МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ИСТ

Кафедра «Информационные системы»

Дисциплина «Разработка профессиональных приложений»

**КУРСОВАЯ РАБОТА**

Тема Разработка программы «Банк «Вы банкрот». Исполнитель»

Выполнил студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Белянин Н. Н./ подпись инициалы, фамилия

Курс второй Группа ПИбд-21

Направление/специальность 09.03.04 «Программная инженерия»

Руководитель ст. преподаватель Эгов Е.Н.

должность, ученая степень, ученое звание фамилия, имя, отчество

Дата сдачи:

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г.

Дата защиты:

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ульяновск 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ИСТ

Кафедра «Информационные системы»

Дисциплина «Разработка профессиональных приложений»

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Студенту ПИбд-21 Белянин Н. Н.

группа фамилия, инициалы

Тема работы Разработка программы «Банк «Вы банкрот». Исполнитель»

Срок сдачи законченной работы «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г.

**Исходные данные к работе:** описание задания по теме, утвержденной распоряжением деканата ФИСТ

**Рекомендуемая литература:** курс лекций по дисциплине «Разработка профессиональных приложений», методические указания к лабораторным работам по дисциплине «Разработка профессиональных приложений», интернет-источники.

**Содержание пояснительной записки** (перечень подлежащих разработке вопросов)

Введение. Описание актуальности задачи.

Первая глава. Описание предметной области, поиск аналогов, ТЗ.

Вторая глава. Представление диаграмм с их описанием.

Третья глава. Представление руководств пользователя и программиста для разработанного проекта

Заключение. Выводы по проделанной работе, какие технологии были применены и освоены.

Список литературы.

Приложение. Листинг кода

**Перечень графического материала** (с точным указанием обязательных чертежей) Диаграммы UML: диаграмма вариантов использования (use-case), диаграмма последовательности (sequence), диаграмма состояний. ER-диаграмма. Скриншоты разработанного программного продукта

Руководитель ст. преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ /Эгов Е.Н./

должность подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г.

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Белянин Н. Н./

подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**ОТЗЫВ руководителя на курсовую работу**

студента Белянина Никиты Николаевича

фамилия, имя и отчество

Факультет ИСТ группа ПИбд-21 курс второй

Дисциплина «Разработка профессиональных приложений»

Тема работы Разработка программы «Банк «Вы банкрот». Исполнитель»

Отмечаются следующие моменты: актуальность темы исследования; соответствие содержания и структуры курсовой работы ее теме; степень разработанности проблемы, наиболее интересно исследованные вопросы. Оценивается степень самостоятельности и инициативы студента; умение пользоваться различными источниками информации; уровень его теоретической подготовки; умение анализировать научные материалы, делать практические выводы; знание основных концепций, научной и специальной литературы по избранной теме. Содержится оценка проекта (работы) руководителем.

Руководитель ст. преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ /Эгов Е.Н./

должность, учёная степень, ученое звание подпись инициалы, фамилия

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г.

Содержание

[**Содержание 4**](#_Toc167922458)

[**Введение 5**](#_Toc167922459)

[**Первая глава 6**](#_Toc167922460)

[*Описание предметной области* 6](#_Toc167922461)

[*Поиск аналогов* 7](#_Toc167922462)

[*Техническое задание на разработку программного продукта «Банк «Вы банкрот». Исполнитель»* 11](#_Toc167922463)

[**Вторая глава 22**](#_Toc167922464)

[*Диаграмма вариантов использования* 22](#_Toc167922465)

[*Диаграмма последовательности* 23](#_Toc167922466)

[*Диаграмма состояний* 24](#_Toc167922467)

[*ER-диаграмма* 26](#_Toc167922468)

[*Применяемые технологии* 29](#_Toc167922469)

[*Распределение обязанностей* 31](#_Toc167922470)

[**Третья глава 32**](#_Toc167922471)

[*Руководство пользователя для разработанного проекта* 32](#_Toc167922472)

[*Руководство программиста для разработанного проекта* 44](#_Toc167922473)

[**Заключение 53**](#_Toc167922474)

[**Приложение 56**](#_Toc167922475)

Введение

На сегодняшний день информационные технологии становятся неотъемлемой частью нашей жизни. Они применяются повсеместно: в образовательных программах, в бизнесе и менеджменте, в медицине, в управлении, в планировании и т.д.

Все чаще руководителям требуется внедрить некоторое программное обеспечение в свое предприятие, чтобы облегчить работу. Например, с помощью грамотного программного обеспечения можно контролировать все процессы, происходящие при выполнении поставленной задачи, экономить время на рутинных и заурядных операциях, эффективно использовать ресурсы и, конечно, повышать качество товаров и услуг, предоставляемых клиентам. Помимо этого, стоит не забывать про автоматизацию бизнес-процессов, что может значительно улучшить работу предприятия. При выборе программного продукта важно учитывать такие факторы, как функциональность, стоимость, простота использования и возможность интеграции с уже существующими системами.

Темой данной курсовой работы является разработка программы для ведения учёта банковских карт и операций на снятие и пополнения наличных. Пользователи могут оформлять карты и совершать по ним различные операции. Программа предоставляет возможность получения отчётов по картам и сведений о переводах, пополнениях и получениях денежных средств в формате текстовых документов.

Первая глава

*Описание предметной области*

У данного проекта в качестве предметной области выступает банк «Вы банкрот». Каждому человеку хотя бы раз в жизни приходилось пользоваться услугами банковской системы. Весь мир непрерывно развивается, появляются новые технологии. Цифровизация начинает охватывать множество аспектов жизни людей, включая бизнес, образование, здравоохранение, государственное управление и повседневные задачи. Её влияние на мир в 21 веке очень огромно. Денежная система также не осталась без изменений.

На протяжении всего существования цивилизации экономика претерпевала всё новые и новые изменения. Постепенно от натурального хозяйства и бартера люди пришли к торговле и денежному обмену. Сейчас на смену бумажным купюрам и ассигнациям пришли электронные деньги и криптовалюта. В мире стали востребованы специалисты, которые смогут грамотно спроектировать и разработать новую финансовую систему, основанную на информационных технологиях. Банкиры вынуждены обращаться за помощью к программистам, которые построят безопасную, надежную, быструю и гибкую банковскую систему.

Кроме того, банк позволяет не только хранить денежные сбережения людей, но и манипулировать с ними. Пользователи могут дистанционно в режиме онлайн совершать покупки, переводить между счетами деньги или оплачивать коммунальные услуги. Также для клиентов и работников банка нужно предусмотреть возможность получения информации за определённый период по тому или иному счёту карты. Это позволит увеличить безопасность, информативность и удобство сервисом. Однако во всех этих процессах необходима автоматизация работы.

Решением этой проблемы является создание приложения для клиентов и сотрудников банка «Вы банкрот», позволяющее им взаимодействовать и получать необходимую информацию. Приложение должно обеспечить для исполнителя возможность работы с картами, выдаваемыми в банке, с заявками на получение наличных, а также с операциями пополнения счёта денежными средствами.

*Поиск аналогов*

При проектировании и разработке программы, связанной с работой банковской системой, стоит изучить различные аналоги, чтобы увидеть плюсы и минусы программных продуктов, продумать качественную концепцию разработки, определить цели и задачи. На данный момент уже существует множество приложений с удобным и качественным интерфейсом, предназначенным для клиентов, кассиров и банковских работников.

Рассмотрим следующие аналоги для указанной нами предметной области:

1. Коммерческий банк АО «ТИНЬКОФФ БАНК (https://www.tinkoff.ru/about/)

Тинькофф – онлайн-экосистема, основанная на финансовых и лайфстайл-услугами. Одна из самых быстрорастущих финансовых организаций в России, предоставляющая широкий спектр банковских услуг населению и бизнесу. Относительно молодой игрок на рынке, он за короткий срок сумел завоевать доверие клиентов благодаря инновационным подходам и цифровым технологиям. Сегодня Банк Тинькофф предлагает широкий выбор продуктов и услуг: от привычных депозитов, кредитов и карт до инвестиций и страхования. Клиентам доступны различные форматы обслуживания: онлайн-банкинг, мобильное приложение, крупную сеть отделений и пунктов выдачи. Благодаря своим уникальным цифровым решениям и инновационным продуктам, банк Тинькофф успешно конкурирует с традиционными финансовыми учреждениями и привлекает все больше клиентов.

Достоинства:

* Быстро и просто оформить банковскую карту в режиме онлайн и отделении банка.
* Быстрый и простой вывод средств на карту Тинькофф.
* Удобный и интуитивно понятный торговый терминал.
* Качественное мобильное приложение онлайн банка.
* Быстро и просто открыть инвестиционный счёт.
* Качественное мобильное приложение для торговли акциями, облигациями и инвестициями.
* Выгодные акции для новых клиентов, такие как месяц торговли без убытков, и т.п.
* Присутствует программа лояльности для постоянных клиентов.

Недостатки:

* Малый грейс период (Беспроцентный период по кредитной карте).
* Снятие наличных с комиссией.
* На снятие наличных не действует льготный период.
* Повышенный процент при снятии наличных.
* Нет наличия процентов на остаток собственных средств.

1. Банк ПАО «Сбербанк России» (https://www.sberbank.com/ru/about)

Сбербанк крупнейший конгломерат в России, а такжже в Центральной и Восточной Европе. Сбербанк предоставляет услуги в розничном и корпоративном банкинге: различные виды кредитования, лизинга и финансирования, выпуск кредитных карт и предоставление банковских ячеек, потребительские кредиты и зарплатные проекты, управление активами, брокерские услуги. В рамках стратегии Сбербанка по его трансформации в технологическую компанию, трехзначными темпами растет доля небанковских услуг — электронная коммерция, каршеринг, телеком, страхование, медицина. Всё это делает банк одним из передовых на сегодняшний день, а его банковское приложение одним из лучших.

Достоинства:

* Быстрое и удобное оформление банковских карт.
* Быстрое и качественное мобильное приложение «Сбербанк Онлайн».
* Удобный вывод средств на банковскую карту.
* Обширная сеть отделений и банкоматов и наличие бонусных программ.
* Большое количество предоставляемых услуг.
* Быстрота денежных переводов.
* Кредиты выдают без залога и поручителя.
* Кредитными и дебетовыми картами можно пользоваться за границей.

Недостатки:

* Большая комиссия при переводе денежных средств на все карты других банков.
* Клиенты сталкиваются с проблемами при оформлении кредитов, поскольку одобрение занимает много времени.
* При открытии счета или депозита нужно заполнить много бумаг, а ставки по вкладам низкие (1–7%).

1. Банк ПАО «ВТБ» (https://www.vtb.ru/about/bank/)

ПАО ВТБ является одним из крупнейших банков в России и имеет значительное влияние на развитие банковской системы страны. Банк активно участвует в различных проектах и инвестициях, что способствует его финансовому росту и укреплению позиций на рынке. Основные направления деятельности ПАО ВТБ включают розничное и корпоративное банковское обслуживание, инвестиционные операции и участие в международных проектах. Банк активно развивает инновационные проекты и внедряет цифровые технологии, чтобы обеспечить удобство и эффективность обслуживания клиентов

Достоинства:

* Удобное оформление банковских карт.
* Широкая география присутствия отделений банка.
* Разнообразие продуктов и услуг.
* Инновационные технологии.
* Профессиональная команда.

Недостатки:

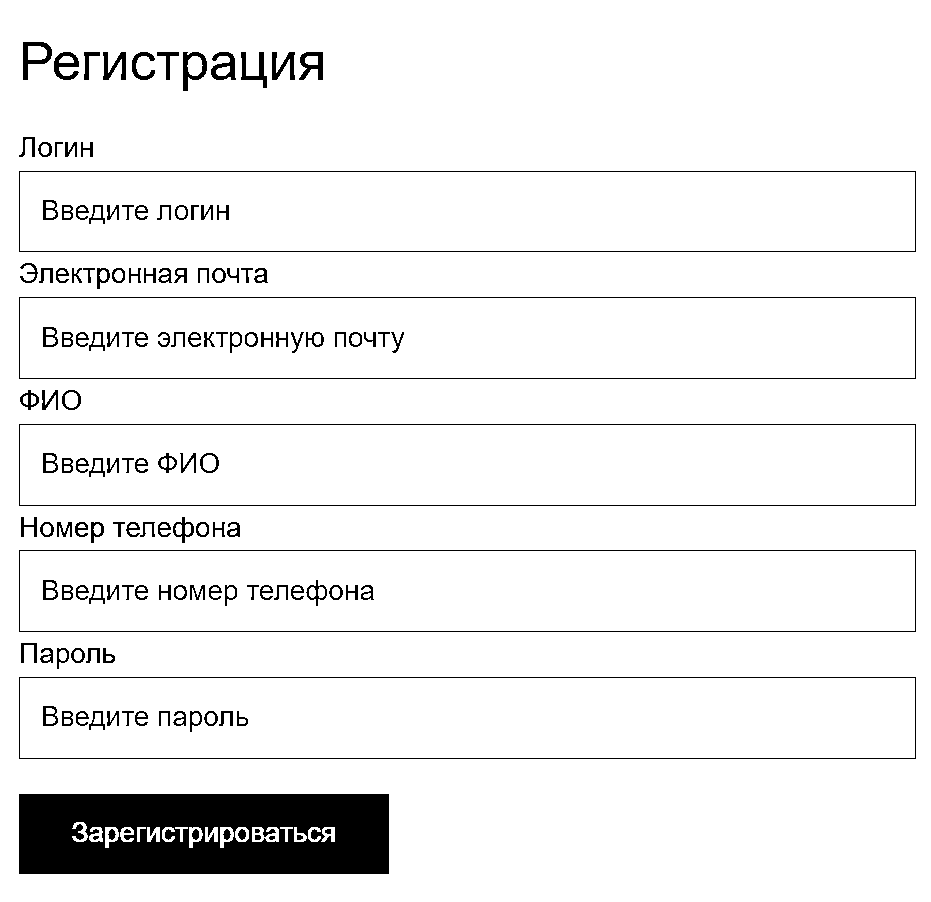
* Ограниченный ассортимент услуг: ВТБ может предлагать ограниченный ассортимент услуг по сравнению с некоторыми другими банками.
* Высокие комиссии: ВТБ может взимать высокие комиссии за некоторые свои услуги, что может быть неприемлемо для некоторых клиентов.
* Ограниченное мобильное приложение: Мобильное приложение ВТБ может иметь ограниченный функционал и не всегда быть удобным для использования.
* Сложность онлайн-банкинга.
* Неразвитая сеть отделений.
* Зависимость от государственных субсидий.
* Медленное и некачественное обслуживание клиентов.
* Плохая информативность клиентов.
* Длительное время обработки операций: Некоторые клиенты могут жаловаться на длительное время обработки транзакций и снятия наличных в банкоматах ВТБ.

*Техническое задание на разработку программного продукта «Банк «Вы банкрот». Исполнитель»*

Программа «Банк «Вы банкрот». Исполнитель» предназначена для ведения учета банковских карт. Пользователи указывают карты, заявки на получение с них наличных и операции на пополнения. Программа предоставляет возможность получения отчетов по картам.

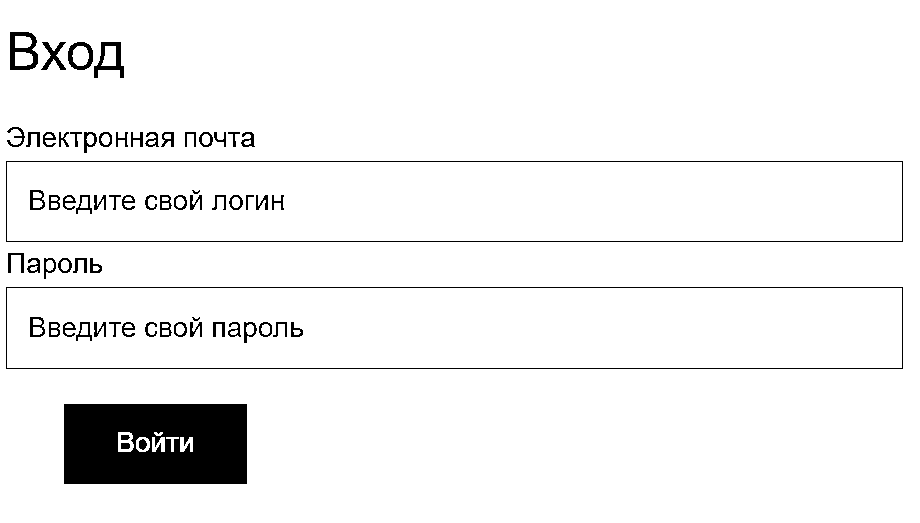
Основной функционал программы:

* Регистрация. Для регистрации пользователь должен заполнить следующие данные:
  + 1. Электронная почта. Электронная почта должна быть уникальна, иметь длину не более 50 символов. Обязательно к заполнению.
    2. Свое ФИО. Обязательно к заполнению
    3. Номер телефона. Уникален. Обязательно к заполнению.
    4. Пароль. Обязательно к заполнению. Не менее 10 символов.



Регистрация

* Авторизация. Пользователь вводит логин (электронную почту) и пароль. Если такой пользователь есть в системе, то происходит переход к основной форме. Если такого пользователя нет в системе, выводится сообщение об этом с просьбой ввести заново данные.



* Основная форма. Состоит из пунктов меню и логотипа. Через пункты меню можно перейти на формы работы с картами, операциями снятия и пополнения и на форму получения отчета по финансовым манипуляциям. Дополнительно, если реализовано, то вызов формы рекомендательной подсистемы.



Формирование банковских карт (CRUD). Имеется форма со списком всех банковских карт (Рисунок 1.1) и кнопки для создания (Рисунок 1.2), изменения (Рисунок 1.3) и удаления (Рисунок 1.4) банковской карты.

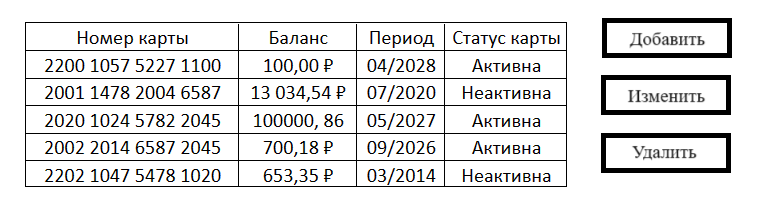
По банковской карте хранятся следующие сведения:

Номер карты.

Сумма денежных средств на карте

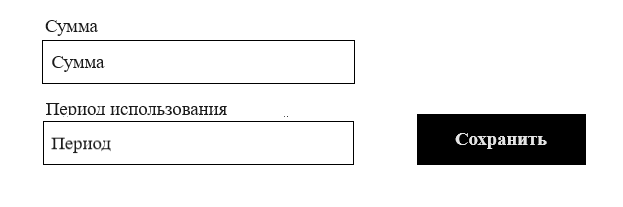
Период

Статус

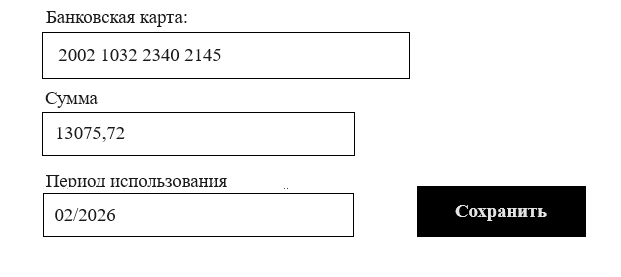


*Рисунок 1.1. Список банковских карт*

Номер карты и статус заполняются автоматически, заполнять следует только период пользования и сумму на карте



*Рисунок 1.2. Добавление новой карты*



*Рисунок 1.3. Изменение данных карты*



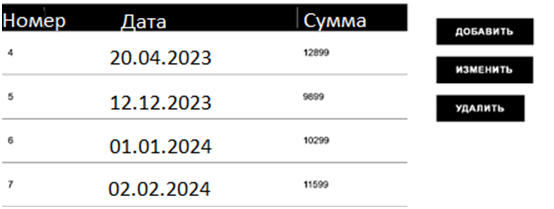
*Рисунок 1.4. Удаление банковской карты*

Формирование операций на пополнение для каждой банковской карты (CRUD). Имеется форма со списком всех банковских карт клиента (Рисунок 1.5) и кнопки для создания (Рисунок 1.6), изменения (Рисунок 1.7) и удаления (Рисунок 1.8) операции на пополнение.

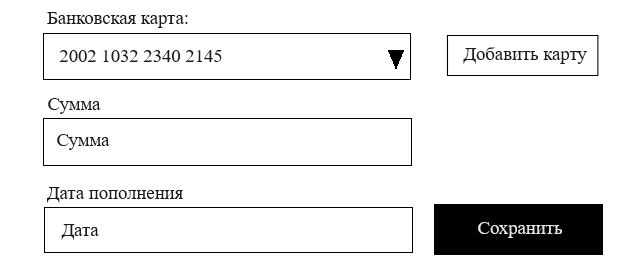
По операциям на пополнение хранятся следующие сведения:

Сумма

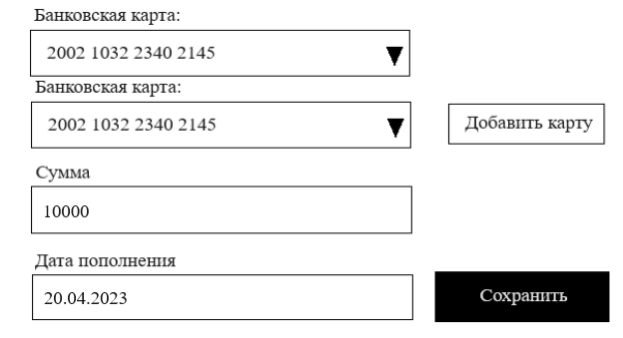
Дата операции



*Рисунок 1.5. Список операций на пополнение*



*Рисунок 1.6. Добавление операции на пополнение*



*Рисунок 1.7. Изменение операции на пополнение (списки пополнений, привязанных к данным картам)*



*Рисунок 1.8. Удаление операции на пополнение*

Формирование списка заявок на получение (CRUD). Имеется форма со списком всех заявок (Рисунок 1.9) и кнопки для создания (Рисунок 1.10), изменения (Рисунок 1.11) и удаления (Рисунок 1.12) заявок.

По конференции хранятся следующие сведения:

Сумма.

Дата заявки.

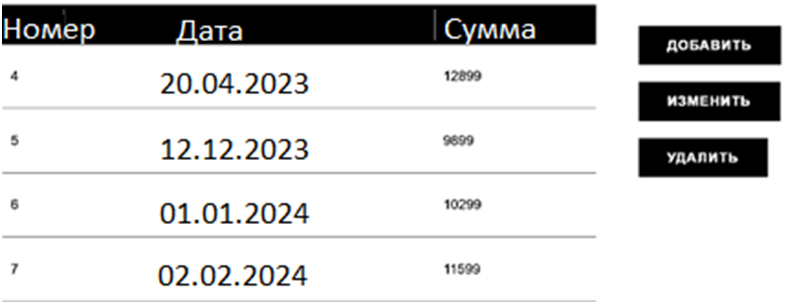


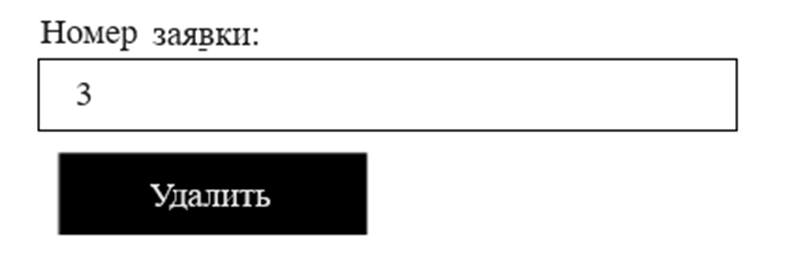
Рисунок 1.9. Список заявок



*Рисунок 1.10. Добавление заявки*



*Рисунок 1.11. Изменение заявки*



*Рисунок 1.12. Удаление заявки*

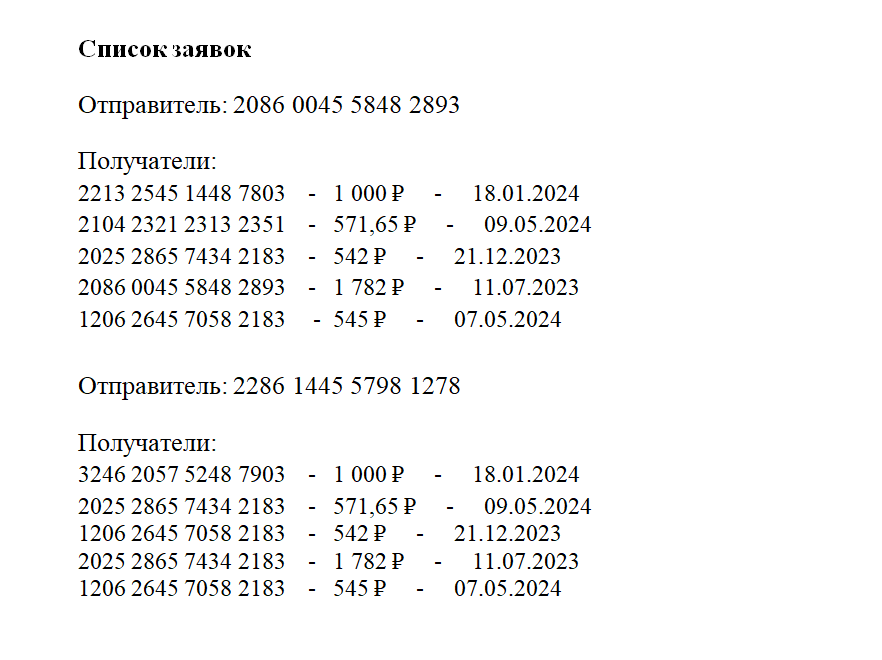
* Получение списка. Пользователь может получать списки заявок по выбранным картам в формате текстового документа или электронной таблицы. Пользователь отмечает интересующие его карты, указывает формат файла (doc или xls). (Рисунок 1.13).



карты

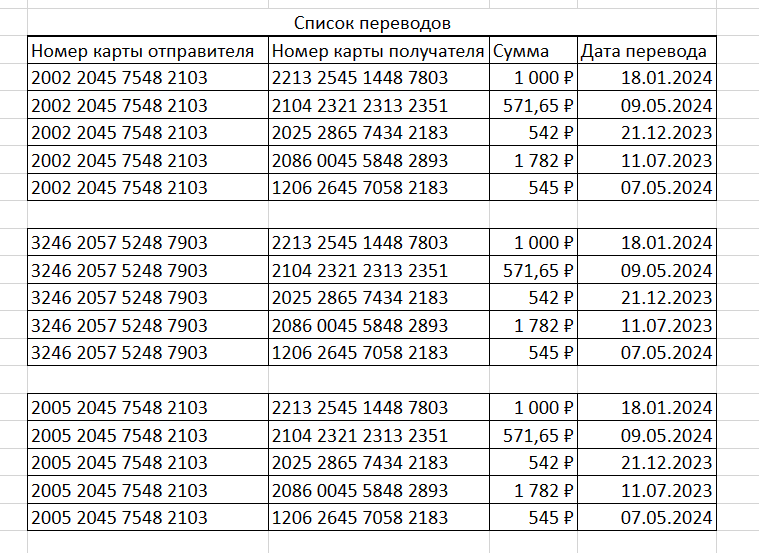
Рисунок 1.13. Форма для формирования отчета в формате docx/xlsx

При выводе в doc-файл формат документа будет следующим:



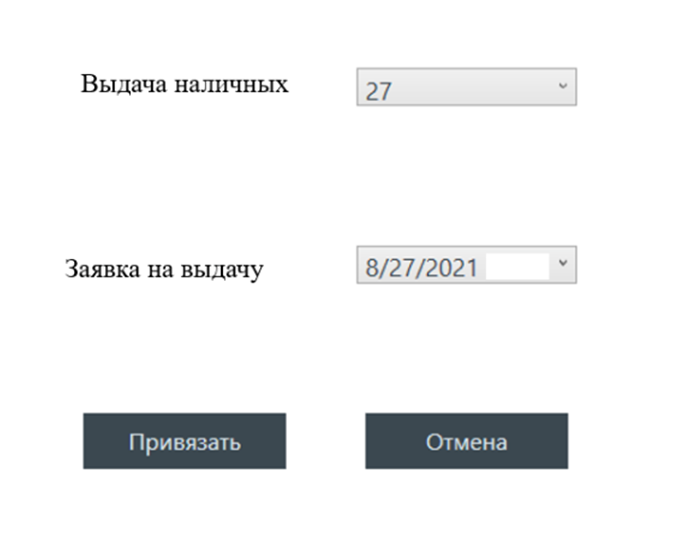
*Рисунок 1.14. Пример вывода в doc – формате*

При выводе в xls-файл формат документа будет следующим:



*Рисунок 1.15. Пример вывода в xls - формате*

* Привязка к операции на перевод денег. Пользователь может выбрать номер перевода, операцию на перевод и связать их (Рисунок 1.16).



Перевод наличных

Операция на перевод

Рисунок 1.16. Форма для привязки переводов

Пользователь может получить сведения за период по картам, операциями на пополнение и выдачами наличных. Для этого он должен указать интересующий его период, выбрать вариант получения сведений (отправка на почту или вывод на форму) и получить отчет (Рисунок 1.17).



Рисунок 1.18. Форма для формирования отчета в формате pdf



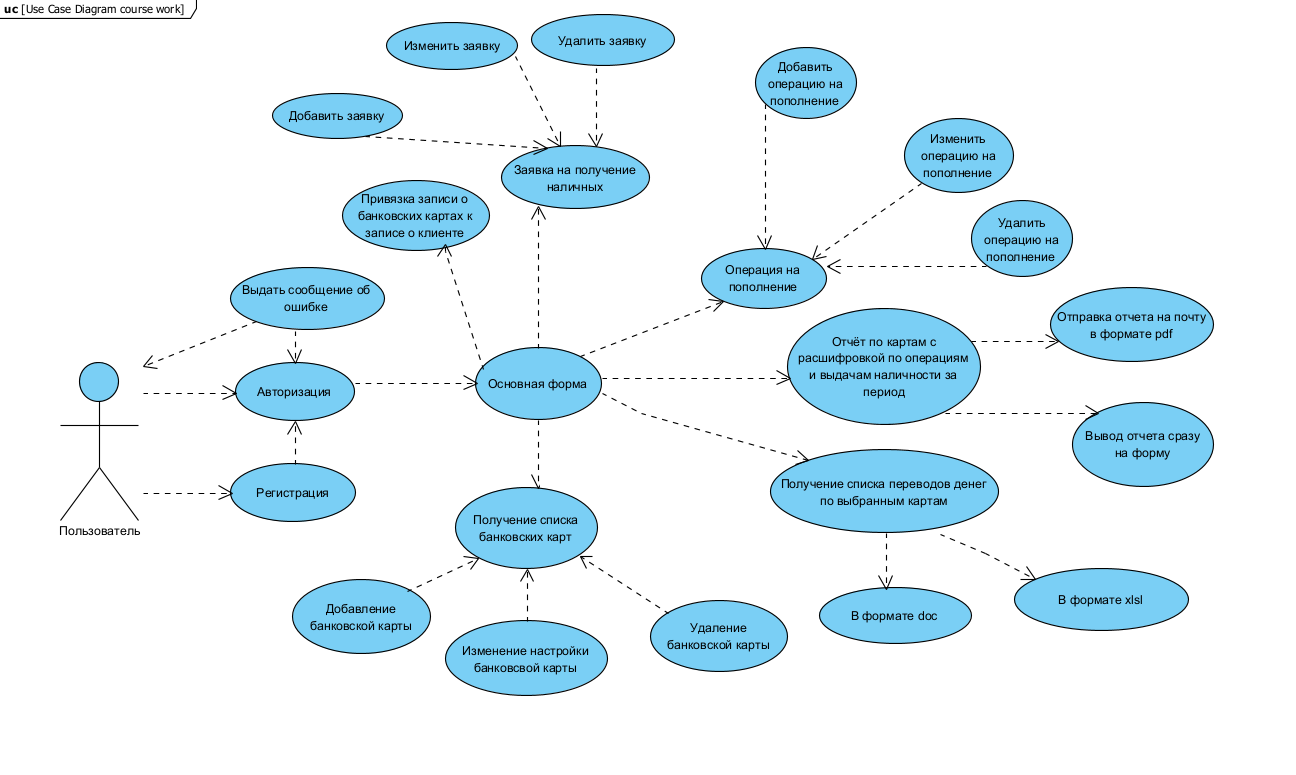
*Рисунок 1.19 Пример вывода отчета в формате pdf*

Вторая глава

*Диаграмма вариантов использования*

Диаграмма вариантов использования отражает отношения между актёрами и прецедентами и является составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Основная цель построения диаграммы вариантов использования для данной работы – определить действующих лиц (актёров), их взаимодействие с системой и ожидаемую функциональность системы.



*Рисунок 2.1. Диаграмма вариантов использования (UseCase)*

Пользователь имеет 2 возможных действия в системе: регистрация или авторизация (если такой пользователь уже есть в системе). Оба этих действия имеют общую часть функционала – действие «Проверка корректности введенных данных». У этого действия имеется необязательное действие «Выдать сообщение об ошибке».

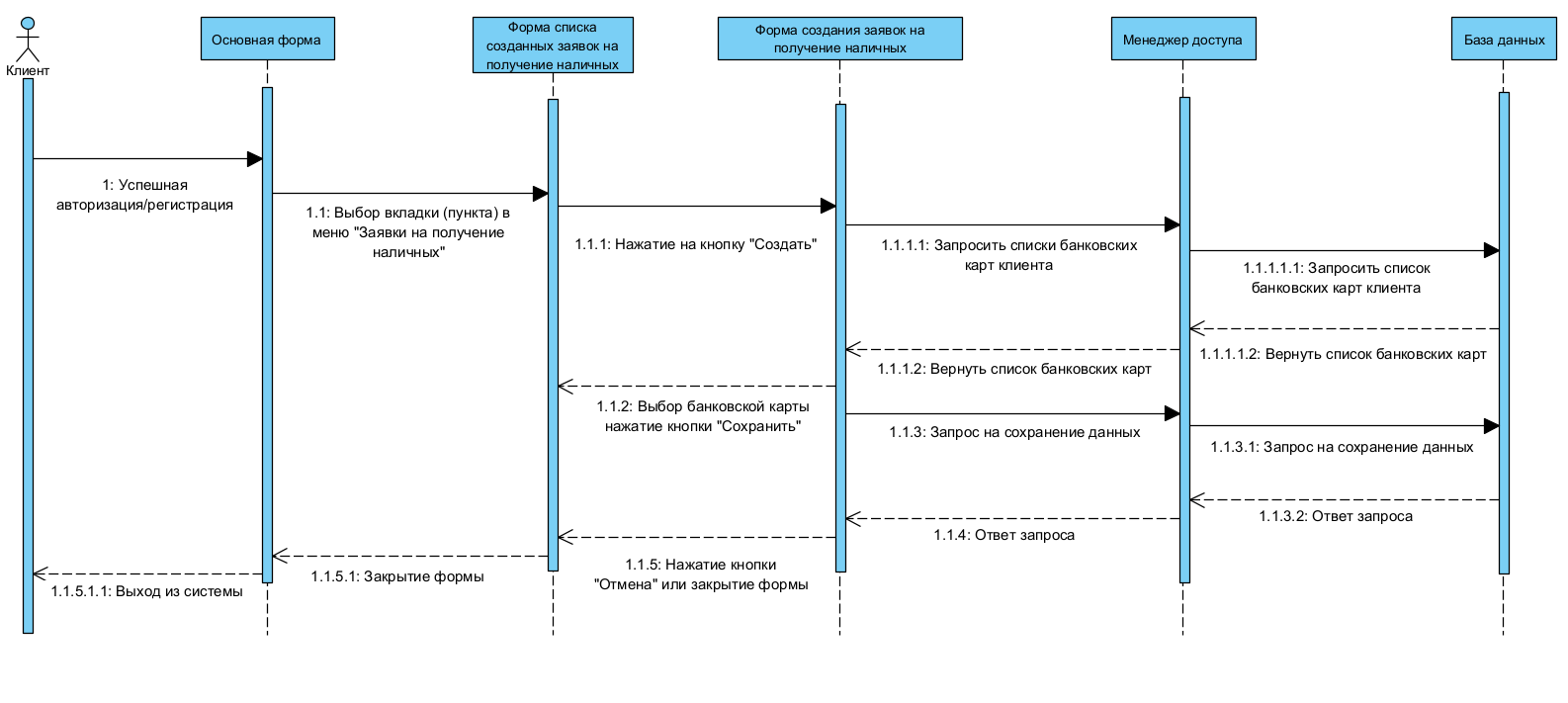
При выполнении действия «Авторизация» выполняется вход с лица клиента, после чего он переходит на основную форму (страницу).

После успешной авторизации клиенту необходимо выбрать пункт «Получить список банковских карт». После создания банковской карты можем перейти в пункт «Операции на пополнение», где происходит передача денежных средств на карту клиента, затем можем перейти на «Заявка на получение наличных», где происходит передача денежных средств клиенту, либо совершить перевод по карте, перейдя на «Операции с переводами денег». После того как совершены действия по карте, описанные выше, можем получить список переводов денег по выбранным картам в формате .doc или .xls.

Также после создания банковской карты и её пополнения, клиент может получать отчет по карте. Получить отчет по карте за период с возможностью просмотра на форме (странице) или же с отправкой на почту в формате pdf.

*Диаграмма последовательности*

Диаграмма последовательности – это диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл какого-либо определённого объекта (создание-деятельность-уничтожение некой сущности) и взаимодействие актёров (действующих лиц) ИС в рамках какого-либо определённого прецедента (отправка запросов и получение ответов).



*Рисунок 2.2. Диаграмма последовательности*

На данной диаграмме есть один актер – Клиент. В диаграмме отображена последовательность запроса данных по банковским картам при создании заявки на получение наличных.

Сначала на основной форме выбираем вкладку «Заявки», затем в новой форме нажимаем кнопку создать и появляется третья форма, на которой из списка выбираем банковскую карту, делаем запрос из базы данных. После того, как заявка создана, нажимаем «Сохранить», и данные о сохранении передаются в базу данных. Если сохранять изменения не нужно, нажимаем «Отмена»

*Диаграмма состояний*

Диаграмма состояний представляет динамическое поведение сущностей, на основе спецификации их реакции на восприятие некоторых конкретных событий. Визуализирует модель поведения объекта или системы в различных состояниях. Она позволяет описать все возможные состояния объекта, а также переходы между ними в ответ на определённые события.

Используются следующие условные обозначения:

• круг, обозначающий начальное состояние;

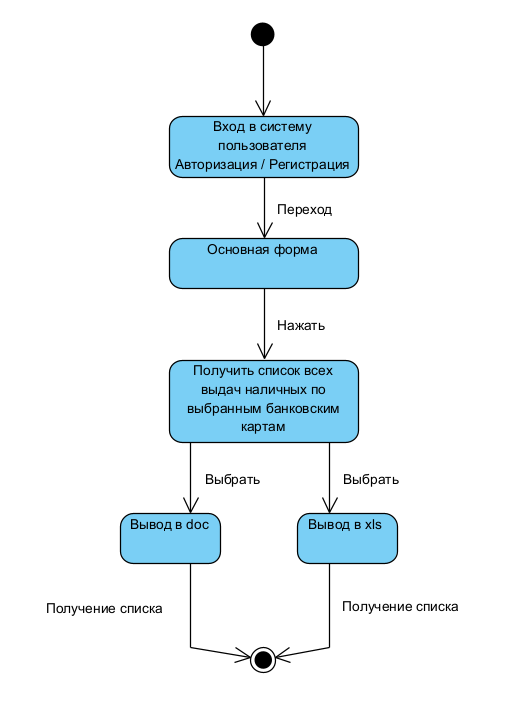
• окружность с маленьким кругом внутри, обозначающая конечное состояние (если есть);

• скруглённый прямоугольник, обозначающий состояние;

• стрелка, обозначающая переход;

• толстая горизонтальная линия с либо множеством входящих линий и одной выходящей, либо одной входящей линией и множеством выходящих, обозначающая объединение и разветвление соответственно.

Диаграмма состояний напоминает блок-схему и показывает поведение системы в тех или иных условиях. Например, на данной диаграмме демонстрируется поведение системы при выводе списка всех выдач наличных по банковским картам. Если пользователь существует, то происходит переход на основную форму, где можно будет запросить список. Пользователю предлагается выбор в каком формате он может получить список по картам.



*Рисунок 2.3. Диаграмма состояний*

*ER-диаграмма*

В рамках курсовой работы требуется составить ER-диаграмму сущностей, которые будут использоваться в разрабатываемом проекте. Выделить необходимые для реализации атрибуты, задать тип данных.

ER-модель — модель данных, позволяющая описывать концептуальные схемы предметной области. С её помощью можно выделить ключевые сущности и обозначить связи, которые могут устанавливаться между этими сущностями.

Необходимо дать названия сущностям и атрибутам. Кроме того, атрибутам нужно прописать тип данных. А также выстроить связи между сущностями.

ER-диаграмма содержит следующие сущности:

* Клиент – пользователь.
* Кассир – пользователь.
* Банковская карта – банковская карта привязана к пользователю и по ней совершаются заявки на получение наличных и операции на пополнение.
* Банковский счёт – это счёт, который привязан к кассиру, по которому совершаются переводы и выдача наличных.
* Заявки на получение наличных – это получение наличных с банковской карты клиента.
* Операции на пополнения – внесение наличных на банковскую карту клиента.
* Выдача наличных – снятие денег со счёта кассира и перевод в наличные.
* Переводы денег – это трансфер денег на другой счёт.

ER-диаграмма содержит следующие связи между сущностями:

Клиент — заявки на получение наличных (Один ко многим), поскольку в рамках работы программы только один клиент взаимодействует с множеством заявок на получение наличных.

Клиент — банковская карта (Один ко многим), поскольку в рамках работы программы только один клиент взаимодействует с множеством собственных, доступных банковских карт.

Клиент — операции на пополнение (Один ко многим), поскольку в рамках работы программы только один клиент взаимодействует с множеством операций на пополнение.

Банковская карта — заявки на получение наличных (многие ко многим), поскольку в рамках работы программы клиенту предоставляется возможность снимать наличные с разных, доступных ему, банковских карт и таких заявок может быть много.

Банковская карта — операции на пополнение (многие ко многим), поскольку в рамках работы программы клиенту предоставляется возможность пополнять все свои банковские карты и таких пополнений можно производить большое множество.

Кассир — выдача наличных (Один ко многим), поскольку при работе с выплатой денег в рамках выполнения данной программы только один кассир взаимодействует с множеством выдач наличных.

Кассир — счёт (Один ко многим), поскольку в рамках работы данной программы один кассир взаимодействует с множеством, доступных ему, счетов в банке.

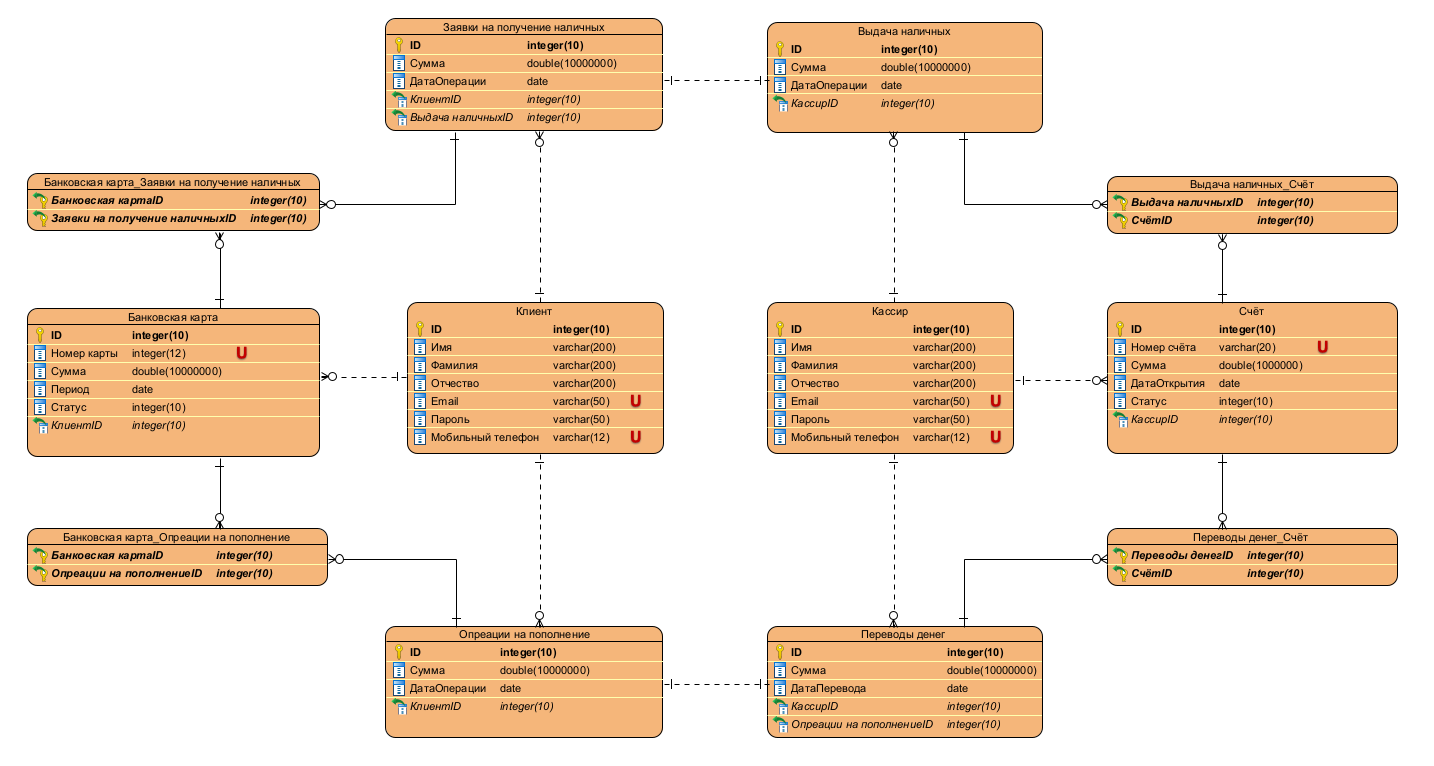
Кассир — переводы денег (Один ко многим), поскольку в рамках работы данной программы один кассир производит трансфер денег между счетами и таких переводов может быть много.

Счёт — выдача наличных (Многие ко многим), поскольку кассир при работе может выбирать, доступные ему, счета и выдавать наличные множество раз.

Счёт — переводы денег (Многие ко многим), поскольку кассир при работе может выбирать множество, доступных ему, счетов и пополнять их наличными.

Заявка на получение наличных — выдача наличных (Один ко многим), поскольку в рамках выполнения программы заявка на получение наличных соответствует одной конкретной выдаче наличных. Совершаться их может множество, но клиенты и кассиры запрашивают информацию по конкретно выполненным выдачам и заявкам на получение.

Операция на пополнение — перевод денег (Один ко многим), поскольку в рамках выполнения программы одна операция соответствует одному конкретному переводу денег и в дальнейшем для получения информации клиенты и кассиры будут запрашивать данных по определённым, соответствующим друг другу переводам и пополнениям.



*Рисунок 2.5. ER-диаграмма*

## *Применяемые технологии*

При разработке программы присутствует реализация концепции DAL, в котором есть как минимум 4 следующих уровня: контракты, бизнес-логика, интерфейс пользователя и хранение данных. Есть блок, отвечающий за реализацию интерфейсов и два блока, отвечающих за внешний вид форм со стороны Клиента и Кассира. Так же два блока, в которых находятся сущности для Клиента и Кассира и блок сервера с базой данных.

