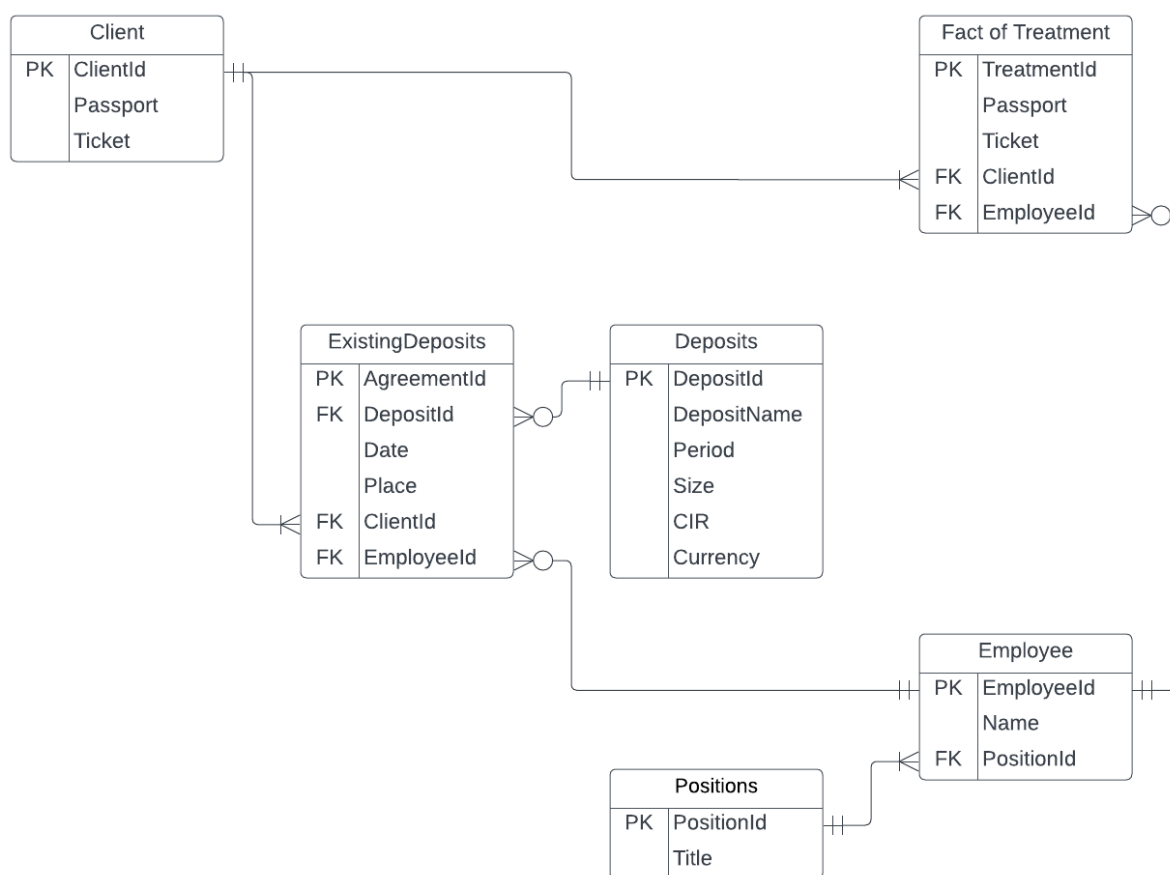


Рисунок 4 - Диаграмма сущностей

3.2 База данных

База данных — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе [16]. 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.

Данные в наиболее распространенных типах современных баз данных обычно хранятся в виде строк и столбцов, формирующих таблицу. Этими данными можно легко управлять, изменять, обновлять, контролировать и упорядочивать. В большинстве баз данных для записи и запросов данных используется язык структурированных запросов (SQL).

Для базы данных обычно требуется комплексное программное обеспечение, которое называется системой управления базами данных (СУБД). СУБД служит интерфейсом между базой данных и пользователями или программами, предоставляя пользователям возможность получать и обновлять информацию, а также управлять ее упорядочением и оптимизацией. СУБД обеспечивает контроль и управление данными, позволяя выполнять различные административные операции, такие как мониторинг производительности, настройка, а также резервное копирование и восстановление.

Для создания и управление базой данных в работе использовалась СУБД SQL Server Management Studio (SSMS) от Microsoft. Таблицы в базе данных разрабатывались на основании диаграммы сущностей (Рисунок 4) и в соответствии с принципом нормализации базы данных, с упором на 3 нормальную форму [18], [19].

Таким образом, получается следующая база данных (Рисунки 5-7):

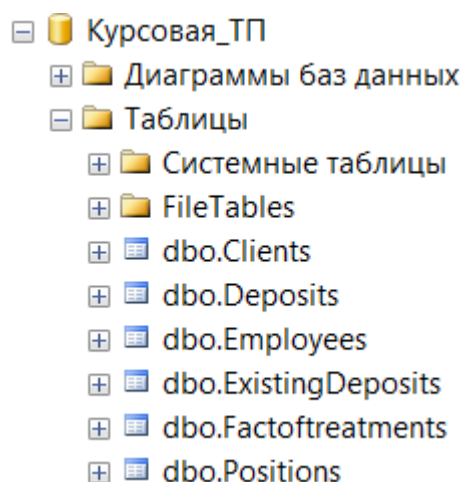


Рисунок 5 – Общий вид таблиц в базе данных

	Имя столбца	Тип данных	Разрешить знач...
🔑	AgreementId	int	<input type="checkbox"/>
	DepositId	int	<input type="checkbox"/>
	Date	datetime	<input type="checkbox"/>
	Place	varchar(20)	<input type="checkbox"/>
	ClientId	int	<input type="checkbox"/>
	ClientAge	int	<input checked="" type="checkbox"/>
	EmployeeId	int	<input type="checkbox"/>
			<input type="checkbox"/>

Рисунок 6 – Устройство таблицы в базе данных на примере таблицы оформленных вкладов

	AgreementId	DepositId	Date	Place	ClientId	ClientAge	EmployeeId
	25	2	2023-03-22 ...	main office	4	99	2
	31	7	2023-03-16 ...	filial	6	12	18
➤*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Рисунок 7 – Пример записей в таблице оформленных вкладов

Каждая таблица в базе отражает ранее описанную сущность (Рисунок 3), а каждое поле характеризует её параметр.

3.3 Data Access Layer

Следующим шагом в разработке системы является создание DAL - слоя доступа к данным [20].

Уровень 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 (Рисунок 3).

DAL отвечает за выполнение таких задач, как:

- 1) подключение к системе хранения данных и управление подключением;
- 2) генерация и выполнение SQL-запросов или других команд доступа к данным для извлечения и хранения данных;
- 3) отображение данных из системы хранения данных в объекты данных приложения и наоборот;
- 4) обработка ошибок и исключений, связанных с доступом к данным;
- 5) 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.

Всякий раз, когда приложению (BLL) требуется конкретный запрос данных или обновление, запрос отправляется на уровень доступа к данным, который затем предпринимает соответствующие действия, чтобы обеспечить выполнение запрошенных данных или обновления, что приводит к уровню абстракции, при котором бизнес-уровень будет знать только, какую функцию или методы вызывать, и ему не нужно знать фактическую архитектуру или детали всей базы данных.

Любые организации, отслеживающие данные (например, учетные записи, платежную информацию или другие записи), нуждаются в системах, обеспечивающих постоянное хранение, которое обычно организовано в базу данных. Реляционная база данных состоит из данных, организованных по строкам и столбцам. Их можно подключить к другим таблицам с помощью первичных и внешних ключей.

CRUD [21] (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.

В таблице ниже показано, что означает каждая операция CRUD (Таблица 1).

Таблица 1 – Значение каждой из CRUD операций

Письмо	Операция	Функция
C	Создавать	Вставлять
R	Читать	Выбирать
U	Обновлять	Редактировать
D	Удалить	Удалить

Для SQL карты CRUD для вставки, выбора, обновления и удаления соответственно. Такие операции, как управление безопасностью, управление транзакциями, доступ и разрешение, а также оптимизация производительности, основаны на CRUD.

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 контексте реализуемой системы CRUD операции не раз использовались при работе с данными – операции указанные ранее составляли основу функционала системы, позволяли организовать взаимодействие между базой данных и другими слоями.

Обозначив основные термины, можно приступить к реализации компонента – в общем смысле, DAL составлен из моделей, контроллеров, Application context и связи с БД (Рисунок 8).

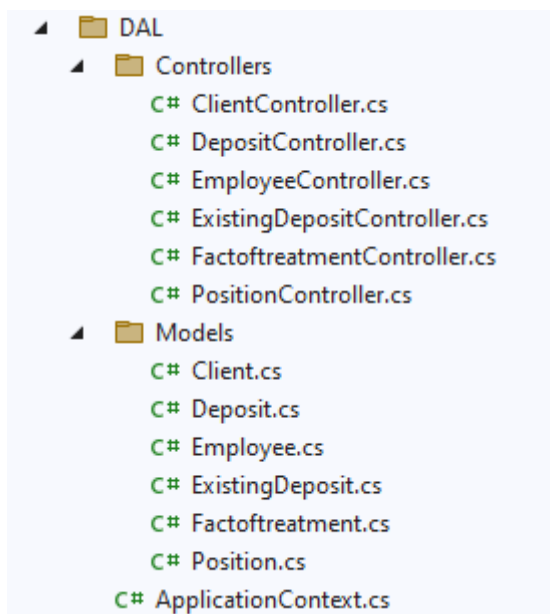


Рисунок 8 – Общий вид слоя доступа к данным

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 Clients (рисунок 9).

	Column Name	Data Type	Allow Nulls
►	ClientId	int	<input type="checkbox"/>
	Passport	varchar(15)	<input type="checkbox"/>
	Ticket	varchar(15)	<input type="checkbox"/>
			<input type="checkbox"/>

Рисунок 9 – таблица Clients

```

using System.ComponentModel.DataAnnotations;

namespace DAL
{
    public class Client
    {
        [Key]
        public int ClientId { get; set; }
        public string? Passport { get; set; }
        public string? Ticket { get; set; }
    }
}

```

Для остальных таблиц были также созданы соответствующие классы.

Однако моделей и контроллеров для работы с базой данных недостаточно, чтобы реализовать полноценное взаимодействие. Была использована технология Entity Framework Core [22]. 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»

```

using System;
using System.Collections.Generic;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using System.Text;
using System.IO;
using DAL;

namespace DAL
{
    public class ApplicationContext : DbContext
    {
        public DbSet<Employee> Employees { get; set; }
        public DbSet<Client> Clients { get; set; }
        public DbSet<Deposit> Deposits { get; set; }
        public DbSet<ExistingDeposit> ExistingDeposits { get; set; }
        public DbSet<Factoftreatment> Factoftreatments { get; set; }
        public DbSet<Position> Positions { 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 ApplicationDbContext 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=DESKTOP-
9QCIIQ1\\SQLEXPRESS;Database=Курсовая_ТП;Trusted_Connection=True;MultipleActiveResultSets=true;TrustServerCertificate=True"
  }
}

```

Контекст данных, через который идет взаимодействие с базой данных, передается в приложение в качестве сервиса через механизм внедрения зависимостей. Поэтому для работы с Entity Framework Core необходимо добавить класс контекста данных в коллекцию сервисов приложения. Для этого в startup.cs добавим следующий код (листинг 4):

Листинг 4 – Подключение класса контекста данных

```

string con = Configuration.GetConnectionString("DefaultConnection");
services.AddDbContext<ApplicationContext>(options =>
options.UseSqlServer(con));

```

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

В качестве примера ниже показан класс, реализующий CRUD операции (листинг 5):

Листинг 5 – Пример контроллера, реализующего CRUD операции

```
using DAL;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace Deposit_project.DAL.Controllers
{
    [ApiController]
    [Route("/Client")]
    public class ClientController : ControllerBase
    {
        ApplicationDbContext db;
        public ClientController (ApplicationDbContext context)
        {
            db = context;
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<Client>>> Get()
        {
            return await db.Clients.ToListAsync();
        }

        // POST api/users
        [HttpPost]
        public async Task<ActionResult<Client>> Post(Client client)
        {
            if (client == null)
            {
                return BadRequest();
            }

            db.Clients.Add(client);
            await db.SaveChangesAsync();
            return Ok(client);
        }

        // PUT api/users/
        [HttpPut]
        public async Task<ActionResult<Client>> Put(Client client)
        {
            if (client == null)
            {
                return BadRequest();
            }
            db.Update(client);
            await db.SaveChangesAsync();
            return Ok(client);
        }
    }
}
```

```

    }

    // DELETE api/users/5
    [HttpDelete("{Id}")]
    public async Task<ActionResult<Client>> Delete(int Id)
    {
        Client client = db.Clients.FirstOrDefault(x => x.ClientId == Id);
        if (client == null)
        {
            return NotFound();
        }
        db.Clients.Remove(client);
        await db.SaveChangesAsync();
        return Ok(client);
    }
}
}

```

3.4 Слой бизнес-логики

Для пояснения необходимости слоя бизнес-логики [23] вернёмся к ранее использованному изображению (Рисунок 3).

Как правило, в приложении определяются слои пользовательского интерфейса, бизнес-логики и доступа к данным. 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.

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

HTTP [24] — широко распространённый протокол передачи данных. Этот протокол предполагает использование клиент-серверной структуры передачи данных. Клиентское приложение формирует запрос и отправляет его на сервер, после чего серверное программное обеспечение обрабатывает данный запрос, формирует ответ и передаёт его обратно клиенту. После этого

клиентское приложение может продолжить отправлять другие запросы, которые будут обработаны аналогичным образом.

Задача, которая традиционно решается с помощью протокола HTTP — обмен данными между пользовательским приложением, осуществляющим доступ к веб-ресурсам (обычно это веб-браузер) и веб-сервером.

HTTP определяет множество методов запроса, которые указывают, какое желаемое действие выполнится для данного ресурса [25].

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 1.

Для совершения CRUD операций были реализованы классы, совершающие данные операции, позволяющие создавать и считывать записи из соответствующих таблиц в базе данных.

В качестве примера ниже показан класс, реализующий данные операции (листинг 6):

Листинг 6 – Класс ClientController

```
using Microsoft.AspNetCore.Components;  
using Microsoft.AspNetCore.Mvc;  
using System.Text.Json;  
using System.Text.Json.Nodes;  
using Domen2Vozvrashenie.Domain.Models;  
  
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?.Passport}");
        if (dalClient == null)
            return BadRequest();
        else
            return dalClient;
    }
}

```

3.5 User interface

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

Основой нашего веб-сайта являются HTML, CSS, JavaScript [26].

Что такое HTML

Это один из основных строительных блоков сети. HTML, или «язык гипертекстовой разметки», существует в той или иной форме примерно с 1993 года, когда он был создан физиком Тимом Бернерсом-Ли. Сейчас это пятое поколение HTML5.

HTML — это не язык программирования, а «язык разметки». Например, он использует синтаксис тегов для изменения способа отображения текста. Он также может определять, где изображения размещаются на странице.

Что такое CSS

CSS или «каскадные таблицы стилей» работают с HTML для создания формата веб-страниц. Он работает поверх основы страницы, созданной с помощью HTML. Это также помогает адаптировать страницы к различным форматам, оптимизированным для настольных компьютеров или мобильных устройств. Последним стандартом является CSS3, то есть они относятся к третьему поколению.

CSS — это то, что придает веб-сайтам безупречный и профессиональный вид. Это также позволяет добавлять интерактивные элементы, такие как цвет фона, заголовки, формы, графика и многое другое, что делает веб-сайты и веб-приложения гораздо более привлекательными для зрителя.

Что такое JavaScript

JavaScript, который часто сокращают до JS, — это еще один интерфейсный язык программирования, который является одной из основных технологий Всемирной паутины, наряду с HTML и CSS. По состоянию на 2022 год 98% веб-сайтов используют JavaScript на стороне клиента для поведения веб-страниц, часто включая сторонние библиотеки, такие как JQuery. 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 [27].

React — это декларативная JavaScript-библиотека для создания пользовательских интерфейсов. Она позволяет собирать

сложный UI из маленьких изолированных кусочков кода, называемых «компонентами».

В разработке веб-интерфейса было использовано REST API через Fetch API и Axios. В React есть несколько способов использования REST API, например с помощью встроенного метода `fetch()`, который позволяет получать данные от сервера или API, используя URL-адрес. Этот метод учитывает такие понятия, как CORS и заголовок HTTP Origin. `fetch()` всегда возвращает промис, указывающий на ответ от сервера. Кроме того, возможно передать объект параметров инициализации в `fetch()` для настройки запроса. После получения запроса доступны встроенные методы для структурирования и обработки данных.

Axios — это другой HTTP-клиент, предназначенный для браузера и node.js. Этот HTTP-клиент также основан на промисах и позволяет использовать асинхронность в коде. Он дает возможность перехватывать и отменять запросы и обеспечивает безопасность от подделки межсайтовых запросов на стороне клиента.

В слое рассматриваемом слое пользовательского интерфейса используется шаблон проектирования MVC [28] («модель-представление-контроллер»). Общая схема архитектурной реализации шаблона представлена на рисунке 10. MVC разделяет программный код пользовательского интерфейса на три основные составляющие: модель, представление и контроллер. Модель содержит программные сущности, полученные из программного шлюза ИС, в то время как представление описывает программные сущности, используемые для создания пользовательского интерфейса. Контроллер соединяет представление и модель, преобразуя пользовательские запросы в правильные программные сущности модели, а также выполняет действия взаимодействия с программным шлюзом приложения. Применение шаблона проектирования MVC позволяет отделить бизнес-логику, находящуюся в модели, от логики создания интерфейсов.