Слой доступа к данным (Data Access Layer — DAL) в программном обеспечении — это слой компьютерной программы, который предоставляет упрощённый доступ к данным, хранимым в постоянном хранилище какого-либо типа, таком как реляционная база данных. Данная концепция позволяет обеспечить возможность смены хранилища данных без изменения бизнес-логики проекта. В основе концепции заложен принцип управляемой работы с данными, который определяет унифицированный и контролируемый способ доступа к различным данным для приложения. Предлагается создавать интерфейсы и описывать методы работы с данными в хранилище. При этом реализаций этих интерфейсов может быть множество, в зависимости от различных хранилищ данных, с которыми предстоит работать. Бизнес-логика использует не конкретный способ хранения данных, а абстракции в виде интерфейсов. Этот акроним в основном используется в Microsoft ASP.NET окружении. Например, DAL может возвращать ссылку на объект (в ООП) с его атрибутами вместо строк полей из таблицы базы данных. Это позволяет создавать клиентские (или пользовательские) модули с более высоким уровнем абстракции.

В качестве интерфейса пользователя будет выступать Web-клиент, разработанный на ASP.NET. ASP.NET(Active Server Pages.NET) – это платформа создания веб-приложений и веб-сервисов на языке программирования C#. Она позволяет разработчикам создавать динамические веб-страницы, работать с базами данных, обрабатывать веб-формы и многое другое. ASP.NET работает на платформе Microsoft Windows и использует технологию .NET Framework для компиляции и выполнения кода.

*Распределение обязанностей*

Крюков Алексей будет реализовывать роль «Поручитель», а именно интерфейс, реализацию базы данных, бизнес логику, а также часть базы данных и бизнес логики, где связываются записи выдачи наличности с заявками на получение наличных.  
 Белянин Никита будет реализовывать роль «Исполнитель», а именно интерфейс, реализацию базы данных, бизнес логику, а также часть базы данных и бизнес логики, где связываются переводы денег с операциями на пополнение.  
 Пояснительная записка, дизайн веб-приложения, реализация ER-диаграммы, работа с git делается совместно.

# **Третья глава**

## *Руководство пользователя для разработанного проекта*

Основное стартовое окно программы

После запуска нашего Web-приложения, открывается стартовое окно (Рисунок 3.1). В верхней части сайта (header) представлен логотип, название Web-клиента, функциональные опции работы с приложением и кнопки для входа и регистрации. В основной части окна основная информация и просьба авторизоваться для дальнейшей работы. В нижней части (Footer) указан автор. Пользователю необходимо нажать "Войти", если он уже зарегистрировался или "Регистрация", если ещё этого не делал. Нажав на опции в верхней части окна, пользователя перебросит на основное главное окно, с просьбой авторизоваться. Без прохождения аутентификации дальнейшая работа с приложением будет недоступна.

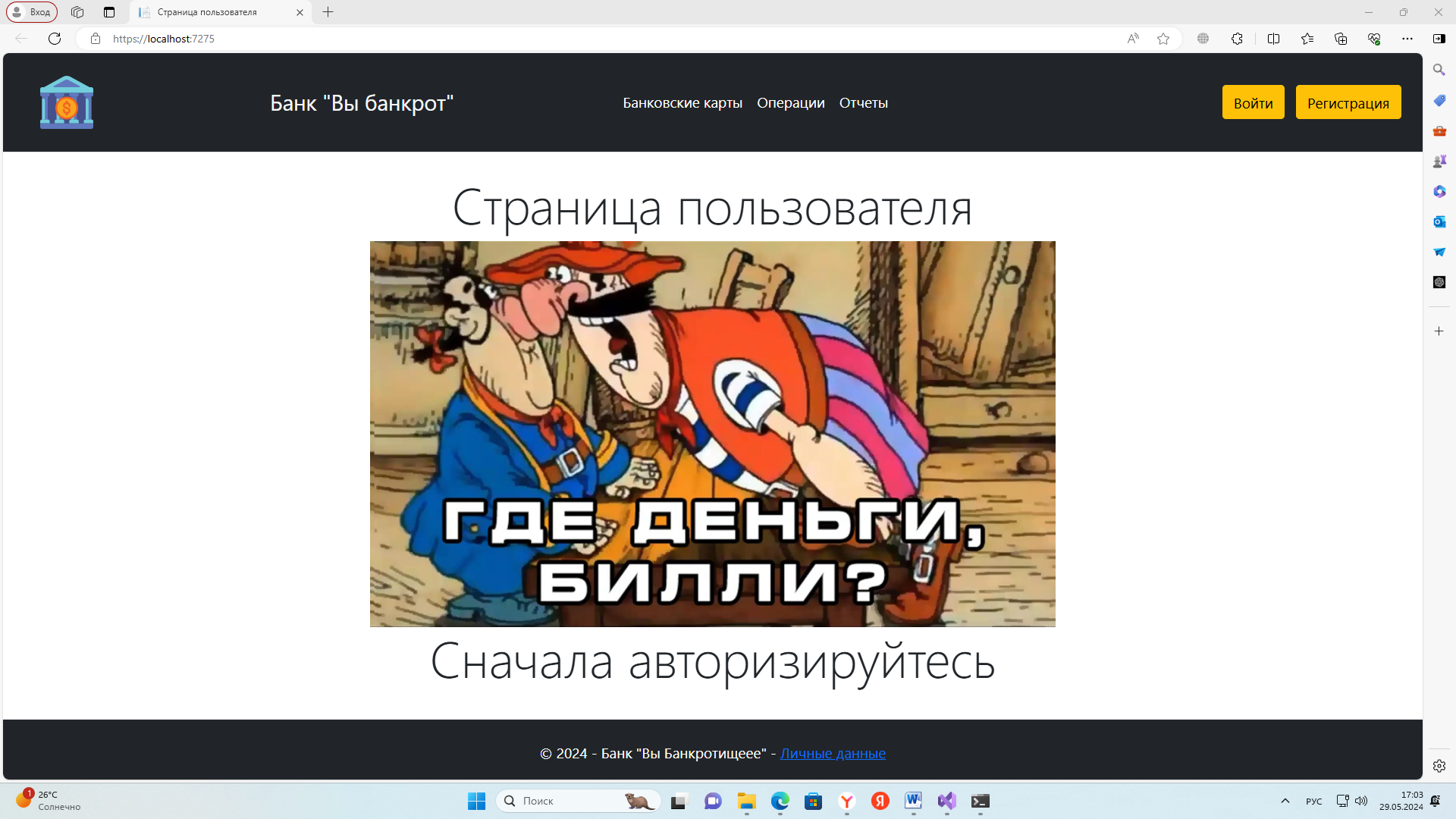


Рисунок 3.1. Главное окно программы

Окно "Вход в приложение"

Кликнув по кнопке "Войти", пользователь перейдёт на новое окно (Рисунок 3.2), где в теле окна два поля для заполнения логина (электронной почты) и пароля от аккаунта и кнопка входа. Пользователю нужно заполнить все поля и нажать на кнопку "Войти". При правильных данных откроется основная форма.

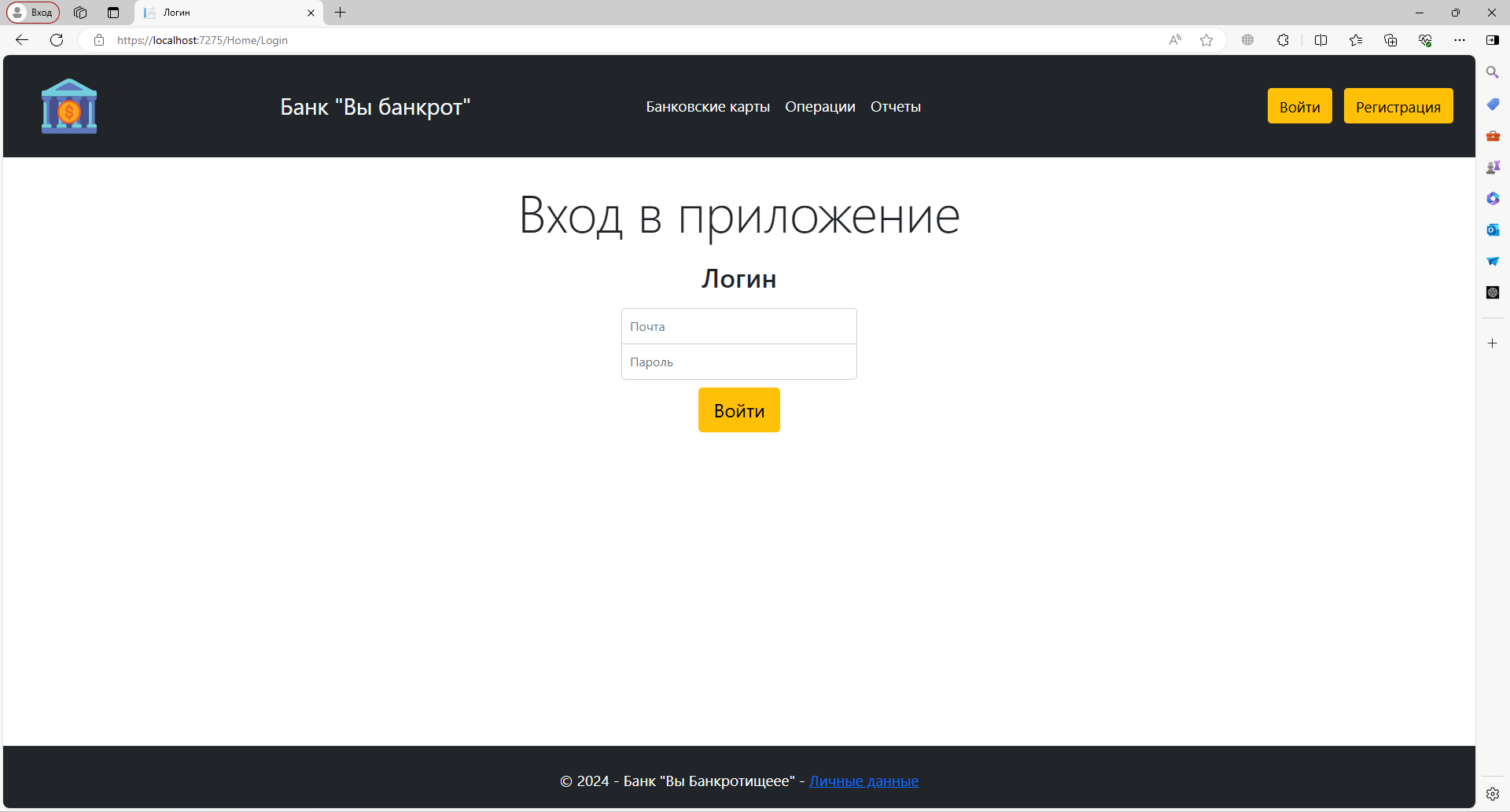


Рисунок 3.2. Вход в приложение

Окно "Регистрация"

Кликнув по кнопке "Регистрация", пользователь перейдёт на новое окно (Рисунок 3.3), где в теле окна шесть полей для заполнения: Почта, Пароль, Имя, Фамилия, Отчество, Телефон и кнопка для регистрации. Пользователю нужно корректно заполнить все поля и нажать на кнопку "Регистрация". При правильных данных пользователь зарегистрируется и автоматически вернётся на основную страницу.

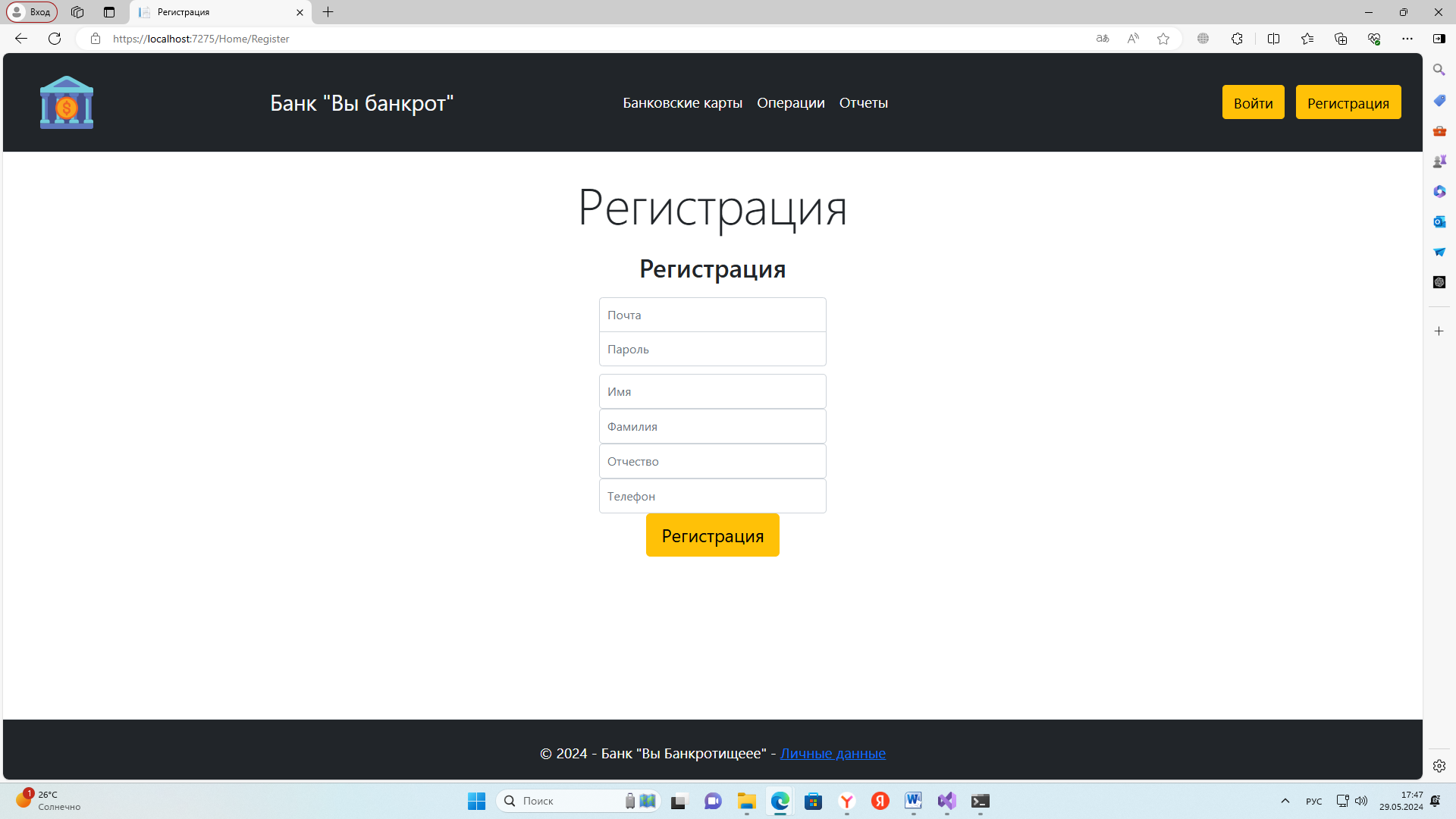


Рисунок 3.3. Регистрация

Главное меню

Откроется главное меню, где пользователя поприветствуют и можно будет выбрать интересующую опцию (Рисунок 3.4).

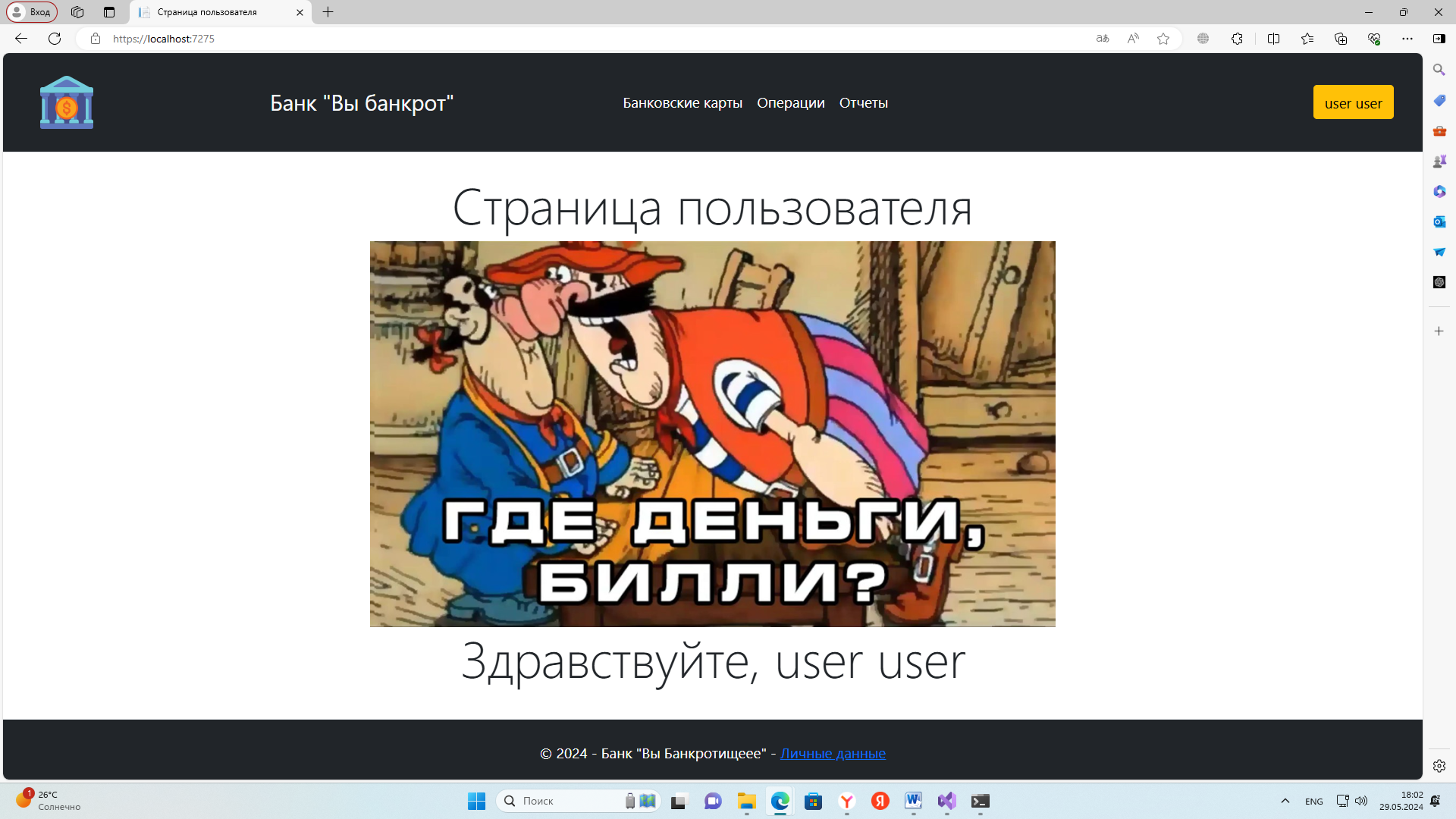


Рисунок 3.4. Главное меню

Окно "Личные данные"

Если кликнуть по кнопке с именем и фамилией в верхнем левом углу, то пользователь перейдёт в окно "Личные данные" (Рисунок 3.5), где он сможет отредактировать сведения о ФИО, почте, номере телефона и пароле. В окне будет представлены поля с данными, которые можно отредактировать. После, нажав на кнопку "Сохранить", данные зафиксируются, и пользователь перенесётся на главное меню. Также предусмотрен выход из Web-приложения. Кликнув по кнопке "Выйти", пользователь покинет приложение и попадёт на гостевую страницу. После ему заново нужно будет авторизоваться

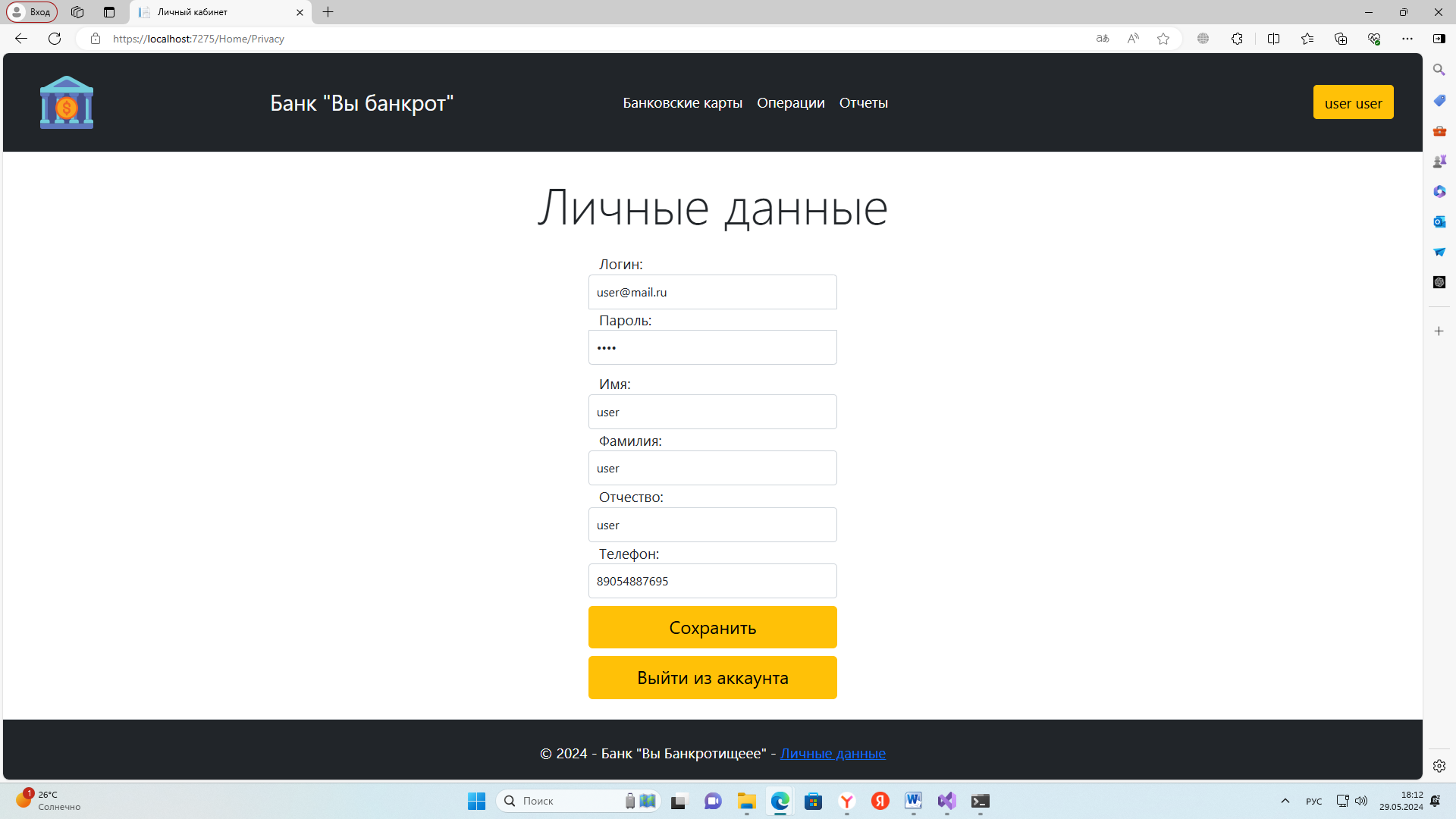


Рисунок 3.5. Личные данные

Банковские карты

Кликнув по опции "Банковские карты" на главной странице, пользователь попадёт на вкладку со списком его банковских карт (Рисунок 3.6). Там ему будет представлены все сведения: баланс, номер карты, CVC и срок действия. Также предусмотрена кнопка для оформления новой банковской карты. Нажав, на кнопку "Создать карту", пользователь попадёт на окно с созданием банковской карты. Если у пользователя нет, доступных банковских карт, ему необходимо её оформить.

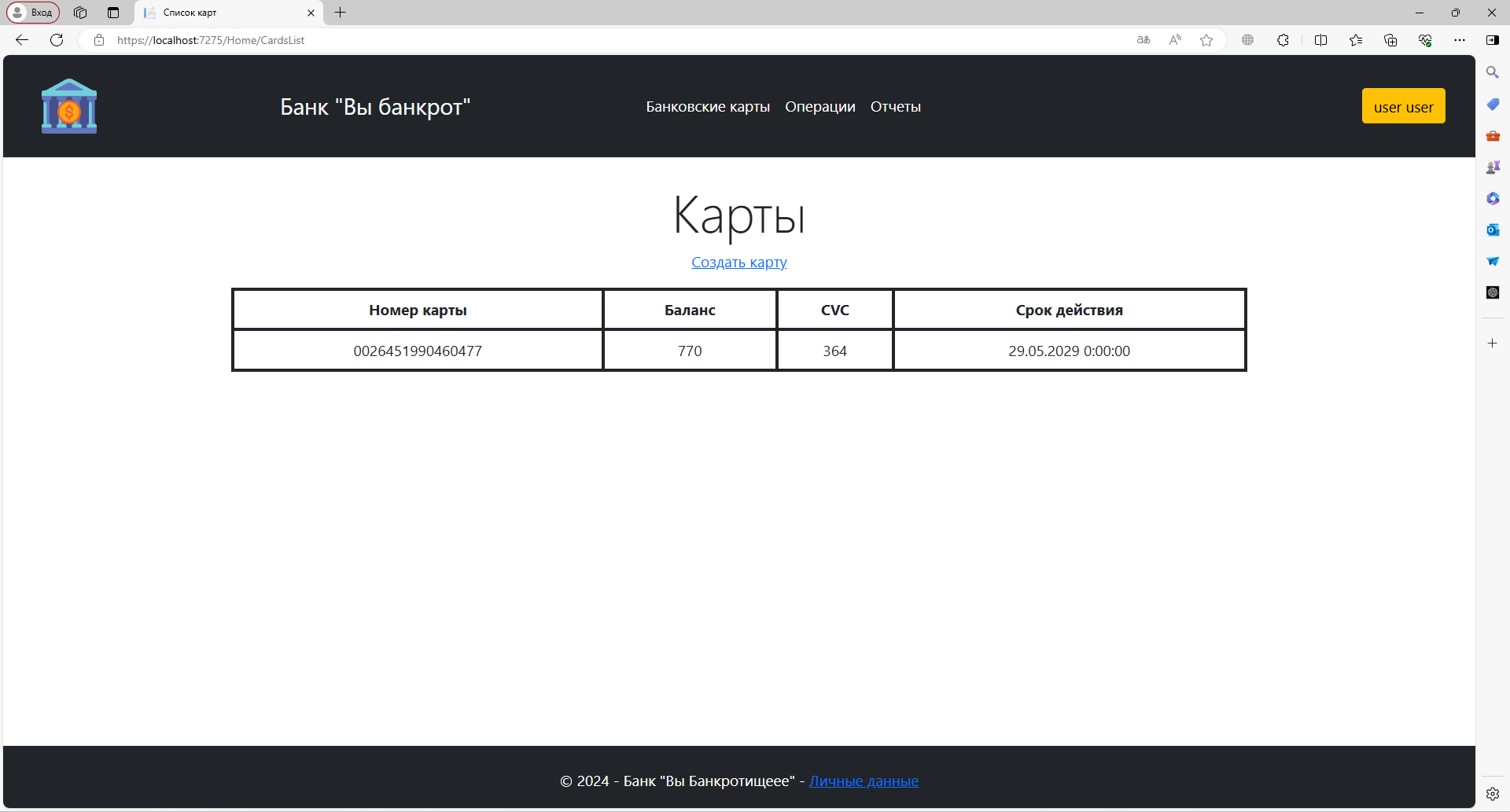


Рисунок 3.6. Список карт

Создание банковской карты

Перейдя на вкладку "Создать карту" (Рисунок 3.7), пользователю нужно будет ознакомиться с полями, которые он может отредактировать по необходимости. Сами поля генерируются автоматически, которые не были ещё заняты другими пользователями. Срок действия банковской карты определяется на 5 лет. Далее пользователю предстоит нажать на кнопку "Создание" и у него создастся банковская карта. Клиента автоматически перенаправит на список всех доступных ему карт, там же отобразится только что созданная. Данную операцию он может повторить при необходимости, ограничений на количество нет.

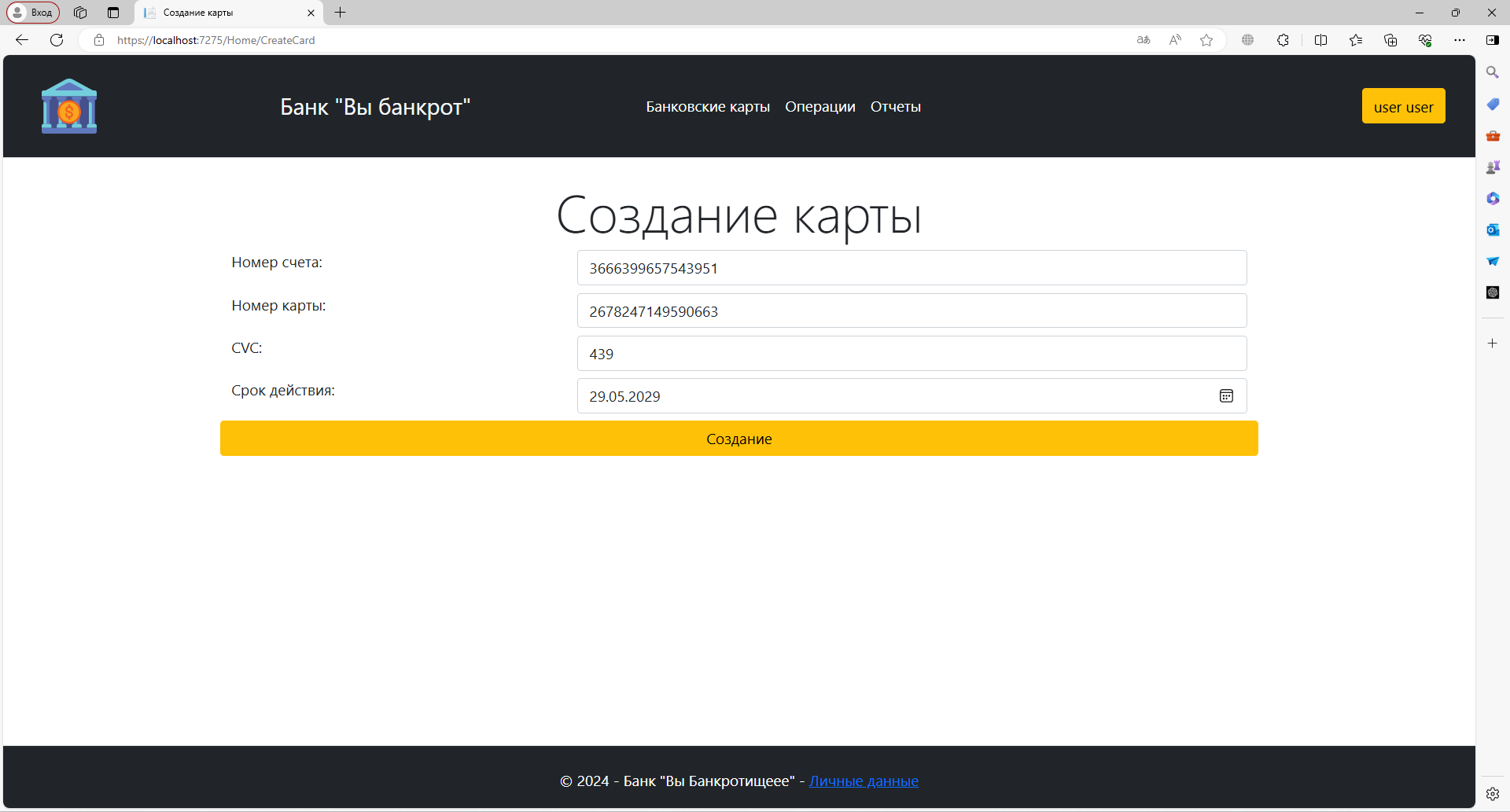


Рисунок 3.7. Создание карты

Операции на пополнение

Далее пользователь может оставить заявку на пополнение карты и на снятие. Перейдя на вкладку с заявками на начисление и на снятие. Наведя курсор мыши на "Операции", появится выпадающий список. Выбрав "Заявки на начисления" (Рисунок 3.8), клиент попадёт на вкладку с пополнением карты (Рисунок 3.9). Там ему будут предоставлены сведения по всем заявкам на операции пополнения по его банковским картам: номер карты, сумма, статус, дата открытия и закрытия заявки. По кнопке "Пополнить средства" клиент может оставить новую заявку на пополнение банковской карты.

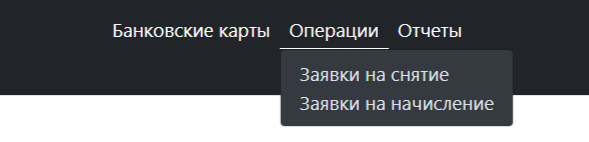


Рисунок 3.8. Выпадающий список

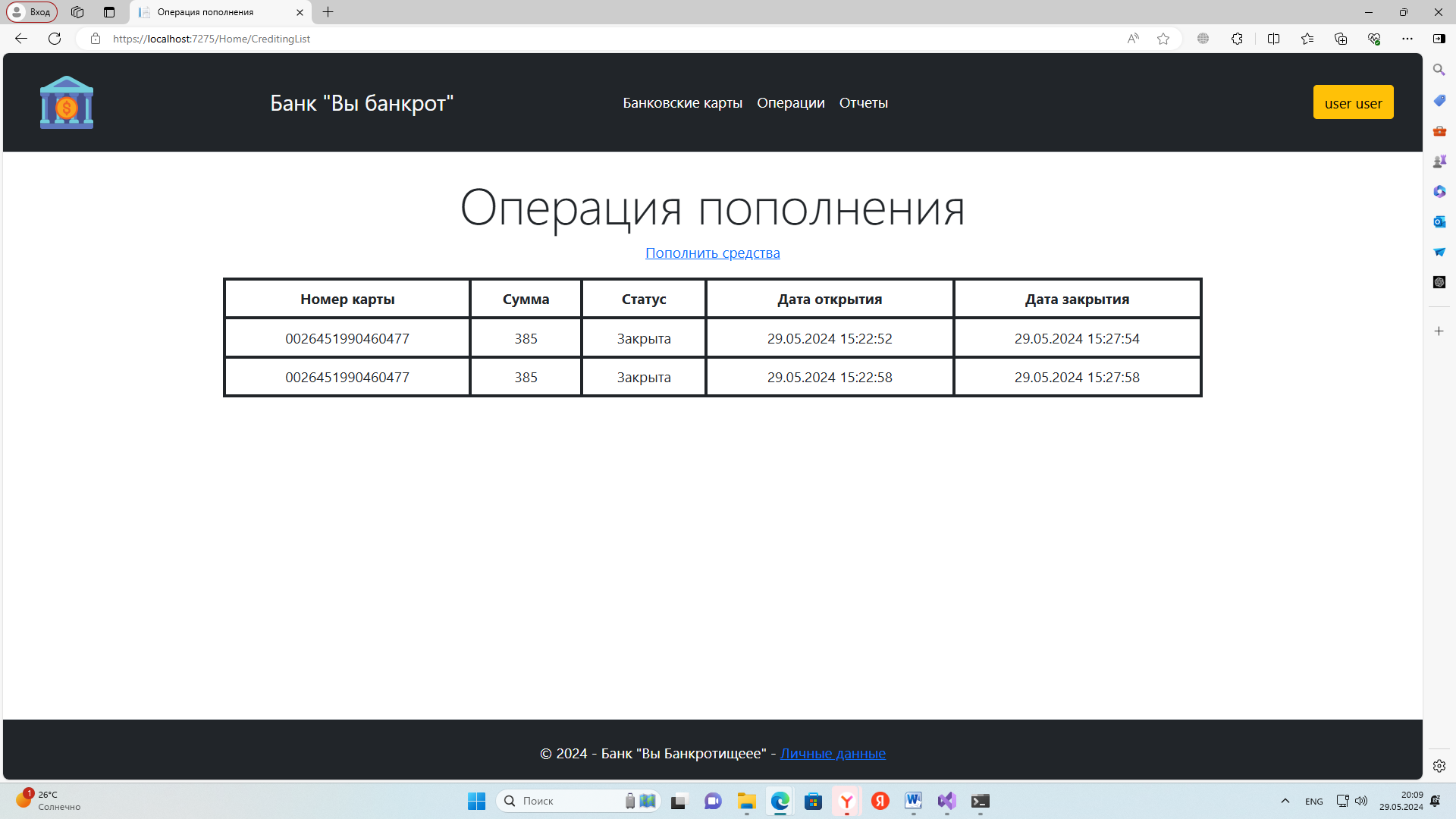


Рисунок 3.9. Операция пополнения

Создание операции на пополнение

Перейдя на вкладку с созданием операции на пополнение (Рисунок 3.10), пользователь из выпадающего списка должен выбрать нужную карту и указать сумму на которую он хочет пополнить карту. Далее нажать на кнопку "Создание". Далее пользователя перенесёт на список всех созданных заявок, если ему их одобрили, то заявки будут со статусом "Закрыта" или "Отклонена", если карта была просрочена или операция невозможна. Таких заявок клиент может создавать неограниченное количество.

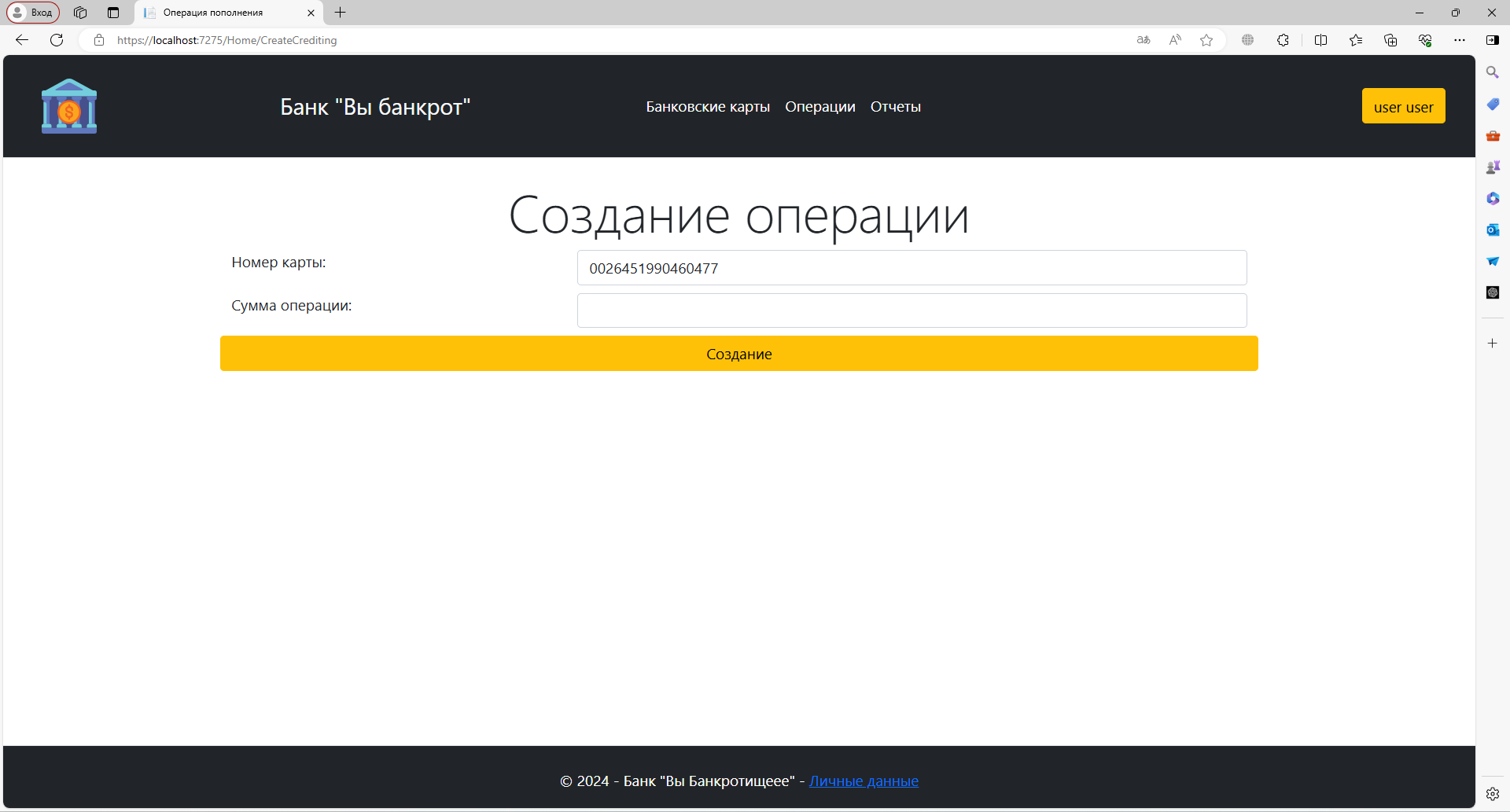


Рисунок 3.10. Создание операции на пополнение

Операции на снятие

Далее пользователь может оставить заявку на снятие наличных с банковской карты. Перейдя на вкладку с заявками на снятие. Наведя курсор мыши на "Операции", появится выпадающий список. Выбрав "Заявки на снятие" (Рисунок 3.8), клиент попадёт на вкладку со снятиями наличных с карты (Рисунок 3.11). Там ему будут предоставлены сведения по всем заявкам на операции снятия наличных по его банковским картам: номер карты, сумма, статус, дата открытия и закрытия заявки. По кнопке "Снять средства" клиент может оставить новую заявку на снятие наличных с банковской карты.

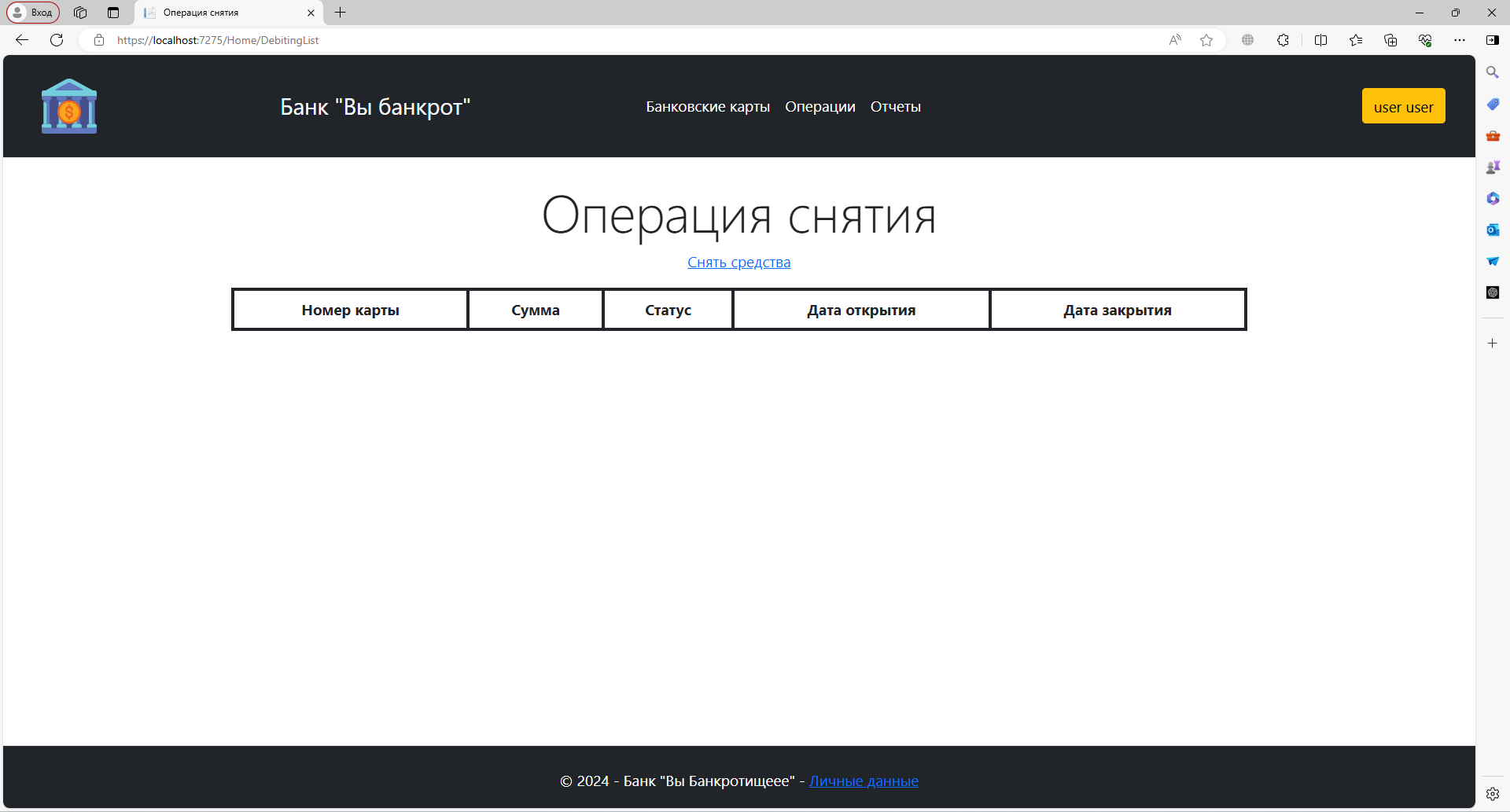


Рисунок 3.11. Операция снятия

Создание операции на снятие

Перейдя на вкладку с созданием операции на снятие (Рисунок 3.12), пользователь из выпадающего списка должен выбрать нужную банковскую карту и указать сумму, которую он хочет снять с карты. Далее нажать на кнопку "Создание". Далее пользователя перенесёт на список всех созданных заявок, если ему их одобрили, то заявки будут со статусом "Закрыта" или "Отклонена", в зависимости от того была ли карта просрочена или операция невозможна. Таких заявок клиент может создавать неограниченное количество. А их уже одобрит кассир.