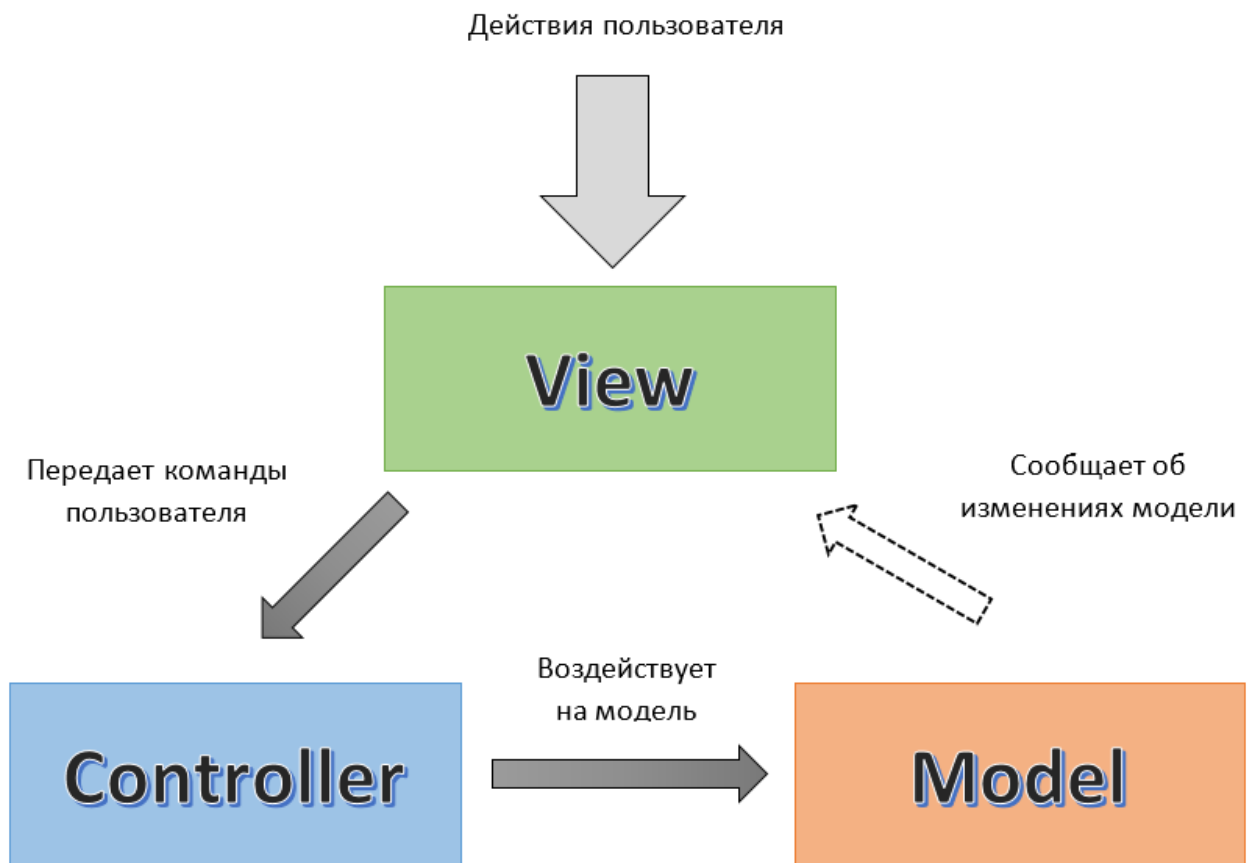


Рисунок 10 - Шаблон проектирования MVC

Сам сайт делится на несколько компонентов, в зависимости от поставленных задач.

Начало любого компонента сопровождалось импортированием необходимых дополнений, как пример раздел импорта внутри главного компонента App.js (листинг 7):

Листинг 7 – импорт в App.js

```
import './index.css';  
  
import './App.css'  
  
import logo from './images/волга 4.jpg';  
  
import { Link } from "react-router-dom";
```


Для того, чтобы сделать базовое одностраничное приложение React многостраничным потребовалось внести изменение в файл index.js (листинг 8):

Листинг 8 – изменённый index.js

```
import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

import Deposits from './Deposits';

import FormPage from './FormPage';

import AboutPage from './AboutPage';

import MiddlePage from './MiddlePage';

import {

  createBrowserRouter,

  RouterProvider,

} from "react-router-dom";

const router = createBrowserRouter([

  {

    path: "/",

    element: <App />,

  },

  {

    path: "/deposit",

    element: <Deposits/>,

  },
```

```

{
  path: "/form",
  element: <FormPage/>,
},
{
  path: "/about",
  element: <AboutPage/>,
},
{
  path: "/middle",
  element: <MiddlePage/>,
},
]);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <RouterProvider router={router} />
);

reportWebVitals();

```

Для координирования работы веб-сайта и домена использовался файл `setupProxy.js` (листинг 9):

Листинг 9 - файл `setupProxy.js`

```

const { createProxyMiddleware } = require('http-proxy-middleware');

module.exports = function(app) {

  app.use(

    '/api',

```

```

createProxyMiddleware({
  target: 'http://localhost:5192',
  changeOrigin: true,
})
);
};

```

В качестве примера компонента, можно привести MiddlePage.js, который реализует взаимодействие с БД и доменом посредством операций GET, POST, DELETE (Листинг 10).

Листинг 10 – Реализация компонента MiddlePage.js

```

function MiddlePage() {
  return (
    <body>
      <section className="contact4">
        <div className="section-center clearfix">
          <br/>
          <br/>
          <article className="main-container mid" id='contacts'>
            <a><Link to='/middle' className='btn middle-btn middle-btn-main' id='middle-btn-main'
              onClick={menuClick3}>Вывести вкладки</Link></a>
            <a><Link to='/middle' className='btn middle-btn middle-btn-main2' id='middle-btn-
              main2' onClick={menuClick4}>Скрыть</Link></a>
          </article>
          <div className='buttons'>
            <a><Link to='/middle' className='btn middle-btn first-middle' id="nav-toggle"
              onClick={menuClick}>Сортировать</Link>
            <ul className="nav-links-middle" id="nav-links-middle">

```

```

</li>

    <a href="#home" className="nav-link-middle scroll-link">По id вклада</a>

</li>

<li>

    <a href="#about" className="nav-link-middle scroll-link">По дате</a>

</li>

<li>

    <a href="#services" className="nav-link-middle scroll-link">По размеру</a>

</li>

<li>

    <a href="#featured" className="nav-link-middle scroll-link">По возрасту </a>

</li>

</ul>

</a>

<a><Link      to='/middle'      className='btn      middle-btn'      id="nav-toggle"
onClick={ menuClick2 }>Изменить</Link>

    <ul className="nav-links-middle" id="nav-links-middle2">

        <li>

            <a href="#home" className="nav-link-middle2 scroll-link">Процент по вкладу</a>

        </li>

        <li>

            <a href="#about" className="nav-link-middle2 scroll-link">Начисление ставки</a>

        </li>

        <li>

            <a href="#services" className="nav-link-middle2 scroll-link">Валюту</a>

        </li>

```

```

    <li>

      <a href="#featured" className="nav-link-middle2 scroll-link">Размер вклада</a>

    </li>

  </ul>

</a>

<a><Link to='/form' className='btn middle-btn'>Добавить</Link></a>

<a><Link to='/middle' className='btn middle-btn'>Удалить</Link></a>

</div>

<a><Link to="/" className='btn form-back2'>Вернуться на главную</Link> </a>

</div>

</section>

</body>

);

}

export default MiddlePage;

```

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

4 Тестирование системы

Поскольку, говоря о текущем проекте, мы имеем дело с многоуровневым приложением, тестировать его работу стоит соответствующе - отдельно проверяя работу каждого его 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.

Для начала оценим корректность функционирования DAL на примере таблицы Client (Рисунок 11).

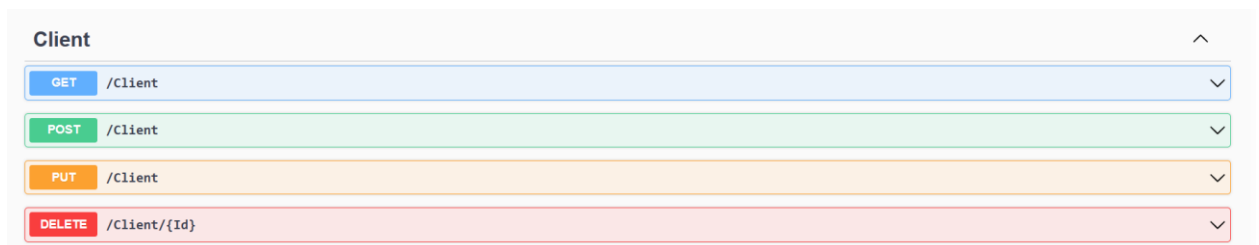


Рисунок 11 – Таблица Client в Swagger UI

Для этого проверим все возможные операции, начиная с GET (Рисунок 12).

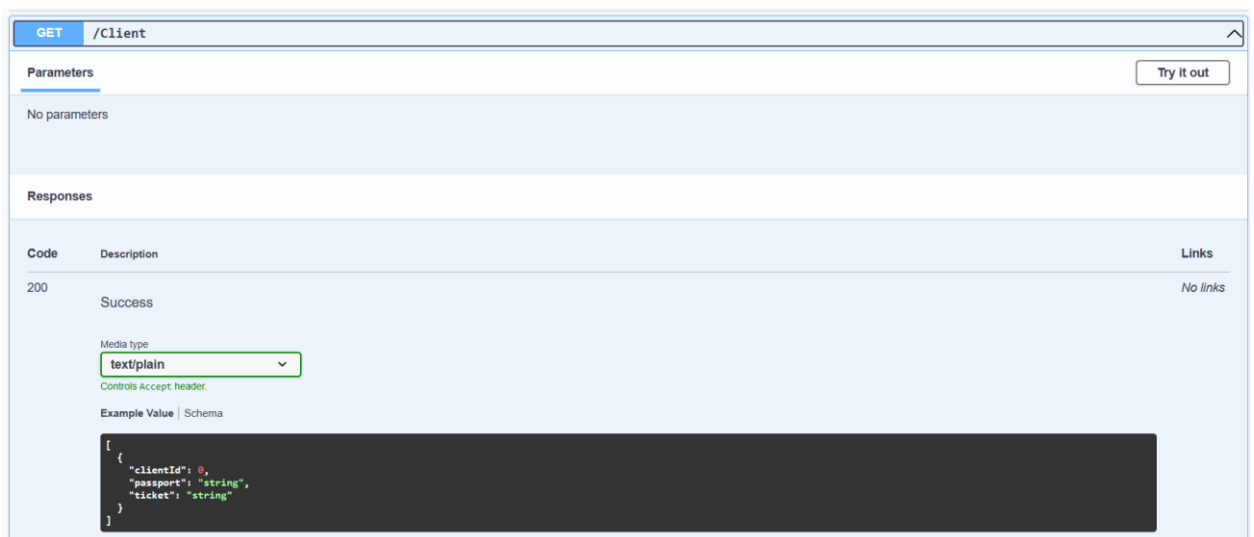


Рисунок 12 – Исходный интерфейс операции GET

Для выполнения операции GET выберем опцию Try it out в правом верхнем углу. Меню соответственно изменится (Рисунок 13).