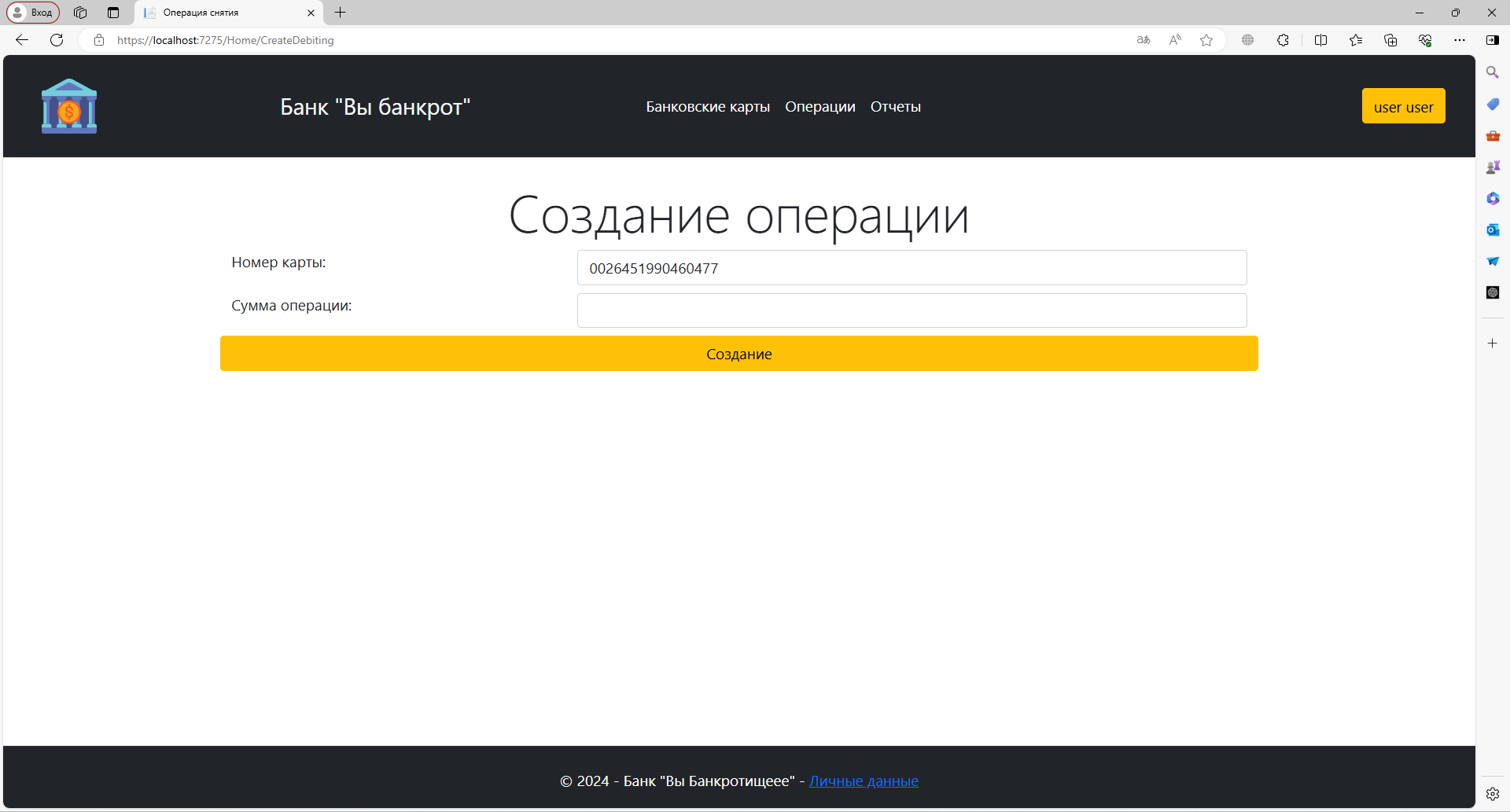


Рисунок 3.12. Создание операции на снятие

Отчёт по банковским картам

Далее пользователь может получить отчёт по банковским картам и отчёт за период. Перейдя на вкладку с отчётом по банковским картам. Наведя курсор мыши на "Отчёты", появится выпадающий список. Выбрав "Отчёты по банковским картам" (Рисунок 3.13), клиент попадёт на вкладку с отчётами по картам (Рисунок 3.14).

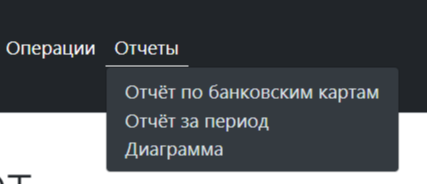


Рисунок 3.13. Отчёты

Далее перейдя на вкладку с "Отчётами по банковским картам", пользователю необходимо выбрать нужные банковские карты, для этого кликнув по кнопке (с ползунком, можно выбрать несколько карт) и нажать "Создать отчёт". На странице появится сгенерированный отчёт по заявкам. Кликнув по "Excel отчёты" и "Word отчёты", из данных кнопок появится выпадающий список, который предоставит возможность отправить данные отчёты на почту клиенту для дальнейшего изучения.

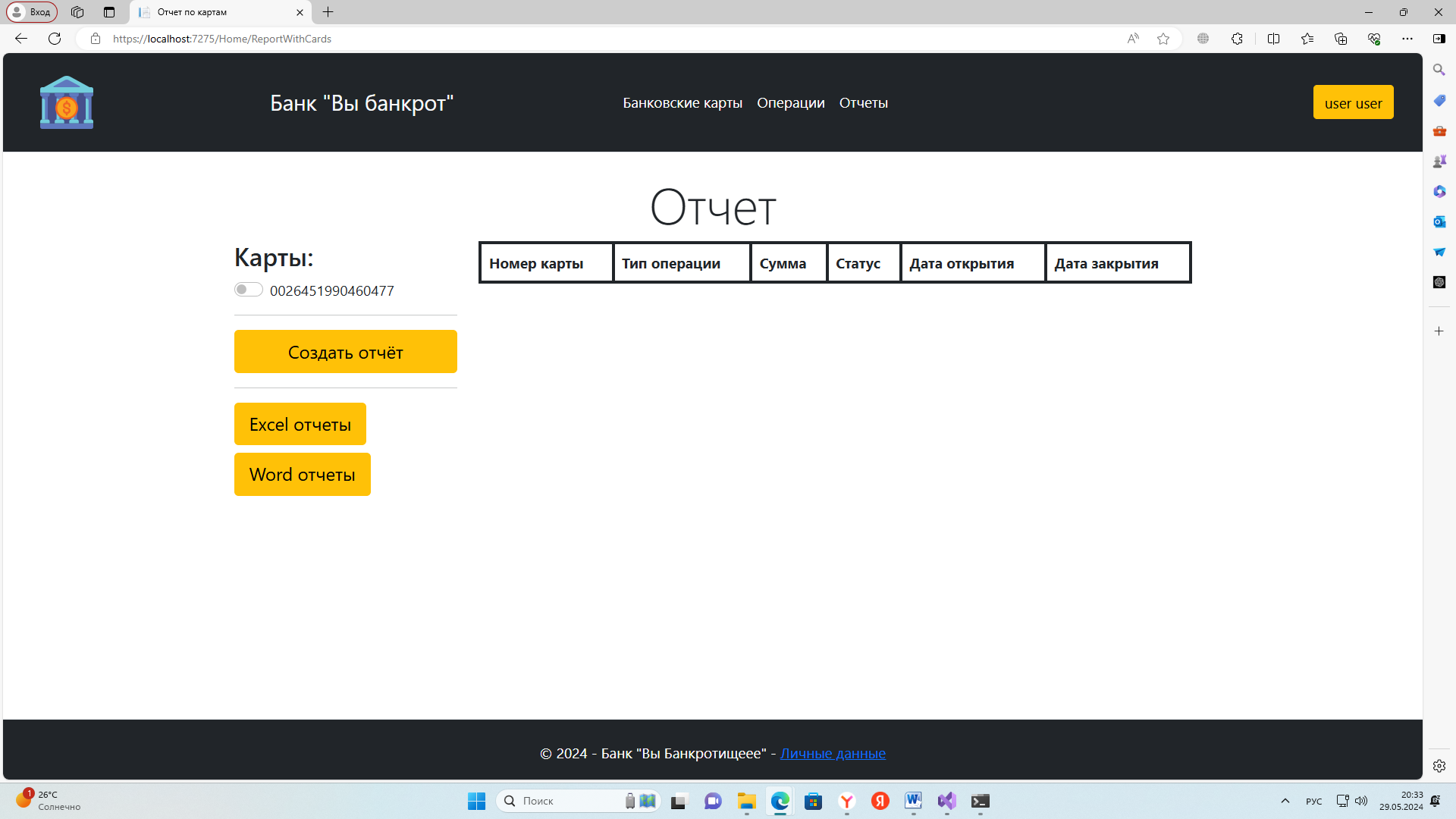


Рисунок 3.14. Отчёт по банковским картам

Отчёт за период

Далее пользователь может получить отчёт за период по переводам, снятиям и пополнениям. Перейдя на вкладку с отчётом за период. Наведя курсор мыши на "Отчёты", появится выпадающий список. Выбрав "Отчёты за период" (Рисунок 3.13), клиент попадёт на вкладку с отчётами за период (Рисунок 3.15). Пользователю нужно указать дату начала периода и дату конца периода (выбрав в календаре сроки) и кликнуть на кнопку "Сформировать отчёт" и отчёт Pdf сразу же отправится на почту.

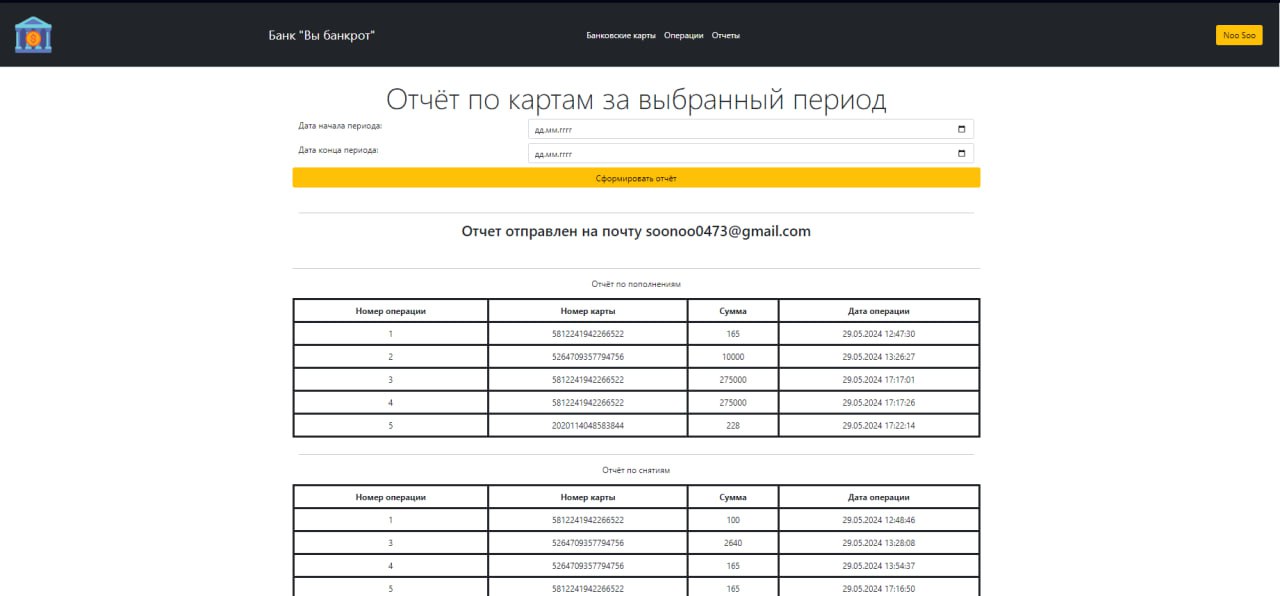


Рисунок 3.15. Отчёт по картам за выбранный период

Диаграмма

Далее пользователь может получить диаграмму за месяц по банковским картам. Наведя курсор мыши на "Отчёты", появится выпадающий список. Выбрав "Диаграмма" (Рисунок 3.13), клиент попадёт на вкладку с диаграммой по выбранной банковской карте (Рисунок 3.16). Далее перейдя на вкладку "Диаграмма", пользователю необходимо выбрать нужную банковскую карту, для этого из выпадающего списка выбрать карту и нажать на кнопку "Выбрать". На странице появится сгенерированная диаграмма по месяцам.

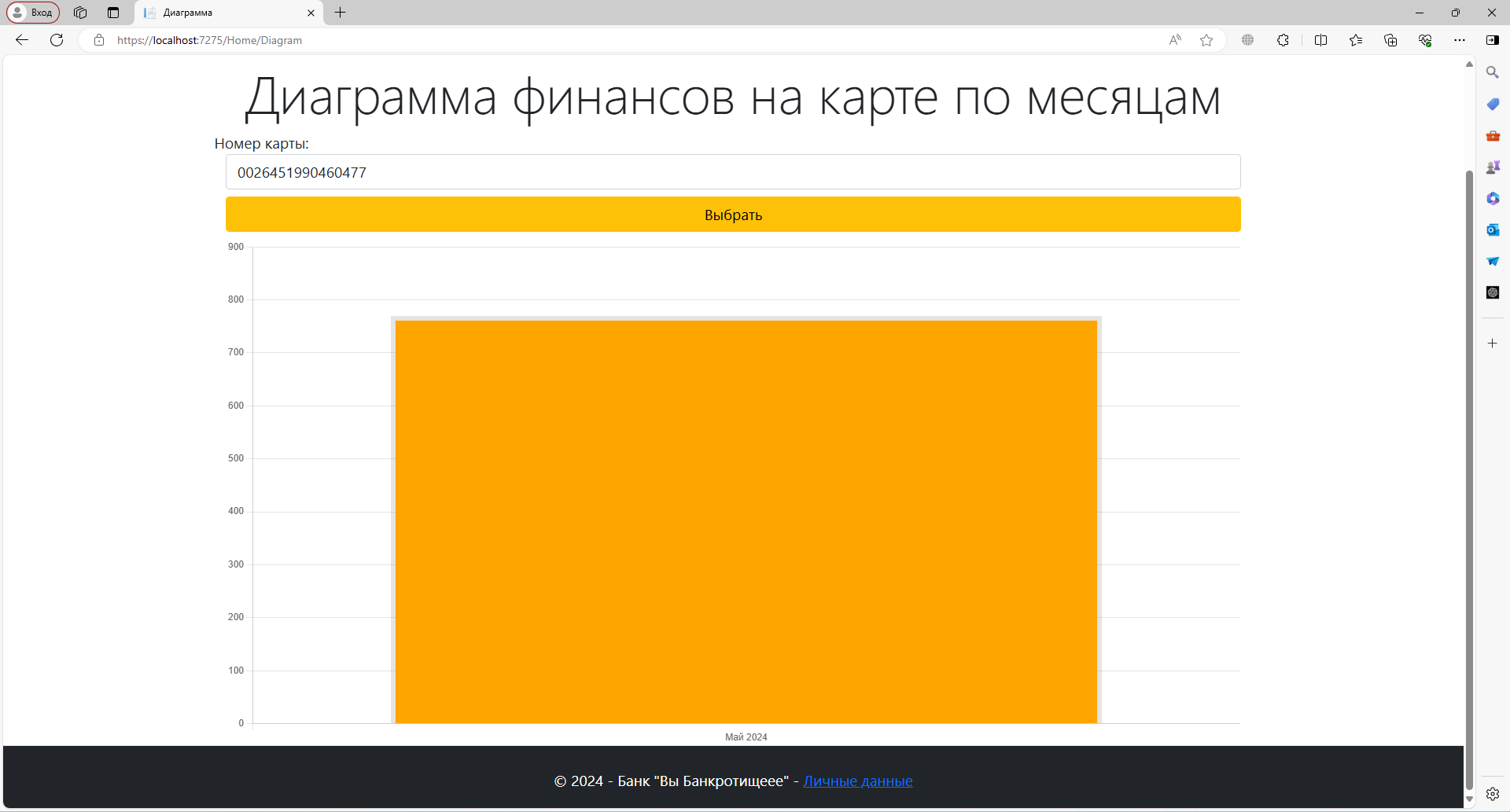


Рисунок 3.16. Диаграмма финансов по месяцам

*Руководство программиста для разработанного проекта*

1. В проекте BankBusinessLogic находится бизнес логика, логика форматирования отчётов формата pdf, docx, xlsx.
   1. В папке BusinessLogics находится бизнес логика:
      1. В папке Cashier:
         1. AccountLogic.cs - логика счетов
         2. MoneyTransferLogic.cs – логика переводов.
         3. CashierLogic.cs - логика кассиров.
         4. CashWithdrawalLogic – выдачи наличных
      2. В папке Client:
         1. CardLogic.cs – логика карт.
         2. DebitingLogic.cs – логика снятия.
         3. CreditingLogic.cs - логика поплнения.
         4. ClientLogic.cs - логика клиентов.
      3. В папке Reports:
         1. ReportLogicCashier.cs - логика отчёта для кассира.
         2. ReportLogicClient.cs - логика отчёта для клиентов.
   2. В папке MailWorker находится основная логика отправки отчетов на почту:
      1. AbstractMailWorker – абстрактный класс работы с почтой
      2. MailKitWorker – класс работы с почтой
   3. В папке OfficePackage находится основная логика формирования файлов с обработкой нужных данных:
      1. AbstractSaveToExcel.cs – логика абстрактного класса для файла Excel.
      2. AbstractSaveToWord.cs – логика абстрактного класса для файла Word.
      3. AbstractSaveToPdf.cs – логика абстрактного класса для файла Pdf.
      4. В папке HelperEnums находится логика классов перечислений для форматирования отчетов формата pdf, docx, xlsx:
         1. ExcelStyleInfoType – для отчёта xls.
         2. PdfParagraphAlignmentType – для отчёта docx.
         3. WordJustificationType – для отчёта pdf.
      5. В папке HelperModels находится логика форматирования отчетов формата pdf, docx, xls:
         1. ExcelCellParameters.cs – клеток в формате xlsx.
         2. ExcelInfo.cs – формирование имени файла.
         3. ExcelMergeParameters.cs – слияния в формате xlsx.
         4. WordInfo.cs – передачи данных в формате docx.
         5. PdfInfo.cs – передачи данных в формате pdf.
         6. PdfRowParameters.cs – строк в формате pdf.
         7. PdfParagraph.cs – абзацов pdf.
         8. WordParagraph.cs – абзацов docx.
         9. WordTextProperties.cs – выравнивания в формате docx.
      6. В папке Implements реализация абстрактных классов сохранения отчётов
         1. SaveToExcel.cs – сохранение в формате xlsx.
         2. SaveToPdf.cs – сохранение в формате pdf.
         3. SaveToWord.cs – сохранение в формате docx
2. В проекте BankContracts находятся модели привязки, бизнес логика, классы, для передачи данных от интерфейса, модель просмотра и модель поиска.
   1. В папке BindingModels находятся модели привязки:
      1. В папке Cashier:
         1. AccountBindingModel.cs – модель привязки для счета.
         2. CashierBindingModel.cs – модель привязки для кассира.
         3. MoneyTransferBindingModel.cs – модель привязки для переводов.
         4. CashWithdrawalBindingModel.cs – модель привязки для выдачи.
      2. В папке Client:
         1. DebitingBindingModel.cs – модель привязки для снятия.
         2. CardBindingModel.cs – модель привязки для карт.
         3. CreditingBindingModel.cs – модель привязки для пополнения.
         4. ClientBindingModel.cs – модель привязки для клиентов.
      3. В папке Messages:
         1. MailConfigBindingModel.cs – модель привязки для конфигурации писем.
         2. MailSendInfoBindingModel – модель привязки для содержания писем.
         3. MessageInfoBindingModel – модель привязки для содержания писем.
      4. В папке Reports:
         1. ReportBindingModel.cs – модель привязки для отчётов.
         2. ReportSupportBindingModel.cs – вспомогательная модель привязки для отчётов.
   2. В папке BusinessLogicsContracts находятся интерфейсы для бизнес-логики:
      1. В папке Cashier:
         1. IAccountLogic.cs – интерфейс счета
         2. ICashierLogic.cs – интерфейс кассира.
         3. IMoneyTransferLogic – интерфейс певода.
         4. ICashWithdrawalLogic.cs – интерфейс выдачи.
      2. В папке Client:
         1. ICardLogic.cs – интерфейс карты.
         2. IDebitingLogic.cs – интерфейс снятия.
         3. ICreditingLogic.cs – интерфейс пополнения.
         4. IClientLogic – интерфейс клиента.
      3. В папке Reports:
         1. IReportCashierLogic.cs – интерфейс отчётов кассира.
         2. IReportClientLogic – интерфейс клиента.
   3. В папке StoragesContracts находятся интерфейсы по хранилищу:
      1. В папке Cashier:
         1. IAccountStorage.cs – интерфейс хранилища счета.
         2. ICashierStorage.cs – интерфейс хранилища кассира.
         3. ICashWithdrawal.cs – интерфейс хранилища выдачи.
         4. IMoneyTransferStorage.cs – интерфейс хранилища переводов.
      2. В папке Clients:
         1. ICardStorage.cs – интерфейс хранилища карт.
         2. IDebitingStorage.cs – интерфейс хранилища снятия.
         3. ICreditingStorage.cs – интерфейс хранилища пополнения.
         4. IClientStorage.cs – интерфейс хранилища клиентов.
   4. В папке ViewModels находится модель просмотра:
      1. В папке Cashier:
         1. В папке Diagram:
            1. CashierDiagramElementsViewModel- отображение элементов диаграммы
            2. CashierDiagramViewModel – отображение диаграммы
         2. В папке View Models:
            1. AccountViewModel.cs – модель просмотра счета.
            2. CashierViewModel.cs – модель просмотра кассира.
            3. CashWithdrawalViewModel.cs – модель просмотра выдачи.
            4. MoneyTransferViewModel.cs – модель просмотра переводов.
      2. В папке Client:
         1. В папке Diagram:
            1. ClientDiagramElementsViewModel- отображение элементов диаграммы
            2. ClientDiagramViewModel – отображение диаграммы
         2. В папке View Models:
            1. ClientViewModel.cs – модель просмотра клиентов.
            2. DebitingViewModel.cs – модель просмотра снятия.
            3. CreditingViewModel.cs – модель просмотра пополнения.
            4. CardViewModel.cs – модель просмотра карт.
      3. В папке Reports:
         1. ReportCashierAccountsViewModel- для просмотра отчета по счетам
         2. ReportCashierViewModel -модель для отчета по кассиру
         3. CheckboxViewModel – модель для проверки
         4. ReportClientCardsViewModel -модель для просмотра отчета карт по клиентам
         5. ReportClientViewModel – отчет по клиентам
         6. ReportViewModel – модель отчета
         7. FileViewModel – модель для записи
         8. ReportCashierViewModelForHTML – модель для отображения у кассира
         9. ReportClientViewModelForHTML – модель для отображения у клиента
   5. В папке SearchModels находится модель поиска:
      1. В папке Cashier:
         1. AccountSearchModel – модель поиска по счетам.
         2. MoneyTransferModel – модель поиска по переводам.
         3. CashWithdrawalModel – модель поиска по выдаче наличных.
         4. CashierSearchModel – модель поиска кассира.
      2. В папке Client:
         1. CardSearchModel – модель поиска карт.
         2. DebitingSearchModel – модель поиска снятия.
         3. CreditingSearchModel – модель поиска пополнения.
         4. ClientSearchModel – модель поиска клиента.
3. В проекте BankDataBaseImplement находится база данных и все её прилегающие:
   1. BankDataBase.cs – подключение и создание базы данных отеля.
   2. В папке Implements находятся склады моделей:
      1. В папке CashierImplements:
         1. MoneyTransferStorage.cs – склад модели по переводам.
         2. AccountStorage.cs – склад модели по счетам.
         3. CashierStorage.cs – склад модели кассирам.
         4. CashWithdrawalStorage.cs – склад снятию.
      2. В папке ClientImplements
         1. CardStorage.cs – склад модели карт.
         2. DebitingStorage.cs – склад модели снятия.
         3. CreditingStorage.cs – склад модели пополнения.
         4. ClientStorage.cs – склад модели клиентов.
   3. В папке Migrations находятся параметры, столбцы и таблицы базы данных.
   4. В папке Models находятся модели и их связи:
      1. В папке CashierModels:
         1. Account.cs – модели и связи по счетам
         2. Cashier.cs – модели и связи по кассирам
         3. MoneyTransfer.cs – модели и связи по переводам
         4. CashWithdrawal.cs – модели и связи по снятию
      2. В папке ClientModels:
         1. Card.cs – модели и связи картам.
         2. Debiting.cs – модели и связи снятия.
         3. Crediting.cs – модели и связи пополнения.
         4. CreditingClient.cs – модели и связи клиентов.
4. В проекте BankDataModels находятся интерфейсы сущностей с описанием их полей.
5. В проекте BankRestApi находятся контроллеры для работы с сущностями.
6. В проекте BankClientApp находится контроллер, использующий контроллеры из BankRestApi и с помощью этого контроллера данные передаются с форм и на формы
7. В проекте BankСashierApp находится контроллер, использующий контроллеры из BankRestApi и с помощью этого контроллера данные передаются с форм и на формы

# **Заключение**

В ходе выполнения курсовой работы была реализована программа, основанная на концепции DAL с 4 уровнями (слоями) доступа и работы: контракты, бизнес-логика, интерфейс пользователя и хранение данных. Веб-приложение, разработанный на ASP.NET использует базу данных (SQLEXPRESS) для хранения и работы со сведениями о клиентах банка, их банковских картах и операциях по ним. В этой программе демонстрируется информация о клиентах, банковских картах и операциях, в которых они задействованы. Программа позволяет кассиру в любой момент узнать, какие заявки оставил клиент и совершить с ними манипуляции: одобрить/отклонить. Также клиент может получать отчёты по совершённым операциям, по банковским картам, которые придут к нему на электронную почту. Есть блок, отвечающий за реализацию интерфейсов и два блока, отвечающих за внешний вид форм со стороны Клиента и Кассира. Так же два блока, в которых находятся сущности для Клиента и Кассира и блок сервера с базой данных.

В итоге при работе с курсовым проектом была реализована и изучена работа с базами данных через Visual Studio и работа с отчётами, кроме того, для реализации интерфейса использовался Web-клиент, разработанный на ASP.NET.

**Список литературы**

1. Программирование на языке C#, платформа .NET Framework / ProfessorWeb. .Net & Web Programming [Электронный ресурс] / Режим доступа (URL): https://professorweb.ru/ . – Загл. с экрана (Дата обращения: 27.03.2024).
2. Справочник по SQL Server Express LocalDB [Электронный ресурс] (Дата последней публикации: 23.05.2023) / Режим доступа (URL): https://learn.microsoft.com/ru-ru/sql/relational-databases/sql-server-express-localdb-reference?view=sql-server-ver16 . – Загл. с экрана (Дата обращения: 27.04.2024).
3. Data Access Layer как инструмент управления хранением данных [Электронный ресурс] (Дата последней публикации: 31.08.2015) / Режим доступа (URL): https://habr.com/ru/companies/custis/articles/265785/ . – Загл. с экрана (Дата обращения: 10.03.2024).
4. Microsoft Learn. Официальный сайт Microsoft [Электронный ресурс] / Режим доступа (URL): https://learn.microsoft.com/. – Загл. с экрана (Дата обращения: 29.03.2024).
5. StackOverflow. Форум для программистов [Электронный ресурс] / Режим доступа (URL): https://ru.stackoverflow.com/. – Загл. с экрана (Дата обращения: 14.05.2024).
6. Бабич А. В. Введение в UML / А. В. Бабич. – Москва.: Издательство НОУ "Интуит", 2016. – 209с. / [Электронный ресурс] Режим доступа (URL): https://www.intuit.ru/goods\_store/ebooks/8359 (Дата обращения: 26.02.2023).
7. Как использовать REST API: полное руководство для начинающих / Денис Суворов [Электронный ресурс] (Дата последней публикации: 31.07.2022) // Режим доступа (URL): https://proglib.io/p/kak-ispolzovat-rest-api-polnoe-rukovodstvo-dlya-nachinayushchih-2022-07-31. – Загл. с экрана (Дата обращения: 08.05.2024).
8. Informicus. Диаграммы вариантов использования [Электронный ресурс] / Режим доступа (URL): https://www.informicus.ru/mps04.html. – Загл. с экрана (Дата обращения: 21.02.2024).
9. Теория и практика UML. Диаграмма последовательности [Электронный ресурс] / Режим доступа (URL): http://it-gost.ru/articles/view\_articles/94 . – Загл. с экрана (Дата обращения: 08.02.2024).
10. Pro Git. 2nd Edition [Электронный ресурс] / Режим доступа: https://git-scm.com/book/ru/v2 (Дата обращения: 20.05.2024).
11. Flaticon. Vector Icons and Stickers – PNG, SVG, EPS, PSD and CSS [Электронный ресурс] / Режим доступа (URL): https://www.flaticon.com/?ref=iamxk (Дата обращения: 23.05.2024).
12. Bootstrap. Bootstrap The most popular HTML, CSS and JS library in the world [Электронный ресурс] / Режим доступа (URL) : https://getbootstrap.com/ (Дата обращения: 22.05.2024)
13. Nicepage. Все о работе с CSS и HTML [Электронный ресурс] / Режим доступа (URL) : https://nicepage.com/
14. Хабр Q&A — вопросы и ответы для IT-специалистов [Электронный ресурс] / Режим доступа (URL) : https://qna.habr.com/
15. METANIT.COM. Сайт о программировании [Электронный ресурс] / Режим доступа (URL) : https://metanit.com/sharp/ . – Загл. с экрана (Дата обращения: 15.05.2024).

# **Приложение**

Листинг кода

Проект BankBusinessLogic:

Папка BusinessLogics:

Папка Cashier

[AccountLogic.cs](https://git.is.ulstu.ru/Belnik/CourseWork_Bank/src/branch/main/Bank/BankBusinessLogic/BusinessLogic/Cashier/AccountLogic.cs)

using BankContracts.BindingModels.Cashier;

using BankContracts.BusinessLogicsContracts.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.StoragesContracts.Cashier;

using BankContracts.ViewModels.Cashier.Diagram;

using BankContracts.ViewModels.Cashier.ViewModels;

using BankDataModels.Enums;

using Microsoft.Extensions.Logging;

namespace BankBusinessLogic.BusinessLogics.Cashier

{

public class AccountLogic : IAccountLogic

{

private readonly ILogger \_logger;

private readonly IAccountStorage \_accountStorage;

private readonly ICashWithdrawalLogic \_cashWithdrawalLogic;

private readonly IMoneyTransferLogic \_moneyTransferLogic;

public AccountLogic(ILogger<AccountLogic> logger, IAccountStorage accountStorage,

ICashWithdrawalLogic cashWithdrawalLogic, IMoneyTransferLogic moneyTransferLogic)

{

\_logger = logger;

\_accountStorage = accountStorage;

\_cashWithdrawalLogic = cashWithdrawalLogic;

\_moneyTransferLogic = moneyTransferLogic;

}

public AccountViewModel? ReadElement(AccountSearchModel model)

{

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

\_logger.LogInformation("ReadElement. AccountNumber:{Name}. Id:{Id}", model.AccountNumber, model?.Id);

var element = \_accountStorage.GetElement(model);

if (element == null)

{

\_logger.LogWarning("ReadElement element not found");

return null;

}

\_logger.LogInformation("ReadElement find. Id:{Id}", element.Id);

return element;

}

public List<AccountViewModel>? ReadList(AccountSearchModel? model)

{

//\_logger.LogInformation("ReadList. AccountNumber:{AccountNumber}. Id:{Id}", model.AccountNumber, model?.Id);

//list хранит весь список в случае, если model пришло со значением null на вход метода

var list = model == null ? \_accountStorage.GetFullList() : \_accountStorage.GetFilteredList(model);

if (list == null)

{

\_logger.LogWarning("ReadList return null list");

return null;

}

\_logger.LogInformation("ReadList. Count:{Count}", list.Count);

return list;

}

//метод, отвечающий за изменение баланса счёта

public bool ChangeBalance(AccountSearchModel? model, int sum)

{

//ищем счёт

var account = ReadElement(model);

if (account == null)

{

throw new ArgumentNullException("Счёт не найден", nameof(account));

}

//проверяем возможность операции снятия (sum может быть отрицательной)

if (sum + account.Balance < 0)

{

return false;

}

//обновляем балланс счёта

\_accountStorage.Update(new AccountBindingModel

{

Id = account.Id,

Balance = account.Balance + sum

});

return true;

}

public bool Create(AccountBindingModel model)

{

CheckModel(model);

if (\_accountStorage.Insert(model) == null)

{

\_logger.LogWarning("Insert operation failed");

return false;

}

return true;

}

public bool Update(AccountBindingModel model)

{

CheckModel(model);

if (\_accountStorage.Update(model) == null)

{

\_logger.LogWarning("Update operation failed");

return false;

}

return true;

}

public bool Delete(AccountBindingModel model)

{

CheckModel(model, false);

\_logger.LogInformation("Delete. Id:{Id}", model.Id);

if (\_accountStorage.Delete(model) == null)

{

\_logger.LogWarning("Delete operation failed");

return false;

}

return true;

}

public List<CashierDiagramElementsViewModel> GetMonthInfo(int AccountId)

{

Dictionary<(int, int), int> cashWithdrawals = \_cashWithdrawalLogic.ReadList(new CashWithdrawalSearchModel()

{

AccountId = AccountId,

}).Where(x => x.DebitingStatus == StatusEnum.Закрыта ).GroupBy(x => new { x.DateOperation.Month, x.DateOperation.Year })

.Select(x => new { x.Key.Month, x.Key.Year, Sum = x.Select(y => y.Sum).Sum() }).ToDictionary(x => (x.Month, x.Year), x => (x.Sum));

Dictionary<(int, int), int> moneyTransfers = \_moneyTransferLogic.ReadList(new MoneyTransferSearchModel()

{

AccountPayeeId = AccountId,

}).GroupBy(x => new { x.DateOperation.Month, x.DateOperation.Year })

.Select(x => new { x.Key.Month, x.Key.Year, Sum = x.Select(y => y.Sum).Sum() }).ToDictionary(x => (x.Month, x.Year), x => (x.Sum));

Dictionary<(int, int), int> moneyTransfersDebiting = \_moneyTransferLogic.ReadList(new MoneyTransferSearchModel()

{

AccountSenderId = AccountId,

}).GroupBy(x => new { x.DateOperation.Month, x.DateOperation.Year })

.Select(x => new { x.Key.Month, x.Key.Year, Sum = x.Select(y => y.Sum).Sum() }).ToDictionary(x => (x.Month, x.Year), x => (x.Sum));

List<CashierDiagramElementsViewModel> result = new();

int sum = 0;

foreach (var key in cashWithdrawals.Keys.Union(moneyTransfers.Keys).Union(moneyTransfers.Keys).OrderBy(x => x.Item1 \* 12 + x.Item2))

{

if (cashWithdrawals.ContainsKey(key)) sum += cashWithdrawals.GetValueOrDefault(key);

if (moneyTransfers.ContainsKey(key)) sum += moneyTransfers.GetValueOrDefault(key);

if (moneyTransfers.ContainsKey(key)) sum -= moneyTransfersDebiting.GetValueOrDefault(key);

result.Add(new CashierDiagramElementsViewModel() { Name = Enum.GetName(typeof(Months), key.Item1) + " " + key.Item2.ToString(), Value = sum });

}

return result;

}

//проверка входного аргумента для методов Insert, Update и Delete

private void CheckModel(AccountBindingModel model, bool withParams = true)

{

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

//так как при удалении передаём как параметр false

if (!withParams)

{

return;

}

//проверка на наличие номера счёта

if (string.IsNullOrEmpty(model.AccountNumber))

{

throw new ArgumentNullException("Отсутствие номера у счёта", nameof(model.AccountNumber));

}

//проверка на наличие id владельца

if (model.CashierId < 0)

{

throw new ArgumentNullException("Некорректный Id владельца счёта", nameof(model.CashierId));

}

//проверка на наличие id кассира, создавшего счёт

if (model.CashierId < 0)

{

throw new ArgumentNullException("Некорректный Id кассира, открывшего счёт", nameof(model.CashierId));

}

//проверка на наличие пароля счёта

if (string.IsNullOrEmpty(model.PasswordAccount) )

{

throw new ArgumentNullException("Некорректный пароль счёта", nameof(model.PasswordAccount));

}

if (model.Balance < 0) {

throw new ArgumentNullException("Изначальный баланс аккаунта не может быть < 0", nameof(model.Balance));

}

//проверка на корректную дату открытия счёта

if (model.DateOpen > DateTime.Now)

{

throw new ArgumentNullException("Дата открытия счёта не может быть ранее текущей", nameof(model.DateOpen));

}

\_logger.LogInformation("Account. AccountNumber:{AccountNumber}. PasswordAccount:{PasswordAccount}. ClientId:{ClientId}. " +

"CashierId:{CashierId}. DateOpen:{DateOpen}. Id:{Id}",

model.AccountNumber, model.PasswordAccount, model.ClientId, model.CashierId, model.DateOpen, model.Id);

//для проверка на наличие такого же счёта

var element = \_accountStorage.GetElement(new AccountSearchModel

{

AccountNumber = model.AccountNumber,

});

//если элемент найден и его Id не совпадает с Id переданного объекта

if (element != null && element.Id != model.Id)

{

throw new InvalidOperationException("Счёт с таким номером уже существует");

}

}

}

}

CashWithdrawalLogic.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.BindingModels.Client;

using BankContracts.BusinessLogicsContracts.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.StoragesContracts.Cashier;

using BankContracts.StoragesContracts.Client;

using BankContracts.ViewModels.Cashier.ViewModels;

using BankDataModels.Enums;

using Microsoft.Extensions.Logging;

namespace BankBusinessLogic.BusinessLogics.Cashier

{

public class CashWithdrawalLogic : ICashWithdrawalLogic

{

private readonly ILogger \_logger;

private readonly ICashWithdrawalStorage \_cashWithdrawalStorage;

private readonly IDebitingStorage \_debitingStorage;

public CashWithdrawalLogic(ILogger<CashWithdrawalLogic> logger, ICashWithdrawalStorage cashWithdrawalStorage,

IDebitingStorage debitingStorage)

{

\_logger = logger;

\_cashWithdrawalStorage = cashWithdrawalStorage;

\_debitingStorage = debitingStorage;

}

public CashWithdrawalViewModel? ReadElement(CashWithdrawalSearchModel model)

{

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

\_logger.LogInformation("ReadElement. AccountId:{AccountId}. Sum:{Sum}. DateOperation:{DateOperation}. Id:{Id}",

model.AccountId, model.Sum, model.DateTo, model?.Id);

var element = \_cashWithdrawalStorage.GetElement(model);

if (element == null)

{

\_logger.LogWarning("ReadElement element not found");

return null;

}

\_logger.LogInformation("ReadElement find. Id:{Id}", element.Id);

return element;

}

public List<CashWithdrawalViewModel>? ReadList(CashWithdrawalSearchModel? model)

{

\_logger.LogInformation("ReadElement. AccountId:{AccountId}. Sum:{Sum}. DateOperation:{DateOperation}. Id:{Id}",

model?.AccountId, model?.Sum, model?.DateTo, model?.Id);

//list хранит весь список в случае, если model пришло со значением null на вход метода

var list = model == null ? \_cashWithdrawalStorage.GetFullList() : \_cashWithdrawalStorage.GetFilteredList(model);

if (list == null)

{

\_logger.LogWarning("ReadList return null list");

return null;

}

\_logger.LogInformation("ReadList. Count:{Count}", list.Count);

return list;

}

public bool Create(CashWithdrawalBindingModel model, bool flag)

{

CheckModel(model);

if (flag)

{

if (\_cashWithdrawalStorage.Insert(model) == null)

{

\_logger.LogWarning("Insert operation failed");

return false;

}

\_debitingStorage.Update(new DebitingBindingModel

{

Id = model.DebitingId,

DateClose = DateTime.Now,

Status = StatusEnum.Закрыта

});

}

else

{

\_debitingStorage.Update(new DebitingBindingModel

{

Id = model.DebitingId,

DateClose = DateTime.Now,

Status = StatusEnum.Отклонено

});

}

return true;

}

public bool Update(CashWithdrawalBindingModel model)

{

CheckModel(model);

if (\_cashWithdrawalStorage.Update(model) == null)

{

\_logger.LogWarning("Update operation failed");

return false;

}

return true;

}

public bool Delete(CashWithdrawalBindingModel model)

{

CheckModel(model, false);

\_logger.LogInformation("Delete. Id:{Id}", model.Id);

if (\_cashWithdrawalStorage.Delete(model) == null)

{

\_logger.LogWarning("Delete operation failed");

return false;

}

return true;

}

//проверка входного аргумента для методов Insert, Update и Delete

private void CheckModel(CashWithdrawalBindingModel model, bool withParams = true)

{

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

//так как при удалении передаём как параметр false

if (!withParams)

{

return;

}

//проверка на корректность Id счёта

if (model.AccountId < 0)

{

throw new ArgumentNullException("Некорректный Id счёта", nameof(model.AccountId));

}

//проверка на корректность снимаемой суммы

if (model.Sum <= 0)

{

throw new ArgumentNullException("Снимаемая сумма не может раняться нулю или быть меньше его", nameof(model.Sum));

}

//проверка на корректную дату операции

if (model.DateOperation > DateTime.Now)

{

throw new ArgumentNullException("Дата операции не может быть позднее текущей", nameof(model.DateOperation));

}

\_logger.LogInformation("CashWithdrawal: AccountId:{AccountId}. Sum:{Sum}. DateOperation:{DateOperation}. Id:{Id}",

model.AccountId, model.Sum, model.DateOperation, model?.Id);

}

}

}

CashierLogic.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.BusinessLogicsContracts.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.StoragesContracts.Cashier;

using BankContracts.ViewModels.Cashier.ViewModels;

using Microsoft.Extensions.Logging;

namespace BankBusinessLogic.BusinessLogics.Cashier

{

public class CashierLogic : ICashierLogic

{

private readonly ILogger \_logger;

private readonly ICashierStorage \_cashierStorage;

public CashierLogic(ILogger<CashierLogic> logger, ICashierStorage cashierStorage)

{

\_logger = logger;

\_cashierStorage = cashierStorage;

}

public CashierViewModel? ReadElement(CashierSearchModel model)

{

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

\_logger.LogInformation("ReadElement. CashierName:{Name}. CashierSurname:{Surname}. CashierPatronymic:{Patronymic}. Id:{Id}",

model.Name, model.Surname, model.Patronymic, model?.Id);

var element = \_cashierStorage.GetElement(model);

if (element == null)

{

\_logger.LogWarning("ReadElement element not found");

return null;

}

\_logger.LogInformation("ReadElement find. Id:{Id}", element.Id);

return element;

}

public List<CashierViewModel>? ReadList(CashierSearchModel? model)

{

\_logger.LogInformation("ReadList. CashierName:{Name}. CashierSurname:{Surname}. CashierPatronymic:{Patronymic}. Id:{Id}",

model.Name, model.Surname, model.Patronymic, model?.Id);

//list хранит весь список в случае, если model пришло со значением null на вход метода

var list = model == null ? \_cashierStorage.GetFullList() : \_cashierStorage.GetFilteredList(model);

if (list == null)

{

\_logger.LogWarning("ReadList return null list");

return null;

}

\_logger.LogInformation("ReadList. Count:{Count}", list.Count);

return list;

}

public bool Create(CashierBindingModel model)

{

CheckModel(model);

if (\_cashierStorage.Insert(model) == null)

{

\_logger.LogWarning("Insert operation failed");

return false;

}

return true;

}

public bool Update(CashierBindingModel model)

{

CheckModel(model);

if (\_cashierStorage.Update(model) == null)

{

\_logger.LogWarning("Update operation failed");

return false;

}

return true;

}

public bool Delete(CashierBindingModel model)

{

CheckModel(model, false);

\_logger.LogInformation("Delete. Id:{Id}", model.Id);

if (\_cashierStorage.Delete(model) == null)

{

\_logger.LogWarning("Delete operation failed");

return false;

}

return true;

}

//проверка входного аргумента для методов Insert, Update и Delete

private void CheckModel(CashierBindingModel model, bool withParams = true)

{

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

//так как при удалении передаём как параметр false

if (!withParams)

{

return;

}

//проверка на наличие имени

if (string.IsNullOrEmpty(model.Name))

{

throw new ArgumentNullException("Отсутствие имени в учётной записи", nameof(model.Name));

}

//проверка на наличие фамилия

if (string.IsNullOrEmpty(model.Surname))

{

throw new ArgumentNullException("Отсутствие фамилии в учётной записи", nameof(model.Name));

}

//проверка на наличие отчество

if (string.IsNullOrEmpty(model.Patronymic))

{

throw new ArgumentNullException("Отсутствие отчества в учётной записи", nameof(model.Name));

}

//проверка на наличие почты

if (string.IsNullOrEmpty(model.Email))

{

throw new ArgumentNullException("Отсутствие почты в учётной записи (логина)", nameof(model.Email));

}

//проверка на наличие пароля

if (string.IsNullOrEmpty(model.Password))

{

throw new ArgumentNullException("Отсутствие пароля в учётной записи", nameof(model.Password));

}

\_logger.LogInformation("Cashier. CashierName:{Name}. CashierSurname:{Surname}. CashierPatronymic:{Patronymic}. " +

"Email:{Email}. Password:{Password}. Id:{Id}",

model.Name, model.Surname, model.Patronymic, model.Email, model.Password, model.Id);

//для проверка на наличие такого же аккаунта

var element = \_cashierStorage.GetElement(new CashierSearchModel

{

Email = model.Email,

});

//если элемент найден и его Id не совпадает с Id переданного объекта

if (element != null && element.Id != model.Id)

{

throw new InvalidOperationException("Аккаунт с таким логином уже есть");

}

}

}

}

MoneyTransferLogic.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.BindingModels.Client;

using BankContracts.BusinessLogicsContracts.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.StoragesContracts.Cashier;

using BankContracts.StoragesContracts.Client;

using BankContracts.ViewModels.Cashier.ViewModels;

using BankDataModels.Enums;

using Microsoft.Extensions.Logging;

namespace BankBusinessLogic.BusinessLogics.Cashier

{

public class MoneyTransferLogic : IMoneyTransferLogic

{

private readonly ILogger \_logger;

private readonly IMoneyTransferStorage \_moneyTransferStorage;

public readonly ICreditingStorage \_creditingStorage;

public MoneyTransferLogic(ILogger<MoneyTransferLogic> logger, IMoneyTransferStorage moneyTransferStorage, ICreditingStorage creditingStorage)

{

\_logger = logger;

\_moneyTransferStorage = moneyTransferStorage;

\_creditingStorage = creditingStorage;

}

public MoneyTransferViewModel? ReadElement(MoneyTransferSearchModel model)

{

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

\_logger.LogInformation("ReadElement. AccountSenderId:{AccountSenderId}. AccountPayeeId:{AccountPayeeId}. Sum:{Sum}. Id:{Id}",

model.AccountSenderId, model.AccountPayeeId, model.Sum, model?.Id);

var element = \_moneyTransferStorage.GetElement(model);

if (element == null)

{

\_logger.LogWarning("ReadElement element not found");

return null;

}

\_logger.LogInformation("ReadElement find. Id:{Id}", element.Id);

return element;

}

public List<MoneyTransferViewModel>? ReadList(MoneyTransferSearchModel? model)

{

\_logger.LogInformation("ReadElement. AccountSenderId:{AccountSenderId}. AccountPayeeId:{AccountPayeeId}. Sum:{Sum}. Id:{Id}",

model?.AccountSenderId, model?.AccountPayeeId, model?.Sum, model?.Id);

//list хранит весь список в случае, если model пришло со значением null на вход метода

var list = model == null ? \_moneyTransferStorage.GetFullList() : \_moneyTransferStorage.GetFilteredList(model);

if (list == null)

{

\_logger.LogWarning("ReadList return null list");

return null;

}

\_logger.LogInformation("ReadList. Count:{Count}", list.Count);

return list;

}

public bool Create(MoneyTransferBindingModel model)

{

CheckModel(model);

if (\_moneyTransferStorage.Insert(model) == null)

{

\_logger.LogWarning("Insert operation failed");

return false;

}

//проверка на то, что это зачисление на карту, а не перевод между счетами

if (model.CreditingId.HasValue)

{

\_creditingStorage.Update(new CreditingBindingModel

{

Id = model.CreditingId.Value,

DateClose = DateTime.Now,

Status = StatusEnum.Закрыта

});

}

return true;

}

public bool Update(MoneyTransferBindingModel model)

{

CheckModel(model);

if (\_moneyTransferStorage.Update(model) == null)

{

\_logger.LogWarning("Update operation failed");

return false;

}

return true;

}

public bool Delete(MoneyTransferBindingModel model)

{

CheckModel(model, false);

\_logger.LogInformation("Delete. Id:{Id}", model.Id);

if (\_moneyTransferStorage.Delete(model) == null)

{

\_logger.LogWarning("Delete operation failed");

return false;