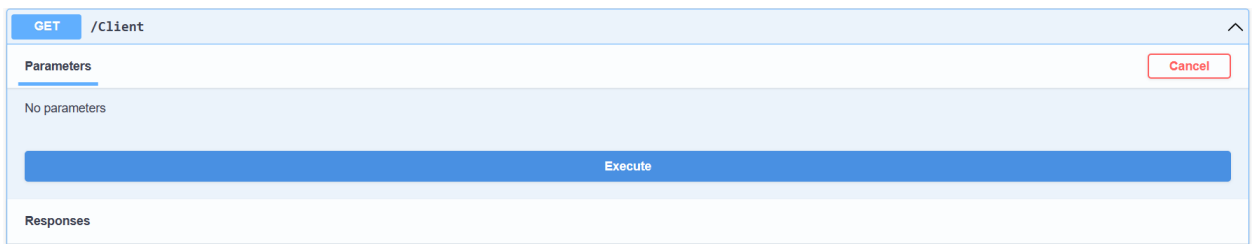


Рисунок 13 – Результат выбора опции Try it out

В изменившемся окне, нажмём на Execute (Рисунок 14).



Рисунок 14 – Результат команды Execute

В расширившемся окне, появился результат операции GET, схожим образом будут выполнены и последующие POST, PUT, DELETE.

Операция POST (Рисунок 15)

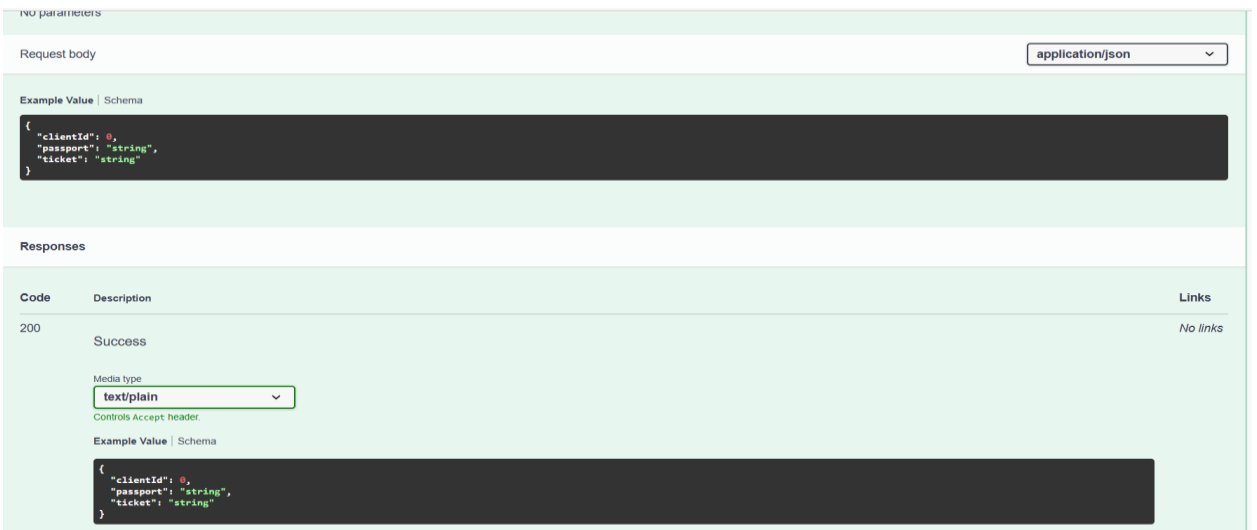


Рисунок 15 – Исходный интерфейс операции POST

Выполним команду Execute (Рисунок 16).



Рисунок 16 – Результат команды Execute

Операция PUT (Рисунок 17)

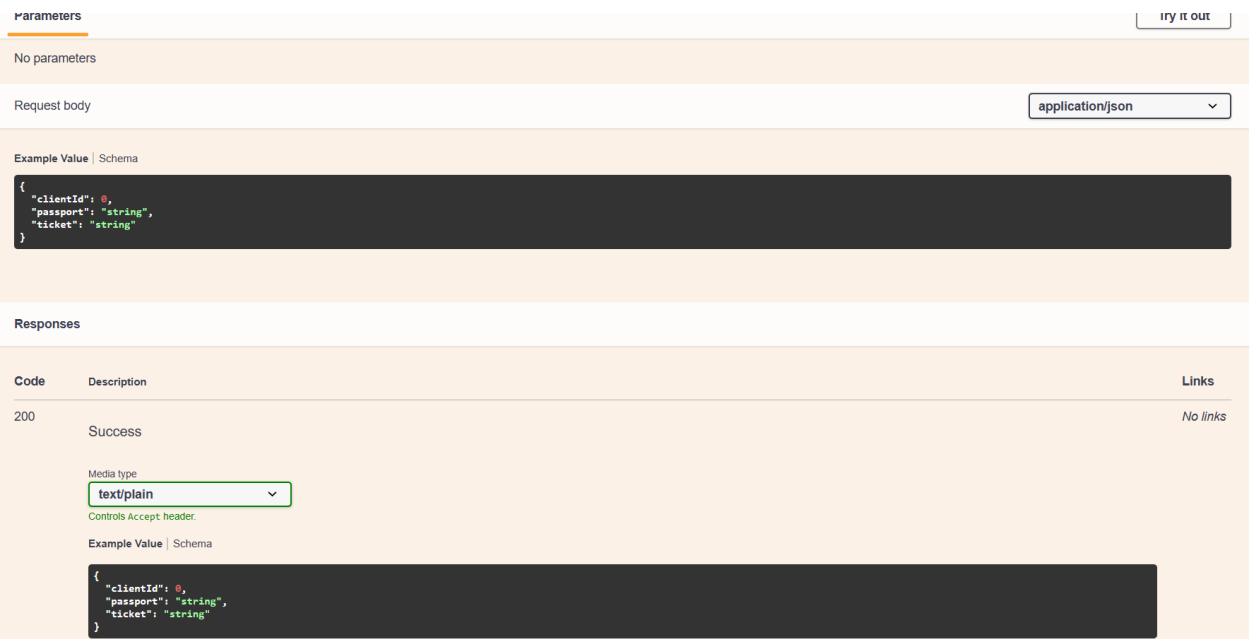


Рисунок 17 – Исходный интерфейс операции PUT

Выполним команду Execute (Рисунок 18).



Рисунок 18 – Результат команды Execute

Операция DELETE (Рисунок 19)

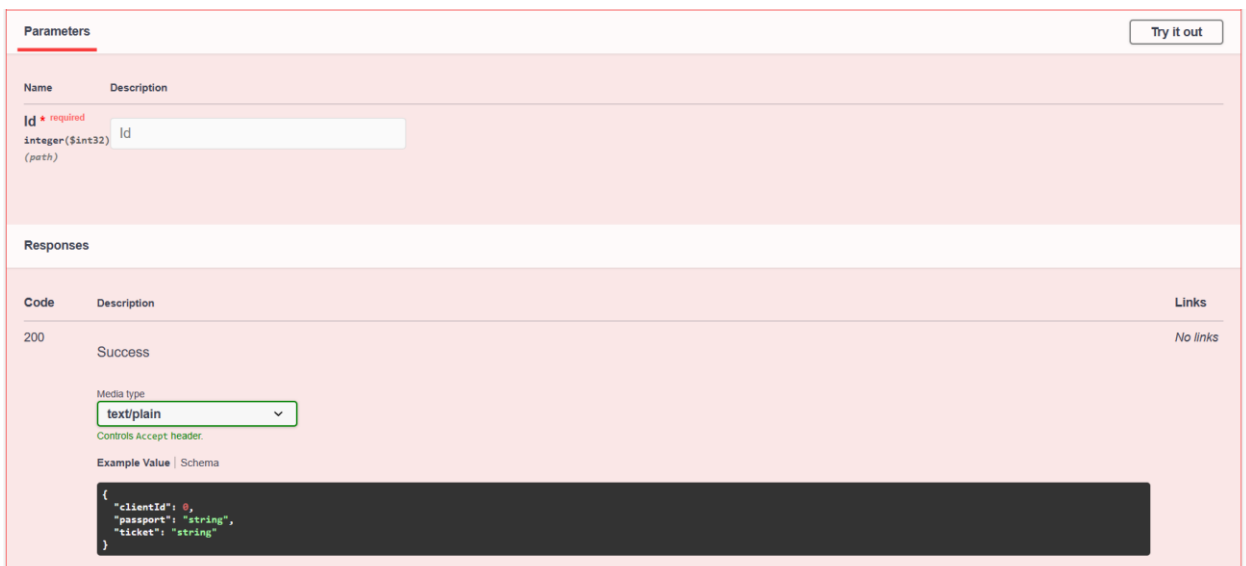


Рисунок 19 – Исходный интерфейс операции DELETE

Выполним команду Execute (Рисунок 20).



Рисунок 20 – Результат команды Execute

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

Теперь к слою бизнес-логики. Проверка его работы будет выполняться схожим образом, через веб-Фреймворк Swagger UI на примере таблицы Clients (Рисунок 21).

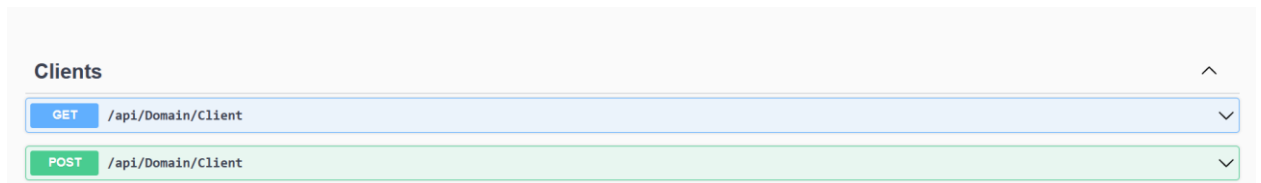


Рисунок 21 – Таблица Client в Swagger UI

Операция GET (Рисунок 22)

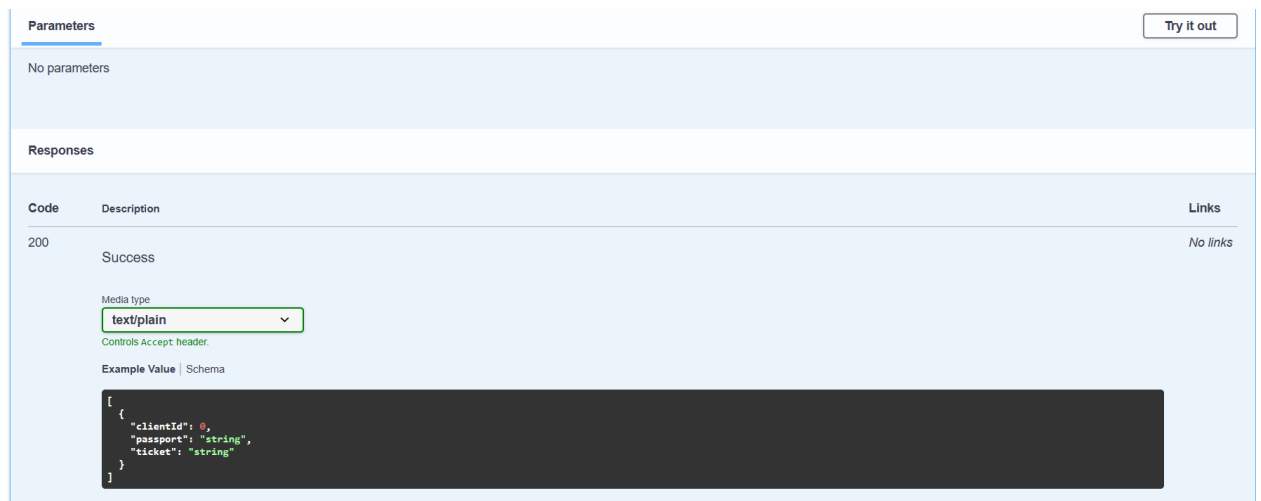


Рисунок 22 – Исходный интерфейс операции GET

Выполним команду Execute (Рисунок 23).

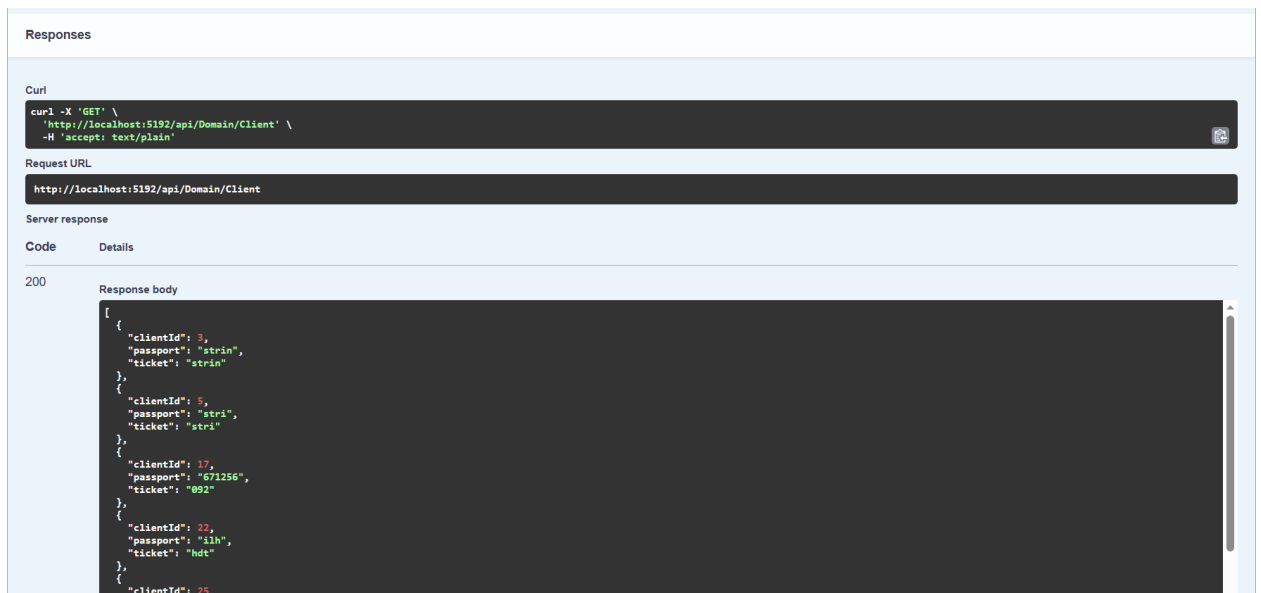


Рисунок 23 – Результат команды Execute

Операция POST (Рисунок 24)

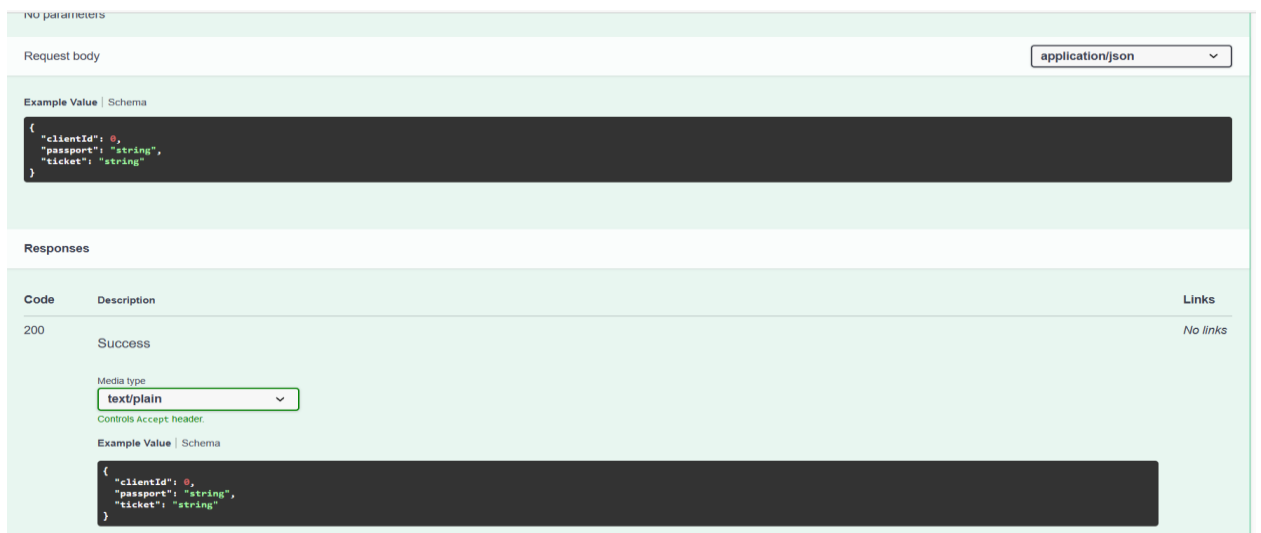


Рисунок 24 – Исходный интерфейс операции POST

Выполним команду Execute (Рисунок 25).

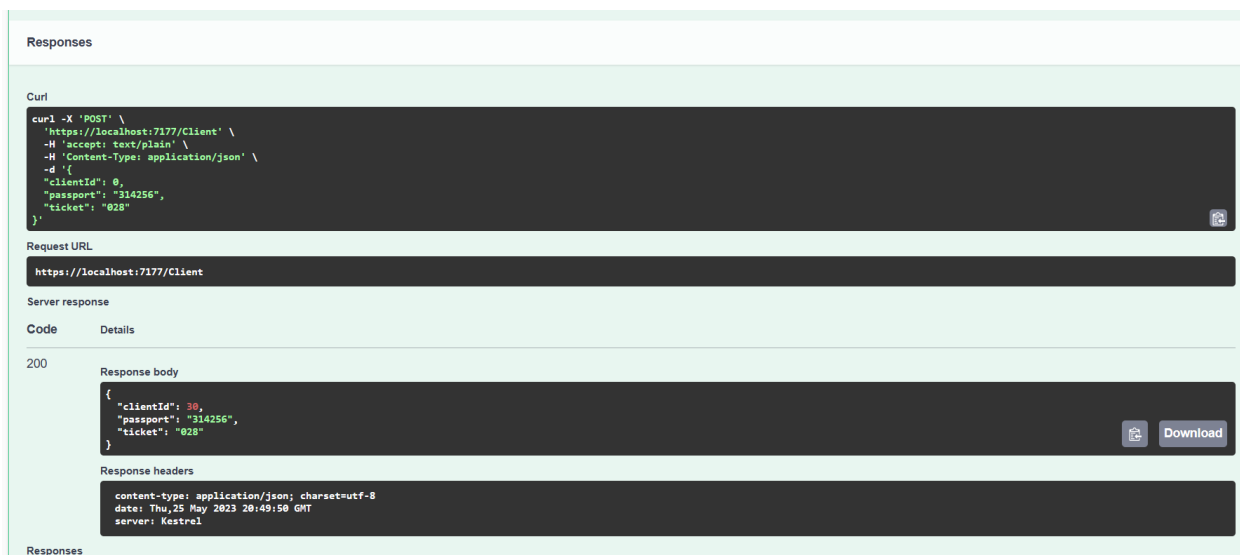


Рисунок 25 – Результат команды Execute

Следующим этапом тестирования является просмотр веб-сайта, начнём рассмотрение с главного раздела (Рисунок 26).