}

return true;

}

//проверка входного аргумента для методов Insert, Update и Delete

private void CheckModel(MoneyTransferBindingModel model, bool withParams = true)

{

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

//так как при удалении передаём как параметр false

if (!withParams)

{

return;

}

//проверка корректности Id счёта отправителя

if (model.AccountSenderId < 0)

{

throw new ArgumentNullException("Отсутствие Id у счёта отправителя", nameof(model.AccountSenderId));

}

//проверка корректности Id счёта получателя

if (model.AccountPayeeId < 0)

{

throw new ArgumentNullException("Отсутствие Id у счёта получателя", nameof(model.AccountPayeeId));

}

//проверка на корректную сумму перевода

if (model.Sum <= 0)

{

throw new ArgumentNullException("Сумма перевода не может раняться нулю или быть меньше его", nameof(model.Sum));

}

//проверка на корректную дату открытия счёта

if (model.DateOperation > DateTime.Now)

{

throw new ArgumentNullException("Дата операции не может быть ранее текущей", nameof(model.DateOperation));

}

\_logger.LogInformation("ReadElement. AccountSenderId:{AccountSenderId}. AccountPayeeId:{AccountPayeeId}. Sum:{Sum}. DateOperation:{DateOperation}. Id:{Id}",

model.AccountSenderId, model.AccountPayeeId, model.Sum, model.DateOperation, model?.Id);

}

}

}

Папка Client

CardLogic.cs

using Microsoft.Extensions.Logging;

using BankContracts.ViewModels.Client.Diagram;

using BankDataModels.Enums;

using BankContracts.BusinessLogicsContracts.Client;

using BankContracts.StoragesContracts.Client;

using BankContracts.BusinessLogicsContracts.Cashier;

using BankContracts.BindingModels.Client;

using BankContracts.SearchModels.Client;

using BankContracts.ViewModels.Client.ViewModels;

using BankContracts.SearchModels.Cashier;

namespace BankBusinessLogic.BusinessLogics.Client

{

public class CardLogic : ICardLogic

{

private readonly ILogger \_logger;

private readonly ICardStorage \_cardStorage;

private readonly IAccountLogic \_accountLogic;

private readonly IDebitingLogic \_debitingLogic;

private readonly ICreditingLogic \_creditingLogic;

public CardLogic(ILogger<CardLogic> logger, ICardStorage cardStorage, IAccountLogic accountLogic,

IDebitingLogic debitingLogic, ICreditingLogic creditingLogic) {

\_logger = logger;

\_cardStorage = cardStorage;

\_accountLogic = accountLogic;

\_debitingLogic = debitingLogic;

\_creditingLogic = creditingLogic;

}

public bool Create(CardBindingModel model)

{

CheckModel(model);

if (\_cardStorage.Insert(model) == null)

{

\_logger.LogWarning("Insert operation failed");

return false;

}

return true;

}

public bool Delete(CardBindingModel model)

{

CheckModel(model, false);

\_logger.LogInformation("Delete. Id:{Id}", model.Id);

if (\_cardStorage.Delete(model) == null)

{

\_logger.LogWarning("Delete operation failed");

return false;

}

return true;

}

public CardViewModel? ReadElement(CardSearchModel model)

{

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

\_logger.LogInformation("ReadElement. CardNumber:{Number}.Id:{ Id}", model.Number, model.Id);

var element = \_cardStorage.GetElement(model);

if (element == null)

{

\_logger.LogWarning("ReadElement element not found");

return null;

}

\_logger.LogInformation("ReadElement find. Id:{Id}", element.Id);

return element;

}

public List<CardViewModel>? ReadList(CardSearchModel? model)

{

\_logger.LogInformation("ReadList. CardId:{Id}", model?.Id);

var list = model == null ? \_cardStorage.GetFullList() : \_cardStorage.GetFilteredList(model);

if (list == null)

{

\_logger.LogWarning("ReadList return null list");

return null;

}

\_logger.LogInformation("ReadList. Count:{Count}", list.Count);

return list;

}

public bool Update(CardBindingModel model)

{

CheckModel(model);

if (\_cardStorage.Update(model) == null)

{

\_logger.LogWarning("Update operation failed");

return false;

}

return true;

}

public List<ClientDiagramElementsViewModel> GetMonthInfo(int CardId) {

Dictionary<(int?, int?), int> debitings = \_debitingLogic.ReadList(new DebitingSearchModel()

{

CardId = CardId,

Status = StatusEnum.Закрыта

}).GroupBy(x => new { x.DateClose?.Month, x.DateClose?.Year })

.Select(x => new { x.Key.Month, x.Key.Year, Sum = x.Select(y => y.Sum).Sum()}).ToDictionary(x => (x.Month, x.Year), x => (x.Sum));

Dictionary<(int?, int?), int> creditings = \_creditingLogic.ReadList(new CreditingSearchModel()

{

CardId = CardId,

Status = StatusEnum.Закрыта

}).GroupBy(x => new { x.DateClose?.Month, x.DateClose?.Year })

.Select(x => new { x.Key.Month, x.Key.Year, Sum = x.Select(y => y.Sum).Sum() }).ToDictionary(x => (x.Month, x.Year), x => (x.Sum));

List<ClientDiagramElementsViewModel> result = new();

foreach (var key in debitings.Keys.Union(creditings.Keys).OrderBy(x => x.Item1 \* 12 + x.Item2)) {

int sum = 0;

if (debitings.ContainsKey(key)) sum -= debitings.GetValueOrDefault(key);

if (creditings.ContainsKey(key)) sum += creditings.GetValueOrDefault(key);

result.Add(new ClientDiagramElementsViewModel() { Name = Enum.GetName(typeof(Months), key.Item1) + " " + key.Item2.ToString(), Value = sum});

}

return result;

}

private void CheckModel(CardBindingModel model, bool withParams = true)

{

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

if (!withParams)

{

return;

}

if (string.IsNullOrEmpty(model.Number) || model.Number.Length != 16)

{

throw new ArgumentNullException("Неправильный номер карты", nameof(model.Number));

}

if (string.IsNullOrEmpty(model.CVC) || model.CVC.Length != 3)

{

throw new ArgumentNullException("Неправильный СVC карты", nameof(model.CVC));

}

if (model.Period < DateTime.Now)

{

throw new ArgumentNullException("Нет периода действия", nameof(model.Period));

}

var cardElement = \_cardStorage.GetElement(new CardSearchModel

{

Number = model.Number,

});

if (cardElement != null && cardElement.Id != model.Id)

{

throw new InvalidOperationException("Карта с таким ноиером уже есть");

}

var accountElement = \_accountLogic.ReadElement(new AccountSearchModel

{

Id = model.AccountId,

ClientId = model.ClientID

});

if (accountElement != null && accountElement.ClientId != model.ClientID)

{

throw new InvalidOperationException("Это не счёт данного клиента");

}

\_logger.LogInformation("Card. Number:{Number}.CVC:{CVC}.ClientId:{ClientID}.Patronymic:{Period}.Id:{Id}",

model.Number, model.CVC, model.Period.ToString(), model.ClientID, model.Id);

}

}

}

ClientLogic.cs

using BankContracts.BindingModels.Client;

using BankContracts.BusinessLogicsContracts.Client;

using BankContracts.SearchModels.Client;

using BankContracts.StoragesContracts.Client;

using BankContracts.ViewModels.Client.ViewModels;

using Microsoft.Extensions.Logging;

namespace BankBusinessLogic.BusinessLogics.Client

{

public class ClientLogic : IClientLogic

{

private readonly ILogger \_logger;

private readonly IClientStorage \_clientStorage;

public ClientLogic(ILogger<ClientLogic> logger, IClientStorage clientStorage) {

\_logger = logger;

\_clientStorage = clientStorage;

}

public bool Create(ClientBindingModel model)

{

CheckModel(model);

if (\_clientStorage.Insert(model) == null)

{

\_logger.LogWarning("Insert operation failed");

return false;

}

return true;

}

public bool Delete(ClientBindingModel model)

{

CheckModel(model, false);

\_logger.LogInformation("Delete. Id:{Id}", model.Id);

if (\_clientStorage.Delete(model) == null)

{

\_logger.LogWarning("Delete operation failed");

return false;

}

return true;

}

public ClientViewModel? ReadElement(ClientSearchModel model)

{

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

\_logger.LogInformation("ReadElement. Name:{Name}.Surname:{Surname}.Patronymic:{Patronymic}.Id:{ Id}", model.Name, model.Surname, model.Patronymic, model.Id);

var element = \_clientStorage.GetElement(model);

if (element == null)

{

\_logger.LogWarning("ReadElement element not found");

return null;

}

\_logger.LogInformation("ReadElement find. Id:{Id}", element.Id);

return element;

}

public List<ClientViewModel>? ReadList(ClientSearchModel? model)

{

\_logger.LogInformation("ReadList. ClientId:{Id}", model?.Id);

var list = model == null ? \_clientStorage.GetFullList() : \_clientStorage.GetFilteredList(model);

if (list == null)

{

\_logger.LogWarning("ReadList return null list");

return null;

}

\_logger.LogInformation("ReadList. Count:{Count}", list.Count);

return list;

}

public bool Update(ClientBindingModel model)

{

CheckModel(model);

if (\_clientStorage.Update(model) == null)

{

\_logger.LogWarning("Update operation failed");

return false;

}

return true;

}

private void CheckModel(ClientBindingModel model, bool withParams = true)

{

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

if (!withParams)

{

return;

}

if (string.IsNullOrEmpty(model.Name))

{

throw new ArgumentNullException("Нет имени пользователя", nameof(model.Name));

}

if (string.IsNullOrEmpty(model.Surname))

{

throw new ArgumentNullException("Нет фамилии пользователя", nameof(model.Surname));

}

if (string.IsNullOrEmpty(model.Patronymic))

{

throw new ArgumentNullException("Нет отчества пользователя", nameof(model.Patronymic));

}

if (string.IsNullOrEmpty(model.Email))

{

throw new ArgumentNullException("Нет почты пользователя", nameof(model.Email));

}

if (string.IsNullOrEmpty(model.Password))

{

throw new ArgumentNullException("Нет пароля пользователя", nameof(model.Password));

}

if (string.IsNullOrEmpty(model.Telephone))

{

throw new ArgumentNullException("Нет телефона пользователя", nameof(model.Telephone));

}

\_logger.LogInformation("Client. Name:{Name}.Surname:{Surname}.Patronymic:{Patronymic}.Email:{Email}.Password:{Password}.Telephone:{Telephone}.Id:{Id}",

model.Name, model.Surname, model.Patronymic, model.Email, model.Password, model.Telephone, model.Id);

var element = \_clientStorage.GetElement(new ClientSearchModel

{

Email = model.Email,

});

if (element != null && element.Id != model.Id)

{

throw new InvalidOperationException("Клиент с такой почтой уже есть");

}

}

}

}

CreditingLogic.cs

using Microsoft.Extensions.Logging;

using BankDataModels.Enums;

using BankContracts.BusinessLogicsContracts.Client;

using BankContracts.StoragesContracts.Client;

using BankContracts.BindingModels.Client;

using BankContracts.ViewModels.Client.ViewModels;

using BankContracts.SearchModels.Client;

namespace BankBusinessLogic.BusinessLogics.Client

{

public class CreditingLogic : ICreditingLogic

{

private readonly ILogger \_logger;

private readonly ICreditingStorage \_creditingStorage;

private readonly ICardStorage \_cardStorage;

public CreditingLogic(ILogger<CreditingLogic> logger, ICreditingStorage creditingStorage, ICardStorage cardStorage) {

\_logger = logger;

\_creditingStorage = creditingStorage;

\_cardStorage = cardStorage;

}

public bool Create(CreditingBindingModel model)

{

CheckModel(model);

if (!CheckCardPeriod(model))

{

model.Status = StatusEnum.Карта\_просрочена;

}

else

{

model.Status = StatusEnum.Открыта;

}

if (\_creditingStorage.Insert(model) == null)

{

\_logger.LogWarning("Insert operation failed");

return false;

}

return true;

}

public bool Delete(CreditingBindingModel model)

{

CheckModel(model, false);

\_logger.LogInformation("Delete. Id:{Id}", model.Id);

if (\_creditingStorage.Delete(model) == null)

{

\_logger.LogWarning("Delete operation failed");

return false;

}

return true;

}

public CreditingViewModel? ReadElement(CreditingSearchModel model)

{

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

\_logger.LogInformation("ReadElement. CreditingId:{ Id }", model.Id);

var element = \_creditingStorage.GetElement(model);

if (element == null)

{

\_logger.LogWarning("ReadElement element not found");

return null;

}

\_logger.LogInformation("ReadElement find. Id:{Id}", element.Id);

return element;

}

public List<CreditingViewModel>? ReadList(CreditingSearchModel? model)

{

\_logger.LogInformation("ReadList. CreditingId:{Id}", model?.Id);

var list = model == null ? \_creditingStorage.GetFullList() : \_creditingStorage.GetFilteredList(model);

if (list == null)

{

\_logger.LogWarning("ReadList return null list");

return null;

}

\_logger.LogInformation("ReadList. Count:{Count}", list.Count);

return list;

}

public bool Update(CreditingBindingModel model)

{

CheckModel(model);

if (\_creditingStorage.Update(model) == null)

{

\_logger.LogWarning("Update operation failed");

return false;

}

return true;

}

private void CheckModel(CreditingBindingModel model, bool withParams = true)

{

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

if (!withParams)

{

return;

}

if (model.Sum <= 0)

{

throw new ArgumentNullException("Сумма операции должна быть больше 0", nameof(model.Sum));

}

if (model.DateOpen > DateTime.Now)

{

throw new ArgumentNullException("Дата не может быть меньше текущего времени", nameof(model.DateOpen));

}

\_logger.LogInformation("Crediting. Sum:{Sum}.CardId:{CardId}.Date:{date}.Id:{Id}",

model.Sum, model.CardId, model.DateOpen.ToString(), model.Id);

}

//проверка карты на просроченность

bool CheckCardPeriod(CreditingBindingModel model)

{

var card = \_cardStorage.GetElement(new CardSearchModel

{

Id = model.CardId

});

//если карта просрочена

if (card.Period < DateTime.Now)

{

return false;

}

return true;

}

}

}

DebitingLogic.cs

using Microsoft.Extensions.Logging;

using BankDataModels.Enums;

using BankContracts.BusinessLogicsContracts.Client;

using BankContracts.StoragesContracts.Client;

using BankContracts.BindingModels.Client;

using BankContracts.SearchModels.Client;

using BankContracts.ViewModels.Client.ViewModels;

namespace BankBusinessLogic.BusinessLogics.Client

{

public class DebitingLogic : IDebitingLogic

{

private readonly ILogger \_logger;

private readonly IDebitingStorage \_debitingStorage;

private readonly ICardStorage \_cardStorage;

public DebitingLogic(ILogger<DebitingLogic> logger, IDebitingStorage debitingStorage, ICardStorage cardStorage) {

\_logger = logger;

\_debitingStorage = debitingStorage;

\_cardStorage = cardStorage;

}

public bool Create(DebitingBindingModel model)

{

CheckModel(model);

if (!CheckCardPeriod(model))

{

model.Status = StatusEnum.Карта\_просрочена;

}

else

{

model.Status = StatusEnum.Открыта;

}

if (\_debitingStorage.Insert(model) == null)

{

\_logger.LogWarning("Insert operation failed");

return false;

}

return true;

}

public bool Delete(DebitingBindingModel model)

{

CheckModel(model, false);

\_logger.LogInformation("Delete. Id:{Id}", model.Id);

if (\_debitingStorage.Delete(model) == null)

{

\_logger.LogWarning("Delete operation failed");

return false;

}

return true;

}

public DebitingViewModel? ReadElement(DebitingSearchModel model)

{

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

\_logger.LogInformation("ReadElement. DebitingId:{ Id }", model.Id);

var element = \_debitingStorage.GetElement(model);

if (element == null)

{

\_logger.LogWarning("ReadElement element not found");

return null;

}

\_logger.LogInformation("ReadElement find. Id:{Id}", element.Id);

return element;

}

public List<DebitingViewModel>? ReadList(DebitingSearchModel? model)

{

\_logger.LogInformation("ReadList. DebitingId:{Id}", model?.Id);

var list = model == null ? \_debitingStorage.GetFullList() : \_debitingStorage.GetFilteredList(model);

if (list == null)

{

\_logger.LogWarning("ReadList return null list");

return null;

}

\_logger.LogInformation("ReadList. Count:{Count}", list.Count);

return list;

}

public bool Update(DebitingBindingModel model)

{

CheckModel(model);

if (\_debitingStorage.Update(model) == null)

{

\_logger.LogWarning("Update operation failed");

return false;

}

return true;

}

private void CheckModel(DebitingBindingModel model, bool withParams = true)

{

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

if (!withParams)

{

return;

}

if (model.Sum <= 0)

{

throw new ArgumentNullException("Сумма операции должна быть больше 0", nameof(model.Sum));

}

if (model.DateClose < model.DateOpen)

{

throw new ArgumentNullException("Дата закрытия не может быть меньше даты открытия", nameof(model.DateClose));

}

\_logger.LogInformation("Debiting. Sum:{Sum}.CardId:{CardId}.DateOpen:{DateOpen}.DateOpen:{DateOpen}.Id:{Id}",

model.Sum, model.CardId, model.DateOpen.ToString(), model.DateClose.ToString(), model.Id);

}

//проверка карты на просроченность

bool CheckCardPeriod(DebitingBindingModel model)

{

var card = \_cardStorage.GetElement(new CardSearchModel

{

Id = model.CardId

});

//если карта просрочена

if (card.Period < DateTime.Now)

{

return false;

}

return true;

}

}

}

Папка Reports

ReportCashierLogic.cs

using BankBusinessLogic.MailWorker;

using BankBusinessLogic.OfficePackage;

using BankBusinessLogic.OfficePackage.HelperModels;

using BankContracts.BindingModels.Reports;

using BankContracts.BusinessLogicsContracts.Reports;

using BankContracts.SearchModels.Cashier;

using BankContracts.SearchModels.Client;

using BankContracts.StoragesContracts.Cashier;

using BankContracts.StoragesContracts.Client;

using BankContracts.ViewModels.Client.ViewModels;

using BankContracts.ViewModels.Reports;

using BankContracts.ViewModels.Reports.Cashier;

using BankDataModels.Enums;

namespace BankBusinessLogic.BusinessLogics.Reports

{

public class ReportCashierLogic : IReportCashierLogic

{

private readonly IMoneyTransferStorage \_moneyTransferStorage;

private readonly ICashWithdrawalStorage \_cashWithdrawalStorage;

private readonly IClientStorage \_clientStorage;

private readonly IDebitingStorage \_debitingStorage;

private readonly ICardStorage \_cardStorage;

private readonly AbstractSaveToExcel \_saveToExcel;

private readonly AbstractSaveToWord \_saveToWord;

private readonly AbstractSaveToPdf \_saveToPdf;

private readonly MailKitWorker \_mailKitWorker;

//инициализируем поля класса через контейнер

public ReportCashierLogic(IMoneyTransferStorage moneyTransferStorage, ICashWithdrawalStorage cashWithdrawalStorage,

IClientStorage clientStorage, AbstractSaveToExcel saveToExcel,

AbstractSaveToWord saveToWord, AbstractSaveToPdf saveToPdf,

IDebitingStorage debitingStorage, ICardStorage cardStorage, MailKitWorker mailKitWorker)

{

\_moneyTransferStorage = moneyTransferStorage;

\_cashWithdrawalStorage = cashWithdrawalStorage;

\_saveToExcel = saveToExcel;

\_saveToWord = saveToWord;

\_saveToPdf = saveToPdf;

\_clientStorage = clientStorage;

\_debitingStorage = debitingStorage;

\_cardStorage = cardStorage;

\_mailKitWorker = mailKitWorker;

}

//формирование списка переводов между счетами за период

public List<ReportCashierViewModel>? GetMoneyTransfers(ReportBindingModel model)

{

return \_moneyTransferStorage.GetFilteredList(new MoneyTransferSearchModel { ClientId = model.ClientId, DateFrom = model.DateFrom, DateTo = model.DateTo})

.Select(x => new ReportCashierViewModel

{

OperationId = x.Id,

DateComplite = x.DateOperation,

AccountPayeeNumber = x.AccountPayeeNumber,

AccountSenderNumber = x.AccountSenderNumber,

SumOperation = x.Sum

})

.ToList();

}

//формирование списка выдаци наличных со счёта за период

public List<ReportCashierViewModel>? GetCashWithrawals(ReportBindingModel model)

{

return \_cashWithdrawalStorage.GetFilteredList(new CashWithdrawalSearchModel { ClientId = model.ClientId, DateFrom = model.DateFrom, DateTo = model.DateTo })

.Select(x => new ReportCashierViewModel

{

OperationId = x.Id,

DebitingId = x.DebitingId,

AccountPayeeNumber = x.AccountNumber,

DateComplite = x.DateOperation,

SumOperation = x.Sum

})

.ToList();

}

//формирование списка выдаци наличных со счёта за период

public List<DebitingViewModel>? GetDebitings(ReportBindingModel model)

{

List<int> CardIdList = new();

var list = \_cardStorage.GetFilteredList(new CardSearchModel

{

AccountId = model.AccountId

});

foreach(var index in list)

{

CardIdList.Add(index.Id);

}

List<DebitingViewModel> totalList = new();

foreach(var index in CardIdList)

{

var result = \_debitingStorage.GetFilteredList(new DebitingSearchModel

{

CardId = index

});

totalList.AddRange(result);

}

return totalList;

}

//формирование полного имени клиента для отчёта

public string GetFullName(ReportBindingModel model)

{

var client = \_clientStorage.GetElement(new ClientSearchModel

{

Id = model.ClientId

});

return client.Surname + " " + client.Name + " " + client.Patronymic;

}

//Сохранение мороженных в файл-Word

public void SaveAccountsToWordFile(ReportBindingModel model)

{

\_saveToWord.CreateDoc(new WordInfo

{

FileName = model.FileName,

Title = "Заявки на снятия со счёта",

Debiting = GetDebitings(model)

}, OfficeOperationEnum.Для\_кассира);

byte[] word = System.IO.File.ReadAllBytes("../BankRestAPI/Отчёт по зявкам на снятие.docx");

File.Delete("../BankRestAPI/Отчёт по зявкам на снятие.docx");

\_mailKitWorker.SendMailAsync(new()

{

MailAddress = model.Email,

Subject = "Отчёт по счетам",

Text = $"Отчёт по состоянию на {DateTime.Now.ToString()}",

File = word,

Role = model.Role,

TypeDoc = TypeDocEnum.WORD

});

}

//Сохранение заготовок с указаеним изделий в файл-Excel

public void SaveAccountsToExcelFile(ReportBindingModel model)

{

\_saveToExcel.CreateReport(new ExcelInfo

{

FileName = model.FileName,

Title = "Заявки на счёт",

Debiting = GetDebitings(model)

}, OfficeOperationEnum.Для\_кассира);

byte[] excel = System.IO.File.ReadAllBytes("../BankRestAPI/Отчёт по зявкам на снятие.xlsx");

File.Delete("../BankRestAPI/Отчёт по зявкам на снятие.xlsx");

\_mailKitWorker.SendMailAsync(new()

{

MailAddress = model.Email,

Subject = "Отчёт по счетам",

Text = $"Отчёт по состоянию на {DateTime.Now.ToString()}",

File = excel,

Role = model.Role,

TypeDoc = TypeDocEnum.EXCEL

});

}

//Сохранение заказов в файл-Pdf

public ReportCashierViewModelForHTML SaveAccountsToPdfFile(ReportBindingModel model)

{

var listMoneyTransfers = GetMoneyTransfers(model);

var listCashWithdrawals = GetCashWithrawals(model);

\_saveToPdf.CreateDoc(new PdfInfo

{

ForClient = false,

FileName = model.FileName,

FullClientName = GetFullName(model),

Title = "Отчёт по операциям списаний со счёта и переводов между счетами",

DateFrom = model.DateFrom!.Value,

DateTo = model.DateTo!.Value,

ReportMoneyTransfer = listMoneyTransfers,

ReportCashWithdrawal = listCashWithdrawals

});

byte[] pdf = System.IO.File.ReadAllBytes("../BankRestAPI/Отчёт\_по\_счетам.pdf");

File.Delete("../BankRestAPI/Отчёт\_по\_счетам.pdf");

\_mailKitWorker.SendMailAsync(new()

{

MailAddress = model.Email,

Subject = "Отчёт по счетам",

Text = $"За период с {model.DateFrom} " +

$"по {model.DateTo}.",

File = pdf,

Role = model.Role,

TypeDoc = TypeDocEnum.PDF

});

//возврат полученных списков для отображения на вебе

return new ReportCashierViewModelForHTML

{

ReportCashWithdrawal = listCashWithdrawals,

ReportMoneyTransfer = listMoneyTransfers

};

}

}

}

ReportClientLogic.cs

using BankBusinessLogic.OfficePackage.HelperModels;

using BankBusinessLogic.OfficePackage;

using BankDataModels.Enums;

using BankBusinessLogic.MailWorker;

using BankContracts.BusinessLogicsContracts.Reports;

using BankContracts.StoragesContracts.Client;

using BankContracts.StoragesContracts.Cashier;

using BankContracts.ViewModels.Reports.Client;

using BankContracts.BindingModels.Reports;

using BankContracts.SearchModels.Client;

using BankContracts.ViewModels.Cashier.ViewModels;

using BankContracts.SearchModels.Cashier;

using BankContracts.ViewModels.Client.ViewModels;

using BankContracts.ViewModels.Reports;

namespace BankBusinessLogic.BusinessLogics.Reports

{

public class ReportClientLogic : IReportClientLogic

{

private readonly ICreditingStorage \_creditingStorage;

private readonly IDebitingStorage \_debitingStorage;

private readonly ICardStorage \_cardStorage;

private readonly IMoneyTransferStorage \_moneyTransferStorage;

private readonly IClientStorage \_clientStorage;

private readonly AbstractSaveToExcel \_saveToExcel;

private readonly AbstractSaveToWord \_saveToWord;

private readonly AbstractSaveToPdf \_saveToPdf;

private readonly MailKitWorker \_mailKitWorker;

public ReportClientLogic(ICreditingStorage creditingStorage, IDebitingStorage debitingStorage,

AbstractSaveToExcel saveToExcel, AbstractSaveToWord saveToWord, AbstractSaveToPdf saveToPdf,

ICardStorage cardStorage, IMoneyTransferStorage moneyTransferStorage,

MailKitWorker mailKitWorker, IClientStorage clientStorage)

{

\_creditingStorage = creditingStorage;

\_debitingStorage = debitingStorage;

\_cardStorage = cardStorage;

\_moneyTransferStorage = moneyTransferStorage;

\_clientStorage = clientStorage;

\_saveToExcel = saveToExcel;

\_saveToWord = saveToWord;

\_saveToPdf = saveToPdf;

\_mailKitWorker = mailKitWorker;

}

public List<ReportClientViewModel>? GetCrediting(ReportBindingModel model)

{

return \_creditingStorage.GetFilteredList(new CreditingSearchModel

{

DateFrom = model.DateFrom,

DateTo = model.DateTo,

}).Select(x => new ReportClientViewModel

{

OperationId = x.Id,

CardNumber = x.CardNumber,

SumOperation = x.Sum,

DateComplite = x.DateOpen

}).ToList();

}

public List<ReportClientViewModel>? GetDebiting(ReportBindingModel model)

{

return \_debitingStorage.GetFilteredList(new DebitingSearchModel

{

DateTo = model.DateFrom,

DateFrom = model.DateTo,

}).Select(x => new ReportClientViewModel

{

OperationId = x.Id,

CardNumber = x.CardNumber,

SumOperation = x.Sum,

DateComplite = x.DateClose

}).ToList();

}

//для excel отчёта по переводам между счетов

public List<MoneyTransferViewModel>? GetMoneyTransfer(ReportBindingModel model)

{

//список счетов по выбранным картам

List<int> accountId = new();

foreach(var index in model.CardList)

{

accountId.Add(\_cardStorage.GetElement(new CardSearchModel { Id = index}).AccountId);

}

var list = accountId.ToHashSet().ToList();

List<MoneyTransferViewModel> totalList = new();

foreach (var index in list)

{

var result = \_moneyTransferStorage.GetFilteredList(new MoneyTransferSearchModel

{

AccountSenderId = index,

AccountPayeeId = index

}).OrderBy(x => x.AccountSenderId).ToList();

totalList.AddRange(result);

}

return totalList;

}

//для excel отчёта по пополнениям карты

public List<CreditingViewModel> GetExcelCrediting(ReportBindingModel model)

{

List<CreditingViewModel> totalList = new();

foreach (var index in model.CardList)

{

var result = \_creditingStorage.GetFilteredList(new CreditingSearchModel

{

CardId = index

});

totalList.AddRange(result);

}

return totalList;

}

//для excel отчёта по снятиям с карты

public List<DebitingViewModel> GetExcelDebiting(ReportBindingModel model)

{

List<DebitingViewModel> totalList = new();

foreach (var index in model.CardList)

{

var result = \_debitingStorage.GetFilteredList(new DebitingSearchModel

{

CardId = index

});

totalList.AddRange(result);

}

return totalList;

}

public void SaveToExcelFile(ReportBindingModel model, OfficeOperationEnum operationEnum)

{

byte[] excel = Array.Empty<byte>();

if (operationEnum == OfficeOperationEnum.Между\_cчетами)

{

\_saveToExcel.CreateReport(new ExcelInfo

{

FileName = model.FileName,

Title = "Отчёт по переводам",

MoneyTransfer = GetMoneyTransfer(model)

}, operationEnum);

excel = System.IO.File.ReadAllBytes("../BankRestAPI/Отчёт по переводам.xlsx");

File.Delete("../BankRestAPI/Отчёт по переводам.xlsx");

}

if (operationEnum == OfficeOperationEnum.Пополнение\_карт)

{

\_saveToExcel.CreateReport(new ExcelInfo

{

FileName = model.FileName,

Title = "Отчёт по пополнениям (переводам из налички на карту)",

Crediting = GetExcelCrediting(model)

}, operationEnum);

excel = System.IO.File.ReadAllBytes("../BankRestAPI/Отчёт по пополнениям.xlsx");

File.Delete("../BankRestAPI/Отчёт по пополнениям.xlsx");

}

if (operationEnum == OfficeOperationEnum.Cнятие\_с\_карты)

{

\_saveToExcel.CreateReport(new ExcelInfo

{

FileName = model.FileName,

Title = "Отчёт по снятиям денежных средств",

Debiting = GetExcelDebiting(model)

}, operationEnum);

excel = System.IO.File.ReadAllBytes("../BankRestAPI/Отчёт по снятиям.xlsx");

File.Delete("../BankRestAPI/Отчёт по снятиям.xlsx");

}

\_mailKitWorker.SendMailAsync(new()

{

MailAddress = model.Email,

Subject = "Отчёт по картам",

Text = $"Отчёт по состоянию на {DateTime.Now.ToString()}",

File = excel,

Role = model.Role,

TypeDoc = TypeDocEnum.EXCEL

});

}

public void SaveToWordFile(ReportBindingModel model, OfficeOperationEnum operationEnum)

{

byte[] word = Array.Empty<byte>();

if (operationEnum == OfficeOperationEnum.Между\_cчетами)

{

\_saveToWord.CreateDoc(new WordInfo

{

FileName = model.FileName,

Title = "Отчёт по переводам",

MoneyTransfer = GetMoneyTransfer(model)

}, operationEnum);

word = System.IO.File.ReadAllBytes("../BankRestAPI/Отчёт по переводам.docx");

File.Delete("../BankRestAPI/Отчёт по переводам.docx");

}

if (operationEnum == OfficeOperationEnum.Пополнение\_карт)

{

\_saveToWord.CreateDoc(new WordInfo

{

FileName = model.FileName,

Title = "Отчёт по пополнениям (переводам из налички на карту)",

Crediting = GetExcelCrediting(model)

}, operationEnum);

word = System.IO.File.ReadAllBytes("../BankRestAPI/Отчёт по пополнениям.docx");

File.Delete("../BankRestAPI/Отчёт по пополнениям.docx");

}

if (operationEnum == OfficeOperationEnum.Cнятие\_с\_карты)

{

\_saveToWord.CreateDoc(new WordInfo

{

FileName = model.FileName,

Title = "Отчёт по снятиям денежных средств",

Debiting = GetExcelDebiting(model)

}, operationEnum);

word = System.IO.File.ReadAllBytes("../BankRestAPI/Отчёт по снятиям.docx");

File.Delete("../BankRestAPI/Отчёт по снятиям.docx");

}

\_mailKitWorker.SendMailAsync(new()

{

MailAddress = model.Email,

Subject = "Отчёт по картам",

Text = $"Отчёт по состоянию на {DateTime.Now.ToString()}",

File = word,

Role = model.Role,

TypeDoc = TypeDocEnum.WORD

});

}

//отчёт в формате PDF для клиента

public ReportClientViewModelForHTML SaveClientReportToPdfFile(ReportBindingModel model)

{

var listCreditings = GetCrediting(model);

var listDebitings = GetDebiting(model);

\_saveToPdf.CreateDoc(new PdfInfo

{

FileName = model.FileName,

Title = "Отчёт по операциям с картами",

DateFrom = model.DateFrom!.Value,

DateTo = model.DateTo!.Value,

ReportCrediting = listCreditings,

ReportDebiting = listDebitings

});

byte[] pdf = System.IO.File.ReadAllBytes("../BankRestAPI/Отчёт\_по\_картам.pdf");

\_mailKitWorker.SendMailAsync(new()

{

MailAddress = model.Email,

Subject = "Отчёт по картам",

Text = $"За период с {model.DateFrom} " +

$"по {model.DateTo}.",

File = pdf,

Role = model.Role,

TypeDoc = TypeDocEnum.PDF

});

File.Delete("../BankRestAPI/Отчёт\_по\_картам.pdf");

//возврат полученных списков для отображения на вебе

return new ReportClientViewModelForHTML

{

ReportCrediting = listCreditings,

ReportDebiting = listDebitings

};

}

}

}

Папка [MailWorker](https://git.is.ulstu.ru/Belnik/CourseWork_Bank/src/branch/main/Bank/BankBusinessLogic/MailWorker)

[AbstractMailWorker.cs](https://git.is.ulstu.ru/Belnik/CourseWork_Bank/src/branch/main/Bank/BankBusinessLogic/MailWorker/AbstractMailWorker.cs)

using BankContracts.BindingModels.Messages;

using BankContracts.BusinessLogicsContracts.Client;

using Microsoft.Extensions.Logging;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankBusinessLogic.MailWorker

{

public abstract class AbstractMailWorker

{

protected string \_mailLogin = string.Empty;

protected string \_mailPassword = string.Empty;

protected string \_smtpClientHost = string.Empty;

protected int \_smtpClientPort;

protected string \_popHost = string.Empty;

protected int \_popPort;

private readonly IClientLogic \_clientLogic;

private readonly ILogger \_logger;

// Конструктор

public AbstractMailWorker(ILogger<AbstractMailWorker> logger, IClientLogic clientLogic)

{

\_logger = logger;

\_clientLogic = clientLogic;

}

public void MailConfig(MailConfigBindingModel config)

{

\_mailLogin = config.MailLogin;

\_mailPassword = config.MailPassword;

\_smtpClientHost = config.SmtpClientHost;

\_smtpClientPort = config.SmtpClientPort;

\_popHost = config.PopHost;

\_popPort = config.PopPort;

\_logger.LogDebug("Config: {login}, {password}, {clientHost}, {clientPort}, {popHost}, {popPort}",

\_mailLogin, \_mailPassword, \_smtpClientHost, \_smtpClientPort, \_popHost, \_popPort);

}

public async void MailSendAsync(MailSendInfoBindingModel info)

{

if (string.IsNullOrEmpty(\_mailLogin) || string.IsNullOrEmpty(\_mailPassword))

{

return;

}

if (string.IsNullOrEmpty(\_smtpClientHost) || \_smtpClientPort == 0)

{

return;

}

if (string.IsNullOrEmpty(info.MailAddress) || string.IsNullOrEmpty(info.Subject) || string.IsNullOrEmpty(info.Text))

{

return;

}

\_logger.LogDebug("Send Mail: {To}, {Subject}", info.MailAddress, info.Subject);

await SendMailAsync(info);

}

protected abstract Task SendMailAsync(MailSendInfoBindingModel info);

}

}

[MailKitWorker.cs](https://git.is.ulstu.ru/Belnik/CourseWork_Bank/src/branch/main/Bank/BankBusinessLogic/MailWorker/MailKitWorker.cs)

using BankContracts.BusinessLogicsContracts;

using BankContracts.BindingModels;

using Microsoft.Extensions.Logging;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Mail;

using System.Net;

using System.Text;

using System.Threading.Tasks;

using MailKit.Net.Pop3;

using MailKit.Security;

using BankDataModels.Enums;

using DocumentFormat.OpenXml.EMMA;

using BankContracts.BindingModels.Messages;

namespace BankBusinessLogic.MailWorker

{

//класс, отвечающий за отправку письма

public class MailKitWorker

{

private string \_mailLogin = string.Empty;

private string \_mailPassword = string.Empty;

private string \_smtpClientHost = string.Empty;

private int \_smtpClientPort;

private readonly ILogger logger;

public MailKitWorker(ILogger<MailKitWorker> logger)

{

this.logger = logger;

}

public void MailConfig(MailConfigBindingModel config)

{

\_mailLogin = config.MailLogin;

\_mailPassword = config.MailPassword;

\_smtpClientHost = config.SmtpClientHost;

\_smtpClientPort = config.SmtpClientPort;

}

public async void SendMailAsync(MailSendInfoBindingModel info)

{

using var objMailMessage = new MailMessage();

using var objSmtpClient = new SmtpClient(\_smtpClientHost, \_smtpClientPort);

try

{

objMailMessage.From = new MailAddress(\_mailLogin);

objMailMessage.To.Add(new MailAddress(info.MailAddress));

objMailMessage.Subject = info.Subject;

objMailMessage.Body = info.Text;

objMailMessage.SubjectEncoding = Encoding.UTF8;

objMailMessage.BodyEncoding = Encoding.UTF8;

MemoryStream ms = new(info.File);

if(info.Role == MailsEnum.Клиент)

{

if(info.TypeDoc == TypeDocEnum.PDF)

{

objMailMessage.Attachments.Add(new Attachment(ms, "Отчёт\_для\_клиента.pdf", "application/pdf"));

}

if (info.TypeDoc == TypeDocEnum.EXCEL)

{

objMailMessage.Attachments.Add(new Attachment(ms, "Отчёт\_для\_клиента.xlsx", "application/xlsx"));

}

if (info.TypeDoc == TypeDocEnum.WORD)

{

objMailMessage.Attachments.Add(new Attachment(ms, "Отчёт\_для\_клиента.docx", "application/docx"));

}

}

else

{

if (info.TypeDoc == TypeDocEnum.PDF)

{

objMailMessage.Attachments.Add(new Attachment(ms, "Отчёт\_для\_кассира.pdf", "application/pdf"));

}

if (info.TypeDoc == TypeDocEnum.EXCEL)

{

objMailMessage.Attachments.Add(new Attachment(ms, "Отчёт\_для\_кассира.xlsx", "application/xlsx"));

}

if (info.TypeDoc == TypeDocEnum.WORD)

{

objMailMessage.Attachments.Add(new Attachment(ms, "Отчёт\_для\_кассира.docx", "application/docx"));

}

}

objSmtpClient.UseDefaultCredentials = false;

objSmtpClient.EnableSsl = true;

objSmtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;

objSmtpClient.Credentials = new NetworkCredential(\_mailLogin, \_mailPassword);

await Task.Run(() => objSmtpClient.Send(objMailMessage));

}

catch (Exception)

{

throw;

}

}

}

}

Папка OfficePackage

[AbstractSaveToExcel.cs](https://git.is.ulstu.ru/Belnik/CourseWork_Bank/src/branch/main/Bank/BankBusinessLogic/OfficePackage/AbstractSaveToExcel.cs)

using BankBusinessLogic.OfficePackage.HelperEnums;

using BankBusinessLogic.OfficePackage.HelperModels;

using BankDataModels.Enums;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankBusinessLogic.OfficePackage

{

public abstract class AbstractSaveToExcel

{

//Создание отчета. Описание методов ниже

public void CreateReport(ExcelInfo info, OfficeOperationEnum operationEnum)

{

if(operationEnum == OfficeOperationEnum.Между\_cчетами)

{

CreateMoneyTransferExcel(info);

}

if (operationEnum == OfficeOperationEnum.Пополнение\_карт)

{

CreateCreditingExcel(info);

}

if (operationEnum == OfficeOperationEnum.Cнятие\_с\_карты)

{

CreateDebitingExcel(info);

}

if (operationEnum == OfficeOperationEnum.Для\_кассира)

{

CreateCashierExcel(info);

}

}

private void CreateMoneyTransferExcel(ExcelInfo info)

{

CreateExcel(info);

//вставляет заголовок

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = 1,

Text = info.Title,

StyleInfo = ExcelStyleInfoType.Title

});

//соединяет 3 ячейки для заголовка

MergeCells(new ExcelMergeParameters

{

CellFromName = "A1",

CellToName = "C1"

});

//номер строчки в докуметне

uint rowIndex = 2;

foreach (var mt in info.MoneyTransfer)

{

//вставляет номер перевода

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = rowIndex,

Text = "Перевод №" + mt.Id,

StyleInfo = ExcelStyleInfoType.Text

});

rowIndex++;

//строчка с номером счёта отправителя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "B",

RowIndex = rowIndex,

Text = "Номер счёта отправителя: ",

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

//вставка номера отправителя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = mt.AccountSenderNumber,

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

rowIndex++;

//строчка с номером счёта получателя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "B",

RowIndex = rowIndex,

Text = "Номер счёта получателя: ",

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

//вставка номера отправителя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = mt.AccountPayeeNumber,

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

rowIndex++;

//Вставляет слово "Сумма перевода"

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = rowIndex,

Text = "Сумма перевода: ",

StyleInfo = ExcelStyleInfoType.Text

});

//подсчитывает общее кол-во заготовок в изделии

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = mt.Sum.ToString(),

StyleInfo = ExcelStyleInfoType.Text

});

rowIndex++;

//Вставляет слово "Сумма перевода"

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = rowIndex,

Text = "Дата операции: ",

StyleInfo = ExcelStyleInfoType.Text

});

//подсчитывает общее кол-во заготовок в изделии

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = mt.DateOperation.ToString(),

StyleInfo = ExcelStyleInfoType.Text

});

rowIndex++;

}

rowIndex++;

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = rowIndex,

Text = "Суммарный объём переводов: ",

StyleInfo = ExcelStyleInfoType.Text

});

MergeCells(new ExcelMergeParameters

{

CellFromName = "A" + rowIndex.ToString(),

CellToName = "B" + rowIndex.ToString()

});

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = info.MoneyTransfer.Sum(x => x.Sum).ToString(),

StyleInfo = ExcelStyleInfoType.Text

});

SaveExcel(info);

}

private void CreateCreditingExcel(ExcelInfo info)

{

CreateExcel(info);

//вставляет заголовок

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = 1,

Text = info.Title,

StyleInfo = ExcelStyleInfoType.Title

});

//соединяет 3 ячейки для заголовка

MergeCells(new ExcelMergeParameters

{

CellFromName = "A1",

CellToName = "E2"

});

//номер строчки в докуметне

uint rowIndex = 3;

//string supportNumber = string.Empty;

foreach (var cr in info.Crediting)

{

//вставляет номер перевода

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = rowIndex,

Text = "Пополнение №" + cr.Id,

StyleInfo = ExcelStyleInfoType.Text

});

rowIndex++;

//строчка с номером счёта отправителя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "B",

RowIndex = rowIndex,

Text = "Номер карты: ",

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

//вставка номера отправителя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = cr.CardNumber,

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

rowIndex++;

//строчка с номером счёта получателя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "B",

RowIndex = rowIndex,

Text = "Сумма пополнения: ",

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

//вставка номера отправителя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = cr.Sum.ToString(),

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

rowIndex++;

}

rowIndex++;

//формирование списка поступления по каждой карте

var dict = new Dictionary<string, int>();

foreach(var elem in info.Crediting)

{

if (dict.ContainsKey(elem.CardNumber))

{

dict[elem.CardNumber] += elem.Sum;

}

else

{

dict[elem.CardNumber] = elem.Sum;

}

}

foreach(var elem in dict)

{

//строчка с номером счёта получателя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "B",

RowIndex = rowIndex,

Text = "Сумма пополнения на карту №" + elem.Key + ":",

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

//вставка номера отправителя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = elem.Value.ToString(),

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

rowIndex++;

}

rowIndex++;

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = rowIndex,

Text = "Суммарный объём пополнений: ",

StyleInfo = ExcelStyleInfoType.Text

});

MergeCells(new ExcelMergeParameters

{

CellFromName = "A" + rowIndex.ToString(),

CellToName = "B" + rowIndex.ToString()

});

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = info.Crediting.Sum(x => x.Sum).ToString(),

StyleInfo = ExcelStyleInfoType.Text

});

SaveExcel(info);

}

private void CreateDebitingExcel(ExcelInfo info)

{

CreateExcel(info);

//вставляет заголовок

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = 1,

Text = info.Title,

StyleInfo = ExcelStyleInfoType.Title

});

//соединяет 3 ячейки для заголовка

MergeCells(new ExcelMergeParameters

{

CellFromName = "A1",

CellToName = "E2"

});

//номер строчки в докуметне

uint rowIndex = 3;

foreach (var cr in info.Debiting)

{

//вставляет номер перевода

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = rowIndex,

Text = "Снятие №" + cr.Id,

StyleInfo = ExcelStyleInfoType.Text

});

rowIndex++;

//строчка с номером счёта отправителя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "B",

RowIndex = rowIndex,

Text = "Номер карты: ",

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

//вставка номера отправителя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = cr.CardNumber,

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

rowIndex++;

//строчка с номером счёта получателя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "B",

RowIndex = rowIndex,

Text = "Сумма снятия: ",

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

//вставка номера отправителя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = cr.Sum.ToString(),

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

rowIndex++;

}

rowIndex++;

//формирование списка поступления по каждой карте

var dict = new Dictionary<string, int>();

foreach (var elem in info.Debiting)

{

if (dict.ContainsKey(elem.CardNumber))

{

dict[elem.CardNumber] += elem.Sum;

}

else

{

dict[elem.CardNumber] = elem.Sum;

}

}

foreach (var elem in dict)

{

//строчка с номером счёта получателя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "B",

RowIndex = rowIndex,

Text = "Сумма снятия с карты №" + elem.Key + ":",

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

//вставка номера отправителя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = elem.Value.ToString(),

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

rowIndex++;

}

rowIndex++;

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = rowIndex,

Text = "Суммарный объём снятий: ",

StyleInfo = ExcelStyleInfoType.Text

});

MergeCells(new ExcelMergeParameters

{

CellFromName = "A" + rowIndex.ToString(),

CellToName = "B" + rowIndex.ToString()

});

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = info.Debiting.Sum(x => x.Sum).ToString(),

StyleInfo = ExcelStyleInfoType.Text

});

SaveExcel(info);

}

private void CreateCashierExcel(ExcelInfo info)

{

CreateExcel(info);

//вставляет заголовок

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = 1,

Text = info.Title,

StyleInfo = ExcelStyleInfoType.Title

});

//соединяет 3 ячейки для заголовка

MergeCells(new ExcelMergeParameters

{

CellFromName = "A1",

CellToName = "E2"

});

//номер строчки в докуметне

uint rowIndex = 3;

foreach (var cr in info.Debiting)