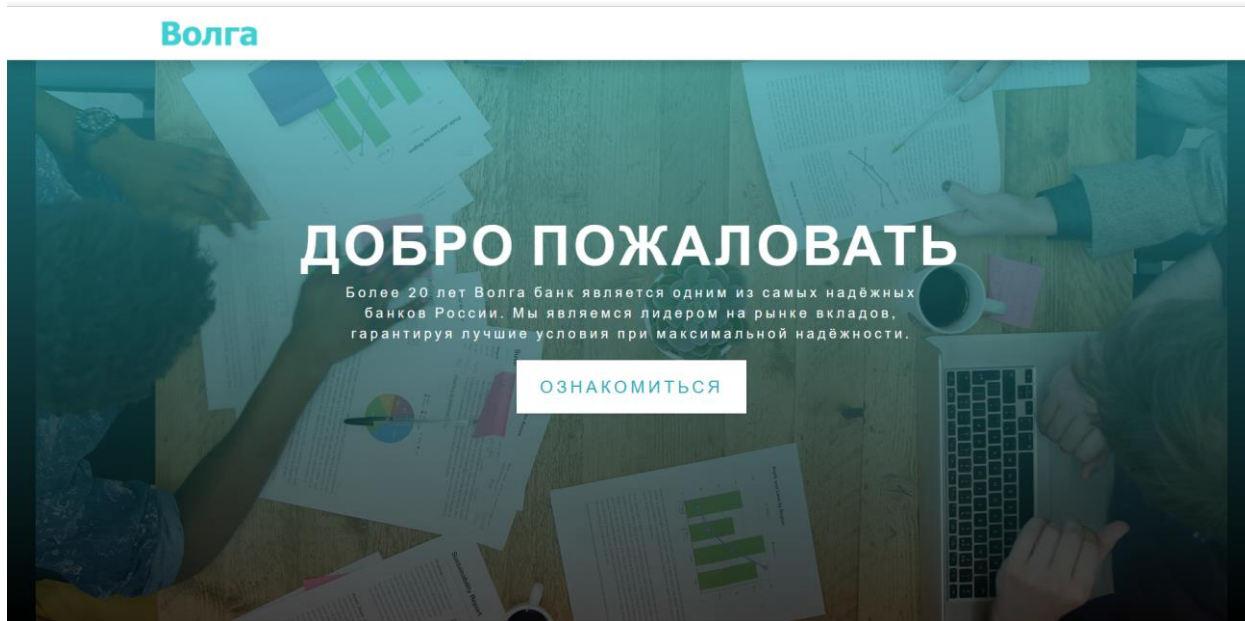


Рисунок 26 – Начало главной страницы

Общая информация о компании содержится в последующем разделе (Рисунки 27-29).

ПОЧЕМУ МЫ?



Качество-Превыше Всего

Мы дорожим нашими клиентами, поэтому постоянно работаем над тем, чтобы улучшить качество предоставляемых нами услуг.

Вклады в Волга Банк застрахованы Центральным банком РФ, вы можете быть уверены в безопасности своих средств.

[БОЛЬШЕ ОБ ЭТОМ](#)

Рисунок 27 – Раздел о нас главной страницы

НАШ ПОДХОД

Клиенты - наше всё

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

[ВЕРНУТЬСЯ](#)

Рисунок 28 – Страница с расширенным разделом о нас

НАШИ ПРЕИМУЩЕСТВА

Широкий Выбор Валют

Вклады в Волга банке можно оформлять разными валютами, среди них - доллары, рубли, тенге, юани.

Выгодная Ставка

Наши вклады являются самыми выгодными на рынке, миллионы людей доверяют нам свои сбережения.


Надёжность

Все вклады в банке находятся под надёжной защитой федерального законодательства Российской Федерации. В случае непредвиденных обстоятельств обязательства перед вкладчиками перейдут к Центробанку. Ваши средства всегда в безопасности.

Рисунок 29 – Раздел наши преимущества главной страницы

Общий и расширенный раздел вкладов (Рисунки 30-31)

ЛУЧШИЕ ОПЦИИ




От 9,5% Годовых

Пенсионный

Для тех, кто думает наперёд

Рубль 30 Лет От 5 Миллионов

ОФОРМИТЬ




От 5,5% Годовых

Инвестиционный

Для опытных инвесторов

Доллар 5 Лет От 2 Миллионов

ОФОРМИТЬ



От 6,5% Годовых

Семейный

Для опытных родителей






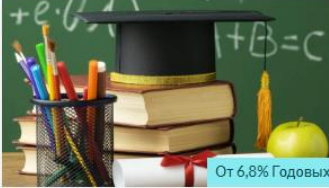
Евро 10 Лет От 3 Миллионов

ОФОРМИТЬ

ВСЕ ВКЛАДЫ

Рисунок 30 – Раздел вкладов главной страницы

СПИСОК ВКЛАДОВ

 От 9,5% Годовых	 От 5,5% Годовых	 От 6,5% Годовых
Пенсионный Для тех, кто думает наперёд Рубль 30 Лет От 5 Миллионов ОФОРМИТЬ	Инвестиционный Для опытных инвесторов Доллар 5 Лет От 2 Миллионов ОФОРМИТЬ	Семейный Для опытных родителей Евро 10 Лет От 3 Миллионов ОФОРМИТЬ
 От 4,5% Годовых	 От 10,5% Годовых	 От 6,8% Годовых
Личный Для тех, кто хочет быть уверен в завтрашнем дне Рубль 2,5 Года От 2 Миллионов ОФОРМИТЬ	Восточный Если вы предпочитаете Азиатские валюты Юань 5 Лет От 4 Миллионов ОФОРМИТЬ	Учебный Если вы всерьёз задумались о будущей учёбе Рубль 3 Года От 1 Миллиона ОФОРМИТЬ

[ВЕРНУТЬСЯ](#)

Рисунок 31 – Раздел обратной связи главной страницы

Последним на сайте является раздел обратной связи в виде общей и специализированной формы, для пользователей, желающих оформить вклад (Рисунки 32-33).

Хотите получать уведомления о наших новостях?

Подпишитесь на рассылку и будьте в курсе новостей

[Подтвердить](#)

[Главная](#) [О Нас](#) [Преимущества](#) [Услуги](#)

© Волга банк. Все права защищены

Рисунок 32 – Главный раздел страницы списка вкладов

Рисунок 33 – Главный раздел страницы с формой на заявку по оформлению вклада

После просмотра веб-сайта становится возможным изучение его функционала. Основная бизнес-операция – оформление вклада осуществляется пользователем посредством выбора соответствующей кнопки в вариантах вклада, соответствующие кнопки продемонстрированы на рисунках 30 и 31.

После нажатия кнопки «Оформить вклад» клиент пере адресуется на страницу с формой, представленную на рисунке 33.

После заполнения данных данные считываются, а пользователь уведомляется о обработки заявки (Рисунок 34).

The image shows a web application interface for confirming a pension application. A white modal dialog box is overlaid on top of the form, displaying a success message in Russian. The form itself is light blue and contains several input fields. The first field is a dropdown menu with the text 'ПЕНСИОННЫЙ' and a downward arrow. Below it are two text input fields containing the numbers '2500000' and '25'. The next section is titled 'ДАТА ПОДАЧИ ЗАЯВКИ' and contains a date input field with the text '25.05.2023' and a calendar icon. At the bottom of the form is a large cyan button with the text 'ПОДТВЕРДИТЬ'.

Сообщение с localhost:3000:

Ваша заявка отправлена на рассмотрение

ОК

ПОДТВЕРДИТЕ ВЫБОР

ПЕНСИОННЫЙ

2500000

25

ДАТА ПОДАЧИ ЗАЯВКИ

25.05.2023

ПОДТВЕРДИТЬ

Рисунок 34 – Заполненная форма и уведомление о принятии заявки

Помимо основной операции, в веб-сайте существует возможность расширенного взаимодействия с системой (Рисунок 35).

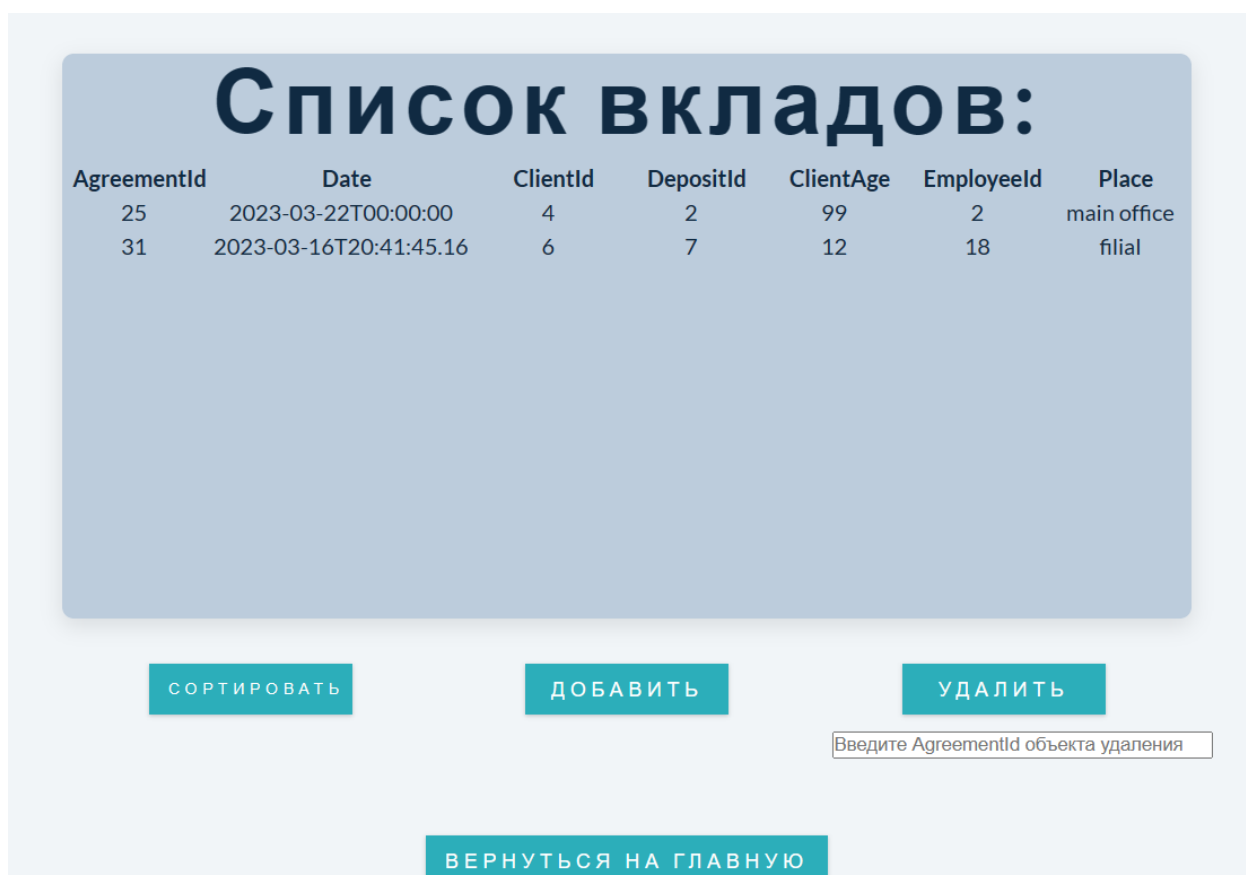


Рисунок 35 – Окно расширенного взаимодействия с системой

Данное окно позволяет лицу принятия решения производить соответствующие манипуляции с уже имеющимися данными из базы данных.

Фактически, данное окно связывает все слои системы, организуя конкретные возможности для их взаимодействия. Например, возможна операция сортировки, фильтрации при нажатии на соответствующую кнопку (Рисунок 38).

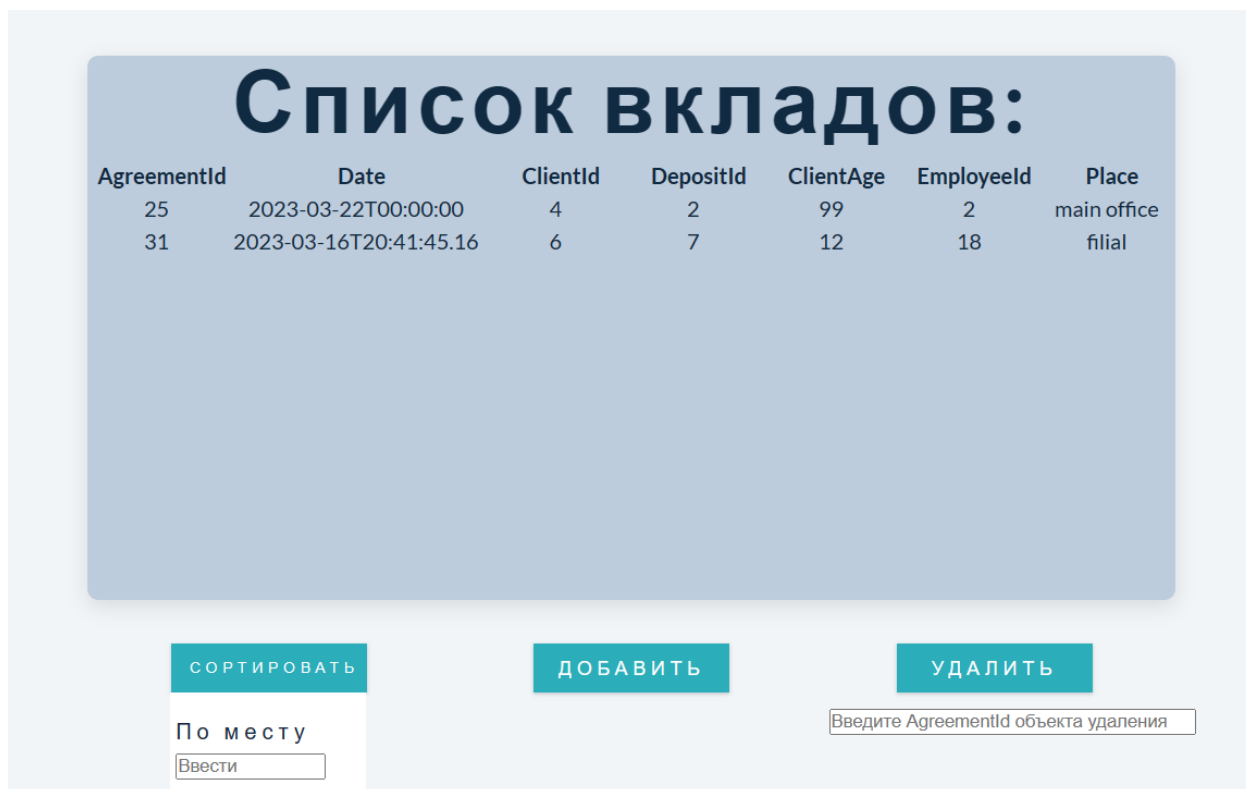


Рисунок 36 – Раскрывшееся меню команды сортировать

Для удобства, рассмотрим сортировку(фильтрацию) по месту оформления договора, например, оставим только ту запись, которая отвечает за вклад, оформленный в главном офисе банка (Рисунок 37).

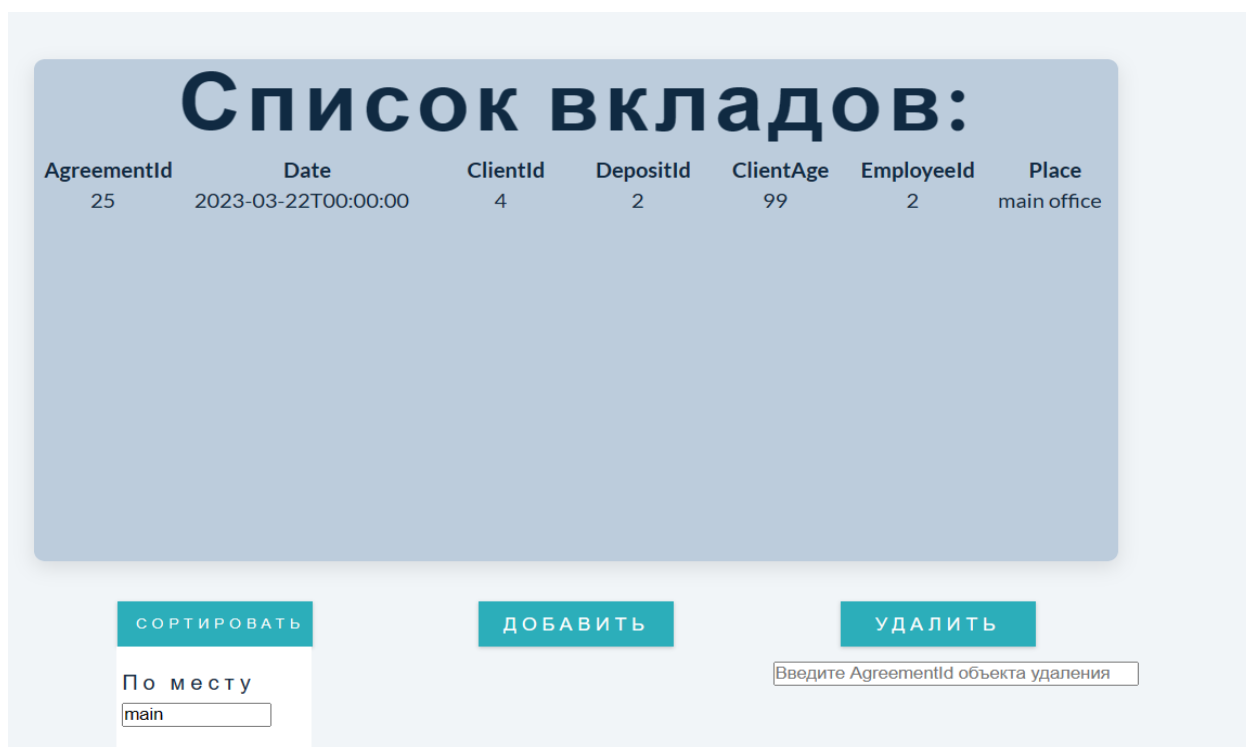


Рисунок 37 – Результат фильтрации

По мере заполнения поля, система сортирует имеющиеся записи и выводит результат.