{

//вставляет номер перевода

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = rowIndex,

Text = "Заявка №" + cr.Id,

StyleInfo = ExcelStyleInfoType.Text

});

rowIndex++;

//строчка с номером счёта получателя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "B",

RowIndex = rowIndex,

Text = "Сумма заявки: ",

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

//вставка суммы заявки

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = cr.Sum.ToString(),

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

rowIndex++;

//строчка с номером счёта получателя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "B",

RowIndex = rowIndex,

Text = "Статус: ",

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

//вставка номера отправителя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = cr.Status.ToString(),

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

rowIndex++;

//строчка с номером счёта получателя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "B",

RowIndex = rowIndex,

Text = "Дата: ",

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

//вставка номера отправителя

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = cr.DateClose == null ? "В обработке" : cr.DateClose.ToString(),

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

rowIndex++;

}

rowIndex++;

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = rowIndex,

Text = "Суммарный объём открытых заявок на снятие: ",

StyleInfo = ExcelStyleInfoType.Text

});

MergeCells(new ExcelMergeParameters

{

CellFromName = "A" + rowIndex.ToString(),

CellToName = "C" + rowIndex.ToString()

});

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "D",

RowIndex = rowIndex,

Text = info.Debiting.Where(x => x.Status == StatusEnum.Открыта).Sum(x => x.Sum).ToString(),

StyleInfo = ExcelStyleInfoType.Text

});

rowIndex += 2;

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = rowIndex,

Text = "Суммарный объём закртытых заявок на снятие: ",

StyleInfo = ExcelStyleInfoType.Text

});

MergeCells(new ExcelMergeParameters

{

CellFromName = "A" + rowIndex.ToString(),

CellToName = "C" + rowIndex.ToString()

});

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "D",

RowIndex = rowIndex,

Text = info.Debiting.Where(x => x.Status == StatusEnum.Закрыта).Sum(x => x.Sum).ToString(),

StyleInfo = ExcelStyleInfoType.Text

});

SaveExcel(info);

}

//Создание excel-файла

protected abstract void CreateExcel(ExcelInfo info);

//Добавляем новую ячейку в лист

protected abstract void InsertCellInWorksheet(ExcelCellParameters excelParams);

//Объединение ячеек

protected abstract void MergeCells(ExcelMergeParameters excelParams);

//Сохранение файла

protected abstract void SaveExcel(ExcelInfo info);

}

}

[AbstractSaveToPdf.cs](https://git.is.ulstu.ru/Belnik/CourseWork_Bank/src/branch/main/Bank/BankBusinessLogic/OfficePackage/AbstractSaveToPdf.cs)

using BankBusinessLogic.OfficePackage.HelperEnums;

using BankBusinessLogic.OfficePackage.HelperModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankBusinessLogic.OfficePackage

{

public abstract class AbstractSaveToPdf

{

//публичный метод создания документа. Описание методов ниже

public void CreateDoc(PdfInfo info)

{

if(info.ForClient)

{

CreateDocClient(info);

}

else

{

CreateDocCashier(info);

}

}

#region Отчёт для клиента

public void CreateDocClient(PdfInfo info)

{

CreatePdf(info);

CreateParagraph(new PdfParagraph

{

Text = info.Title + $"\nот {DateTime.Now.ToShortDateString()}",

Style = "NormalTitle",

ParagraphAlignment = PdfParagraphAlignmentType.Center

});

CreateParagraph(new PdfParagraph

{

Text = $"Расчётный период: с {info.DateFrom.ToShortDateString()} по {info.DateTo.ToShortDateString()}",

Style = "Normal",

ParagraphAlignment = PdfParagraphAlignmentType.Center

});

//параграф с отчётом на пополнения

CreateParagraph(new PdfParagraph { Text = "Отчёт по пополнениям", Style = "Normal", ParagraphAlignment = PdfParagraphAlignmentType.Center });

CreateTable(new List<string> { "3cm", "3cm", "5cm", "5cm" });

CreateRow(new PdfRowParameters

{

Texts = new List<string> { "Номер операции", "Номер карты", "Сумма", "Дата операции" },

Style = "NormalTitle",

ParagraphAlignment = PdfParagraphAlignmentType.Center

});

foreach (var report in info.ReportCrediting)

{

CreateRow(new PdfRowParameters

{

Texts = new List<string> { report.OperationId.ToString(), report.CardNumber, report.SumOperation.ToString(), report.DateComplite.ToString() },

Style = "Normal",

ParagraphAlignment = PdfParagraphAlignmentType.Left

});

}

//подсчёт суммы операций на пополнение

CreateParagraph(new PdfParagraph { Text = $"Итоговая сумма поступлений за период: {info.ReportCrediting.Sum(x => x.SumOperation)}\t", Style = "Normal", ParagraphAlignment = PdfParagraphAlignmentType.Right });

//отчёт с отчётом на снятие

CreateParagraph(new PdfParagraph { Text = "Отчёт по снятиям", Style = "Normal", ParagraphAlignment = PdfParagraphAlignmentType.Center });

CreateTable(new List<string> { "3cm", "3cm", "5cm", "5cm" });

CreateRow(new PdfRowParameters

{

Texts = new List<string> { "Номер операции", "Номер карты", "Сумма", "Дата операции" },

Style = "NormalTitle",

ParagraphAlignment = PdfParagraphAlignmentType.Center

});

foreach (var report in info.ReportDebiting)

{

CreateRow(new PdfRowParameters

{

Texts = new List<string> { report.OperationId.ToString(), report.CardNumber, report.SumOperation.ToString(), report.DateComplite.ToString() },

Style = "Normal",

ParagraphAlignment = PdfParagraphAlignmentType.Left

});

}

//подсчёт суммы операций на пополнение

CreateParagraph(new PdfParagraph { Text = $"Итоговая сумма снятий за период: {info.ReportDebiting.Sum(x => x.SumOperation)}\t", Style = "Normal", ParagraphAlignment = PdfParagraphAlignmentType.Right });

SavePdf(info);

}

#endregion

#region Отчёт для кассира

//создание отчёта для кассира

public void CreateDocCashier(PdfInfo info)

{

CreatePdf(info);

CreateParagraph(new PdfParagraph

{

Text = info.Title + $"\nот {DateTime.Now.ToShortDateString()}\nФИО клиента: {info.FullClientName}",

Style = "NormalTitle",

ParagraphAlignment = PdfParagraphAlignmentType.Center

});

CreateParagraph(new PdfParagraph

{

Text = $"Расчётный период: с {info.DateFrom.ToShortDateString()} по {info.DateTo.ToShortDateString()}",

Style = "Normal",

ParagraphAlignment = PdfParagraphAlignmentType.Center

});

//параграф с отчётом по выдаче наличных с карт

CreateParagraph(new PdfParagraph { Text = "Отчёт по выдаче наличных со счёта", Style = "Normal", ParagraphAlignment = PdfParagraphAlignmentType.Center });

CreateTable(new List<string> { "3.5cm", "3.5cm", "5cm", "5cm" });

CreateRow(new PdfRowParameters

{

Texts = new List<string> { "Номер операции", "Номер счёта", "Сумма операции", "Дата операции" },

Style = "NormalTitle",

ParagraphAlignment = PdfParagraphAlignmentType.Center

});

foreach (var report in info.ReportCashWithdrawal)

{

CreateRow(new PdfRowParameters

{

Texts = new List<string> { report.OperationId.ToString(), report.AccountPayeeNumber, report.SumOperation.ToString(), report.DateComplite.ToShortDateString(), },

Style = "Normal",

ParagraphAlignment = PdfParagraphAlignmentType.Left

});

}

CreateParagraph(new PdfParagraph { Text = $"Итоговая сумма снятий за период: {info.ReportCashWithdrawal.Sum(x => x.SumOperation)}\t", Style = "Normal", ParagraphAlignment = PdfParagraphAlignmentType.Right });

//параграф с отчётом по переводу денег со счёта на счёт

CreateParagraph(new PdfParagraph { Text = "Отчёт по денежным переводам между счетами", Style = "Normal", ParagraphAlignment = PdfParagraphAlignmentType.Center });

CreateTable(new List<string> { "3cm", "3cm", "3cm", "4cm", "4cm" });

CreateRow(new PdfRowParameters

{

Texts = new List<string> { "Номер операции", "Номер счёта отправителя", "Номер счёта получателя", "Сумма операции", "Дата операции" },

Style = "NormalTitle",

ParagraphAlignment = PdfParagraphAlignmentType.Center

});

foreach (var report in info.ReportMoneyTransfer)

{

CreateRow(new PdfRowParameters

{

Texts = new List<string> { report.OperationId.ToString(), report.AccountSenderNumber, report.AccountPayeeNumber, report.SumOperation.ToString(), report.DateComplite.ToShortDateString(), },

Style = "Normal",

ParagraphAlignment = PdfParagraphAlignmentType.Left

});

}

CreateParagraph(new PdfParagraph { Text = $"Итоговая сумма переводов за период: {info.ReportMoneyTransfer.Sum(x => x.SumOperation)}\t", Style = "Normal", ParagraphAlignment = PdfParagraphAlignmentType.Right });

SavePdf(info);

}

#endregion

/// Создание pdf-файла

protected abstract void CreatePdf(PdfInfo info);

/// Создание параграфа с текстом

protected abstract void CreateParagraph(PdfParagraph paragraph);

/// Создание таблицы

protected abstract void CreateTable(List<string> columns);

/// Создание и заполнение строки

protected abstract void CreateRow(PdfRowParameters rowParameters);

/// Сохранение файла

protected abstract void SavePdf(PdfInfo info);

}

}

AbstractSaveToWord.cs

using BankBusinessLogic.OfficePackage.HelperEnums;

using BankBusinessLogic.OfficePackage.HelperModels;

using BankDataModels.Enums;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankBusinessLogic.OfficePackage

{

public abstract class AbstractSaveToWord

{

//метод создания документа

public void CreateDoc(WordInfo info, OfficeOperationEnum operationEnum)

{

if (operationEnum == OfficeOperationEnum.Между\_cчетами)

{

CreateMoneyTransferWord(info);

}

if (operationEnum == OfficeOperationEnum.Пополнение\_карт)

{

CreateCreditingWord(info);

}

if (operationEnum == OfficeOperationEnum.Cнятие\_с\_карты)

{

CreateDebitingWord(info);

}

if (operationEnum == OfficeOperationEnum.Для\_кассира)

{

CreateCashierWord(info);

}

}

private void CreateMoneyTransferWord(WordInfo info)

{

CreateWord(info);

CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)> { (info.Title, new WordTextProperties { Bold = true, Size = "24" }) },

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Center

}

});

foreach (var transfer in info.MoneyTransfer)

{

List<List<(string, WordTextProperties)>> rowList = new()

{

new()

{

new("Номер счёта отправителя", new WordTextProperties { Bold = true, Size = "20" } ),

new("Номер счёта получателя", new WordTextProperties { Bold = true, Size = "20" } ),

new("Сумма операции", new WordTextProperties { Bold = true, Size = "20" } ),

new("Дата перевода", new WordTextProperties { Bold = true, Size = "20" } )

}

};

CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)> { ("Перевод №" + transfer.Id.ToString(), new WordTextProperties { Bold = true, Size = "24" }) },

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Center

}

});

List<(string, WordTextProperties)> cellList = new()

{

new(transfer.AccountSenderNumber, new WordTextProperties { Size = "20" }),

new(transfer.AccountPayeeNumber, new WordTextProperties { Size = "20" }),

new(transfer.Sum.ToString(), new WordTextProperties { Size = "20"}),

new(transfer.DateOperation.ToString(), new WordTextProperties { Size = "20"}),

};

rowList.Add(cellList);

CreateTable(new WordParagraph

{

RowTexts = rowList,

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Center

}

});

}

CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)> { ("Суммарный объём переводов: " + info.MoneyTransfer.Sum(x => x.Sum).ToString(),

new WordTextProperties { Bold = true, Size = "24" }) },

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Both

}

});

SaveWord(info);

}

private void CreateCreditingWord(WordInfo info)

{

CreateWord(info);

CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)> { (info.Title, new WordTextProperties { Bold = true, Size = "24" }) },

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Center

}

});

foreach (var crediting in info.Crediting)

{

List<List<(string, WordTextProperties)>> rowList = new()

{

new()

{

new("Номер карты", new WordTextProperties { Bold = true, Size = "24" } ),

new("Сумма пополнения", new WordTextProperties { Bold = true, Size = "24" } ),

new("Дата выполнения", new WordTextProperties { Bold = true, Size = "24" } )

}

};

CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)> { ("Пополнение №" + crediting.Id.ToString(), new WordTextProperties { Bold = true, Size = "24" }) },

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Center

}

});

List<(string, WordTextProperties)> cellList = new()

{

new(crediting.CardNumber, new WordTextProperties { Size = "24" }),

new(crediting.Sum.ToString(), new WordTextProperties { Size = "24" }),

new(crediting.DateClose == null ? "В обработке" : crediting.DateClose.ToString(), new WordTextProperties { Size = "24" })

};

rowList.Add(cellList);

CreateTable(new WordParagraph

{

RowTexts = rowList,

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Center

}

});

}

//формирование списка поступления по каждой карте

var dict = new Dictionary<string, int>();

foreach (var elem in info.Crediting)

{

if (dict.ContainsKey(elem.CardNumber))

{

dict[elem.CardNumber] += elem.Sum;

}

else

{

dict[elem.CardNumber] = elem.Sum;

}

}

foreach (var elem in dict)

{

CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)> { ("Суммарное пополнение на карту №" + elem.Key + ": " + elem.Value.ToString(), new WordTextProperties { Bold = true, Size = "24" }) },

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Both

}

});

}

SaveWord(info);

}

private void CreateDebitingWord(WordInfo info)

{

CreateWord(info);

CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)> { (info.Title, new WordTextProperties { Bold = true, Size = "24" }) },

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Center

}

});

foreach (var crediting in info.Debiting)

{

List<List<(string, WordTextProperties)>> rowList = new()

{

new()

{

new("Номер карты", new WordTextProperties { Bold = true, Size = "24" } ),

new("Сумма снятия", new WordTextProperties { Bold = true, Size = "24" } ),

new("Дата выполнения", new WordTextProperties { Bold = true, Size = "24" } )

}

};

CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)> { ("Снятие №" + crediting.Id.ToString(), new WordTextProperties { Bold = true, Size = "24" }) },

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Center

}

});

List<(string, WordTextProperties)> cellList = new()

{

new(crediting.CardNumber, new WordTextProperties { Size = "24" }),

new(crediting.Sum.ToString(), new WordTextProperties { Size = "24" }),

new(crediting.DateClose == null ? "В обработке" : crediting.DateClose.ToString(), new WordTextProperties { Size = "24" })

};

rowList.Add(cellList);

CreateTable(new WordParagraph

{

RowTexts = rowList,

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Center

}

});

}

//формирование списка поступления по каждой карте

var dict = new Dictionary<string, int>();

foreach (var elem in info.Debiting)

{

if (dict.ContainsKey(elem.CardNumber))

{

dict[elem.CardNumber] += elem.Sum;

}

else

{

dict[elem.CardNumber] = elem.Sum;

}

}

foreach (var elem in dict)

{

CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)> { ("Суммарное снятие с карты №" + elem.Key + ": " + elem.Value.ToString(), new WordTextProperties { Bold = true, Size = "24" }) },

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Both

}

});

}

SaveWord(info);

}

private void CreateCashierWord(WordInfo info)

{

CreateWord(info);

CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)> { (info.Title, new WordTextProperties { Bold = true, Size = "24" }) },

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Center

}

});

foreach (var crediting in info.Debiting)

{

List<List<(string, WordTextProperties)>> rowList = new()

{

new()

{

new("Сумма заявки", new WordTextProperties { Bold = true, Size = "24" } ),

new("Дата открытия", new WordTextProperties { Bold = true, Size = "24" } ),

new("Статус", new WordTextProperties { Bold = true, Size = "24" } )

}

};

CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)> { ("Заявка №" + crediting.Id.ToString(), new WordTextProperties { Bold = true, Size = "24" }) },

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Center

}

});

List<(string, WordTextProperties)> cellList = new()

{

new(crediting.Sum.ToString(), new WordTextProperties { Size = "24" }),

new(crediting.DateOpen.ToString(), new WordTextProperties { Size = "24" }),

new(crediting.Status.ToString(), new WordTextProperties { Size = "24" })

};

rowList.Add(cellList);

CreateTable(new WordParagraph

{

RowTexts = rowList,

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Center

}

});

}

SaveWord(info);

}

// Создание doc-файла

protected abstract void CreateWord(WordInfo info);

// Создание абзаца с текстом

protected abstract void CreateParagraph(WordParagraph paragraph);

//Создание таблицы

protected abstract void CreateTable(WordParagraph paragraph);

// Сохранение файла

protected abstract void SaveWord(WordInfo info);

}

}

Папка HelperEnums

[ExcelStyleInfoType.cs](https://git.is.ulstu.ru/Belnik/CourseWork_Bank/src/branch/main/Bank/BankBusinessLogic/OfficePackage/HelperEnums/ExcelStyleInfoType.cs)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankBusinessLogic.OfficePackage.HelperEnums

{

//вспомогательное перечисление для оформления exel

public enum ExcelStyleInfoType

{

//заголовок

Title,

//просто текст

Text,

//текст в рамке

TextWithBorder

}

}

PdfParagraphAlignmentType.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankBusinessLogic.OfficePackage.HelperEnums

{

//вспомогательное перечисление для оформления pdf документа

public enum PdfParagraphAlignmentType

{

//либо по центру

Center,

//либо с левого края

Left,

//либо с правого края

Right

}

}

WordJustificationType.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankBusinessLogic.OfficePackage.HelperEnums

{

//вспомогательное перечисление для настройки формата word документа

public enum WordJustificationType

{

//выравниваем либо по центру

Center,

//либо на всю ширину

Both

}

}

Папка [HelperModels](https://git.is.ulstu.ru/Belnik/CourseWork_Bank/src/branch/main/Bank/BankBusinessLogic/OfficePackage/HelperModels)

ExcelCellParameters.cs

using BankBusinessLogic.OfficePackage.HelperEnums;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankBusinessLogic.OfficePackage.HelperModels

{

//информация по ячейке в таблице excel

public class ExcelCellParameters

{

//название колонки

public string ColumnName { get; set; } = string.Empty;

//строка

public uint RowIndex { get; set; }

//тект в ячейке

public string Text { get; set; } = string.Empty;

//геттер для того, чтобы не искать каждый раз

public string CellReference => $"{ColumnName}{RowIndex}";

//в каком стиле выводить информацию

public ExcelStyleInfoType StyleInfo { get; set; }

}

}

ExcelInfo.cs

using BankContracts.ViewModels.Cashier.ViewModels;

using BankContracts.ViewModels.Client.ViewModels;

namespace BankBusinessLogic.OfficePackage.HelperModels

{

//информация по excel файлу, который хотим создать

public class ExcelInfo

{

//название файла

public string FileName { get; set; } = string.Empty;

//заголовок

public string Title { get; set; } = string.Empty;

//списки для отчёта клиента

public List<MoneyTransferViewModel> MoneyTransfer { get; set; } = new();

public List<CreditingViewModel> Crediting { get; set; } = new();

//список для отчёта кассира и клиента

public List<DebitingViewModel> Debiting { get; set; } = new();

}

}

ExcelMergeParameters.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankBusinessLogic.OfficePackage.HelperModels

{

//информация для объединения ячеек

public class ExcelMergeParameters

{

public string CellFromName { get; set; } = string.Empty;

public string CellToName { get; set; } = string.Empty;

//гетер для указания диапазона для объединения, чтобы каждый раз его не вычислять

public string Merge => $"{CellFromName}:{CellToName}";

}

}

PdfInfo.cs

using BankContracts.ViewModels.Reports.Cashier;

using BankContracts.ViewModels.Reports.Client;

namespace BankBusinessLogic.OfficePackage.HelperModels

{

//общая информация по pdf файлу

public class PdfInfo

{

public string FileName { get; set; } = string.Empty;

public string Title { get; set; } = string.Empty;

public DateTime DateFrom { get; set; }

public DateTime DateTo { get; set; }

//по умолчанию отчёт делается для клиента

public bool ForClient { get; set; } = true;

//для передачи полного имени клиента в отчёт

public string FullClientName { get; set; } = string.Empty;

//перечень заказов за указанный период для вывода/сохранения

public List<ReportClientViewModel> ReportCrediting { get; set; } = new();

//перечень заказов за указанный период для вывода/сохранения

public List<ReportClientViewModel> ReportDebiting { get; set; } = new();

//перечень переводов со счёта на счёт

public List<ReportCashierViewModel> ReportMoneyTransfer { get; set; } = new();

//перечень зачислений денежных средств на карту (т. е. на её счёт)

public List<ReportCashierViewModel> ReportCashWithdrawal { get; set; } = new();

}

}

PdfParagraph.cs

using BankBusinessLogic.OfficePackage.HelperEnums;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankBusinessLogic.OfficePackage.HelperModels

{

//информация п параграфу в pdf документе

public class PdfParagraph

{

public string Text { get; set; } = string.Empty;

public string Style { get; set; } = string.Empty;

//информация по выравниванию текста в параграфе

public PdfParagraphAlignmentType ParagraphAlignment { get; set; }

}

}

PdfRowParameters.cs

using BankBusinessLogic.OfficePackage.HelperEnums;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankBusinessLogic.OfficePackage.HelperModels

{

//информация по параметрам строк таблицы

public class PdfRowParameters

{

//набор текстов

public List<string> Texts { get; set; } = new();

//стиль к текстам

public string Style { get; set; } = string.Empty;

//как выравниваем

public PdfParagraphAlignmentType ParagraphAlignment { get; set; }

}

}

WordInfo.cs

using BankContracts.ViewModels.Cashier.ViewModels;

using BankContracts.ViewModels.Client.ViewModels;

namespace BankBusinessLogic.OfficePackage.HelperModels

{

//общая информация по документу

public class WordInfo

{

public string FileName { get; set; } = string.Empty;

public string Title { get; set; } = string.Empty;

//списки для отчёта клиента

public List<MoneyTransferViewModel> MoneyTransfer { get; set; } = new();

public List<CreditingViewModel> Crediting { get; set; } = new();

//список для отчёта кассира и клиента

public List<DebitingViewModel> Debiting { get; set; } = new();

}

}

WordParagraph.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankBusinessLogic.OfficePackage.HelperModels

{

//модель параграфов, которые есть в тексте

public class WordParagraph

{

//набор текстов в абзаце (для случая, если в абзаце текст разных стилей)

public List<(string, WordTextProperties)> Texts { get; set; } = new();

//свойства параграфа, если они есть

public WordTextProperties? TextProperties { get; set; }

public List<List<(string, WordTextProperties)>> RowTexts { get; set; } = new();

}

}

WordTextProperties.cs

using BankBusinessLogic.OfficePackage.HelperEnums;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankBusinessLogic.OfficePackage.HelperModels

{

//модель свойств текста, которые нам нужны в word документе

public class WordTextProperties

{

//размере текста

public string Size { get; set; } = string.Empty;

//надо ли делать его жирным

public bool Bold { get; set; }

//выравнивание

public WordJustificationType JustificationType { get; set; }

}

}

Папка Implements

SaveToExcel.cs

using BankBusinessLogic.OfficePackage.HelperModels;

using BankBusinessLogic.OfficePackage.HelperEnums;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Spreadsheet;

using DocumentFormat.OpenXml.Office2010.Excel;

using DocumentFormat.OpenXml.Office2013.Excel;

using DocumentFormat.OpenXml.Office2016.Excel;

using DocumentFormat.OpenXml;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankBusinessLogic.OfficePackage.Implements

{

public class SaveToExcel : AbstractSaveToExcel

{

private SpreadsheetDocument? \_spreadsheetDocument;

private SharedStringTablePart? \_shareStringPart;

private Worksheet? \_worksheet;

//Настройка стилей для файла

private static void CreateStyles(WorkbookPart workbookpart)

{

var sp = workbookpart.AddNewPart<WorkbookStylesPart>();

sp.Stylesheet = new Stylesheet();

//настройка шрифта простого текста

var fonts = new Fonts() { Count = 2U, KnownFonts = true };

var fontUsual = new Font();

fontUsual.Append(new FontSize() { Val = 12D });

fontUsual.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color() { Theme = 1U });

fontUsual.Append(new FontName() { Val = "Times New Roman" });

fontUsual.Append(new FontFamilyNumbering() { Val = 2 });

fontUsual.Append(new FontScheme() { Val = FontSchemeValues.Minor });

//настройка шрифта заголока

var fontTitle = new Font();

fontTitle.Append(new Bold());

fontTitle.Append(new FontSize() { Val = 14D });

fontTitle.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color() { Theme = 1U });

fontTitle.Append(new FontName() { Val = "Times New Roman" });

fontTitle.Append(new FontFamilyNumbering() { Val = 2 });

fontTitle.Append(new FontScheme() { Val = FontSchemeValues.Minor });

//добавление созданных шрифтов

fonts.Append(fontUsual);

fonts.Append(fontTitle);

//создание заливки

var fills = new Fills() { Count = 2U };

var fill1 = new Fill();

fill1.Append(new PatternFill()

{

PatternType = PatternValues.None

});

var fill2 = new Fill();

fill2.Append(new PatternFill()

{

PatternType = PatternValues.Gray125

});

fills.Append(fill1);

fills.Append(fill2);

//стиль границ ячейки - незакрашенный (для заголовка) и закрашенный

var borders = new Borders() { Count = 2U };

var borderNoBorder = new Border();

borderNoBorder.Append(new LeftBorder());

borderNoBorder.Append(new RightBorder());

borderNoBorder.Append(new TopBorder());

borderNoBorder.Append(new BottomBorder());

borderNoBorder.Append(new DiagonalBorder());

var borderThin = new Border();

var leftBorder = new LeftBorder() { Style = BorderStyleValues.Thin };

leftBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color() { Indexed = 64U });

var rightBorder = new RightBorder()

{

Style = BorderStyleValues.Thin

};

rightBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color() { Indexed = 64U });

var topBorder = new TopBorder() { Style = BorderStyleValues.Thin };

topBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color() { Indexed = 64U });

var bottomBorder = new BottomBorder()

{

Style = BorderStyleValues.Thin

};

bottomBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color() { Indexed = 64U });

borderThin.Append(leftBorder);

borderThin.Append(rightBorder);

borderThin.Append(topBorder);

borderThin.Append(bottomBorder);

borderThin.Append(new DiagonalBorder());

borders.Append(borderNoBorder);

borders.Append(borderThin);

//формирование CellFormat из комбинаций шрифтов, заливок и т. д.

var cellStyleFormats = new CellStyleFormats() { Count = 1U };

var cellFormatStyle = new CellFormat()

{

NumberFormatId = 0U,

FontId = 0U,

FillId = 0U,

BorderId = 0U

};

cellStyleFormats.Append(cellFormatStyle);

var cellFormats = new CellFormats() { Count = 3U };

var cellFormatFont = new CellFormat()

{

NumberFormatId = 0U,

FontId = 0U,

FillId = 0U,

BorderId = 0U,

FormatId = 0U,

ApplyFont = true

};

var cellFormatFontAndBorder = new CellFormat()

{

NumberFormatId = 0U,

FontId = 0U,

FillId = 0U,

BorderId = 1U,

FormatId = 0U,

ApplyFont = true,

ApplyBorder = true

};

var cellFormatTitle = new CellFormat()

{

NumberFormatId = 0U,

FontId = 1U,

FillId = 0U,

BorderId = 0U,

FormatId = 0U,

Alignment = new Alignment()

{

Vertical = VerticalAlignmentValues.Center,

WrapText = true,

Horizontal = HorizontalAlignmentValues.Center

},

ApplyFont = true

};

//по итогу создали 3 стиля

cellFormats.Append(cellFormatFont);

cellFormats.Append(cellFormatFontAndBorder);

cellFormats.Append(cellFormatTitle);

var cellStyles = new CellStyles() { Count = 1U };

cellStyles.Append(new CellStyle()

{

Name = "Normal",

FormatId = 0U,

BuiltinId = 0U

});

var differentialFormats = new DocumentFormat.OpenXml.Office2013.Excel.DifferentialFormats() { Count = 0U };

var tableStyles = new TableStyles()

{

Count = 0U,

DefaultTableStyle = "TableStyleMedium2",

DefaultPivotStyle = "PivotStyleLight16"

};

var stylesheetExtensionList = new StylesheetExtensionList();

var stylesheetExtension1 = new StylesheetExtension()

{

Uri = "{EB79DEF2-80B8-43e5-95BD-54CBDDF9020C}"

};

stylesheetExtension1.AddNamespaceDeclaration("x14", "http://schemas.microsoft.com/office/spreadsheetml/2009/9/main");

stylesheetExtension1.Append(new SlicerStyles()

{

DefaultSlicerStyle = "SlicerStyleLight1"

});

var stylesheetExtension2 = new StylesheetExtension()

{

Uri = "{9260A510-F301-46a8-8635-F512D64BE5F5}"

};

stylesheetExtension2.AddNamespaceDeclaration("x15", "http://schemas.microsoft.com/office/spreadsheetml/2010/11/main");

stylesheetExtension2.Append(new TimelineStyles()

{

DefaultTimelineStyle = "TimeSlicerStyleLight1"

});

stylesheetExtensionList.Append(stylesheetExtension1);

stylesheetExtensionList.Append(stylesheetExtension2);

sp.Stylesheet.Append(fonts);

sp.Stylesheet.Append(fills);

sp.Stylesheet.Append(borders);

sp.Stylesheet.Append(cellStyleFormats);

sp.Stylesheet.Append(cellFormats);

sp.Stylesheet.Append(cellStyles);

sp.Stylesheet.Append(differentialFormats);

sp.Stylesheet.Append(tableStyles);

sp.Stylesheet.Append(stylesheetExtensionList);

}

//Получение номера стиля (одного из 3-х нами созданных) из типа

private static uint GetStyleValue(ExcelStyleInfoType styleInfo)

{

return styleInfo switch

{

ExcelStyleInfoType.Title => 2U,

ExcelStyleInfoType.TextWithBorder => 1U,

ExcelStyleInfoType.Text => 0U,

\_ => 0U,

};

}

protected override void CreateExcel(ExcelInfo info)

{

//создаём документ Excel

\_spreadsheetDocument = SpreadsheetDocument.Create(info.FileName, SpreadsheetDocumentType.Workbook);

// Создаем книгу (в ней хранятся листы)

var workbookpart = \_spreadsheetDocument.AddWorkbookPart();

workbookpart.Workbook = new Workbook();

CreateStyles(workbookpart);

// Получаем/создаем хранилище текстов для книги

\_shareStringPart = \_spreadsheetDocument.WorkbookPart!.GetPartsOfType<SharedStringTablePart>().Any()

? \_spreadsheetDocument.WorkbookPart.GetPartsOfType<SharedStringTablePart>().First()

: \_spreadsheetDocument.WorkbookPart.AddNewPart<SharedStringTablePart>();

// Создаем SharedStringTable, если его нет

if (\_shareStringPart.SharedStringTable == null)

{

\_shareStringPart.SharedStringTable = new SharedStringTable();

}

// Создаем лист в книгу

var worksheetPart = workbookpart.AddNewPart<WorksheetPart>();

worksheetPart.Worksheet = new Worksheet(new SheetData());

// Добавляем лист в книгу

var sheets = \_spreadsheetDocument.WorkbookPart.Workbook.AppendChild(new Sheets());

var sheet = new Sheet()

{

Id = \_spreadsheetDocument.WorkbookPart.GetIdOfPart(worksheetPart),

SheetId = 1,

Name = "Лист"

};

sheets.Append(sheet);

\_worksheet = worksheetPart.Worksheet;

}

//метод вставки в лист книги

protected override void InsertCellInWorksheet(ExcelCellParameters excelParams)

{

if (\_worksheet == null || \_shareStringPart == null)

{

return;

}

var sheetData = \_worksheet.GetFirstChild<SheetData>();

if (sheetData == null)

{

return;

}

// Ищем строку, либо добавляем ее

Row row;

if (sheetData.Elements<Row>().Where(r => r.RowIndex! == excelParams.RowIndex).Any())

{

row = sheetData.Elements<Row>().Where(r => r.RowIndex! == excelParams.RowIndex).First();

}

else

{

row = new Row() { RowIndex = excelParams.RowIndex };

sheetData.Append(row);

}

// Ищем нужную ячейку

Cell cell;

if (row.Elements<Cell>().Where(c => c.CellReference!.Value == excelParams.CellReference).Any())

{

cell = row.Elements<Cell>().Where(c => c.CellReference!.Value == excelParams.CellReference).First();

}

else

{

// Все ячейки должны быть последовательно друг за другом расположены

// нужно определить, после какой вставлять

Cell? refCell = null;

foreach (Cell rowCell in row.Elements<Cell>())

{

if (string.Compare(rowCell.CellReference!.Value, excelParams.CellReference, true) > 0)

{

refCell = rowCell;

break;

}

}

var newCell = new Cell()

{

CellReference = excelParams.CellReference

};

row.InsertBefore(newCell, refCell);

cell = newCell;

}

// вставляем новый текст

\_shareStringPart.SharedStringTable.AppendChild(new SharedStringItem(new Text(excelParams.Text)));

\_shareStringPart.SharedStringTable.Save();

cell.CellValue = new CellValue((\_shareStringPart.SharedStringTable.Elements<SharedStringItem>().Count() - 1).ToString());

cell.DataType = new EnumValue<CellValues>(CellValues.SharedString);

cell.StyleIndex = GetStyleValue(excelParams.StyleInfo);

}

//метод объединения ячеек

protected override void MergeCells(ExcelMergeParameters excelParams)

{

if (\_worksheet == null)

{

return;

}

MergeCells mergeCells;

if (\_worksheet.Elements<MergeCells>().Any())

{

mergeCells = \_worksheet.Elements<MergeCells>().First();

}

else

{

mergeCells = new MergeCells();

if (\_worksheet.Elements<CustomSheetView>().Any())

{

\_worksheet.InsertAfter(mergeCells, \_worksheet.Elements<CustomSheetView>().First());

}

else

{

\_worksheet.InsertAfter(mergeCells, \_worksheet.Elements<SheetData>().First());

}

}

var mergeCell = new MergeCell()

{

Reference = new StringValue(excelParams.Merge)

};

mergeCells.Append(mergeCell);

}

protected override void SaveExcel(ExcelInfo info)

{

if (\_spreadsheetDocument == null)

{

return;

}

\_spreadsheetDocument.WorkbookPart!.Workbook.Save();

\_spreadsheetDocument.Close();

}

}

}

SaveToPdf.cs

using BankBusinessLogic.OfficePackage.HelperEnums;

using BankBusinessLogic.OfficePackage.HelperModels;

using MigraDoc.DocumentObjectModel;

using MigraDoc.DocumentObjectModel.Tables;

using MigraDoc.Rendering;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankBusinessLogic.OfficePackage.Implements

{

//реализация астрактного класса создания pdf документа

public class SaveToPdf : AbstractSaveToPdf

{

private Document? \_document;

private Section? \_section;

private Table? \_table;

//преобразование необходимого типа выравнивания в соотвествующее выравнивание в MigraDoc

private static ParagraphAlignment GetParagraphAlignment(PdfParagraphAlignmentType type)

{

return type switch

{

PdfParagraphAlignmentType.Center => ParagraphAlignment.Center,

PdfParagraphAlignmentType.Left => ParagraphAlignment.Left,

PdfParagraphAlignmentType.Right => ParagraphAlignment.Right,

\_ => ParagraphAlignment.Justify,

};

}

//Создание стилей для документа

private static void DefineStyles(Document document)

{

var style = document.Styles["Normal"];

style.Font.Name = "Times New Roman";

style.Font.Size = 14;

style = document.Styles.AddStyle("NormalTitle", "Normal");

style.Font.Bold = true;

}

protected override void CreatePdf(PdfInfo info)

{

//создаём документ

\_document = new Document();

//передаём для него стили

DefineStyles(\_document);

//получение первой секции документа

\_section = \_document.AddSection();

}

protected override void CreateParagraph(PdfParagraph pdfParagraph)

{

if (\_section == null)

{

return;

}

var paragraph = \_section.AddParagraph(pdfParagraph.Text);

paragraph.Format.SpaceAfter = "1cm";

paragraph.Format.Alignment = GetParagraphAlignment(pdfParagraph.ParagraphAlignment);

paragraph.Style = pdfParagraph.Style;

}

protected override void CreateTable(List<string> columns)

{

if (\_document == null)

{

return;

}

//добавляем таблицу в документ как последнюю секцию (?)

\_table = \_document.LastSection.AddTable();

foreach (var elem in columns)

{

\_table.AddColumn(elem);

}

}

protected override void CreateRow(PdfRowParameters rowParameters)

{

if (\_table == null)

{

return;

}

//добавление строки в таблицу

var row = \_table.AddRow();

for (int i = 0; i < rowParameters.Texts.Count; ++i)

{

//ячейка добавляется добавлением параграфа

row.Cells[i].AddParagraph(rowParameters.Texts[i]);

if (!string.IsNullOrEmpty(rowParameters.Style))

{

row.Cells[i].Style = rowParameters.Style;

}

Unit borderWidth = 0.5;

row.Cells[i].Borders.Left.Width = borderWidth;

row.Cells[i].Borders.Right.Width = borderWidth;

row.Cells[i].Borders.Top.Width = borderWidth;

row.Cells[i].Borders.Bottom.Width = borderWidth;

row.Cells[i].Format.Alignment = GetParagraphAlignment(rowParameters.ParagraphAlignment);

row.Cells[i].VerticalAlignment = VerticalAlignment.Center;

}

}

protected override void SavePdf(PdfInfo info)

{

var renderer = new PdfDocumentRenderer(true)

{

Document = \_document

};

renderer.RenderDocument();

renderer.PdfDocument.Save(info.FileName);

}

}

}

SaveToPdf.cs

using BankBusinessLogic.OfficePackage.HelperEnums;

using BankBusinessLogic.OfficePackage.HelperModels;

using MigraDoc.DocumentObjectModel;

using MigraDoc.DocumentObjectModel.Tables;

using MigraDoc.Rendering;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankBusinessLogic.OfficePackage.Implements

{

//реализация астрактного класса создания pdf документа

public class SaveToPdf : AbstractSaveToPdf

{

private Document? \_document;

private Section? \_section;

private Table? \_table;

//преобразование необходимого типа выравнивания в соотвествующее выравнивание в MigraDoc

private static ParagraphAlignment GetParagraphAlignment(PdfParagraphAlignmentType type)

{

return type switch

{

PdfParagraphAlignmentType.Center => ParagraphAlignment.Center,

PdfParagraphAlignmentType.Left => ParagraphAlignment.Left,

PdfParagraphAlignmentType.Right => ParagraphAlignment.Right,

\_ => ParagraphAlignment.Justify,

};

}

//Создание стилей для документа

private static void DefineStyles(Document document)

{

var style = document.Styles["Normal"];

style.Font.Name = "Times New Roman";

style.Font.Size = 14;

style = document.Styles.AddStyle("NormalTitle", "Normal");

style.Font.Bold = true;

}

protected override void CreatePdf(PdfInfo info)

{

//создаём документ

\_document = new Document();

//передаём для него стили

DefineStyles(\_document);

//получение первой секции документа

\_section = \_document.AddSection();

}

protected override void CreateParagraph(PdfParagraph pdfParagraph)

{

if (\_section == null)

{

return;

}

var paragraph = \_section.AddParagraph(pdfParagraph.Text);

paragraph.Format.SpaceAfter = "1cm";

paragraph.Format.Alignment = GetParagraphAlignment(pdfParagraph.ParagraphAlignment);

paragraph.Style = pdfParagraph.Style;

}

protected override void CreateTable(List<string> columns)

{

if (\_document == null)

{

return;

}

//добавляем таблицу в документ как последнюю секцию (?)

\_table = \_document.LastSection.AddTable();

foreach (var elem in columns)

{

\_table.AddColumn(elem);

}

}

protected override void CreateRow(PdfRowParameters rowParameters)

{

if (\_table == null)

{

return;

}

//добавление строки в таблицу

var row = \_table.AddRow();

for (int i = 0; i < rowParameters.Texts.Count; ++i)

{

//ячейка добавляется добавлением параграфа

row.Cells[i].AddParagraph(rowParameters.Texts[i]);

if (!string.IsNullOrEmpty(rowParameters.Style))

{

row.Cells[i].Style = rowParameters.Style;

}

Unit borderWidth = 0.5;

row.Cells[i].Borders.Left.Width = borderWidth;

row.Cells[i].Borders.Right.Width = borderWidth;

row.Cells[i].Borders.Top.Width = borderWidth;

row.Cells[i].Borders.Bottom.Width = borderWidth;

row.Cells[i].Format.Alignment = GetParagraphAlignment(rowParameters.ParagraphAlignment);

row.Cells[i].VerticalAlignment = VerticalAlignment.Center;

}

}

protected override void SavePdf(PdfInfo info)

{

var renderer = new PdfDocumentRenderer(true)

{

Document = \_document

};

renderer.RenderDocument();

renderer.PdfDocument.Save(info.FileName);

}

}

}

SaveToWord.cs

using BankBusinessLogic.OfficePackage.HelperEnums;

using BankBusinessLogic.OfficePackage.HelperModels;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

namespace BankBusinessLogic.OfficePackage.Implements

{

//реализация абстрактного класса сохранения в word

public class SaveToWord : AbstractSaveToWord

{

private WordprocessingDocument? \_wordDocument;

private Body? \_docBody;

//Получение типа выравнивания

private static JustificationValues GetJustificationValues(WordJustificationType type)

{

//выравнивание слева будет в том случае, если передаётся неизвестный тип выравнивания

return type switch

{

WordJustificationType.Both => JustificationValues.Both,

WordJustificationType.Center => JustificationValues.Center,

\_ => JustificationValues.Left,

};

}

//Настройки страницы

private static SectionProperties CreateSectionProperties()

{

var properties = new SectionProperties();

//прописываем портретную ориентацию

var pageSize = new PageSize

{

Orient = PageOrientationValues.Portrait

};

properties.AppendChild(pageSize);

return properties;

}

//Задание форматирования для абзаца

private static ParagraphProperties? CreateParagraphProperties(WordTextProperties? paragraphProperties)

{

if (paragraphProperties == null)

{

return null;

}

var properties = new ParagraphProperties();

//вытаскиваем выравнивание текста

properties.AppendChild(new Justification()

{

Val = GetJustificationValues(paragraphProperties.JustificationType)

});

properties.AppendChild(new SpacingBetweenLines

{

LineRule = LineSpacingRuleValues.Auto

});

properties.AppendChild(new Indentation());

var paragraphMarkRunProperties = new ParagraphMarkRunProperties();

if (!string.IsNullOrEmpty(paragraphProperties.Size))

{

paragraphMarkRunProperties.AppendChild(new FontSize

{

Val = paragraphProperties.Size

});

}

properties.AppendChild(paragraphMarkRunProperties);

return properties;

}

protected override void CreateWord(WordInfo info)

{

//создаём документ word

\_wordDocument = WordprocessingDocument.Create(info.FileName, WordprocessingDocumentType.Document);

//вытаскиваем главную часть из вордовского документа

MainDocumentPart mainPart = \_wordDocument.AddMainDocumentPart();

mainPart.Document = new Document();

//генерируем тело основной части документа

\_docBody = mainPart.Document.AppendChild(new Body());

}

protected override void CreateParagraph(WordParagraph paragraph)

{

//проверка на то, был ли вызван WordprocessingDocument.Create (создался ли документ) и есть ли вообще параграф для вставки

if (\_docBody == null || paragraph == null)

{

return;

}

var docParagraph = new Paragraph();

//добавляем свойства параграфа

docParagraph.AppendChild(CreateParagraphProperties(paragraph.TextProperties));

//вставляем блоки текста (их называют Run)

foreach (var run in paragraph.Texts)

{

var docRun = new Run();

var properties = new RunProperties();

//задание свойств текста - размер и жирность

properties.AppendChild(new FontSize { Val = run.Item2.Size });

if (run.Item2.Bold)

{

properties.AppendChild(new Bold());

}

docRun.AppendChild(properties);

docRun.AppendChild(new Text

{

Text = run.Item1,

Space = SpaceProcessingModeValues.Preserve

});

docParagraph.AppendChild(docRun);

}

\_docBody.AppendChild(docParagraph);

}

//метод, отвечающий за создание таблицы

protected override void CreateTable(WordParagraph paragraph)

{

if (\_docBody == null || paragraph == null)

{

return;

}

Table table = new Table();

var tableProp = new TableProperties();

tableProp.AppendChild(new TableLayout { Type = TableLayoutValues.Fixed });

tableProp.AppendChild(new TableBorders(

new TopBorder() { Val = new EnumValue<BorderValues>(BorderValues.Single), Size = 4 },

new LeftBorder() { Val = new EnumValue<BorderValues>(BorderValues.Single), Size = 4 },

new RightBorder() { Val = new EnumValue<BorderValues>(BorderValues.Single), Size = 4 },

new BottomBorder() { Val = new EnumValue<BorderValues>(BorderValues.Single), Size = 4 },

new InsideHorizontalBorder() { Val = new EnumValue<BorderValues>(BorderValues.Single), Size = 4 },

new InsideVerticalBorder() { Val = new EnumValue<BorderValues>(BorderValues.Single), Size = 4 }

));

tableProp.AppendChild(new TableWidth { Type = TableWidthUnitValues.Auto });

table.AppendChild(tableProp);

TableGrid tableGrid = new TableGrid();

for (int j = 0; j < paragraph.RowTexts[0].Count; ++j)

{

tableGrid.AppendChild(new GridColumn() { Width = "2500" });

}

table.AppendChild(tableGrid);

for (int i = 0; i < paragraph.RowTexts.Count; ++i)

{

TableRow docRow = new TableRow();

for (int j = 0; j < paragraph.RowTexts[i].Count; ++j)

{

var docParagraph = new Paragraph();

docParagraph.AppendChild(CreateParagraphProperties(paragraph.RowTexts[i][j].Item2));

var docRun = new Run();

var properties = new RunProperties();

properties.AppendChild(new FontSize { Val = paragraph.RowTexts[i][j].Item2.Size });

if (paragraph.RowTexts[i][j].Item2.Bold)

{

properties.AppendChild(new Bold());

}

docRun.AppendChild(properties);

docRun.AppendChild(new Text { Text = paragraph.RowTexts[i][j].Item1, Space = SpaceProcessingModeValues.Preserve });

docParagraph.AppendChild(docRun);

TableCell docCell = new TableCell();

docCell.AppendChild(docParagraph);

docRow.AppendChild(docCell);

}

table.AppendChild(docRow);

}

\_docBody.AppendChild(table);

}

//метод сохранения документа

protected override void SaveWord(WordInfo info)

{

if (\_docBody == null || \_wordDocument == null)

{

return;

}

//вставляем информацию по секциям (смотри, что является входным параметром)

\_docBody.AppendChild(CreateSectionProperties());

//сохраняем документ

\_wordDocument.MainDocumentPart!.Document.Save();

\_wordDocument.Close();

}

}

}

Проект BankContracts

Папка BindingModels

Папка Cashier

AccountBindingModel.cs

using BankDataModels.Models.Cashier;

namespace BankContracts.BindingModels.Cashier

{

public class AccountBindingModel : IAccountModel

{

public int Id { get; set; }

public int CashierId { get; set; }

public int ClientId { get; set; }

public string AccountNumber { get; set; } = string.Empty;

public string PasswordAccount { get; set; } = string.Empty;

public double Balance { get; set; }

public DateTime DateOpen { get; set; } = DateTime.Now;

}

}

CashWithdrawalBindingModel.cs

using BankDataModels.Models.Cashier;

namespace BankContracts.BindingModels.Cashier

{

public class CashWithdrawalBindingModel : ICashWithdrawalModel

{

public int Id { get; set; }

public int DebitingId { get; set; }

public int AccountId { get; set; }

public int CashierId { get; set; }

public int Sum { get; set; }

public DateTime DateOperation { get; set; } = DateTime.Now;

}

}

CashierBindingModel.cs

using BankDataModels.Models.Cashier;

namespace BankContracts.BindingModels.Cashier

{

public class CashierBindingModel : ICashierModel

{

public int Id { get; set; }

public string Password { get; set; } = string.Empty;

public string Name { get; set; } = string.Empty;

public string Surname { get; set; } = string.Empty;

public string Patronymic { get; set; } = string.Empty;

public string Email { get; set; } = string.Empty;

public string Telephone { get; set; } = string.Empty;

}

}

MoneyTransferBindingModel.cs

using BankDataModels.Models.Cashier;

namespace BankContracts.BindingModels.Cashier

{

public class MoneyTransferBindingModel : IMoneyTransferModel

{

public int Id { get; set; }

public int Sum { get; set; }

public int? AccountSenderId { get; set; }

public int AccountPayeeId { get; set; }

public DateTime DateOperation { get; set; } = DateTime.Now;

public int? CreditingId { get; set; }

public int CashierId { get; set; }

}

}

Папка Client

CardBindingModel.cs

using BankDataModels.Models.Client;

namespace BankContracts.BindingModels.Client

{

public class CardBindingModel : ICardModel

{

public int Id { get; set; }

public int ClientID { get; set; }

public int AccountId { get; set; }

public string Number { get; set; } = string.Empty;

public string CVC { get; set; } = string.Empty;

public DateTime Period { get; set; } = DateTime.Now;

}

}

ClientBindingModel.cs

using BankDataModels.Models.Client;

namespace BankContracts.BindingModels.Client

{

public class ClientBindingModel : IClientModel

{

public int Id { get; set; }

public string Password { get; set; } = string.Empty;

public string Name { get; set; } = string.Empty;

public string Surname { get; set; } = string.Empty;

public string Patronymic { get; set; } = string.Empty;

public string Email { get; set; } = string.Empty;

public string Telephone { get; set; } = string.Empty;

}

}

CreditingBindingModel.cs

using BankDataModels.Enums;

using BankDataModels.Models.Client;

namespace BankContracts.BindingModels.Client

{

public class CreditingBindingModel : ICreditingModel

{

public int Id { get; set; }

public int CardId { get; set; }

public int Sum { get; set; }

public DateTime DateOpen { get; set; } = DateTime.Now;

public DateTime? DateClose { get; set; }

public StatusEnum Status { get; set; }

}

}

DebitingBindingModel.cs

using BankDataModels.Enums;

using BankDataModels.Models.Client;

namespace BankContracts.BindingModels.Client

{

public class DebitingBindingModel : IDebitingModel

{

public int Id { get; set; }

public int CardId { get; set; }

public int Sum { get; set; }

public DateTime DateOpen { get; set; } = DateTime.Now;

public DateTime? DateClose { get; set; }

public StatusEnum Status { get; set; } = StatusEnum.Открыта;

}

}

Папка Messages

MailConfigBindingModel.cs

namespace BankContracts.BindingModels.Messages

{

//один из двух классов для передачи данных по почте

public class MailConfigBindingModel

{

public string MailLogin { get; set; } = string.Empty;

public string MailPassword { get; set; } = string.Empty;

public string SmtpClientHost { get; set; } = string.Empty;

public int SmtpClientPort { get; set; }

//можно без них?

public string PopHost { get; set; } = string.Empty;

//можно без них?

public int PopPort { get; set; }

}

}

MailSendInfoBindingModel.cs

using BankDataModels.Enums;

namespace BankContracts.BindingModels.Messages

{

//один из двух классов для обмена информацией по почте

public class MailSendInfoBindingModel

{

public string MailAddress { get; set; } = string.Empty;

public string Subject { get; set; } = string.Empty;

public string Text { get; set; } = string.Empty;

//для отправки pdf

public byte[] File { get; set; } = Array.Empty<byte>();

public MailsEnum Role { get; set; }

public TypeDocEnum TypeDoc { get; set; }

}

}

MessageInfoBindingModel.cs

using BankDataModels.Models;

namespace BankContracts.BindingModels.Messages

{

public class MessageInfoBindingModel : IMessageInfoModel

{

public int Id { get; set; }

public string MessageId { get; set; } = string.Empty;

public int? ClientId { get; set; }

public string SenderName { get; set; } = string.Empty;

public DateTime DateDelivery { get; set; } = DateTime.Now;

public string Subject { get; set; } = string.Empty;

public string Body { get; set; } = string.Empty;

public bool IsRead { get; set; } = false;

public string? Answer { get; set; } = string.Empty;

}

}

Папка Reports

ReportBindingModel.cs

using BankDataModels.Enums;

namespace BankContracts.BindingModels.Reports

{

public class ReportBindingModel

{

public string FileName { get; set; } = string.Empty;

public int? ClientId { get; set; }

public int? AccountId { get; set; }

public List<int>? CardList { get; set; }

public string? ClientFullName { get; set; } = string.Empty;

public DateTime? DateFrom { get; set; }

public DateTime? DateTo { get; set; }

public MailsEnum Role { get; set; }

public string? Email { get; set; }

}

}

ReportSupportBindingModel.cs

namespace BankContracts.BindingModels.Reports

{

//вспомогательная модель для передачи DateTime при формировании отчёта

public class ReportSupportBindingModel

{

public int? ClientId { get; set; }

public int? AccountId { get; set; }

public DateTime? DateFrom { get; set; }

public DateTime? DateTo { get; set; }

//для Excel отчёта клиента

public List<int>? CardList { get; set; }

public string? Email { get; set; }

}

}

Папка BusinessLogicsContracts

Папка Cashier

IAccountLogic.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.ViewModels.Cashier.Diagram;

using BankContracts.ViewModels.Cashier.ViewModels;

namespace BankContracts.BusinessLogicsContracts.Cashier

{

public interface IAccountLogic

{

List<AccountViewModel>? ReadList(AccountSearchModel? model);

AccountViewModel? ReadElement(AccountSearchModel model);

bool ChangeBalance(AccountSearchModel? model, int sum);

bool Create(AccountBindingModel model);

bool Update(AccountBindingModel model);

bool Delete(AccountBindingModel model);

public List<CashierDiagramElementsViewModel> GetMonthInfo(int AccountId);

}

}

ICashWithdrawalLogic.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.ViewModels.Cashier.ViewModels;

namespace BankContracts.BusinessLogicsContracts.Cashier

{

public interface ICashWithdrawalLogic

{

List<CashWithdrawalViewModel>? ReadList(CashWithdrawalSearchModel? model);

CashWithdrawalViewModel? ReadElement(CashWithdrawalSearchModel model);

bool Create(CashWithdrawalBindingModel model, bool flag);

bool Update(CashWithdrawalBindingModel model);

bool Delete(CashWithdrawalBindingModel model);

}

}

ICashierLogic.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.ViewModels.Cashier.ViewModels;

namespace BankContracts.BusinessLogicsContracts.Cashier

{

public interface ICashierLogic

{

List<CashierViewModel>? ReadList(CashierSearchModel? model);

CashierViewModel? ReadElement(CashierSearchModel model);

bool Create(CashierBindingModel model);

bool Update(CashierBindingModel model);

bool Delete(CashierBindingModel model);

}

}

IMoneyTransferLogic.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.ViewModels.Cashier.ViewModels;

namespace BankContracts.BusinessLogicsContracts.Cashier

{

public interface IMoneyTransferLogic

{

List<MoneyTransferViewModel>? ReadList(MoneyTransferSearchModel? model);

MoneyTransferViewModel? ReadElement(MoneyTransferSearchModel model);

bool Create(MoneyTransferBindingModel model);

bool Update(MoneyTransferBindingModel model);

bool Delete(MoneyTransferBindingModel model);

}

}

Папка Client

ICardLogic.cs

using BankContracts.BindingModels.Client;

using BankContracts.SearchModels.Client;

using BankContracts.ViewModels.Client.Diagram;

using BankContracts.ViewModels.Client.ViewModels;

namespace BankContracts.BusinessLogicsContracts.Client

{

public interface ICardLogic

{

List<CardViewModel>? ReadList(CardSearchModel? model);

CardViewModel? ReadElement(CardSearchModel model);

bool Create(CardBindingModel model);

bool Update(CardBindingModel model);

bool Delete(CardBindingModel model);

public List<ClientDiagramElementsViewModel> GetMonthInfo(int CardId);

}

}

IClientLogic.cs

using BankContracts.BindingModels.Client;

using BankContracts.SearchModels.Client;

using BankContracts.ViewModels.Client.ViewModels;

namespace BankContracts.BusinessLogicsContracts.Client

{

public interface IClientLogic

{

List<ClientViewModel>? ReadList(ClientSearchModel? model);

ClientViewModel? ReadElement(ClientSearchModel model);

bool Create(ClientBindingModel model);

bool Update(ClientBindingModel model);

bool Delete(ClientBindingModel model);

}

}

ICreditingLogic.cs

using BankContracts.BindingModels.Client;

using BankContracts.SearchModels.Client;

using BankContracts.ViewModels.Client.ViewModels;

namespace BankContracts.BusinessLogicsContracts.Client

{

public interface ICreditingLogic

{

List<CreditingViewModel>? ReadList(CreditingSearchModel? model);

CreditingViewModel? ReadElement(CreditingSearchModel model);

bool Create(CreditingBindingModel model);

bool Update(CreditingBindingModel model);

bool Delete(CreditingBindingModel model);

}

}

IDebitingLogic.cs

using BankContracts.BindingModels.Client;

using BankContracts.SearchModels.Client;

using BankContracts.ViewModels.Client.ViewModels;

namespace BankContracts.BusinessLogicsContracts.Client

{

public interface IDebitingLogic

{

List<DebitingViewModel>? ReadList(DebitingSearchModel? model);

DebitingViewModel? ReadElement(DebitingSearchModel model);

bool Create(DebitingBindingModel model);

bool Update(DebitingBindingModel model);

bool Delete(DebitingBindingModel model);

}

}

IReportCashierLogic.cs

using BankContracts.BindingModels.Reports;

using BankContracts.ViewModels.Reports;

using BankContracts.ViewModels.Reports.Cashier;

namespace BankContracts.BusinessLogicsContracts.Reports

{

public interface IReportCashierLogic

{

List<ReportCashierViewModel>? GetMoneyTransfers(ReportBindingModel model);

List<ReportCashierViewModel>? GetCashWithrawals(ReportBindingModel model);

//Сохранение отчёта по счетам в файл-Word

void SaveAccountsToWordFile(ReportBindingModel model);

//Сохранение отчёта по счетам в файл-Excel

void SaveAccountsToExcelFile(ReportBindingModel model);

//Сохранение отчёта по счетам в файл-Pdf

ReportCashierViewModelForHTML SaveAccountsToPdfFile(ReportBindingModel model);

}

}

IReportClientLogic.cs

using BankContracts.BindingModels.Reports;

using BankContracts.ViewModels.Reports;

using BankContracts.ViewModels.Reports.Client;

using BankDataModels.Enums;

namespace BankContracts.BusinessLogicsContracts.Reports

{

public interface IReportClientLogic

{

List<ReportClientViewModel>? GetCrediting(ReportBindingModel model);

List<ReportClientViewModel>? GetDebiting(ReportBindingModel model);

//Сохранение отчёта по картам в файл-Word

void SaveToWordFile(ReportBindingModel model, OfficeOperationEnum operationEnum);

//Сохранение отчёта по картам в файл-Excel

void SaveToExcelFile(ReportBindingModel model, OfficeOperationEnum operationEnum);

//Сохранение отчёта по картам в файл-Pdf

ReportClientViewModelForHTML SaveClientReportToPdfFile(ReportBindingModel model);

}

}

Папка SearchModels

Папка Cashier

AccountSearchModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.SearchModels.Cashier

{

public class AccountSearchModel

{

public int? Id { get; set; }

public string? AccountNumber { get; set; } = string.Empty;

public int? CashierId { get; set; }

public int? ClientId { get; set; }

public string? PasswordAccount { get; set; } = string.Empty;

public double? Balance { get; set; }

public DateTime? DateOpen { get; set; }

}

}

CashWithdrawalSearchModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.SearchModels.Cashier

{

public class CashWithdrawalSearchModel

{

public int? Id { get; set; }

public int? DebitingId { get; set; }

public int? AccountId { get; set; }

public int? CashierId { get; set; }

public int? ClientId { get; set; }

public int? Sum { get; set; }

public DateTime? DateFrom { get; set; }

public DateTime? DateTo { get; set; }

}

}

CashierSearchModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.SearchModels.Cashier

{

public class CashierSearchModel

{

public int? Id { get; set; }

public string? Name { get; set; }

public string? Surname { get; set; }

public string? Patronymic { get; set; }

public string? Email { get; set; }

public string? Password { get; set; }

public string? Telephone { get; set; }

}

}

MoneyTransferSearchModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.SearchModels.Cashier

{

public class MoneyTransferSearchModel

{

public int? Id { get; set; }

public int? Sum { get; set; }

public int? ClientId { get; set; }

public int? CashierId { get; set; }

public int? AccountSenderId { get; set; }

public int? AccountPayeeId { get; set; }

public DateTime? DateFrom { get; set; }

public DateTime? DateTo { get; set; }

}

}

Папка Client

CardSearchModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.SearchModels.Client

{

public class CardSearchModel

{

public int? Id { get; set; }

public int? ClientID { get; set; }

public int? AccountId { get; set; }

public string? Number { get; set; }

public string? CVC { get; set; }

public DateTime? Period { get; set; }

}

}

ClientSearchModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.SearchModels.Client

{

public class ClientSearchModel

{

public int? Id { get; set; }

public string? Name { get; set; }

public string? Surname { get; set; }

public string? Patronymic { get; set; }

public string? Email { get; set; }

public string? Password { get; set; }

}

}

CreditingSearchModel.cs

using BankDataModels.Enums;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.SearchModels.Client

{

public class CreditingSearchModel

{

public int? Id { get; set; }

public int? CardId { get; set; }

public int? Sum { get; set; }

public int? UserId { get; set; }

public DateTime? DateFrom { get; set; }

public DateTime? DateTo { get; set; }

public StatusEnum? Status { get; set; }

}

}

DebitingSearchModel.cs

using BankDataModels.Enums;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.SearchModels.Client

{

public class DebitingSearchModel

{

public int? Id { get; set; }

public int? CardId { get; set; }

public int? UserId { get; set; }

public int? Sum { get; set; }

public DateTime? DateFrom { get; set; }

public DateTime? DateTo { get; set; }

public StatusEnum? Status { get; set; }

}

}

Папка StoragesModels

Папка Cashier

IAccountStorage.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.ViewModels.Cashier.ViewModels;

namespace BankContracts.StoragesContracts.Cashier

{

public interface IAccountStorage

{

List<AccountViewModel> GetFullList();

List<AccountViewModel> GetFilteredList(AccountSearchModel model);

AccountViewModel? GetElement(AccountSearchModel model);

AccountViewModel? Insert(AccountBindingModel model);

AccountViewModel? Update(AccountBindingModel model);

AccountViewModel? Delete(AccountBindingModel model);

}

}

ICashWithdrawalStorage.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.ViewModels.Cashier.ViewModels;

namespace BankContracts.StoragesContracts.Cashier

{

public interface ICashWithdrawalStorage

{

List<CashWithdrawalViewModel> GetFullList();

List<CashWithdrawalViewModel> GetFilteredList(CashWithdrawalSearchModel model);

CashWithdrawalViewModel? GetElement(CashWithdrawalSearchModel model);

CashWithdrawalViewModel? Insert(CashWithdrawalBindingModel model);

CashWithdrawalViewModel? Update(CashWithdrawalBindingModel model);

CashWithdrawalViewModel? Delete(CashWithdrawalBindingModel model);

}

}

ICashierStorage.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.ViewModels.Cashier.ViewModels;

namespace BankContracts.StoragesContracts.Cashier

{

public interface ICashierStorage

{

List<CashierViewModel> GetFullList();

List<CashierViewModel> GetFilteredList(CashierSearchModel model);

CashierViewModel? GetElement(CashierSearchModel model);

CashierViewModel? Insert(CashierBindingModel model);

CashierViewModel? Update(CashierBindingModel model);

CashierViewModel? Delete(CashierBindingModel model);

}

}

IMoneyTransferStorage.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.ViewModels.Cashier.ViewModels;

namespace BankContracts.StoragesContracts.Cashier

{

public interface IMoneyTransferStorage

{

List<MoneyTransferViewModel> GetFullList();

List<MoneyTransferViewModel> GetFilteredList(MoneyTransferSearchModel model);

MoneyTransferViewModel? GetElement(MoneyTransferSearchModel model);

MoneyTransferViewModel? Insert(MoneyTransferBindingModel model);

MoneyTransferViewModel? Update(MoneyTransferBindingModel model);

MoneyTransferViewModel? Delete(MoneyTransferBindingModel model);

}

}

Папка Client

ICardStorage.cs

using BankContracts.BindingModels.Client;

using BankContracts.SearchModels.Client;

using BankContracts.ViewModels.Client.ViewModels;

namespace BankContracts.StoragesContracts.Client

{

public interface ICardStorage

{

List<CardViewModel> GetFullList();

List<CardViewModel> GetFilteredList(CardSearchModel model);

CardViewModel? GetElement(CardSearchModel model);

CardViewModel? Insert(CardBindingModel model);

CardViewModel? Update(CardBindingModel model);

CardViewModel? Delete(CardBindingModel model);

}

}

IClientStorage.cs

using BankContracts.BindingModels.Client;

using BankContracts.SearchModels.Client;

using BankContracts.ViewModels.Client.ViewModels;

namespace BankContracts.StoragesContracts.Client

{

public interface IClientStorage

{

List<ClientViewModel> GetFullList();

List<ClientViewModel> GetFilteredList(ClientSearchModel model);

ClientViewModel? GetElement(ClientSearchModel model);

ClientViewModel? Insert(ClientBindingModel model);

ClientViewModel? Update(ClientBindingModel model);

ClientViewModel? Delete(ClientBindingModel model);

}

}

ICreditingStorage.cs

using BankContracts.BindingModels.Client;

using BankContracts.SearchModels.Client;

using BankContracts.ViewModels.Client.ViewModels;

namespace BankContracts.StoragesContracts.Client

{

public interface ICreditingStorage

{

List<CreditingViewModel> GetFullList();

List<CreditingViewModel> GetFilteredList(CreditingSearchModel model);

CreditingViewModel? GetElement(CreditingSearchModel model);

CreditingViewModel? Insert(CreditingBindingModel model);

CreditingViewModel? Update(CreditingBindingModel model);

CreditingViewModel? Delete(CreditingBindingModel model);

}

}

IDebitingStorage.cs

using BankContracts.BindingModels.Client;

using BankContracts.SearchModels.Client;

using BankContracts.ViewModels.Client.ViewModels;

namespace BankContracts.StoragesContracts.Client

{

public interface IDebitingStorage

{

List<DebitingViewModel> GetFullList();

List<DebitingViewModel> GetFilteredList(DebitingSearchModel model);

DebitingViewModel? GetElement(DebitingSearchModel model);

DebitingViewModel? Insert(DebitingBindingModel model);

DebitingViewModel? Update(DebitingBindingModel model);

DebitingViewModel? Delete(DebitingBindingModel model);

}

}

Папка ViewModels

Папка Cashier

Папка Diagram

CashierDiagramElementsViewModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.ViewModels.Cashier.Diagram

{

public class CashierDiagramElementsViewModel

{

public string Name { get; set; } = "Column";

public int Value { get; set; } = 0;

}

}

CashierDiagramViewModel.cs

using BankContracts.ViewModels.Client.Diagram;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.ViewModels.Cashier.Diagram

{

public class CashierDiagramViewModel

{

public string DiagramName { get; set; } = "Diagram Name";

public List<CashierDiagramElementsViewModel> Elements { get; set; } = new();

}

}

Папка ViewModels

AccountViewModel.cs

using BankDataModels.Models.Cashier;

using System.ComponentModel;

namespace BankContracts.ViewModels.Cashier.ViewModels

{

public class AccountViewModel : IAccountModel

{

public int Id { get; set; }

public int CashierId { get; set; }

public int ClientId { get; set; }

[DisplayName("Номер счёта")]

public string AccountNumber { get; set; } = string.Empty;

[DisplayName("Имя")]

public string Name { get; set; } = string.Empty;

[DisplayName("Отчество")]

public string Patronymic { get; set; } = string.Empty;

public string PasswordAccount { get; set; } = string.Empty;

[DisplayName("Баланс")]

public double Balance { get; set; }

[DisplayName("Дата открытия")]

public DateTime DateOpen { get; set; } = DateTime.Now;

}

}

CashWithdrawalViewModel.cs

using BankDataModels.Enums;

using BankDataModels.Models.Cashier;

using System.ComponentModel;

namespace BankContracts.ViewModels.Cashier.ViewModels

{

public class CashWithdrawalViewModel : ICashWithdrawalModel

{

public int Id { get; set; }

public int DebitingId { get; set; }

public StatusEnum DebitingStatus { get; set; }

[DisplayName("Номер заявки на снятие средств")]

public int DebbitingNumber { get; set; }

public int AccountId { get; set; }

[DisplayName("Номер счёта")]

public string AccountNumber { get; set; } = string.Empty;

public int CashierId { get; set; }

[DisplayName("Кассир")]

public string SurmaneCashier { get; set; } = string.Empty;

[DisplayName("Сумма выданных наличных")]

public int Sum { get; set; }

[DisplayName("Дата операции")]

public DateTime DateOperation { get; set; } = DateTime.Now;

}

}

CashierViewModel.cs

using BankDataModels.Models.Cashier;

using System.ComponentModel;

namespace BankContracts.ViewModels.Cashier.ViewModels

{

public class CashierViewModel : ICashierModel

{

public int Id { get; set; }

public string Password { get; set; } = string.Empty;

[DisplayName("Имя")]

public string Name { get; set; } = string.Empty;

[DisplayName("Фамилия")]

public string Surname { get; set; } = string.Empty;

[DisplayName("Отчество")]

public string Patronymic { get; set; } = string.Empty;

[DisplayName("Телефон")]

public string Telephone { get; set; } = string.Empty;

[DisplayName("Почта")]

public string Email { get; set; } = string.Empty;

}

}

ClientSelectViewModel.cs

using BankDataModels.Models;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.ViewModels.Cashier.ViewModels

{

public class ClientSelectViewModel

{

public int Id { get; set; }

public string FullName { get; set; } = string.Empty;

}

}

MoneyTransferViewModel.cs

using BankDataModels.Models.Cashier;

using System.ComponentModel;

namespace BankContracts.ViewModels.Cashier.ViewModels

{

public class MoneyTransferViewModel : IMoneyTransferModel

{

public int Id { get; set; }

[DisplayName("Сумма перевода")]

public int Sum { get; set; }

public int? AccountSenderId { get; set; }

[DisplayName("Номер счёта отп.")]

public string AccountSenderNumber { get; set; } = string.Empty;

public int AccountPayeeId { get; set; }

[DisplayName("Номер счёта получ.")]

public string AccountPayeeNumber { get; set; } = string.Empty;

[DisplayName("Дата операции")]

public DateTime DateOperation { get; set; } = DateTime.Now;

[DisplayName("Номер операции на пополнение")]

public int? CreditingId { get; set; }

public int CashierId { get; set; }

[DisplayName("Фамилия кассира")]

public string CashierSurname { get; set; } = string.Empty;

}

}

Папка Client

Папка Diagram ClientDiagramElementsViewModel.cs

using BankDataModels.Models;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.ViewModels.Client.Diagram

{

public class ClientDiagramElementsViewModel

{

public string Name { get; set; } = "Column";

public int Value { get; set; } = 0;

}

}

ClientDiagramViewModel.cs

using BankDataModels.Models;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.ViewModels.Client.Diagram

{

public class ClientDiagramViewModel

{

public string DiagramName { get; set; } = "Diagram Name";

public List<ClientDiagramElementsViewModel> Elements { get; set; } = new();

}

}

Папка ViewModels

CardViewModel.cs

using BankDataModels.Models.Client;

using System.ComponentModel;

namespace BankContracts.ViewModels.Client.ViewModels

{

public class CardViewModel : ICardModel

{

public int Id { get; set; }

public int ClientID { get; set; }

[DisplayName("Фамилия клиента")]

public string? ClientSurname { get; set; }

public int AccountId { get; set; }

[DisplayName("Номер карты")]

public string Number { get; set; } = string.Empty;

[DisplayName("Баланс карты")]

public double Sum { get; set; }

public string CVC { get; set; } = string.Empty;

[DisplayName("Период действия")]

public DateTime Period { get; set; } = DateTime.Now;

}

}

ClientViewModel.cs

using BankDataModels.Models.Client;

using System.ComponentModel;

namespace BankContracts.ViewModels.Client.ViewModels

{

public class ClientViewModel : IClientModel

{

public int Id { get; set; }

public string Password { get; set; } = string.Empty;

[DisplayName("Имя")]

public string Name { get; set; } = string.Empty;

[DisplayName("Фамилия")]

public string Surname { get; set; } = string.Empty;

[DisplayName("Отчество")]

public string Patronymic { get; set; } = string.Empty;

[DisplayName("Почта")]

public string Email { get; set; } = string.Empty;

[DisplayName("Телефон")]

public string Telephone { get; set; } = string.Empty;

}

}

CreditingViewModel.cs

using BankDataModels.Enums;

using BankDataModels.Models.Client;

using System.ComponentModel;

namespace BankContracts.ViewModels.Client.ViewModels

{

public class CreditingViewModel : ICreditingModel

{

public int Id { get; set; }

public int CardId { get; set; }

[DisplayName("Номер карты")]

public string? CardNumber { get; set; }

[DisplayName("Сумма операции")]

public int Sum { get; set; }

[DisplayName("Дата открытия заявки")]

public DateTime DateOpen { get; set; } = DateTime.Now;

[DisplayName("Дата закрытия заявки")]

public DateTime? DateClose { get; set; }

[DisplayName("Статус заявки")]

public StatusEnum Status { get; set; }

}

}

DebitingViewModel.cs

using BankDataModels.Enums;

using BankDataModels.Models.Client;

using System.ComponentModel;

namespace BankContracts.ViewModels.Client.ViewModels

{

public class DebitingViewModel : IDebitingModel

{

public int Id { get; set; }

public int CardId { get; set; }

[DisplayName("Номер карты")]

public string CardNumber { get; set; } = string.Empty;

[DisplayName("Сумма операции")]

public int Sum { get; set; }

[DisplayName("Дата открытия заявки")]

public DateTime DateOpen { get; set; } = DateTime.Now;

[DisplayName("Дата закрытия заявки")]

public DateTime? DateClose { get; set; }

[DisplayName("Статус заявки")]

public StatusEnum Status { get; set; }

}

}

Папка Reports

FileViewModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.ViewModels.Reports

{

public class FileViewModel

{

public byte[] Bytes { get; set; } = Array.Empty<byte>();

public int[] Test { get; set; } = Array.Empty<int>();

public string StringBytes { get; set; } = string.Empty;

}

}

ReportCashierViewModelForHTML.cs

using BankContracts.ViewModels.Reports.Cashier;

namespace BankContracts.ViewModels.Reports

{

public class ReportCashierViewModelForHTML

{

//перечень переводов со счёта на счёт

public List<ReportCashierViewModel> ReportMoneyTransfer { get; set; } = new();

//перечень зачислений денежных средств на карту (т. е. на её счёт)

public List<ReportCashierViewModel> ReportCashWithdrawal { get; set; } = new();

}

}

ReportClientViewModelForHTML.cs

using BankContracts.ViewModels.Reports.Client;

namespace BankContracts.ViewModels.Reports

{

public class ReportClientViewModelForHTML

{

//перечень заказов за указанный период для вывода/сохранения

public List<ReportClientViewModel> ReportCrediting { get; set; } = new();

//перечень заказов за указанный период для вывода/сохранения

public List<ReportClientViewModel> ReportDebiting { get; set; } = new();

}

}

Папка Cashier

ReportCashierAccountsViewModel.cs

using BankDataModels.Enums;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.ViewModels.Reports.Cashier

{

public class ReportCashierAccountsViewModel

{

public int Sum { get; set; }

public string AccountSenderNumber { get; set; } = "---";

public string AccountPayeeNumber { get; set; } = "---";

public DateTime DateOperation { get; set; } = DateTime.Now;

public string CashierSurname { get; set; } = string.Empty;

public TypeOperationEnum typeOperation { get; set; }

}

}

ReportCashierViewModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.ViewModels.Reports.Cashier

{

public class ReportCashierViewModel

{

public int OperationId { get; set; }

public int DebitingId { get; set; }

public string AccountPayeeNumber { get; set; }

public string? AccountSenderNumber { get; set; }

public double SumOperation { get; set; }

public DateTime DateComplite { get; set; }

}

}

Client

CheckboxViewModel.cs

using BankDataModels.Models;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.ViewModels.Reports.Client

{

public class CheckboxViewModel

{

public int Id { get; set; }

public string LabelName { get; set; }

public bool IsChecked { get; set; }

}

}

ReportClientCardsViewModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.ViewModels.Reports.Client

{

public class ReportClientCardsViewModel

{

public List<CheckboxViewModel>? Cards { get; set; } = new();

public List<ReportViewModel>? Operations { get; set; } = new();

}

}

ReportClientViewModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.ViewModels.Reports.Client

{

public class ReportClientViewModel

{

public int OperationId { get; set; }

public string CardNumber { get; set; }

public double SumOperation { get; set; }

public DateTime? DateComplite { get; set; }

}

}

ReportViewModel.cs

using BankDataModels.Enums;

using BankDataModels.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankContracts.ViewModels.Reports.Client

{

public class ReportViewModel

{

public int Id { get; set; }

public int CardId { get; set; }

public string? CardNumber { get; set; }

public int Sum { get; set; }

public DateTime DateOpen { get; set; }

public DateTime? DateClose { get; set; }

public StatusEnum Status { get; set; }

public TypeOperationEnum TypeOperation { get; set; }

}

}

Проект BankDataModels

IId.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDataModels

{

public interface IId

{

int Id { get; }

}

}

Папка Enums

MailsEnum.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDataModels.Enums

{

public enum MailsEnum

{

Клиент = 0,

Кассир = 1

}

}

Months.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDataModels.Enums

{

public enum Months

{

Январь = 1,

Февраль, Март, Апрель, Май, Июнь,

Июль, Август, Сентябрь, Октябрь, Ноябрь, Декабрь

};

}

OfficeOperationEnum.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDataModels.Enums

{

public enum OfficeOperationEnum

{

Между\_cчетами = 0,

Пополнение\_карт = 1,

Cнятие\_с\_карты = 2,

Для\_кассира = 3

}

}

StatusEnum.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDataModels.Enums

{

public enum StatusEnum

{

Открыта = 0,

Закрыта = 1,

Отклонено = 2,

Карта\_просрочена = 3

}

}

TypeDocEnum.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDataModels.Enums

{

public enum TypeDocEnum

{

PDF = 0,

EXCEL = 1,

WORD = 2

}

}

TypeOperationEnum.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDataModels.Enums

{

public enum TypeOperationEnum

{

Снятие = 1,

Пополнение = 2,

Перевод = 3,

}

}

Папка Models

IMessageInfoModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDataModels.Models

{

public interface IMessageInfoModel : IId

{

string MessageId { get; }

int? ClientId { get; }

string SenderName { get; }

DateTime DateDelivery { get; }

string Subject { get; }

string Body { get; }

public bool IsRead { get; }

public string Answer { get; }

}

}

Папка Cashier

IAccountModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Principal;

using System.Text;

using System.Threading.Tasks;

namespace BankDataModels.Models.Cashier

{

//банковский счёт

public interface IAccountModel : IId

{

string AccountNumber { get; }

int CashierId { get; }

int ClientId { get; }

string PasswordAccount { get; }

double Balance { get; }

DateTime DateOpen { get; }

}

}

ICashWithdrawalModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDataModels.Models.Cashier

{

//выдача наличных

public interface ICashWithdrawalModel : IId

{

int DebitingId { get; }

int AccountId { get; }

int CashierId { get; }

int Sum { get; }

DateTime DateOperation { get; }

}

}

ICashierModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDataModels.Models.Cashier

{

//клиент

public interface ICashierModel : IId

{

string Password { get; }

string Name { get; }

string Surname { get; }

string Patronymic { get; }

string Email { get; }

string Telephone { get; }

}

}

IMoneyTransferModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDataModels.Models.Cashier

{

//перевод денег

public interface IMoneyTransferModel : IId

{

int Sum { get; }

int? AccountSenderId { get; }

int AccountPayeeId { get; }

DateTime DateOperation { get; }

int? CreditingId { get; }

int CashierId { get; }

}

}

Папка Client

ICreditingModel.cs

using BankDataModels.Enums;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDataModels.Models.Client

{

//пополнение карты

public interface ICreditingModel : IId

{

int CardId { get; }

int Sum { get; }

DateTime DateOpen { get; }

DateTime? DateClose { get; }

StatusEnum Status { get; }

}

}

IDebitingModel.cs

using BankDataModels.Enums;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDataModels.Models.Client

{

//снятие денег с карты

public interface IDebitingModel : IId

{

int CardId { get; }

int Sum { get; }

DateTime DateOpen { get; }

DateTime? DateClose { get; }

StatusEnum Status { get; }

}

}

IСardModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDataModels.Models.Client

{

//банковская карта

public interface ICardModel : IId

{

int ClientID { get; }

int AccountId { get; }

string Number { get; }

string CVC { get; }

DateTime Period { get; }

}

}

IСlientModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDataModels.Models.Client

{

//клиент

public interface IClientModel : IId

{

string Password { get; }

string Name { get; }

string Surname { get; }

string Patronymic { get; }

string Email { get; }

string Telephone { get; }

}

}

Проект BankDatabaseImplement

BankDatabase.cs

using BankDatabaseImplement.Models;

using BankDatabaseImplement.Models.CashierModels;

using BankDatabaseImplement.Models.ClientModels;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDatabaseImplement

{

public class BankDatabase : DbContext

{

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

if (optionsBuilder.IsConfigured == false)

{

optionsBuilder.UseSqlServer(@"Data Source= localhost\SQLEXPRESS;Initial Catalog=BankDataBase;Integrated Security=True;MultipleActiveResultSets=True;;TrustServerCertificate=True");

}

base.OnConfiguring(optionsBuilder);

}

public virtual DbSet<Account> Accounts { set; get; }

public virtual DbSet<Card> Cards { set; get; }

public virtual DbSet<Cashier> Cashiers { set; get; }

public virtual DbSet<CashWithdrawal> CashWithdrawals { set; get; }

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

public virtual DbSet<Debiting> Debitings { set; get; }

public virtual DbSet<Crediting> Creditings { set; get; }

public virtual DbSet<MoneyTransfer> MoneyTransfers { set; get; }

}

}

Папка Models  
Папка CashierModels

Account.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.ViewModels.Cashier.ViewModels;

using BankDatabaseImplement.Models.ClientModels;

using BankDataModels.Models.Cashier;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace BankDatabaseImplement.Models.CashierModels

{

public class Account : IAccountModel

{

public int Id { get; set; }

[Required]

public string AccountNumber { get; set; } = string.Empty;

[Required]

public int CashierId { get; set; }

[Required]

public int ClientId { get; set; }

//для передачи ФИО клиента

public virtual Client Client { get; set; }

[Required]

public string PasswordAccount { get; set; } = string.Empty;

[Required]

public double Balance { get; set; }

[Required]

public DateTime DateOpen { get; set; } = DateTime.Now;

//для реализации связи один ко многим со Снятием наличных

[ForeignKey("AccountId")]

public virtual List<CashWithdrawal> CashWithdrawals { get; set; } = new();

//для реализации связи один ко многим с Переводом денег

[NotMapped]

[ForeignKey("AccountSenderId")]

public virtual List<MoneyTransfer> MoneyTransferSenders { get; set; } = new();

[NotMapped]

[ForeignKey("AccountPayeeId")]

public virtual List<MoneyTransfer> MoneyTransferPayees { get; set; } = new();

//для реализации связи один ко многим с Картами

[ForeignKey("AccountId")]

public virtual List<Card> Cards { get; set; } = new();

public static Account Create(BankDatabase context, AccountBindingModel model)

{

return new Account()

{

Id = model.Id,

ClientId = model.ClientId,

Client = context.Clients.First(x => x.Id == model.ClientId),

PasswordAccount = model.PasswordAccount,

Balance = model.Balance,

DateOpen = model.DateOpen,

CashierId = model.CashierId,

AccountNumber = model.AccountNumber

};

}

public void Update(AccountBindingModel model)

{

Balance = model.Balance;

}

public AccountViewModel GetViewModel => new()

{

Id = Id,

CashierId = CashierId,

ClientId = ClientId,

Name = Client.Name,

Patronymic = Client.Patronymic,

AccountNumber = AccountNumber,

Balance = Balance,

PasswordAccount = PasswordAccount,

DateOpen = DateOpen

};

}

}

CashWithdrawal.cs

using BankContracts.BindingModels;

using BankContracts.BindingModels.Cashier;

using BankContracts.ViewModels;

using BankContracts.ViewModels.Cashier.ViewModels;

using BankDatabaseImplement.Models.ClientModels;

using BankDataModels.Models;

using BankDataModels.Models.Cashier;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankDatabaseImplement.Models.CashierModels

{

public class CashWithdrawal : ICashWithdrawalModel

{

public int Id { get; set; }

[Required]

public int DebitingId { get; set; }

//для передачи номера заявки

public virtual Debiting Debiting { get; set; }

[Required]

public int AccountId { get; set; }

//для передачи названия изделия

public virtual Account Account { get; set; }

[Required]

public int CashierId { get; set; }

//для передачи фамилии кассира

public virtual Cashier Cashier { get; set; }

[Required]

public int Sum { get; set; }

[Required]

public DateTime DateOperation { get; set; }

public static CashWithdrawal Create(BankDatabase context, CashWithdrawalBindingModel model)

{

return new CashWithdrawal()

{

Id = model.Id,

AccountId = model.AccountId,

Account = context.Accounts.First(x => x.Id == model.AccountId),

DebitingId = model.DebitingId,

Debiting = context.Debitings.First(x => x.Id == model.DebitingId),

CashierId = model.CashierId,

Cashier = context.Cashiers.First(x => x.Id == model.CashierId),

Sum = model.Sum,

DateOperation = model.DateOperation

};

}

public void Update(CashWithdrawalBindingModel model)

{

DateOperation = model.DateOperation;

}

public CashWithdrawalViewModel GetViewModel => new()

{

Id = Id,

AccountId = AccountId,

CashierId = CashierId,

DebitingId = DebitingId,

AccountNumber = Account.AccountNumber,

SurmaneCashier = Cashier.Surname,

DebbitingNumber = Debiting.Id,

DebitingStatus = Debiting.Status,

Sum = Sum,

DateOperation = DateOperation

};

}

}

Cashier.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.ViewModels.Cashier.ViewModels;

using BankDatabaseImplement.Models.CashierModels;

using BankDataModels.Models.Cashier;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace BankDatabaseImplement.Models.CashierModels

{

public class Cashier : ICashierModel

{

public int Id { get; set; }

[Required]

public string Password { get; set; } = string.Empty;

[Required]

public string Name { get; set; } = string.Empty;

[Required]

public string Surname { get; set; } = string.Empty;

[Required]

public string Patronymic { get; set; } = string.Empty;

[Required]

public string Email { get; set; } = string.Empty;

[Required]

public string Telephone { get; set; } = string.Empty;

//для реализации связи один ко многим со Счётом

[ForeignKey("CashierId")]

public virtual List<Account> Accounts { get; set; } = new();

//для реализации связи один ко многим с заявками на снятие наличных со счёта

[ForeignKey("CashierId")]

public virtual List<CashWithdrawal> CashWithdrawals { get; set; } = new();

//для реализации связи один ко многим с заявками на переводы денег

[ForeignKey("CashierId")]

public virtual List<MoneyTransfer> MoneyTransfers { get; set; } = new();

public static Cashier Create(BankDatabase context, CashierBindingModel model)

{

return new Cashier()

{

Id = model.Id,

Name = model.Name,

Surname = model.Surname,

Patronymic = model.Patronymic,

Email = model.Email,

Telephone = model.Telephone,

Password = model.Password

};

}

public void Update(CashierBindingModel model)

{

Id = model.Id;

Name = model.Name;

Surname = model.Surname;

Patronymic = model.Patronymic;

Email = model.Email;

Telephone = model.Telephone;

Password = model.Password;

}

public CashierViewModel GetViewModel => new()

{

Id = Id,

Name = Name,

Surname = Surname,

Patronymic = Patronymic,

Email = Email,

Telephone = Telephone,

Password = Password

};

}

}

MoneyTransfer.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.ViewModels.Cashier.ViewModels;

using BankDataModels.Models.Cashier;

using System.ComponentModel.DataAnnotations;

namespace BankDatabaseImplement.Models.CashierModels

{

public class MoneyTransfer : IMoneyTransferModel

{

public int Id { get; set; }

[Required]

public int Sum { get; set; }

public int? AccountSenderId { get; set; }

[Required]

public int AccountPayeeId { get; set; }

[Required]

public DateTime DateOperation { get; set; }

public int? CreditingId { get; set; }

public int CashierId { get; set; }

public virtual Cashier Cashier { get; set; }

public virtual Account AccountSender { get; set; }

public virtual Account AccountPayeer { get; set; }

public static MoneyTransfer Create(BankDatabase context, MoneyTransferBindingModel model)

{

return new MoneyTransfer()

{

Id = model.Id,

Sum = model.Sum,

Cashier = context.Cashiers.First(x => x.Id == model.CashierId),

AccountSender = model.AccountSenderId == null ? null : context.Accounts.First(x => x.Id == model.AccountSenderId),

AccountPayeer = context.Accounts.First(x => x.Id == model.AccountPayeeId),

AccountSenderId = model.AccountSenderId,

AccountPayeeId = model.AccountPayeeId,

DateOperation = model.DateOperation,

CreditingId = model.CreditingId,

CashierId = model.CashierId

};

}

public void Update(MoneyTransferBindingModel model)

{

Id = model.Id;

DateOperation = model.DateOperation;

}

public MoneyTransferViewModel GetViewModel => new()

{

Id = Id,

AccountPayeeId = AccountPayeeId,

AccountSenderId = AccountSenderId,

AccountPayeeNumber = AccountPayeer.AccountNumber,

AccountSenderNumber = AccountSenderId == null ? null : AccountSender.AccountNumber,

DateOperation = DateOperation,

Sum = Sum,

CreditingId = CreditingId,

CashierId = CashierId,

CashierSurname = Cashier.Surname

};

}

}

Папка ClientModels

Card.cs

using BankContracts.BindingModels.Client;

using BankContracts.ViewModels.Client.ViewModels;

using BankDatabaseImplement.Models.CashierModels;

using BankDataModels.Models.Client;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace BankDatabaseImplement.Models.ClientModels

{

public class Card : ICardModel

{

public int Id { get; set; }

[Required]

public int ClientID { get; set; }

public virtual Client Client { get; set; }

[Required]

public int AccountId { get; set; }

public virtual Account Account { get; set; }

[Required]

public string Number { get; set; } = string.Empty;

[Required]

public string CVC { get; set; } = string.Empty;

[Required]

public DateTime Period { get; set; } = DateTime.Now;

[ForeignKey("CardId")]

public virtual List<Crediting> Creditings { get; set; } = new();

[ForeignKey("CardId")]

public virtual List<Debiting> Debitings { get; set; } = new();

public CardViewModel GetViewModel => new()

{

Id = Id,

AccountId = AccountId,

ClientID = ClientID,

ClientSurname = Client.Surname,

Number = Number,

Sum = Account.Balance,

Period = Period,

CVC = CVC

};

public static Card Create(BankDatabase context, CardBindingModel model)

{

return new Card()

{

Id = model.Id,

AccountId = model.AccountId,

Account = context.Accounts.First(x => x.Id == model.AccountId),

ClientID = model.ClientID,

Client = context.Clients.First(x => x.Id == model.ClientID),

Number = model.Number,

Period = model.Period,

CVC = model.CVC

};

}

public void Update(CardBindingModel model)

{

Period = model.Period;

}

}

}

Client.cs

using System.ComponentModel.DataAnnotations.Schema;

using System.ComponentModel.DataAnnotations;

using BankDataModels.Models.Client;

using BankContracts.ViewModels.Client.ViewModels;

using BankContracts.BindingModels.Client;

namespace BankDatabaseImplement.Models.ClientModels

{

public class Client : IClientModel

{

public int Id { get; set; }

[Required]

public string Name { get; set; } = string.Empty;

[Required]

public string Surname { get; set; } = string.Empty;

[Required]

public string Patronymic { get; set; } = string.Empty;

[Required]

public string Telephone { get; set; } = string.Empty;

[Required]

public string Email { get; set; } = string.Empty;

[Required]

public string Password { get; set; } = string.Empty;

[ForeignKey("ClientID")]

public virtual List<Card> Cards { get; set; } = new();

public ClientViewModel GetViewModel => new()

{

Id = Id,

Name = Name,

Surname = Surname,

Patronymic = Patronymic,

Telephone = Telephone,

Email = Email,

Password = Password

};

public static Client Create(ClientBindingModel model)

{

return new Client()

{

Id = model.Id,

Name = model.Name,

Surname = model.Surname,

Patronymic = model.Patronymic,

Telephone = model.Telephone,

Email = model.Email,

Password = model.Password

};

}

public void Update(ClientBindingModel model)

{

Name = model.Name;

Surname = model.Surname;

Patronymic = model.Patronymic;

Telephone = model.Telephone;

Email = model.Email;

Password = model.Password;

}

}

}

Crediting.cs

using BankContracts.BindingModels.Client;

using BankContracts.ViewModels.Client.ViewModels;

using BankDatabaseImplement.Models.CashierModels;

using BankDatabaseImplement.Models.ClientModels;

using BankDataModels.Enums;

using BankDataModels.Models.Client;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace BankDatabaseImplement.Models

{

public class Crediting : ICreditingModel

{

public int Id { get; set; }

[Required]

public int CardId { get; set; }

public virtual Card Card { get; set; }

[Required]

public int Sum { get; set; }

[Required]

public DateTime DateOpen { get; set; } = DateTime.Now;

public DateTime? DateClose { get; set; }

[Required]

public StatusEnum Status { get; set; }

//для реализации связи один ко многим с переводом денег

[ForeignKey("CreditingId")]

public virtual List<MoneyTransfer> MoneyTransfers { get; set; } = new();

public CreditingViewModel GetViewModel => new()

{

Id = Id,

CardId = CardId,

CardNumber = Card.Number,

Sum = Sum,

DateOpen = DateOpen,

DateClose = DateClose,

Status = Status

};

public static Crediting Create(BankDatabase context, CreditingBindingModel model)

{

return new Crediting()

{

Id = model.Id,

CardId = model.CardId,

Card = context.Cards.First(x => x.Id == model.CardId),

Sum = model.Sum,

DateOpen = model.DateOpen,

Status = model.Status

};

}

public void Update(CreditingBindingModel model)

{

DateClose = model.DateClose;

Status = model.Status;