Кнопка добавить, отправляет к уже рассмотренному на рисунке 33 окну, поэтому на нём останавливаться не будем. Перейдём к завершающей операции – Операции удаления. Операция удаления работает посредством заполнения соответствующего поля под кнопкой удалить с последующим нажатием самой кнопки, удалим, например запись с AgreementId = 25 (Рисунки 38- 39)

AgreementId	Date	ClientId	DepositId	ClientAge	EmployeeId	Place
25	2023-03-22T00:00:00	4	2	99	2	main office
31	2023-03-16T20:41:45.16	6	7	12	18	filial

СОРТИРОВАТЬ ДОБАВИТЬ УДАЛИТЬ

25

ВЕРНУТЬСЯ НА ГЛАВНУЮ

Рисунок 38 – Ввод значения AgreementId =25, список записей пока не изменён

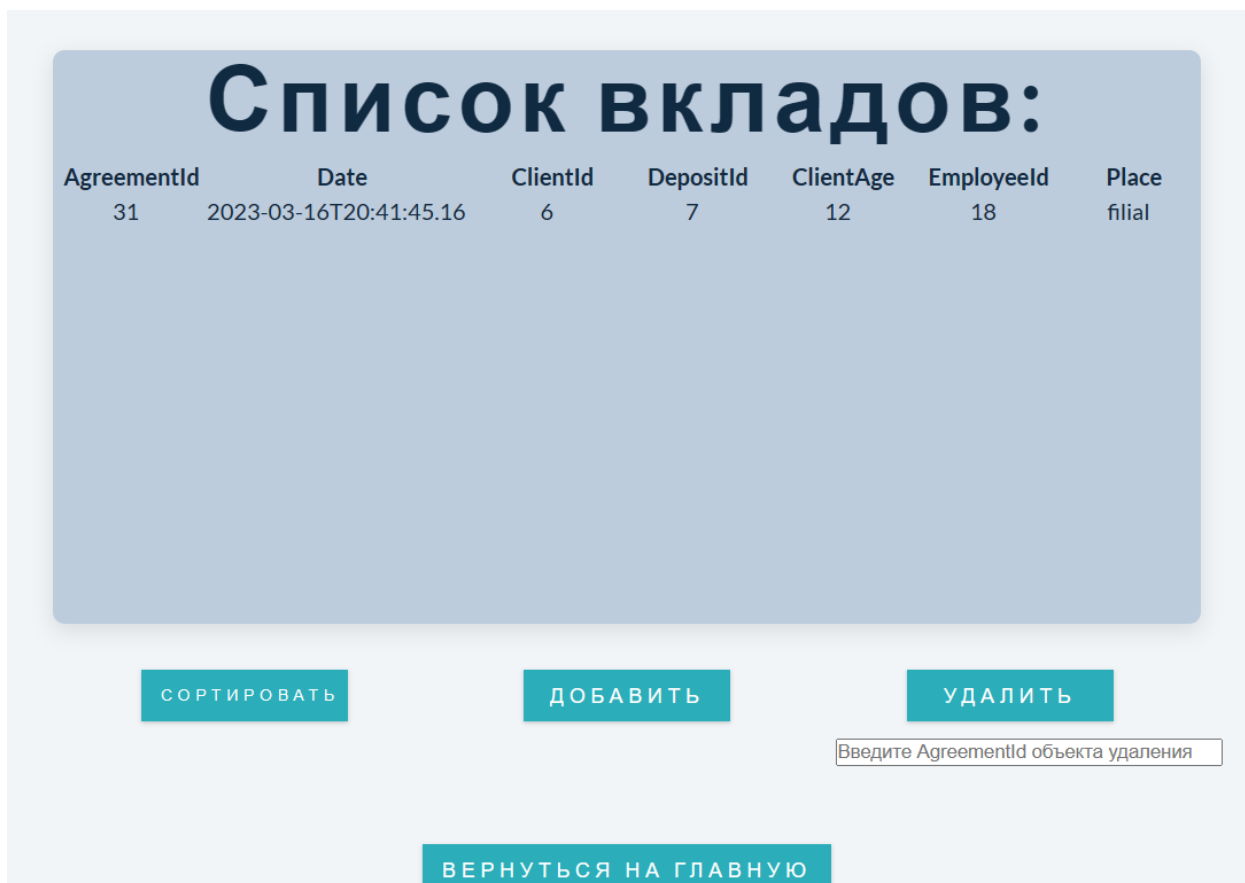


Рисунок 39 – Результат нажатия кнопки удалить, изменённый список записей

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 «Вклад 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 используемых источников

1. Банки инвестируют в информационные технологии // fintechru.org URL: <https://www.fintechru.org/publications/banki-investiruyut-v-informatsionnye-tekhnologii/> (дата обращения: 21.05.2023).
2. Цифровая трансформация российского финансового рынка как ключевая стратегия в пост ковидный период // cyberleninka.ru URL: <https://cyberleninka.ru/article/n/tsifrovaya-transformatsiya-rossiyskogo-finansovogo-rynka-kak-klyuchevaya-strategiya-v-postkovidnyy-period> (дата обращения: 21.05.2023).
3. Современные информационные системы в банковской деятельности // elib.bsu.by URL: <https://elib.bsu.by/bitstream/123456789/216655/1/165-168.pdf> (дата обращения: 21.05.2023).
4. Информационные технологии в банковской сфере: ТОП-5 IT трендов // doczilla.pro URL: <https://doczilla.pro/ru/blog/informacionnye-tekhnologii-v-bankovskoj-sfere-top-5-it-trendov/> (дата обращения: 21.05.2023).
5. Закон Российской Федерации "Федеральный закон от 23.12.2003 N 177-ФЗ (ред. от 18.03.2023) "О страховании вкладов в банках Российской Федерации"" (дата обращения: 21.05.2023).
6. Закон Российской Федерации "Закон Российской Федерации "Федеральный закон "О противодействии легализации (отмыванию) доходов, полученных преступным путем, и финансированию терроризма" " от 07.08.2001 № N 115-ФЗ // Российская газета (дата обращения: 21.05.2023).
7. РЕЙТИНГ БАНКОВ ПО КРЕДИТАМ ФИЗИЧЕСКИМ ЛИЦАМ В 2022 ГОДУ // top-rf.ru URL: <https://top-rf.ru/business/303-rejting-bankov-po-kreditam.html> (дата обращения: 21.05.2023).
8. Что такое API банка, и как с помощью него работать с финансами // vc.ru URL: <https://vc.ru/services/290533-superapp-ili-superservis-chem-otlichayutsya-i-chto-vybrat-dlya-biznesa> (дата обращения: 21.05.2023).

9. Tinkoff Bank's Digital Ecosystem and Platform Strategy - //medium.com URL:

<https://medium.com/search?q=Tinkoff+Bank%27s+Digital+Ecosystem+and+Platform+Strategy> (дата обращения: 21.05.2023).

10. ASSESSMENT METHOD FOR COMPLIANCE OF INFORMATION SECURITY OF THE RUSSIAN BANKING SYSTEM ORGANISATIONS WITH REQUIREMENTS OF STO BR IBBS-1.0-2014* // BANK OF RUSSIA STANDARD URL: http://www.cbr.ru/content/document/file/51218/st-12-14_en.pdf (дата обращения: 21.05.2023).

11. The biggest Russian banks are investing in biometric and NFC technology // Biometricupdate.com URL: <https://www.biometricupdate.com/201903/russian-banks-advised-to-inform-customers-of-biometric-data-collection-after-the-fact> (дата обращения: 21.05.2023).

12. Sberbank, Alfa-Bank go to cloud // Forrester URL: https://www.forrester.com/report/the-forrester-go-to-market-architecture-balances-revenue-obsession-with-customer-obsession/RES178319?ref_search=0_1685377423288 (дата обращения: 21.05.2023).

13. Приложение в браузере: чем заменить привычные мобильные сервисы // trends.rbc.ru URL: <https://trends.rbc.ru/trends/industry/625e988e9a794721e46f03df/> (дата обращения: 21.05.2023).

14. Mobile banking and fintech apps: industry trends for UI design and technologies // advapay.eu URL: <https://advapay.eu/mobile-banking-and-fintech-apps-industry-trends-for-ui-design-and-technologies/> (дата обращения: 21.05.2023).

15. Petre M. UML in practice //2013 35th international conference on software engineering (icse). – IEEE, 2013. – С. 722-731. (дата обращения: 23.05.2023).
16. Что такое база данных? // oracle.com URL:
<https://www.oracle.com/cis/database/what-is-database/> (дата обращения: 23.05.2023).
17. Что такое ER-диаграмма и как ее создать? // Lucidchart URL:
<https://www.lucidchart.com/> (дата обращения: 23.05.2023).
18. Нормализация баз данных простыми словами // info-comp.ru URL:
<https://info-comp.ru/database-normalization> (дата обращения: 24.05.2023).
19. Третья нормальная форма (3NF) базы данных // info-comp.ru URL:
<https://info-comp.ru/third-normal-form> (дата обращения: 24.05.2023).
20. Data-Access Layer // geeksforgeeks.org URL:
<https://www.geeksforgeeks.org/data-access-layer/> (дата обращения: 24.05.2023).
21. Что такое CRUD-операции // bestprogrammer.ru URL:
<https://bestprogrammer.ru/izuchenie/chto-takoe-crud-operatsii> (дата обращения: 24.05.2023).
22. Руководство по Entity Framework Core 7 // metanit.com URL:
<https://metanit.com/sharp/efcore/> (дата обращения: 24.05.2023).
23. Microsoft Developer Division, .NET, and Visual Studio product teams Architecting-Modern-Web-Applications-with-ASP.NET-Core-and-Azure. - Microsoft Corporation: 2020. - 350 с. (дата обращения: 24.05.2023).
24. Простым языком об HTTP // habr.com URL:
<https://habr.com/ru/articles/215117/> (дата обращения: 24.05.2023).
25. Методы HTTP запроса // developer.mozilla.org URL:
<https://developer.mozilla.org/ru/docs/Web/HTTP/Methods> (дата обращения: 24.05.2023).
26. What is HTML, CSS, & Javascript? // ciat.edu URL:
<https://www.ciat.edu/blog/html-css->

javascript/#:~:text=HTML%20defines%20the%20structure%20of,learn%20the
m%20in%20that%20order. (дата обращения: 24.05.2023).

27. Что такое React и как его освоить? // academy.yandex.ru URL:
<https://academy.yandex.ru/journal/что-такое-react-i-kak-ego-osvoit> (дата
обращения: 24.05.2023).

28. Архитектура ASP.NET MVC 5 // professorweb.ru URL:
<https://learn.microsoft.com/ru-ru/aspnet/overviewcsharp> (дата обращения:
25.05.2023).