}

}

}

Debiting.cs

using BankContracts.BindingModels.Client;

using BankContracts.ViewModels.Client.ViewModels;

using BankDatabaseImplement.Models.ClientModels;

using BankDataModels.Enums;

using BankDataModels.Models.Client;

using System.ComponentModel.DataAnnotations;

namespace BankDatabaseImplement.Models

{

public class Debiting : IDebitingModel

{

public int Id { get; set; }

[Required]

public int CardId { get; set; }

public virtual Card Card { get; set; }

[Required]

public int Sum { get; set; }

[Required]

public DateTime DateOpen { get; set; } = DateTime.Now;

public DateTime? DateClose { get; set; }

[Required]

public StatusEnum Status { get; set; }

public DebitingViewModel GetViewModel => new()

{

Id = Id,

CardId = CardId,

CardNumber = Card.Number,

Sum = Sum,

DateOpen = DateOpen,

DateClose = DateClose,

Status = Status

};

public static Debiting Create(BankDatabase context, DebitingBindingModel model)

{

return new Debiting()

{

Id = model.Id,

CardId = model.CardId,

Card = context.Cards.First(x => x.Id == model.CardId),

Sum = model.Sum,

DateOpen = model.DateOpen,

Status = model.Status

};

}

public void Update(DebitingBindingModel model)

{

DateClose = model.DateClose;

Status = model.Status;

}

}

}

Папка Implements

Папка CashierImplements

AccountStorage.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.StoragesContracts.Cashier;

using BankContracts.ViewModels.Cashier.ViewModels;

using BankDatabaseImplement.Models.CashierModels;

using Microsoft.EntityFrameworkCore;

namespace BankDatabaseImplement.Implements.CashierImplements

{

public class AccountStorage : IAccountStorage

{

public List<AccountViewModel> GetFullList()

{

using var context = new BankDatabase();

return context.Accounts

.Include(x => x.Client)

.Select(x => x.GetViewModel)

.ToList();

}

public List<AccountViewModel> GetFilteredList(AccountSearchModel model)

{

using var context = new BankDatabase();

if (!string.IsNullOrEmpty(model.AccountNumber))

{

return context.Accounts

.Include(x => x.Client)

.Where(x => x.AccountNumber.Contains(model.AccountNumber))

.Select(x => x.GetViewModel)

.ToList();

}

if (model.CashierId.HasValue)

{

return context.Accounts

.Include(x => x.Client)

.Where(x => x.CashierId == model.CashierId)

.Select(x => x.GetViewModel)

.ToList();

}

if (model.ClientId.HasValue)

{

return context.Accounts

.Include(x => x.Client)

.Where(x => x.ClientId == model.ClientId)

.Select(x => x.GetViewModel)

.ToList();

}

return context.Accounts

.Include(x => x.Client)

.Select(x => x.GetViewModel)

.ToList();

}

public AccountViewModel? GetElement(AccountSearchModel model)

{

if (string.IsNullOrEmpty(model.AccountNumber) && !model.Id.HasValue)

{

return null;

}

using var context = new BankDatabase();

return context.Accounts

.Include(x => x.Client)

.FirstOrDefault(x => (!string.IsNullOrEmpty(model.AccountNumber) && x.AccountNumber == model.AccountNumber) ||

(model.Id.HasValue && x.Id == model.Id))

?.GetViewModel;

}

public AccountViewModel? Insert(AccountBindingModel model)

{

using var context = new BankDatabase();

var newAccount = Account.Create(context, model);

if (newAccount == null)

{

return null;

}

context.Accounts.Add(newAccount);

context.SaveChanges();

return newAccount.GetViewModel;

}

public AccountViewModel? Update(AccountBindingModel model)

{

using var context = new BankDatabase();

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

try

{

var account = context.Accounts.FirstOrDefault(rec => rec.Id == model.Id);

if (account == null)

{

return null;

}

account.Update(model);

context.SaveChanges();

transaction.Commit();

return context.Accounts

.Include(x => x.Client)

.FirstOrDefault(x => x.Id == model.Id)

?.GetViewModel;

}

catch

{

transaction.Rollback();

throw;

}

}

public AccountViewModel? Delete(AccountBindingModel model)

{

using var context = new BankDatabase();

var element = context.Accounts

.Include(x => x.AccountNumber)

.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.Accounts.Remove(element);

context.SaveChanges();

return element.GetViewModel;

}

return null;

}

}

}

CashWithdrawalStorage.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.StoragesContracts.Cashier;

using BankContracts.ViewModels.Cashier.ViewModels;

using BankDatabaseImplement.Models.CashierModels;

using Microsoft.EntityFrameworkCore;

namespace BankDatabaseImplement.Implements.CashierImplements

{

public class CashWithdrawalStorage : ICashWithdrawalStorage

{

public List<CashWithdrawalViewModel> GetFullList()

{

using var context = new BankDatabase();

return context.CashWithdrawals

.Include(x => x.Cashier)

.Include(x => x.Debiting)

.Include(x => x.Account)

.Select(x => x.GetViewModel)

.ToList();

}

public List<CashWithdrawalViewModel> GetFilteredList(CashWithdrawalSearchModel model)

{

using var context = new BankDatabase();

var result = context.CashWithdrawals.Include(x => x.Cashier)

.Include(x => x.Debiting)

.Include(x => x.Account)

.ToList();

if (model.AccountId.HasValue) result = result.Where(x => x.AccountId == model.AccountId).ToList();

if (model.CashierId.HasValue) result = result.Where(x => x.CashierId == model.CashierId).ToList();

if (model.DateFrom.HasValue) result = result.Where(x => x.DateOperation >= model.DateFrom).ToList();

if (model.DateTo.HasValue) result = result.Where(x => x.DateOperation <= model.DateTo).ToList();

if (model.ClientId.HasValue) result = result.Where(x => x.Account.ClientId == model.ClientId).ToList();

return result.Select(x => x.GetViewModel).ToList();

}

public CashWithdrawalViewModel? GetElement(CashWithdrawalSearchModel model)

{

using var context = new BankDatabase();

return context.CashWithdrawals

.Include(x => x.Cashier)

.Include(x => x.Debiting)

.Include(x => x.Account)

.FirstOrDefault(x => (!(model.AccountId < 0) && x.AccountId == model.AccountId) ||

(model.Id.HasValue && x.Id == model.Id))

?.GetViewModel;

}

public CashWithdrawalViewModel? Insert(CashWithdrawalBindingModel model)

{

using var context = new BankDatabase();

var newCashWithdrawal = CashWithdrawal.Create(context, model);

if (newCashWithdrawal == null)

{

return null;

}

context.CashWithdrawals.Add(newCashWithdrawal);

context.SaveChanges();

return newCashWithdrawal.GetViewModel;

}

public CashWithdrawalViewModel? Update(CashWithdrawalBindingModel model)

{

using var context = new BankDatabase();

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

try

{

var cashWithdrawal = context.CashWithdrawals.FirstOrDefault(rec => rec.Id == model.Id);

if (cashWithdrawal == null)

{

return null;

}

cashWithdrawal.Update(model);

context.SaveChanges();

transaction.Commit();

return cashWithdrawal.GetViewModel;

}

catch

{

transaction.Rollback();

throw;

}

}

public CashWithdrawalViewModel? Delete(CashWithdrawalBindingModel model)

{

using var context = new BankDatabase();

var element = context.CashWithdrawals

.Include(x => x.AccountId)

.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.CashWithdrawals.Remove(element);

context.SaveChanges();

return element.GetViewModel;

}

return null;

}

}

}

CashierStorage.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.StoragesContracts.Cashier;

using BankContracts.ViewModels.Cashier.ViewModels;

using BankDatabaseImplement.Models.CashierModels;

namespace BankDatabaseImplement.Implements.CashierImplements

{

public class CashierStorage : ICashierStorage

{

public List<CashierViewModel> GetFullList()

{

using var context = new BankDatabase();

return context.Cashiers

.Select(x => x.GetViewModel)

.ToList();

}

public List<CashierViewModel> GetFilteredList(CashierSearchModel model)

{

if (string.IsNullOrEmpty(model.Email))

{

return new();

}

using var context = new BankDatabase();

return context.Cashiers

.Where(x => x.Email.Contains(model.Email))

.Select(x => x.GetViewModel)

.ToList();

}

public CashierViewModel? GetElement(CashierSearchModel model)

{

if (string.IsNullOrEmpty(model.Email) && !model.Id.HasValue)

{

return null;

}

using var context = new BankDatabase();

return context.Cashiers

.FirstOrDefault(x => (!string.IsNullOrEmpty(model.Email) && x.Email == model.Email) ||

(model.Id.HasValue && x.Id == model.Id))

?.GetViewModel;

}

public CashierViewModel? Insert(CashierBindingModel model)

{

using var context = new BankDatabase();

var newCashier = Cashier.Create(context, model);

if (newCashier == null)

{

return null;

}

context.Cashiers.Add(newCashier);

context.SaveChanges();

return newCashier.GetViewModel;

}

public CashierViewModel? Update(CashierBindingModel model)

{

using var context = new BankDatabase();

var cashier = context.Cashiers.FirstOrDefault(x => x.Id == model.Id);

if (cashier == null)

{

return null;

}

cashier.Update(model);

context.SaveChanges();

return cashier.GetViewModel;

}

public CashierViewModel? Delete(CashierBindingModel model)

{

using var context = new BankDatabase();

var element = context.Cashiers.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.Cashiers.Remove(element);

context.SaveChanges();

return element.GetViewModel;

}

return null;

}

}

}

MoneyTransferStorage.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.StoragesContracts.Cashier;

using BankContracts.ViewModels.Cashier.ViewModels;

using BankDatabaseImplement.Models.CashierModels;

using Microsoft.EntityFrameworkCore;

namespace BankDatabaseImplement.Implements.CashierImplements

{

public class MoneyTransferStorage : IMoneyTransferStorage

{

public List<MoneyTransferViewModel> GetFullList()

{

using var context = new BankDatabase();

return context.MoneyTransfers

.Include(x => x.AccountPayeer)

.Include(x => x.AccountSender)

.Include(x => x.Cashier)

.Select(x => x.GetViewModel)

.ToList();

}

public List<MoneyTransferViewModel> GetFilteredList(MoneyTransferSearchModel model)

{

using var context = new BankDatabase();

var result = context.MoneyTransfers.Include(x => x.Cashier)

.Include(x => x.AccountPayeer)

.Include(x => x.AccountSender)

.ToList();

if (model.AccountPayeeId.HasValue && model.AccountSenderId.HasValue && model.AccountPayeeId == model.AccountSenderId)

{

return result.Where(x => (x.AccountSenderId == model.AccountSenderId || x.AccountPayeeId == model.AccountPayeeId)

&& x.AccountSender != null)

.Select(x => x.GetViewModel)

.ToList();

}

if (model.AccountPayeeId.HasValue) result = result.Where(x => x.AccountPayeeId == model.AccountPayeeId).ToList();

if (model.AccountSenderId.HasValue) result = result.Where(x => x.AccountSenderId == model.AccountSenderId).ToList();

if (model.CashierId.HasValue) result = result.Where(x => x.CashierId == model.CashierId).ToList();

if (model.DateFrom.HasValue) result = result.Where(x => x.DateOperation >= model.DateFrom).ToList();

if (model.DateTo.HasValue) result = result.Where(x => x.DateOperation <= model.DateTo).ToList();

if (model.ClientId.HasValue)

{

result = result.Where(x => x.AccountSender != null)

.Where(x => (x.AccountPayeer.ClientId == model.ClientId || x.AccountSender.ClientId == model.ClientId))

.ToList();

}

return result.Select(x => x.GetViewModel).ToList();

}

public MoneyTransferViewModel? GetElement(MoneyTransferSearchModel model)

{

if (model.AccountSenderId < 0 && !model.Id.HasValue)

{

return null;

}

using var context = new BankDatabase();

return context.MoneyTransfers

.Include(x => x.AccountPayeer)

.Include(x => x.AccountSender)

.Include(x => x.Cashier)

.FirstOrDefault(x => x.Id == model.Id)

?.GetViewModel;

}

public MoneyTransferViewModel? Insert(MoneyTransferBindingModel model)

{

using var context = new BankDatabase();

var newMoneyTransfer = MoneyTransfer.Create(context, model);

if (newMoneyTransfer == null)

{

return null;

}

context.MoneyTransfers.Add(newMoneyTransfer);

context.SaveChanges();

return newMoneyTransfer.GetViewModel;

}

public MoneyTransferViewModel? Update(MoneyTransferBindingModel model)

{

using var context = new BankDatabase();

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

try

{

var moneyTransfer = context.MoneyTransfers.FirstOrDefault(rec => rec.Id == model.Id);

if (moneyTransfer == null)

{

return null;

}

moneyTransfer.Update(model);

context.SaveChanges();

transaction.Commit();

return moneyTransfer.GetViewModel;

}

catch

{

transaction.Rollback();

throw;

}

}

public MoneyTransferViewModel? Delete(MoneyTransferBindingModel model)

{

using var context = new BankDatabase();

var element = context.MoneyTransfers

.Include(x => x.AccountPayeeId)

.Include(x => x.AccountSenderId)

.Include(x => x.Sum)

.Include(x => x.DateOperation)

.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.MoneyTransfers.Remove(element);

context.SaveChanges();

return element.GetViewModel;

}

return null;

}

}

}

Папка ClientImplements

CardStorage.cs

using BankContracts.BindingModels.Client;

using BankContracts.SearchModels.Client;

using BankContracts.StoragesContracts.Client;

using BankContracts.ViewModels.Client.ViewModels;

using BankDatabaseImplement.Models.ClientModels;

using Microsoft.EntityFrameworkCore;

namespace BankDatabaseImplement.Implements

{

public class CardStorage : ICardStorage

{

public CardViewModel? Delete(CardBindingModel model)

{

using var context = new BankDatabase();

var element = context.Cards.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.Cards.Remove(element);

context.SaveChanges();

return element.GetViewModel;

}

return null;

}

public CardViewModel? GetElement(CardSearchModel model)

{

if (string.IsNullOrEmpty(model.Number) && !model.Id.HasValue)

{

return null;

}

using var context = new BankDatabase();

return context.Cards

.Include(x => x.Client)

.Include(x => x.Account)

.FirstOrDefault(x => (!string.IsNullOrEmpty(model.Number) && x.Number == model.Number) ||

(model.Id.HasValue && x.Id == model.Id))

?.GetViewModel;

}

public List<CardViewModel> GetFilteredList(CardSearchModel model)

{

using var context = new BankDatabase();

if (!string.IsNullOrEmpty(model.Number))

{

return context.Cards

.Include(x => x.Client)

.Include(x => x.Account)

.Where(x => x.Number.Contains(model.Number))

.Select(x => x.GetViewModel)

.ToList();

}

if (model.ClientID.HasValue)

{

return context.Cards

.Include(x => x.Client)

.Include(x => x.Account)

.Where(x => x.ClientID == model.ClientID)

.Select(x => x.GetViewModel)

.ToList();

}

if (model.AccountId.HasValue)

{

return context.Cards

.Include(x => x.Client)

.Include(x => x.Account)

.Where(x => x.AccountId == model.AccountId)

.Select(x => x.GetViewModel)

.ToList();

}

return context.Cards

.Include(x => x.Client)

.Include(x => x.Account)

.Select(x => x.GetViewModel)

.ToList();

}

public List<CardViewModel> GetFullList()

{

using var context = new BankDatabase();

return context.Cards

.Include(x => x.Client)

.Include(x => x.Account)

.Select(x => x.GetViewModel)

.ToList();

}

public CardViewModel? Insert(CardBindingModel model)

{

using var context = new BankDatabase();

var newCard = Card.Create(context, model);

if (newCard == null)

{

return null;

}

context.Cards.Add(newCard);

context.SaveChanges();

return newCard.GetViewModel;

}

public CardViewModel? Update(CardBindingModel model)

{

using var context = new BankDatabase();

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

try

{

var card = context.Cards.FirstOrDefault(rec => rec.Id == model.Id);

if (card == null)

{

return null;

}

card.Update(model);

context.SaveChanges();

transaction.Commit();

return card.GetViewModel;

}

catch

{

transaction.Rollback();

throw;

}

}

}

}

ClientStorage.cs

using BankContracts.BindingModels.Client;

using BankContracts.SearchModels.Client;

using BankContracts.StoragesContracts.Client;

using BankContracts.ViewModels.Client.ViewModels;

using BankDatabaseImplement.Models.ClientModels;

namespace BankDatabaseImplement.Implements.ClientImplements

{

public class ClientStorage : IClientStorage

{

public ClientViewModel? Delete(ClientBindingModel model)

{

using var context = new BankDatabase();

var element = context.Clients.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.Clients.Remove(element);

context.SaveChanges();

return element.GetViewModel;

}

return null;

}

public ClientViewModel? GetElement(ClientSearchModel model)

{

if (string.IsNullOrEmpty(model.Email) && string.IsNullOrEmpty(model.Password) && !model.Id.HasValue)

{

return null;

}

using var context = new BankDatabase();

//сработает для поиска почты для отправки файла

if(model.Id.HasValue && string.IsNullOrEmpty(model.Password))

{

return context.Clients.FirstOrDefault(x => x.Id == model.Id)?.GetViewModel;

}

return context.Clients.FirstOrDefault(x =>

(!string.IsNullOrEmpty(model.Email) && x.Email == model.Email && !string.IsNullOrEmpty(model.Password) && x.Password == model.Password) ||

(model.Id.HasValue && x.Id == model.Id))

?.GetViewModel;

}

public List<ClientViewModel> GetFilteredList(ClientSearchModel model)

{

if (string.IsNullOrEmpty(model.Name))

{

return new();

}

using var context = new BankDatabase();

return context.Clients.Where(x => x.Name.Contains(model.Name)).Select(x => x.GetViewModel).ToList();

}

public List<ClientViewModel> GetFullList()

{

using var context = new BankDatabase();

return context.Clients.Select(x => x.GetViewModel).ToList();

}

public ClientViewModel? Insert(ClientBindingModel model)

{

var newComponent = Client.Create(model);

if (newComponent == null)

{

return null;

}

using var context = new BankDatabase();

context.Clients.Add(newComponent);

context.SaveChanges();

return newComponent.GetViewModel;

}

public ClientViewModel? Update(ClientBindingModel model)

{

using var context = new BankDatabase();

var component = context.Clients.FirstOrDefault(x => x.Id == model.Id);

if (component == null)

{

return null;

}

component.Update(model);

context.SaveChanges();

return component.GetViewModel;

}

}

}

CreditingStorage.cs

using BankContracts.BindingModels.Client;

using BankContracts.SearchModels.Client;

using BankContracts.StoragesContracts.Client;

using BankContracts.ViewModels.Client.ViewModels;

using BankDatabaseImplement.Models;

using Microsoft.EntityFrameworkCore;

namespace BankDatabaseImplement.Implements.ClientImplements

{

public class CreditingStorage : ICreditingStorage

{

public List<CreditingViewModel> GetFullList()

{

using var context = new BankDatabase();

return context.Creditings

.Include(x => x.Card)

.Select(x => x.GetViewModel)

.ToList();

}

public List<CreditingViewModel> GetFilteredList(CreditingSearchModel model)

{

using var context = new BankDatabase();

var result = context.Creditings.Include(x => x.Card).ToList();

if (model.Status.HasValue) result = result.Where(x => x.Status == model.Status).ToList();

if (model.CardId.HasValue) result = result.Where(x => x.CardId == model.CardId).ToList();

if (model.UserId.HasValue)

{

List<int> cards = context.Cards.Where(x => x.ClientID == model.UserId).Select(x => x.Id).ToList();

result = result.Where(x => cards.Contains(x.CardId)).ToList();

}

if (model.DateFrom.HasValue && model.DateTo.HasValue)

{

result = result.Where(x => x.DateOpen >= model.DateFrom && x.DateClose <= model.DateTo).ToList();

}

return result.Select(x => x.GetViewModel).ToList();

}

public CreditingViewModel? GetElement(CreditingSearchModel model)

{

if (model.CardId < 0 && !model.Id.HasValue)

{

return null;

}

using var context = new BankDatabase();

return context.Creditings

.Include(x => x.Card)

.FirstOrDefault(x => x.Id == model.Id)

?.GetViewModel;

}

public CreditingViewModel? Insert(CreditingBindingModel model)

{

using var context = new BankDatabase();

var newCrediting = Crediting.Create(context, model);

if (newCrediting == null)

{

return null;

}

context.Creditings.Add(newCrediting);

context.SaveChanges();

return newCrediting.GetViewModel;

}

public CreditingViewModel? Update(CreditingBindingModel model)

{

using var context = new BankDatabase();

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

try

{

var crediting = context.Creditings.FirstOrDefault(rec => rec.Id == model.Id);

if (crediting == null)

{

return null;

}

crediting.Update(model);

context.SaveChanges();

transaction.Commit();

return context.Creditings

.Include(x => x.Card)

.FirstOrDefault(x => x.Id == model.Id)

?.GetViewModel;

}

catch

{

transaction.Rollback();

throw;

}

}

public CreditingViewModel? Delete(CreditingBindingModel model)

{

using var context = new BankDatabase();

var element = context.Creditings

.Include(x => x.CardId)

.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.Creditings.Remove(element);

context.SaveChanges();

return element.GetViewModel;

}

return null;

}

}

}

DebitingStorage.cs

using BankContracts.BindingModels.Client;

using BankContracts.SearchModels.Client;

using BankContracts.StoragesContracts.Client;

using BankContracts.ViewModels.Client.ViewModels;

using BankDatabaseImplement.Models;

using BankDataModels.Enums;

using Microsoft.EntityFrameworkCore;

namespace BankDatabaseImplement.Implements.ClientImplements

{

public class DebitingStorage : IDebitingStorage

{

public List<DebitingViewModel> GetFullList()

{

using var context = new BankDatabase();

return context.Debitings

.Include(x => x.Card)

.Select(x => x.GetViewModel)

.ToList();

}

public List<DebitingViewModel> GetFilteredList(DebitingSearchModel model)

{

using var context = new BankDatabase();

var result = context.Debitings.Include(x => x.Card).ToList();

//для получения всех заявок на стнятие со статусом "Открыта"

if (model.Status.HasValue) result = result.Where(x => x.Status == model.Status).ToList();

if (model.CardId.HasValue) result = result.Where(x => x.CardId == model.CardId).ToList();

if (model.UserId.HasValue) {

List<int> cards = context.Cards.Where(x => x.ClientID == model.UserId).Select(x => x.Id).ToList();

result = result.Where(x => cards.Contains(x.CardId)).ToList();

}

if (model.DateFrom.HasValue && model.DateTo.HasValue)

{

result = result.Where(x => x.DateOpen <= model.DateFrom && x.DateClose >= model.DateTo && x.Status != StatusEnum.Отклонено).ToList();

}

return result.Select(x => x.GetViewModel).ToList();

}

public DebitingViewModel? GetElement(DebitingSearchModel model)

{

if (model.CardId < 0 && !model.Id.HasValue)

{

return null;

}

using var context = new BankDatabase();

return context.Debitings

.Include(x => x.Card)

.FirstOrDefault(x => (!(model.CardId < 0) && x.CardId == model.CardId) ||

(model.Id.HasValue && x.Id == model.Id))

?.GetViewModel;

}

public DebitingViewModel? Insert(DebitingBindingModel model)

{

using var context = new BankDatabase();

var newDebiting = Debiting.Create(context, model);

if (newDebiting == null)

{

return null;

}

context.Debitings.Add(newDebiting);

context.SaveChanges();

return newDebiting.GetViewModel;

}

public DebitingViewModel? Update(DebitingBindingModel model)

{

using var context = new BankDatabase();

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

try

{

var debiting = context.Debitings.FirstOrDefault(rec => rec.Id == model.Id);

if (debiting == null && debiting.Status == StatusEnum.Закрыта)

{

return null;

}

debiting.Update(model);

context.SaveChanges();

transaction.Commit();

return context.Debitings

.Include(x => x.Card)

.FirstOrDefault(x => x.Id == model.Id)

?.GetViewModel;

}

catch

{

transaction.Rollback();

throw;

}

}

public DebitingViewModel? Delete(DebitingBindingModel model)

{

using var context = new BankDatabase();

var element = context.Debitings

.Include(x => x.CardId)

.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.Debitings.Remove(element);

context.SaveChanges();

return element.GetViewModel;

}

return null;

}

}

}

BankDatabaseModelSnapshot.cs

// <auto-generated />

using System;

using BankDatabaseImplement;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Infrastructure;

using Microsoft.EntityFrameworkCore.Metadata;

using Microsoft.EntityFrameworkCore.Storage.ValueConversion;

#nullable disable

namespace BankDatabaseImplement.Migrations

{

[DbContext(typeof(BankDatabase))]

partial class BankDatabaseModelSnapshot : ModelSnapshot

{

protected override void BuildModel(ModelBuilder modelBuilder)

{

#pragma warning disable 612, 618

modelBuilder

.HasAnnotation("ProductVersion", "7.0.5")

.HasAnnotation("Relational:MaxIdentifierLength", 128);

SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder);

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.Account", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

b.Property<string>("AccountNumber")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<double>("Balance")

.HasColumnType("float");

b.Property<int>("CashierId")

.HasColumnType("int");

b.Property<int>("ClientId")

.HasColumnType("int");

b.Property<DateTime>("DateOpen")

.HasColumnType("datetime2");

b.Property<string>("PasswordAccount")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.HasKey("Id");

b.HasIndex("CashierId");

b.HasIndex("ClientId");

b.ToTable("Accounts");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.Card", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

b.Property<int>("AccountId")

.HasColumnType("int");

b.Property<string>("CVC")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<int>("ClientID")

.HasColumnType("int");

b.Property<string>("Number")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<DateTime>("Period")

.HasColumnType("datetime2");

b.HasKey("Id");

b.HasIndex("AccountId");

b.HasIndex("ClientID");

b.ToTable("Cards");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.CashWithdrawal", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

b.Property<int>("AccountId")

.HasColumnType("int");

b.Property<int>("CashierId")

.HasColumnType("int");

b.Property<DateTime>("DateOperation")

.HasColumnType("datetime2");

b.Property<int>("DebitingId")

.HasColumnType("int");

b.Property<int>("Sum")

.HasColumnType("int");

b.HasKey("Id");

b.HasIndex("AccountId");

b.HasIndex("CashierId");

b.HasIndex("DebitingId");

b.ToTable("CashWithdrawals");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.Cashier", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

b.Property<string>("Email")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Name")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Password")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Patronymic")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Surname")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Telephone")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.HasKey("Id");

b.ToTable("Cashiers");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.Client", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

b.Property<string>("Email")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Name")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Password")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Patronymic")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Surname")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Telephone")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.HasKey("Id");

b.ToTable("Clients");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.Crediting", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

b.Property<int>("CardId")

.HasColumnType("int");

b.Property<DateTime?>("DateClose")

.HasColumnType("datetime2");

b.Property<DateTime>("DateOpen")

.HasColumnType("datetime2");

b.Property<int>("Status")

.HasColumnType("int");

b.Property<int>("Sum")

.HasColumnType("int");

b.HasKey("Id");

b.HasIndex("CardId");

b.ToTable("Creditings");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.Debiting", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

b.Property<int>("CardId")

.HasColumnType("int");

b.Property<DateTime?>("DateClose")

.HasColumnType("datetime2");

b.Property<DateTime>("DateOpen")

.HasColumnType("datetime2");

b.Property<int>("Status")

.HasColumnType("int");

b.Property<int>("Sum")

.HasColumnType("int");

b.HasKey("Id");

b.HasIndex("CardId");

b.ToTable("Debitings");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.MoneyTransfer", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

b.Property<int>("AccountPayeeId")

.HasColumnType("int");

b.Property<int>("AccountPayeerId")

.HasColumnType("int");

b.Property<int?>("AccountSenderId")

.HasColumnType("int");

b.Property<int>("CashierId")

.HasColumnType("int");

b.Property<int?>("CreditingId")

.HasColumnType("int");

b.Property<DateTime>("DateOperation")

.HasColumnType("datetime2");

b.Property<int>("Sum")

.HasColumnType("int");

b.HasKey("Id");

b.HasIndex("AccountPayeerId");

b.HasIndex("AccountSenderId");

b.HasIndex("CashierId");

b.HasIndex("CreditingId");

b.ToTable("MoneyTransfers");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.Account", b =>

{

b.HasOne("BankYouBankruptDatabaseImplement.Models.Cashier", null)

.WithMany("Accounts")

.HasForeignKey("CashierId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.HasOne("BankYouBankruptDatabaseImplement.Models.Client", "Client")

.WithMany()

.HasForeignKey("ClientId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.Navigation("Client");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.Card", b =>

{

b.HasOne("BankYouBankruptDatabaseImplement.Models.Account", null)

.WithMany("Cards")

.HasForeignKey("AccountId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.HasOne("BankYouBankruptDatabaseImplement.Models.Client", "Client")

.WithMany("Cards")

.HasForeignKey("ClientID")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.Navigation("Client");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.CashWithdrawal", b =>

{

b.HasOne("BankYouBankruptDatabaseImplement.Models.Account", "Account")

.WithMany("CashWithdrawals")

.HasForeignKey("AccountId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.HasOne("BankYouBankruptDatabaseImplement.Models.Cashier", "Cashier")

.WithMany("CashWithdrawals")

.HasForeignKey("CashierId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.HasOne("BankYouBankruptDatabaseImplement.Models.Debiting", "Debiting")

.WithMany()

.HasForeignKey("DebitingId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.Navigation("Account");

b.Navigation("Cashier");

b.Navigation("Debiting");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.Crediting", b =>

{

b.HasOne("BankYouBankruptDatabaseImplement.Models.Card", "Card")

.WithMany("Creditings")

.HasForeignKey("CardId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.Navigation("Card");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.Debiting", b =>

{

b.HasOne("BankYouBankruptDatabaseImplement.Models.Card", "Card")

.WithMany("Debitings")

.HasForeignKey("CardId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.Navigation("Card");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.MoneyTransfer", b =>

{

b.HasOne("BankYouBankruptDatabaseImplement.Models.Account", "AccountPayeer")

.WithMany()

.HasForeignKey("AccountPayeerId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.HasOne("BankYouBankruptDatabaseImplement.Models.Account", "AccountSender")

.WithMany()

.HasForeignKey("AccountSenderId");

b.HasOne("BankYouBankruptDatabaseImplement.Models.Cashier", "Cashier")

.WithMany("MoneyTransfers")

.HasForeignKey("CashierId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.HasOne("BankYouBankruptDatabaseImplement.Models.Crediting", null)

.WithMany("MoneyTransfers")

.HasForeignKey("CreditingId");

b.Navigation("AccountPayeer");

b.Navigation("AccountSender");

b.Navigation("Cashier");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.Account", b =>

{

b.Navigation("Cards");

b.Navigation("CashWithdrawals");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.Card", b =>

{

b.Navigation("Creditings");

b.Navigation("Debitings");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.Cashier", b =>

{

b.Navigation("Accounts");

b.Navigation("CashWithdrawals");

b.Navigation("MoneyTransfers");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.Client", b =>

{

b.Navigation("Cards");

});

modelBuilder.Entity("BankYouBankruptDatabaseImplement.Models.Crediting", b =>

{

b.Navigation("MoneyTransfers");

});

#pragma warning restore 612, 618

}

}

}

Проект BankRestAPI

Program.cs

using BankBusinessLogic.BusinessLogics.Cashier;

using BankBusinessLogic.BusinessLogics.Client;

using BankBusinessLogic.BusinessLogics.Reports;

using BankBusinessLogic.MailWorker;

using BankBusinessLogic.OfficePackage;

using BankBusinessLogic.OfficePackage.Implements;

using BankContracts.BindingModels.Messages;

using BankContracts.BusinessLogicsContracts.Cashier;

using BankContracts.BusinessLogicsContracts.Client;

using BankContracts.BusinessLogicsContracts.Reports;

using BankContracts.StoragesContracts.Cashier;

using BankContracts.StoragesContracts.Client;

using BankDatabaseImplement.Implements;

using BankDatabaseImplement.Implements.CashierImplements;

using BankDatabaseImplement.Implements.ClientImplements;

using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

builder.Logging.SetMinimumLevel(LogLevel.Trace);

builder.Logging.AddLog4Net("log4net.config");

// Add services to the container.

builder.Services.AddTransient<IClientStorage, ClientStorage>();

builder.Services.AddTransient<ICashierStorage, CashierStorage>();

builder.Services.AddTransient<IAccountStorage, AccountStorage>();

builder.Services.AddTransient<ICardStorage, CardStorage>();

builder.Services.AddTransient<ICreditingStorage, CreditingStorage>();

builder.Services.AddTransient<IMoneyTransferStorage, MoneyTransferStorage>();

builder.Services.AddTransient<IDebitingStorage, DebitingStorage>();

builder.Services.AddTransient<ICashWithdrawalStorage, CashWithdrawalStorage>();

builder.Services.AddTransient<IClientLogic, ClientLogic>();

builder.Services.AddTransient<ICashierLogic, CashierLogic>();

builder.Services.AddTransient<IAccountLogic, AccountLogic>();

builder.Services.AddTransient<ICardLogic, CardLogic>();

builder.Services.AddTransient<ICreditingLogic, CreditingLogic>();

builder.Services.AddTransient<IMoneyTransferLogic, MoneyTransferLogic>();

builder.Services.AddTransient<IDebitingLogic, DebitingLogic>();

builder.Services.AddTransient<ICashWithdrawalLogic, CashWithdrawalLogic>();

builder.Services.AddTransient<IReportClientLogic, ReportClientLogic>();

builder.Services.AddSingleton<IReportCashierLogic, ReportCashierLogic>();

builder.Services.AddSingleton<MailKitWorker>();

//<2F><><EFBFBD><EFBFBD><EFBFBD> <20><><EFBFBD><EFBFBD><EFBFBD><EFBFBD> <20><><EFBFBD><EFBFBD><EFBFBD><EFBFBD><EFBFBD><EFBFBD><EFBFBD><EFBFBD><EFBFBD> <20><><EFBFBD><EFBFBD><EFBFBD><EFBFBD><EFBFBD>

builder.Services.AddTransient<AbstractSaveToPdf, SaveToPdf>();

builder.Services.AddTransient<AbstractSaveToExcel, SaveToExcel>();

builder.Services.AddTransient<AbstractSaveToWord, SaveToWord>();

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "BankRestAPI", Version = "v1" });

});

var app = builder.Build();

//Mails Service

var mailSender = app.Services.GetService<MailKitWorker>();

mailSender?.MailConfig(new MailConfigBindingModel

{

MailLogin = builder.Configuration?.GetSection("MailLogin")?.Value?.ToString() ?? string.Empty,

MailPassword = builder.Configuration?.GetSection("MailPassword")?.Value?.ToString() ?? string.Empty,

SmtpClientHost = builder.Configuration?.GetSection("SmtpClientHost")?.Value?.ToString() ?? string.Empty,

SmtpClientPort = Convert.ToInt32(builder.Configuration?.GetSection("SmtpClientPort")?.Value?.ToString()),

});

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "BankRestAPI v1"));

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

appsettings.json

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"SmtpClientHost": "smtp.gmail.com",

"SmtpClientPort": "587",

"PopHost": "pop.gmail.com",

"PopPort": "995",

"MailLogin": "furniturecoup73@gmail.com",

"MailPassword": "hwba frfa mmbo bpji"

}

Папка Controllers

AccountController.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.BusinessLogicsContracts.Cashier;

using BankContracts.BusinessLogicsContracts.Client;

using BankContracts.SearchModels.Cashier;

using BankContracts.SearchModels.Client;

using BankContracts.ViewModels.Cashier.Diagram;

using BankContracts.ViewModels.Cashier.ViewModels;

using BankContracts.ViewModels.Client.ViewModels;

using BankDataModels.Enums;

using Microsoft.AspNetCore.Mvc;

namespace BankRestApi.Controllers

{

//указание у контроллера, что Route будет строиться не только по наванию контроллера, но и по названию метода (так как у нас два Post-метода)

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

[ApiController]

public class AccountController : Controller

{

private readonly ILogger \_logger;

private readonly IAccountLogic \_accountLogic;

private readonly IDebitingLogic \_debitingLogic;

private readonly ICreditingLogic \_creditingLogic;

private readonly ICashWithdrawalLogic \_cashLogic;

private readonly IMoneyTransferLogic \_moneyTransferLogic;

public AccountController(IAccountLogic accountLogic, IDebitingLogic debitingLogic, ICashWithdrawalLogic cashLogic,

ICreditingLogic creditingLogic, IMoneyTransferLogic moneyTransferLogic, ILogger<AccountController> logger)

{

\_logger = logger;

\_accountLogic = accountLogic;

\_debitingLogic = debitingLogic;

\_cashLogic = cashLogic;

\_creditingLogic = creditingLogic;

\_moneyTransferLogic = moneyTransferLogic;

}

[HttpGet]

public AccountViewModel? Login(string accountNumber, string passwordAccount)

{

try

{

//попытка найти запись по переданным номеру и паролю счёта

return \_accountLogic.ReadElement(new AccountSearchModel

{

AccountNumber = accountNumber,

PasswordAccount = passwordAccount

});

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

//получаем все имеющиеся счета

[HttpGet]

public List<AccountViewModel>? GetAllAccounts()

{

try

{

return \_accountLogic.ReadList(null);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

//получаем все имеющиеся счета

[HttpGet]

public AccountViewModel? GetAccount(int accountId)

{

try

{

return \_accountLogic.ReadElement(new AccountSearchModel

{

Id = accountId

});

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

[HttpPost]

public void Register(AccountBindingModel model)

{

try

{

//создание счёта

\_accountLogic.Create(model);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка регистрации");

throw;

}

}

[HttpPost]

public void UpdateData(AccountBindingModel model)

{

try

{

//изменение счёта

\_accountLogic.Update(model);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка обновления данных");

throw;

}

}

//найти все счета, которые создал конкретный кассир

[HttpGet]

public List<AccountViewModel>? SearchAccounts(int cashierId)

{

try

{

return \_accountLogic.ReadList(new AccountSearchModel

{

CashierId = cashierId

});

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

//найти все счета клиента

[HttpGet]

public List<AccountViewModel>? SearchAccountsOfCLient(int clientId)

{

try

{

//попытка найти запись по переданным номеру и паролю счёта

return \_accountLogic.ReadList(new AccountSearchModel

{

ClientId = clientId

});

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

[HttpGet]

public List<CashWithdrawalViewModel>? FindAllCashWithdrawal()

{

try

{

return \_cashLogic.ReadList(null);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

[HttpGet]

public List<MoneyTransferViewModel>? FindAllMoneyTransfer()

{

try

{

return \_moneyTransferLogic.ReadList(null);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

//найти все открытые заявки на снятие средств

[HttpGet]

public List<DebitingViewModel>? FindOpenDebiting()

{

try

{

return \_debitingLogic.ReadList(new DebitingSearchModel

{

Status = StatusEnum.Открыта

});

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

//найти все открытые заявки на снятие средств

[HttpGet]

public List<CreditingViewModel>? FindOpenCrediting()

{

try

{

return \_creditingLogic.ReadList(new CreditingSearchModel

{

Status = StatusEnum.Открыта

});

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

//подтверждение заявки на снятие средств со счёта

[HttpPost]

public void CloseDebiting(CashWithdrawalBindingModel CashWithdrawal)

{

try

{

bool flag =\_accountLogic.ChangeBalance(new AccountSearchModel

{

Id = CashWithdrawal.AccountId

}, CashWithdrawal.Sum \* -1);

\_cashLogic.Create(CashWithdrawal, flag);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

//подтверждение заявки на перевод либо пополнение балланса

[HttpPost]

public void CloseCrediting(MoneyTransferBindingModel moneyTransfer)

{

try

{

\_moneyTransferLogic.Create(moneyTransfer);

bool flag = true;

//если есть отправитель, т. е. операция на перевод денег со счёта на счёт

if (moneyTransfer.AccountSenderId.HasValue)

{

flag = \_accountLogic.ChangeBalance(new AccountSearchModel

{

Id = moneyTransfer.AccountSenderId

}, moneyTransfer.Sum \* -1);

}

if (flag)

{

\_accountLogic.ChangeBalance(new AccountSearchModel

{

Id = moneyTransfer.AccountPayeeId

}, moneyTransfer.Sum);

}

else

{

throw new Exception("Недостаточно средств");

}

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

//поиск заявки на зачисление по id

[HttpGet]

public CreditingViewModel FindCrediting(int id)

{

try

{

return \_creditingLogic.ReadElement(new CreditingSearchModel

{

Id = id

});

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

//поиск заявки на снятие по id

[HttpGet]

public DebitingViewModel FindDebiting(int id)

{

try

{

return \_debitingLogic.ReadElement(new DebitingSearchModel

{

Id = id

});

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

[HttpGet]

public List<CashierDiagramElementsViewModel> getAccountMonthResult(int cardId)

{

try

{

return \_accountLogic.GetMonthInfo(cardId);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка получения пользователей");

throw;

}

}

}

}

CardController.cs

using BankContracts.BindingModels.Client;

using BankContracts.BusinessLogicsContracts.Cashier;

using BankContracts.BusinessLogicsContracts.Client;

using BankContracts.SearchModels.Client;

using BankContracts.ViewModels.Client.Diagram;

using BankContracts.ViewModels.Client.ViewModels;

using Microsoft.AspNetCore.Mvc;

using System.Net;

namespace BankRestApi.Controllers

{

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

[ApiController]

public class CardController : Controller

{

private readonly ILogger \_logger;

private readonly ICardLogic \_cardLogic;

private readonly IDebitingLogic \_debitingLogic;

private readonly ICreditingLogic \_creditingLogic;

private readonly IAccountLogic \_accountLogic;

public CardController(ICardLogic cardLogic, IDebitingLogic debitingLogic, ICreditingLogic creditingLogic, IAccountLogic accountLogic, ILogger<ClientController> logger)

{

\_logger = logger;

\_cardLogic = cardLogic;

\_debitingLogic = debitingLogic;

\_creditingLogic = creditingLogic;

\_accountLogic = accountLogic;

}

[HttpGet]

public List<CardViewModel>? GetAllCards()

{

try

{

return \_cardLogic.ReadList(null);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка получения карт");

throw;

}

}

[HttpGet]

public List<CardViewModel>? GetUsersCardsList(int id)

{

try

{

return \_cardLogic.ReadList(new CardSearchModel { ClientID = id });

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка получения карт");

throw;

}

}

[HttpPost]

public void CreateCard(CardBindingModel model) {

try

{

\_cardLogic.Create(model);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка создания карты");

Response.StatusCode = (int)HttpStatusCode.NotAcceptable;

}

}

[HttpPost]

public void CreateDebitingRequest(DebitingBindingModel model) {

try

{

\_debitingLogic.Create(model);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка создания запроса на снятие наличных");

Response.StatusCode = (int)HttpStatusCode.NotAcceptable;

}

}

[HttpPost]

public void CreateCreditingOperation(CreditingBindingModel model) {

try {

\_creditingLogic.Create(model);

}

catch (Exception ex) {

\_logger.LogError(ex, "Ошибка создания операции на пополнение");

Response.StatusCode = (int)HttpStatusCode.NotAcceptable;

}

}

[HttpGet]

public CardViewModel? FindCard(int id)

{

try

{

return \_cardLogic.ReadElement(new CardSearchModel

{

Id = id

});

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

[HttpGet]

public List<ClientDiagramElementsViewModel> getCardMonthResult(int cardId)

{

try

{

return \_cardLogic.GetMonthInfo(cardId);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка получения пользователей");

throw;

}

}

}

}

CashierController.cs

using BankContracts.BindingModels.Cashier;

using BankContracts.BusinessLogicsContracts.Cashier;

using BankContracts.SearchModels.Cashier;

using BankContracts.ViewModels.Cashier.ViewModels;

using Microsoft.AspNetCore.Mvc;

namespace BankRestApi.Controllers

{

//указание у контроллера, что Route будет строиться не только по наванию контроллера, но и по названию метода (так как у нас два Post-метода)

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

[ApiController]

public class CashierController : Controller

{

private readonly ILogger \_logger;

private readonly ICashierLogic \_cashierLogic;

public CashierController(ICashierLogic cashierLogic, ILogger<AccountController> logger)

{

\_logger = logger;

\_cashierLogic = cashierLogic;

}

[HttpGet]

public CashierViewModel? Login(string login, string password)

{

try

{

//попытка найти запись по переданным логину и паролю

return \_cashierLogic.ReadElement(new CashierSearchModel

{

Email = login,

Password = password

});

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

[HttpGet]

public CashierViewModel? GetCashier(int id)

{

try

{

//попытка найти запись по переданным логину и паролю

return \_cashierLogic.ReadElement(new CashierSearchModel

{

Id = id

});

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

[HttpPost]

public void Register(CashierBindingModel model)

{

try

{

//создание клиента

\_cashierLogic.Create(model);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка регистрации");

throw;

}

}

[HttpPost]

public void UpdateData(CashierBindingModel model)

{

try

{

//изменение клиента

\_cashierLogic.Update(model);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка обновления данных");

throw;

}

}

}

}

ClientController.cs

using BankContracts.BindingModels.Client;

using BankContracts.BusinessLogicsContracts.Client;

using BankContracts.SearchModels.Client;

using BankContracts.ViewModels.Client.ViewModels;

using Microsoft.AspNetCore.Mvc;

namespace BankRestApi.Controllers

{

//указание у контроллера, что Route будет строиться не только по наванию контроллера, но и по названию метода (так как у нас два Post-метода)

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

[ApiController]

public class ClientController : Controller

{

private readonly ILogger \_logger;

private readonly IClientLogic \_clientLogic;

private readonly IDebitingLogic \_debitingLogic;

private readonly ICreditingLogic \_creditingLogic;

public ClientController(ILogger<ClientController> logger,

IClientLogic clientLogic, IDebitingLogic debitingLogic, ICreditingLogic creditingLogic)

{

\_logger = logger;

\_clientLogic = clientLogic;

\_debitingLogic = debitingLogic;

\_creditingLogic = creditingLogic;

}

[HttpGet]

public ClientViewModel? Login(string login, string password)

{

try

{

//попытка найти запись по переданным логину и паролю

return \_clientLogic.ReadElement(new ClientSearchModel

{

Email = login,

Password = password

});

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

[HttpGet]

public List<ClientViewModel>? GetAllClients() {

try

{

return \_clientLogic.ReadList(null);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка получения пользователей");

throw;

}

}

[HttpGet]

public ClientViewModel? GetClient(int clientId)

{

try

{

return \_clientLogic.ReadElement(new ClientSearchModel { Id = clientId });

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка получения пользователя");

throw;

}

}

[HttpPost]

public void Register(ClientBindingModel model)

{

try

{

//создание клиента

\_clientLogic.Create(model);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка регистрации");

throw;

}

}

[HttpPost]

public void UpdateData(ClientBindingModel model)

{

try

{

//изменение клиента

\_clientLogic.Update(model);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка обновления данных");

throw;

}

}

[HttpGet]

public List<DebitingViewModel>? getUsersDebitings(int userId) {

try

{

return \_debitingLogic.ReadList(new DebitingSearchModel()

{

UserId = userId

});

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка получения пользователей");

throw;

}

}

[HttpGet]

public List<CreditingViewModel>? getUsersCreditings(int userId)

{

try

{

return \_creditingLogic.ReadList(new CreditingSearchModel()

{

UserId = userId

});

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка получения пользователей");

throw;

}

}

}

}

ReportController.cs

using BankContracts.BindingModels.Reports;

using BankContracts.BusinessLogicsContracts.Reports;

using BankContracts.ViewModels.Reports;

using BankDataModels.Enums;

using Microsoft.AspNetCore.Mvc;

namespace BankRestAPI.Controllers

{

//указание у контроллера, что Route будет строиться не только по наванию контроллера, но и по названию метода (так как у нас два Post-метода)

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

[ApiController]

public class ReportController : Controller

{

private readonly ILogger \_logger;

private readonly IReportClientLogic \_reportClientLogic;

private readonly IReportCashierLogic \_reportCashierLogic;

//хранят данные для отображения отчёта на вебе

private ReportClientViewModelForHTML \_reportClientViewModelForHTML;

private ReportCashierViewModelForHTML \_reportCashierViewModelForHTML;

public ReportController(ILogger<ReportController> logger, IReportClientLogic reportClientLogic, IReportCashierLogic reportCashierLogic)

{

\_logger = logger;

\_reportClientLogic = reportClientLogic;

\_reportCashierLogic = reportCashierLogic;

}

#region Работа с pdf

//метод генерации отчёта за период по картам клиента

[HttpPost]

public ReportClientViewModelForHTML CreateClientReport(ReportSupportBindingModel model)

{

try

{

var result = \_reportClientLogic.SaveClientReportToPdfFile(new ReportBindingModel

{

FileName = "Отчёт\_по\_картам.pdf",

DateFrom = model.DateFrom,

DateTo = model.DateTo,

Role = MailsEnum.Клиент,

Email = model.Email

});

return result;

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

//метод генерации отчёта по всем счетм клиентов

[HttpPost]

public ReportCashierViewModelForHTML CreateCashierReport(ReportSupportBindingModel model)

{

try

{

var result = \_reportCashierLogic.SaveAccountsToPdfFile(new ReportBindingModel

{

FileName = "Отчёт\_по\_счетам.pdf",

ClientId = model.ClientId,

DateFrom = model.DateFrom,

DateTo = model.DateTo,

Role = MailsEnum.Кассир,

Email = model.Email

});

return result;

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

#endregion

#region Работа с Excel

//отчёт клиента Excel по переводу денег

[HttpPost]

public void CreateExcelClient(ReportSupportBindingModel model)

{

try

{

\_reportClientLogic.SaveToExcelFile(new ReportBindingModel

{

FileName = "Отчёт по переводам.xlsx",

CardList = model.CardList,

Email = model.Email

}, OfficeOperationEnum.Между\_cчетами);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

//отчёт клиента Excel по переводу денег

[HttpPost]

public void CreateExcelCrediting(ReportSupportBindingModel model)

{

try

{

\_reportClientLogic.SaveToExcelFile(new ReportBindingModel

{

FileName = "Отчёт по пополнениям.xlsx",

CardList = model.CardList,

Email = model.Email

}, OfficeOperationEnum.Пополнение\_карт);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

//отчёт клиента Excel по выдаче денег

[HttpPost]

public void CreateExcelDebiting(ReportSupportBindingModel model)

{

try

{

\_reportClientLogic.SaveToExcelFile(new ReportBindingModel

{

FileName = "Отчёт по снятиям.xlsx",

CardList = model.CardList,

Email = model.Email

}, OfficeOperationEnum.Cнятие\_с\_карты);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

//отчёт клиента Excel по переводу денег

[HttpPost]

public void CreateExcelCashier(ReportSupportBindingModel model)

{

try

{

\_reportCashierLogic.SaveAccountsToExcelFile(new ReportBindingModel

{

FileName = "Отчёт по зявкам на снятие.xlsx",

AccountId = model.AccountId,

Email = model.Email

});

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

#endregion

#region Работа с word

//отчёт клиента Word по переводу денег

[HttpPost]

public void CreateWordClient(ReportSupportBindingModel model)

{

try

{

\_reportClientLogic.SaveToWordFile(new ReportBindingModel

{

FileName = "Отчёт по переводам.docx",

CardList = model.CardList,

Email = model.Email

}, OfficeOperationEnum.Между\_cчетами);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

//отчёт клиента Word по переводу денег

[HttpPost]

public void CreateWordCrediting(ReportSupportBindingModel model)

{

try

{

\_reportClientLogic.SaveToWordFile(new ReportBindingModel

{

FileName = "Отчёт по пополнениям.docx",

CardList = model.CardList,

Email = model.Email

}, OfficeOperationEnum.Пополнение\_карт);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

//отчёт клиента Word по выдаче денег

[HttpPost]

public void CreateWordDebiting(ReportSupportBindingModel model)

{

try

{

\_reportClientLogic.SaveToWordFile(new ReportBindingModel

{

FileName = "Отчёт по снятиям.docx",

CardList = model.CardList,

Email = model.Email

}, OfficeOperationEnum.Cнятие\_с\_карты);

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

//отчёт клиента Word по переводу денег

[HttpPost]

public void CreateWordCashier(ReportSupportBindingModel model)

{

try

{

\_reportCashierLogic.SaveAccountsToWordFile(new ReportBindingModel

{

FileName = "Отчёт по зявкам на снятие.docx",

AccountId = model.AccountId,

Email = model.Email

});

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка входа в систему");

throw;

}

}

#endregion

}

}

Проект BankClientApp

appsettings.json

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"IPAddress": "http://localhost:5164/"

}

APIClient.cs

using BankContracts.ViewModels.Client.ViewModels;

using Newtonsoft.Json;

using System.Net.Http.Headers;

using System.Text;

namespace BankСlientApp

{

public class APIClient

{

private static readonly HttpClient \_client = new();

public static ClientViewModel? Client { get; set; } = null;

public static string ErrorMessage = string.Empty;

public static void Connect(IConfiguration configuration)

{

\_client.BaseAddress = new Uri(configuration["IPAddress"]);

\_client.DefaultRequestHeaders.Accept.Clear();

\_client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

}

public static void SetErrorMessage(string error)

{

ErrorMessage = error;

}

//Get-запрос

public static T? GetRequest<T>(string requestUrl)

{

var response = \_client.GetAsync(requestUrl);

var result = response.Result.Content.ReadAsStringAsync().Result;

if (response.Result.IsSuccessStatusCode)

{

return JsonConvert.DeserializeObject<T>(result);

}

else

{

throw new Exception(result);

}

}

//Post-запрос

public static void PostRequest<T>(string requestUrl, T model)

{

var json = JsonConvert.SerializeObject(model);

var data = new StringContent(json, Encoding.UTF8, "application/json");

var response = \_client.PostAsync(requestUrl, data);

var result = response.Result.Content.ReadAsStringAsync().Result;

if (!response.Result.IsSuccessStatusCode)

{

throw new Exception(result);

}

}

//Post-запрос для получения данных

public static T? PostRequestReport<T, U>(string requestUrl, U model)

{

var json = JsonConvert.SerializeObject(model);

var data = new StringContent(json, Encoding.UTF8, "application/json");

var response = \_client.PostAsync(requestUrl, data);

var result = response.Result.Content.ReadAsStringAsync().Result;

if (response.Result.IsSuccessStatusCode)

{

return JsonConvert.DeserializeObject<T>(result);

}

else

{

throw new Exception(result);

}

}

}

}

Program.cs

using BankÑlientApp;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews();

var app = builder.Build();

APIClient.Connect(builder.Configuration);

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Enter}/{id?}");

app.Run();

Папка Controllers

HomeController.cs

using BankClientApp.Models;

using BankContracts.ViewModels.Client.Diagram;

using BankDataModels.Enums;

using BankСlientApp;

using Microsoft.AspNetCore.Mvc;

using System.Diagnostics;

using BankContracts.BindingModels.Client;

using BankContracts.BindingModels.Reports;

using BankContracts.ViewModels.Cashier.ViewModels;

using BankContracts.ViewModels.Client.ViewModels;

using BankContracts.ViewModels.Reports.Client;

using BankContracts.ViewModels.Reports;

namespace BankClientApp.Controllers

{

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger)

{

\_logger = logger;

}

#region Профиль, вход и регистрация

[HttpGet]

public IActionResult Enter()

{

return View();

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel

{

RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier

});

}

[HttpGet]

public IActionResult Login()

{

return View();

}

[HttpGet]

public IActionResult ErrorPage()

{

return View();

}

[HttpPost]

public IActionResult Login(string login, string password)

{

if (string.IsNullOrEmpty(login) || string.IsNullOrEmpty(password))

{

APIClient.SetErrorMessage("Введите логин и пароль");

return Redirect("ErrorPage");

}

APIClient.Client = APIClient.GetRequest<ClientViewModel>($"api/Client/Login?login={login}&password={password}");

if (APIClient.Client == null)

{

APIClient.SetErrorMessage("Неверный логин или пароль");

return Redirect("ErrorPage");

}

return Redirect("Enter");

}

[HttpGet]

public IActionResult Register()

{

return View();

}

[HttpPost]

public void Register(string login, string password, string name, string surname, string patronymic, string telephone)

{

if (string.IsNullOrEmpty(login) || string.IsNullOrEmpty(password) || string.IsNullOrEmpty(name)

|| string.IsNullOrEmpty(surname) || string.IsNullOrEmpty(patronymic) || string.IsNullOrEmpty(telephone))

{

APIClient.SetErrorMessage("Проверьте правильность заполнения полей");

Response.Redirect("ErrorPage");

}

APIClient.PostRequest("api/Client/Register", new ClientBindingModel

{

Name = name,

Surname = surname,

Patronymic = patronymic,

Email = login,

Password = password,

Telephone = telephone

});

Response.Redirect("Enter");

return;

}

[HttpPost]

public IActionResult Logout()

{

APIClient.Client = null;

return Redirect("~/Home/Enter");

}

[HttpGet]

public IActionResult Privacy()

{

if (APIClient.Client == null)

{

return Redirect("~/Home/Enter");

}

return View(APIClient.Client);

}

[HttpPost]

public void Privacy(string login, string password, string name, string surname, string patronymic, string telephone)

{

if (APIClient.Client == null)

{

throw new Exception("Вы как сюда попали? Суда вход только авторизованным");

}

if (string.IsNullOrEmpty(login) || string.IsNullOrEmpty(password) || string.IsNullOrEmpty(name)

|| string.IsNullOrEmpty(surname) || string.IsNullOrEmpty(patronymic)

|| string.IsNullOrEmpty(telephone))

{

APIClient.SetErrorMessage("Проверьте правильность заполнения полей");

Response.Redirect("ErrorPage");

}

APIClient.PostRequest("/api/Client/UpdateData", new ClientBindingModel

{

Id = APIClient.Client.Id,

Name = name,

Surname = surname,

Patronymic = patronymic,

Telephone = telephone,

Email = login,

Password = password

});

APIClient.Client.Name = name;

APIClient.Client.Surname = surname;

APIClient.Client.Patronymic = patronymic;

APIClient.Client.Email = login;

APIClient.Client.Password = password;

APIClient.Client.Telephone = telephone;

Response.Redirect("Enter");

}

#endregion

#region Карты

[HttpGet]

public IActionResult CardsList()

{

if (APIClient.Client == null)

{

return Redirect("~/Home/Enter");

}

return View(APIClient.GetRequest<List<CardViewModel>>($"api/Card/GetUsersCardsList?id={APIClient.Client.Id}"));

}

[HttpGet]

public IActionResult CreateCard()

{

if (APIClient.Client == null)

{

return Redirect("~/Home/Enter");

}

ViewBag.Accounts = APIClient.GetRequest<List<AccountViewModel>>($"api/Account/SearchAccountsOfCLient?clientId={APIClient.Client.Id}");

return View();

}

[HttpPost]

public IActionResult CreateCard(string accountId, string number, string cvc, DateTime period)

{

if (APIClient.Client == null)

{

APIClient.SetErrorMessage("Необходимо авторизоваться");

return Redirect("ErrorPage");

}

if (string.IsNullOrEmpty(accountId) || string.IsNullOrEmpty(number) || string.IsNullOrEmpty(cvc)

|| period.Year == 0001 || period <= DateTime.Now)

{

APIClient.SetErrorMessage("Проверьте корректность параметров создаваемой карты");

return Redirect("ErrorPage");

}

APIClient.PostRequest("api/Card/CreateCard", new CardBindingModel

{

ClientID = APIClient.Client.Id,

AccountId = int.Parse(accountId),

Number = number,

CVC = cvc,

Period = period

});

return Redirect("~/Home/CardsList");

}

#endregion

#region Снятие средств

[HttpGet]

public IActionResult DebitingList()

{

if (APIClient.Client == null)

{

return Redirect("~/Home/Enter");

}

return View(APIClient.GetRequest<List<DebitingViewModel>>($"api/Client/getUsersDebitings?userId={APIClient.Client.Id}"));

}

[HttpGet]

public IActionResult CreateDebiting()

{

if (APIClient.Client == null)

{

return Redirect("~/Home/Enter");

}

ViewBag.Cards = APIClient.GetRequest<List<CardViewModel>>($"api/Card/GetUsersCardsList?id={APIClient.Client.Id}");

return View();

}

[HttpPost]

public IActionResult CreateDebiting(string cardId, int sum)

{

if (APIClient.Client == null)

{

APIClient.SetErrorMessage("Необходимо авторизоваться");

return Redirect("ErrorPage");

}

if (string.IsNullOrEmpty(cardId) || sum <= 0)

{

APIClient.SetErrorMessage("Необходимо ввести корректную сумму для снятия");

return Redirect("ErrorPage");

}

APIClient.PostRequest("api/Card/CreateDebitingRequest", new DebitingBindingModel()

{

CardId = int.Parse(cardId),

Sum = sum,

DateOpen = DateTime.Now,

Status = StatusEnum.Открыта

});

return Redirect("~/Home/DebitingList");

}

#endregion

#region Пополнение средств

[HttpGet]

public IActionResult CreditingList()

{

if (APIClient.Client == null)

{

return Redirect("~/Home/Enter");

}

return View(APIClient.GetRequest<List<CreditingViewModel>>($"api/Client/getUsersCreditings?userId={APIClient.Client.Id}"));

}

[HttpGet]

public IActionResult CreateCrediting()

{

if (APIClient.Client == null)

{

return Redirect("~/Home/Enter");

}

ViewBag.Cards = APIClient.GetRequest<List<CardViewModel>>($"api/Card/GetUsersCardsList?id={APIClient.Client.Id}");

return View();

}

[HttpPost]

public IActionResult CreateCrediting(string cardId, int sum)

{

if (APIClient.Client == null)

{

throw new Exception("Не авторизованы");

}

if (string.IsNullOrEmpty(cardId) || sum <= 0)

{

APIClient.SetErrorMessage("Необходимо ввести корректную сумму для пополнения");

return Redirect("ErrorPage");

}

APIClient.PostRequest("api/Card/CreateCreditingOperation", new CreditingBindingModel()

{

CardId = int.Parse(cardId),

Sum = sum,

DateOpen = DateTime.Now,

Status = StatusEnum.Открыта

});

return Redirect("~/Home/CreditingList");

}

#endregion

#region Получение отчёта PDF

[HttpGet]

public IActionResult CreateReport()

{

if (APIClient.Client == null)

{

return Redirect("~/Home/Enter");

}

return View();

}

[HttpPost]

public IActionResult CreateReport(DateTime dateFrom, DateTime dateTo)

{

if (APIClient.Client == null)

{

throw new Exception("Не авторизованы");

}

if (dateFrom == dateTo || dateFrom > dateTo || dateFrom.Year == 0001 || dateTo.Year == 0001)

{

APIClient.SetErrorMessage("Необходимо задать корректные границы периода");

return Redirect("ErrorPage");

}

return View(APIClient.PostRequestReport<ReportClientViewModelForHTML, ReportSupportBindingModel>("api/Report/CreateClientReport", new ReportSupportBindingModel()

{

DateFrom = dateFrom,

DateTo = dateTo,

Email = APIClient.Client.Email

}));

}

#endregion

#region Excel отчёты

//отчёт клиента по переводам

[HttpPost]

public IActionResult CreateExcelReport(List<CheckboxViewModel> cards)

{

if (APIClient.Client == null)

{

APIClient.SetErrorMessage("Не авторизованы");

return Redirect("ErrorPage");

}

if (cards.Count == 0 || cards.Count == cards.Where(x => x.IsChecked == false).ToList().Count)

{

APIClient.SetErrorMessage("Необходимо выбрать хотя-бы 1 карту для отчёта");

return Redirect("ErrorPage");

}

APIClient.PostRequest("api/Report/CreateExcelClient", new ReportSupportBindingModel()

{

CardList = cards.Where(x => x.IsChecked).Select(x => x.Id).ToList(),

Email = APIClient.Client.Email

});

return Redirect("ReportSuccess");

}

//отчёт клиента по пополнениям

[HttpPost]

public IActionResult CreateCreditingExcelReport(List<CheckboxViewModel> cards)

{

if (APIClient.Client == null)

{

APIClient.SetErrorMessage("Не авторизованы");

return Redirect("ErrorPage");

}

if (cards.Count == 0 || cards.Count == cards.Where(x => x.IsChecked == false).ToList().Count)

{

APIClient.SetErrorMessage("Необходимо выбрать хотя-бы 1 карту для отчёта");

return Redirect("ErrorPage");

}

APIClient.PostRequest("api/Report/CreateExcelCrediting", new ReportSupportBindingModel()

{

CardList = cards.Where(x => x.IsChecked).Select(x => x.Id).ToList(),

Email = APIClient.Client.Email

});

return Redirect("ReportSuccess");

}

//отчёт клиента по снятиям

[HttpPost]

public IActionResult CreateDebitingExcelReport(List<CheckboxViewModel> cards)

{

if (APIClient.Client == null)

{

APIClient.SetErrorMessage("Не авторизованы");

return Redirect("ErrorPage");

}

if (cards.Count == 0 || cards.Count == cards.Where(x => x.IsChecked == false).ToList().Count)

{

APIClient.SetErrorMessage("Необходимо выбрать хотя-бы 1 карту для отчёта");

return Redirect("ErrorPage");

}

APIClient.PostRequest("api/Report/CreateExcelDebiting", new ReportSupportBindingModel()

{

CardList = cards.Where(x => x.IsChecked).Select(x => x.Id).ToList(),

Email = APIClient.Client.Email

});

return Redirect("ReportSuccess");

}

#endregion

#region Word отчёты клиента

//отчёт клиента по переводам

[HttpPost]

public IActionResult CreateWordReport(List<CheckboxViewModel> cards)

{

if (APIClient.Client == null)

{

APIClient.SetErrorMessage("Не авторизованы");

return Redirect("ErrorPage");

}

if (cards.Count == 0 || cards.Count == cards.Where(x => x.IsChecked == false).ToList().Count)

{

APIClient.SetErrorMessage("Необходимо выбрать хотя-бы 1 карту для отчёта");

return Redirect("ErrorPage");

}

APIClient.PostRequest("api/Report/CreateWordClient", new ReportSupportBindingModel()

{

CardList = cards.Where(x => x.IsChecked).Select(x => x.Id).ToList(),

Email = APIClient.Client.Email

});

return Redirect("ReportSuccess");

}

//отчёт клиента по пополнениям

[HttpPost]

public IActionResult CreateCreditingWordReport(List<CheckboxViewModel> cards)

{

if (APIClient.Client == null)

{

APIClient.SetErrorMessage("Не авторизованы");

return Redirect("ErrorPage");

}

if (cards.Count == 0 || cards.Count == cards.Where(x => x.IsChecked == false).ToList().Count)

{

APIClient.SetErrorMessage("Необходимо выбрать хотя-бы 1 карту для отчёта");

return Redirect("ErrorPage");

}

APIClient.PostRequest("api/Report/CreateWordCrediting", new ReportSupportBindingModel()

{

CardList = cards.Where(x => x.IsChecked).Select(x => x.Id).ToList(),

Email = APIClient.Client.Email

});

return Redirect("ReportSuccess");

}

//отчёт клиента по снятиям

[HttpPost]

public IActionResult CreateDebitingWordReport(List<CheckboxViewModel> cards)

{

if (APIClient.Client == null)

{

APIClient.SetErrorMessage("Не авторизованы");

return Redirect("ErrorPage");

}

if (cards.Count == 0 || cards.Count == cards.Where(x => x.IsChecked == false).ToList().Count)

{

APIClient.SetErrorMessage("Необходимо выбрать хотя-бы 1 карту для отчёта");

return Redirect("ErrorPage");

}

APIClient.PostRequest("api/Report/CreateWordDebiting", new ReportSupportBindingModel()

{

CardList = cards.Where(x => x.IsChecked).Select(x => x.Id).ToList(),

Email = APIClient.Client.Email

});

return Redirect("ReportSuccess");

}

#endregion

#region Получение отчета по картам

[HttpGet]

public IActionResult ReportWithCards()

{

if (APIClient.Client == null)

{

return Redirect("~/Home/Enter");

}

return View(new ReportClientCardsViewModel()

{

Cards = APIClient.GetRequest<List<CardViewModel>>($"api/Card/GetUsersCardsList?id={APIClient.Client.Id}").Select(x => new CheckboxViewModel()

{

Id = x.Id,

LabelName = x.Number,

IsChecked = false

}).ToList()

});

}

[HttpPost]

public IActionResult ReportWithCards(List<CheckboxViewModel> cards)

{

if (APIClient.Client == null)

{

APIClient.SetErrorMessage("Не авторизованы");

return Redirect("ErrorPage");

}

if (cards.Count == 0 || cards.Count == cards.Where(x => x.IsChecked == false).ToList().Count)

{

APIClient.SetErrorMessage("Необходимо выбрать хотя-бы 1 карту для отчёта");

return Redirect("ErrorPage");

}

List<int> cardList = cards.Where(x => x.IsChecked).Select(x => x.Id).ToList();

List<ReportViewModel> creditings = APIClient.GetRequest<List<CreditingViewModel>>($"api/Client/getUsersCreditings?userId={APIClient.Client.Id}")

.Where(x => cardList.Contains(x.CardId)).Select(x => new ReportViewModel()

{

Id = x.Id,

CardId = x.CardId,

DateOpen = x.DateOpen,

DateClose = x.DateClose,

CardNumber = x.CardNumber,

Status = x.Status,

Sum = x.Sum,

TypeOperation = TypeOperationEnum.Пополнение

}).ToList();

List<ReportViewModel> debitings = APIClient.GetRequest<List<DebitingViewModel>>($"api/Client/getUsersDebitings?userId={APIClient.Client.Id}")

.Where(x => cardList.Contains(x.CardId)).Select(x => new ReportViewModel()

{

Id = x.Id,

CardId = x.CardId,

DateOpen = x.DateOpen,

DateClose = x.DateClose,

CardNumber = x.CardNumber,

Status = x.Status,

Sum = x.Sum,

TypeOperation = TypeOperationEnum.Снятие

}).ToList();

List<ReportViewModel> result = creditings.Concat(debitings).OrderBy(x => x.DateOpen).ToList();

return View(new ReportClientCardsViewModel()

{

Cards = cards,

Operations = result,

});

}

#endregion

#region Диаграмма

[HttpGet]

public IActionResult Diagram()

{

if (APIClient.Client == null)

{

return Redirect("~/Home/Enter");

}

ViewBag.Cards = APIClient.GetRequest<List<CardViewModel>>($"api/Card/GetUsersCardsList?id={APIClient.Client.Id}");

return View();

}

[HttpPost]

public IActionResult Diagram(int cardId)

{

if (APIClient.Client == null)

{

APIClient.SetErrorMessage("Не авторизованы");

return Redirect("ErrorPage");

}

ViewBag.Cards = APIClient.GetRequest<List<CardViewModel>>($"api/Card/GetUsersCardsList?id={APIClient.Client.Id}");

return View(new ClientDiagramViewModel()

{

DiagramName = "Hello World",

Elements = APIClient.GetRequest<List<ClientDiagramElementsViewModel>>($"api/Card/getCardMonthResult?cardId={cardId}")

});

}

#endregion

//сообщение об успешной отправке отчёта на почту

[HttpGet]

public IActionResult ReportSuccess()

{

if (APIClient.Client == null)

{

APIClient.SetErrorMessage("Не авторизованы");

return Redirect("ErrorPage");

}

return View();

}

}

}

Папка Models

ErrorViewModel.cs

namespace BankClientApp.Models

{

public class ErrorViewModel

{

public string? RequestId { get; set; }

public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);

}

}

Папка Properties

launchSettings.json

{

"iisSettings": {

"windowsAuthentication": false,

"anonymousAuthentication": true,

"iisExpress": {

"applicationUrl": "http://localhost:27609",

"sslPort": 44302

}

},

"profiles": {

"BankClientApp": {

"commandName": "Project",

"dotnetRunMessages": true,

"launchBrowser": true,

"applicationUrl": "https://localhost:7125;http://localhost:5201",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

},

"IIS Express": {

"commandName": "IISExpress",

"launchBrowser": true,

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

}

}

}

appsettings.Development.json

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

}

}

Папка Views

Папка Shared

@using BankСlientApp

@{

bool authenticated = APIClient.Client != null;

}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>@ViewData["Title"]</title>

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />

<link rel="stylesheet" href="~/css/site.css" />

<script src="~/lib/jquery/dist/jquery.min.js"></script>

<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

<style>

body {

margin-bottom: 60px;

background-color: white;

}

.nav-main a {

position: relative;

color: #FFFFFF;

cursor: pointer;

line-height: 1;

text-decoration: none;

}

.nav-main a:after {

display: block;

position: absolute;

left: 0;

bottom: 0px;

width: 0;

height: 1px;

background-color: #FFFFFF;

content: "";

transition: width 0.3s ease-out;

}

.nav-main a:hover:after,

.nav-main a:focus:after {

width: 100%;

}

.nav-main .dropdown:hover .dropdown-menu {

display: block;

margin-top: 0;

}

.footer {

position: fixed;

left: 0;

bottom: 0;

width: 100%;

height: 80px;

background-color: #212529

}

.form-signin {

width: 100%;

max-width: 330px;

padding: 15px;

margin: auto;

}

.form-signin .form-control {

position: relative;

box-sizing: border-box;

height: auto;

padding: 10px;

font-size: 16px;

}

.form-signin .form-control:focus {

z-index: 2;

}

.form-signin input[type="email"] {

margin-bottom: -1px;

border-bottom-right-radius: 0;

border-bottom-left-radius: 0;

}

.form-signin input[type="password"] {

margin-bottom: 10px;

border-top-left-radius: 0;

border-top-right-radius: 0;

}

table.table tbody tr td,

table.table thead tr th,

table.table thead {

border-left: solid;

border-right: solid;

border-width: 4px;

border-color: #212529;

}

table {

vertical-align: middle;

}

</style>

</head>

<body class="MyBody">

<header class="d-flex flex-wrap align-items-center justify-content-center justify-content-md-between py-3 px-4 mg-5 mb-4 border-bottom bg-dark ">

<a asp-controller="Home" asp-action="Enter">

<img src="https://img.icons8.com/?size=80&id=CvryVUzkqqMu&format=png" alt="\*" class="navbar-toggler" style="display: block; margin: 0 auto; height:">

</a>

<div class="md-4 mb-2 mb-md-0">

<a asp-controller="Home" asp-action="Enter" class="d-inline-flex link-body-emphasis text-decoration-none">

<span class="fs-4 text-light ">Банк "Вы банкрот"</span>

</a>

</div>

<ul class="nav col-md-auto mb-2 justify-content-center mb-md-0 nav-main">

<li>

<a class="nav-link px-2 link-light" asparea="" asp-controller="Home" asp-action="CardsList">Банковские карты</a>

</li>

<li class="dropdown">

<a href="#" class="nav-link px-2 link-light">Операции</a>

<ul class="dropdown-menu dropdown-menu-dark" aria-labelledby="navbarDarkDropdownMenuLink">

<li><a class="dropdown-item" asp-controller="Home" asp-action="DebitingList">Заявки на снятие</a></li>

<li><a class="dropdown-item" asp-controller="Home" asp-action="CreditingList">Заявки на начисление</a></li>

</ul>

</li>

<li class="dropdown">

<a href="#" class="nav-link px-2 link-light">Отчеты</a>

<ul class="dropdown-menu dropdown-menu-dark" aria-labelledby="navbarDarkDropdownMenuLink">

<li><a class="dropdown-item" asp-controller="Home" asp-controller="Home" asp-action="ReportWithCards">Отчёт по банковским картам</a></li>

<li><a class="dropdown-item" asp-controller="Home" asp-controller="Home" asp-action="CreateReport">Отчёт за период</a></li>

<li><a class="dropdown-item" asp-controller="Home" asp-controller="Home" asp-action="Diagram">Диаграмма</a></li>

</ul>

</li>

</ul>

@{

if (APIClient.Client == null)

{

<div class="col-md-3 text-end">

<a class="btn btn-warning me-2" asp-controller="Home" asp-action="Login">Войти</a>

<a class="btn btn-warning" asp-controller="Home" asp-action="Register">Регистрация</a>

</div>

}

else

{

<div class="col-md-3 text-end">

<a class="btn btn-warning me-2" id="exit" name="exit" asp-controller="Home" asp-action="Privacy">@APIClient.Client.Surname @APIClient.Client.Name</a>

</div>

}

}

</header>

<div class="container">

<main role="main" class="pb-3">

@RenderBody()

</main>

</div>

<footer class="border-top footer text-muted mx-auto">

<div class="container mx-auto text-center mt-4 text-white">

&copy; 2024 - Банк "Вы Банкротищеее" - <a asp-area="" asp-controller="Home" asp-action="Privacy">Личные данные</a>

</div>

</footer>

<script src="~/lib/jquery/dist/jquery.min.js"></script>

<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

<script src="~/js/site.js" asp-append-version="true"></script>

@await RenderSectionAsync("Scripts", required: false)

</body>

</html>

Папка Home

CardsList.cshtml

@using BankContracts.ViewModels.Client.ViewModels

@model List<CardViewModel>

@{

ViewData["Title"] = "Список карт";

}

<div class="text-center">

<h1 class="display-4">Карты</h1>

</div>

<div class="text-center">

@{

if (Model == null)

{

<h3 class="display-4">Сначала авторизируйтесь</h3>

return;

}

<p>

<a asp-action="CreateCard">Создать карту</a>

</p>

<table class="table">

<thead>

<tr>

<th>

Номер карты

</th>

<th>

Баланс

</th>

<th>

CVC

</th>

<th>

Срок действия

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Number)

</td>

<td>

@Html.DisplayFor(modelItem => item.Sum)

</td>

<td>

@Html.DisplayFor(modelItem => item.CVC)

</td>

<td>

@Html.DisplayFor(modelItem => item.Period)

</td>

</tr>

}

</tbody>

</table>

}

</div>

CreateCard.cshtml

@{

ViewData["Title"] = "Создание карты";

}

<div class="text-center">

<h2 class="display-4">Создание карты</h2>

</div>

<form method="post">

<div class="row mb-2">

<div class="col-4">Номер счета:</div>

<div class="col-8">

<select id="accountId" name="accountId" class="form-control" asp-items="@(new SelectList( @ViewBag.Accounts, "Id", "AccountNumber"))"></select>

</div>

</div>

<div class="row mb-2">

<div class="col-4">Номер карты:</div>

<div class="col-8">

<input type="text" class="form-control" name="number" id="number" required />

</div>

</div>

<div class="row mb-2">

<div class="col-4">CVC:</div>

<div class="col-8">

<input type="text" class="form-control" name="cvc" id="cvc" required />

</div>

</div>

<div class="row mb-2">

<div class="col-4">Срок действия:</div>

<div class="col-8">

<input type="date" class="form-control" name="period" id="period" required />

</div>

</div>

<div class="row mb-2">

<input type="submit" value="Создание" style="width: 100%" class="btn btn-warning" />

</div>

</form>

<script>

function createNum(len) {

chrs = '0123456789';

var str = '';

for (var i = 0; i < len; i++) {

var pos = Math.floor(Math.random() \* chrs.length);

str += chrs.substring(pos, pos + 1);

}

return str;

}

document.getElementById("number").value = createNum(16);

document.getElementById("cvc").value = createNum(3);

let year = new Date();

year.setFullYear(year.getFullYear() + 5)

document.getElementById("period").valueAsDate = new Date(year);

</script>

CreateCrediting.cshtml

@{

ViewData["Title"] = "Операция пополнения";

}

<div class="text-center">

<h2 class="display-4">Создание операции</h2>

</div>

<form method="post">

<div class="row mb-2">

<div class="col-4">Номер карты:</div>

<div class="col-8">

<select id="cardId" name="cardId" class="form-control" asp-items="@(new SelectList( @ViewBag.Cards, "Id", "Number"))"></select>

</div>

</div>

<div class="row mb-2">

<div class="col-4">Cумма операции:</div>

<div class="col-8">

<input type="number" class="form-control" name="sum" required autofocus/>

</div>

</div>

<div class="row mb-2">

<input type="submit" value="Создание" style="width: 100%" class="btn btn-warning" />

</div>

</form>

CreateDebiting.cshtml

@{

ViewData["Title"] = "Операция снятия";

}

<div class="text-center">

<h2 class="display-4">Создание операции</h2>

</div>

<form method="post">

<div class="row mb-2">

<div class="col-4">Номер карты:</div>

<div class="col-8">

<select id="cardId" name="cardId" class="form-control" asp-items="@(new SelectList( @ViewBag.Cards, "Id", "Number"))"></select>

</div>

</div>

<div class="row mb-2">

<div class="col-4">Cумма операции:</div>

<div class="col-8">

<input type="number" name="sum" class="form-control" required autofocus />

</div>

</div>

<div class="row mb-2">

<input type="submit" style="width: 100%" value="Создание" class="btn btn-warning" />

</div>

</form>

CreateExcelReport.cshtml

@{

ViewData["Title"] = "Создание Excel отчёта";

}

<div class="text-center">

<h2 class="display-4">Создание отчёта</h2>

</div>

<form method="post">

<div class="row">

<div class="col-4">Выберите карту:</div>

<div class="col-8">

<select id="cardId" name="cardId" class="form-control" asp-items="@(new SelectList( @ViewBag.Cards, "Id", "Number"))"></select>

</div>

<div class="row">

<div class="col-8"></div>

<div class="col-4">

<input type="submit" value="Создание" class="btn btn-warning" />

</div>

</div>

</form>

CreateReport.cshtml

@using BankContracts.ViewModels;

@using BankContracts.ViewModels.Reports

@using BankСlientApp

@model ReportClientViewModelForHTML

@{

ViewData["Title"] = "Создание отчёта";

}

<div class="text-center">

<h2 class="display-4">Отчёт по картам за выбранный период</h2>

</div>

<form method="post">

<div class="row mb-2">

<div class="col-4">Дата начала периода:</div>

<div class="col-8">

<input id="dateFrom" name="dateFrom" class="form-control" type="date" required />

</div>

</div>

<div class="row mb-2">

<div class="col-4">Дата конца периода:</div>

<div class="col-8">

<input id="dateTo" name="dateTo" class="form-control" type="date" required />

</div>

</div>

<div class="row">

<input id="createReport" style="width:100%;" type="submit" value="Сформировать отчёт" class="btn btn-warning" />

</div>

<hr class="mt-5 mb-3" />

@if (Model != null)

{

<div class="row text-center">

<h3>Отчет отправлен на почту @APIClient.Client.Email</h3>

<hr class="mt-5 mb-3" />

<p>Отчёт по пополнениям</p>

<table class="table">

<thead>

<tr>

<th>

Номер операции

</th>

<th>

Номер карты

</th>

<th>

Сумма

</th>

<th>

Дата операции

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.ReportCrediting)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.OperationId)

</td>

<td>

@Html.DisplayFor(modelItem => item.CardNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.SumOperation)

</td>

<td>

@Html.DisplayFor(modelItem => item.DateComplite)

</td>

</tr>

}

</tbody>

</table>

</div>

<hr class="my-12" />

<div class="row text-center">

<p>Отчёт по снятиям</p>

<table class="table">

<thead>

<tr>

<th>

Номер операции

</th>

<th>

Номер карты

</th>

<th>

Сумма

</th>

<th>

Дата операции

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.ReportDebiting)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.OperationId)

</td>

<td>

@Html.DisplayFor(modelItem => item.CardNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.SumOperation)

</td>

<td>

@Html.DisplayFor(modelItem => item.DateComplite)

</td>

</tr>

}

</tbody>

</table>

</div>

}

</form>

CreditingList.cshtml

@using BankContracts.ViewModels.Client.ViewModels

@model List<CreditingViewModel>

@{

ViewData["Title"] = "Операция пополнения";

}

<div class="text-center">

<h1 class="display-4">Операция пополнения</h1>

</div>

<div class="text-center">

@{

if (Model == null)

{

<h3 class="display-4">Сначала авторизируйтесь</h3>

return;

}

<p>

<a asp-action="CreateCrediting">Пополнить средства</a>

</p>

<table class="table">

<thead>

<tr>

<th>

Номер карты

</th>

<th>

Сумма

</th>

<th>

Статус

</th>

<th>

Дата открытия

</th>

<th>

Дата закрытия

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.CardNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.Sum)

</td>

<td>

@item.Status.ToString().Replace("\_", " ")

</td>

<td>

@Html.DisplayFor(modelItem => item.DateOpen)

</td>

<td>

@Html.DisplayFor(modelItem => item.DateClose)

</td>

</tr>

}

</tbody>

</table>

}

</div>

DebitingList.cshtml

@using BankContracts.ViewModels.Client.ViewModels;

@model List<DebitingViewModel>

@{

ViewData["Title"] = "Операция снятия";

}

<div class="text-center">

<h1 class="display-4">Операция снятия</h1>

</div>

<div class="text-center">

@{

if (Model == null)

{

<h3 class="display-4">Сначала авторизируйтесь</h3>

return;

}

<p>

<a asp-action="CreateDebiting">Снять средства</a>

</p>

<table class="table">

<thead>

<tr>

<th>

Номер карты

</th>

<th>

Сумма

</th>

<th>

Статус

</th>

<th>

Дата открытия

</th>

<th>

Дата закрытия

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.CardNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.Sum)

</td>

<td>

@item.Status.ToString().Replace("\_", " ")

</td>

<td>

@Html.DisplayFor(modelItem => item.DateOpen)

</td>

<td>

@Html.DisplayFor(modelItem => item.DateClose)

</td>

</tr>

}

</tbody>

</table>

}

</div>

Diagram.cshtml

@using BankContracts.ViewModels.Client.Diagram

@model ClientDiagramViewModel

@{

ViewData["Title"] = "Диаграмма";

}

<div class="text-center">

<h1 class="display-4">Диаграмма финансов на карте по месяцам</h1>

</div>

<form method="post">

<div class="row mb-2">

<div class="row">Номер карты:</div>

<div class="col">

<select id="cardId" name="cardId" class="form-control" asp-items="@(new SelectList( @ViewBag.Cards, "Id", "Number"))"></select>

</div>

</div>

<div class="row mb-2">

<div class="col">

<input style="width: 100%" type="submit" value="Выбрать" class="btn btn-warning"/>

</div>

</div>

</form>

@if (Model == null) return;

<div id="Diagrams" class="text-center">

<div id="@Model.DiagramName Diagram">

<canvas id="Chart"></canvas>

<div id="params">

@foreach (var info in Model.Elements) {

<input type="hidden" id="@info.Name" value="@info.Value" />

}

</div>

</div>

</div>

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<script>

const diagrams = document.getElementById('Diagrams').childNodes;

let diagram\_name = diagrams[1].id;

console.log(diagram\_name);

let diagram = document.getElementById(diagram\_name).childNodes;

console.log(diagram);

let labels = [];

let data = [];

document.getElementById('params').childNodes.forEach(element => {

if (element.id != undefined) {

labels.push(element.id);

}

});

document.getElementById('params').childNodes.forEach(element => {

if (element.id != undefined) {

data.push(Number(element.value));

}

});

new Chart(diagram.item(1), {

type: 'bar',

data: {

labels: labels,

datasets: [{

label: 'Денег в этом месяце',

data: data,

borderWidth: 6,

backgroundColor: 'rgb(255, 165, 0)'

}]

},

options: {

plugins: {

legend: {

display: false

},

customCanvasBackgroundColor: {

color: 'white',

}

},

scales: {

y: {

suggestedMin: Math.min(data) - Math.min(data) \* -0.1,

suggestedMax: Math.max(data) + Math.max(data) \* 0.1,

}

}

}

}

);

</script>

Enter.cshtml

@using BankСlientApp

@{

ViewData["Title"] = "Страница пользователя";

}

<div class="text-center">

<h1 class="display-4">Страница пользователя</h1>

</div>

<div class="text-center">

@{

<img src="https://i1.ytimg.com/vi/nNQemfCR9Ms/maxresdefault.jpg" alt="\*" class="img-fluid" style="width: 70%; display: block; margin: 0 auto;">

if (APIClient.Client == null)

{

<h3 class="display-4">Сначала авторизируйтесь</h3>

return;

}

<h3 class="display-4">Здравствуйтe, @APIClient.Client.Name @APIClient.Client.Patronymic</h3>

}

</div>

ErrorPage.cshtml

@using BankСlientApp

@{

ViewData["Title"] = "Отправка отчета";

}

<div class="text-center p-5">

<h3 class="display-4">Упс, что-то пошло не так...</h3>

<h3 class="display-4">Ошибка: @APIClient.ErrorMessage</h3>

</div>

Index.cshtml

@using BankСlientApp

@{

ViewData["Title"] = "Добро пожаловать";

}

<div class="text-center">

<h1 class="display-4">Добро пожаловать</h1>

</div>

<div class="text-center mx-auto">

<h2 class="text-center"> Страница пользователя </h2>

<img src="https://i1.ytimg.com/vi/nNQemfCR9Ms/maxresdefault.jpg" alt="\*" class="img-fluid" style="width: 70%; display: block; margin: 0 auto;">

@{

if (APIClient.Client == null)

{

<h3 class="display-3" style="font-size: 50px"> Сначала авторизуйся </h3>

return;

}

<h3 class="display-3"> Здравствуйте, @APIClient.Client.Name @APIClient.Client.Patronymic </h3>

}

</div>

Login.cshtml

@{

ViewData["Title"] = "Логин";

}

<div class="text-center">

<h2 class="display-4">Вход в приложение</h2>

</div>

<form class="form-signin text-center" method="post">

<h1 class="h3 mb-3 font-weight-normal">Логин</h1>

<input type="email" id="login" name="login" class="form-control" placeholder="Почта" required autofocus>

<input type="password" id="password" name="password" class="form-control" placeholder="Пароль" required>

<button class="btn btn-lg btn-warning btn-block" type="submit" asp-controller="Home" asp-action="Login">Войти</button>

</form>

MoneyTransfers.cshtml

@{

ViewData["Title"] = "Переводы меджу счетами";

}

<div class="text-center">

<h2 class="display-4">Перевод между счетами</h2>

</div>

<form class="mx-auto w-75" method="post">

<div class="row mt-5 mb-2">

<div class="col-4">Номер счёта для снятия:</div>

<div class="col-8">

</div>

</div>

<div class="row mt-4 mb-2">

<div class="col-4">Номер счёта для начисления:</div>

<div class="col-8">

</div>

</div>

<div class="row mt-4 mb-2">

<div class="col-4 h3">Сумма перевода:</div>

<div class="col-8">

<input type="number" id="sumMoneyTransfer" class="form-control" name="sumMoneyTransfer" required min=1 value=1 />

</div>

</div>

<div class="row mt-4 mb-2">

<input type="submit" style="width: 100%" value="Перевести" class="btn btn-dark" />

</div>

</form>

Privacy.cshtml

@using BankContracts.ViewModels.Client.ViewModels

@model ClientViewModel

@{

ViewData["Title"] = "Личный кабинет";

}

<div class="text-center">

<h2 class="display-4">Личные данные</h2>

</div>

<form method="post" class="form-signin">

<div class="row">

<div class="col-4">Логин:</div>

<input type="email" id="login" name="login" class="form-control" placeholder="Почта" value=@Html.DisplayFor(modelItem => Model.Email) required>

</div>

<div class="row">

<div class="col-4">Пароль:</div>

<input type="password" id="password" name="password" class="form-control" placeholder="Пароль" value=@Html.DisplayFor(modelItem => Model.Password) required>

</div>

<div class="row">

<div class="col-4">Имя:</div>

<input type="text" id="name" name="name" class="form-control" placeholder="Имя" value=@Html.DisplayFor(modelItem => Model.Name) required>

<div class="col-4">Фамилия:</div>

<input type="text" id="surname" name="surname" class="form-control" placeholder="Фамилия" value=@Html.DisplayFor(modelItem => Model.Surname) required>

<div class="col-4">Отчество:</div>

<input type="text" id="patronymic" name="patronymic" class="form-control" placeholder="Отчество" value=@Html.DisplayFor(modelItem => Model.Patronymic) required>

</div>

<div class="row mb-2">

<div class="col-4">Телефон:</div>

<input type="text" id="telephone" name="telephone" class="form-control" placeholder="Телефон" value=@Html.DisplayFor(modelItem => Model.Telephone) required>

</div>

<div class="row mb-2">

<button class="btn btn-lg btn-warning btn-block mb-2" type="submit" asp-controller="Home" asp-action="Privacy">Coхранить</button>

<button class="btn btn-lg btn-warning btn-block" type="submit" asp-controller="Home" asp-action="Logout">Выйти из аккаунта</button>

</div>

</form>

Register.cshtml

@{

ViewData["Title"] = "Регистрация";

}

<div class="text-center">

<h2 class="display-4">Регистрация</h2>

</div>

<form class="form-signin text-center" method="post">

<h1 class="h3 mb-3 font-weight-normal">Регистрация</h1>

<input type="email" id="login" name="login" class="form-control" placeholder="Почта" required>

<input type="password" id="password" name="password" class="form-control" placeholder="Пароль" required>

<input type="text" id="name" name="name" class="form-control" placeholder="Имя" required>

<input type="text" id="surname" name="surname" class="form-control" placeholder="Фамилия" required>

<input type="text" id="patronymic" name="patronymic" class="form-control" placeholder="Отчество" required>

<input type="text" id="telephone" name="telephone" class="form-control" placeholder="Телефон" required>

<button class="btn btn-lg btn-warning btn-block" type="submit" asp-controller="Home" asp-action="Register">Регистрация</button>

</form>

ReportSuccess.cshtml

@using BankСlientApp

@{

ViewData["Title"] = "Отправка отчета";

}

<div class="text-center">

@{

if (APIClient.Client == null)

{

<h3 class="display-4">Сначала авторизируйтесь</h3>

return;

}

}

<h3 class="display-4">Отчeт был отправлен на почту @APIClient.Client.Email</h3>

</div>

ReportWithCards.cshtml

@using BankContracts.ViewModels.Reports.Client

@using Microsoft.JSInterop;

@inject IJSRuntime JS

@model ReportClientCardsViewModel

@{

ViewData["Title"] = "Отчет по картам";

}

<div class="text-center">

<h1 class="display-4">Отчет</h1>

</div>

<div class="container" sf-type="container" sf-label="Bootstrap Container" sf-uid="2">

<div class="row" sf-type="container" sf-label="Row" sf-uid="3">

<div class="mb-4 mb-md-0 aos-init aos-animate col-md-3" sf-type="container" sf-label="Column" sf-anim-delay="1.5" data-aos="fade-down" data-aos-delay="400" sf-uid="4">

<div sf-type="container" sf-label="Container" class="py-15 h-100 bg-bg-2" sf-uid="5">

<form method="post">

<h3>Карты:</h3>

@for (var item = 0; item < @Model.Cards.Count(); item++)

{

<div class="form-check form-switch">

<input class="form-check-input" type="checkbox" id="flexSwitchCheckDefault" asp-for="@Model.Cards[item].IsChecked">

<label class="form-check-label" for="flexSwitchCheckDefault">@Model.Cards[item].LabelName</label>

<input type="hidden" asp-for="@Model.Cards[item].Id" />

<input type="hidden" asp-for="@Model.Cards[item].LabelName" />

</div>

}

<hr>

<div class="mb-2">

<button style="width:100%" class="btn btn-lg btn-warning btn-block" type="submit" asp-controller="Home" asp-action="ReportWithCards">Создать отчёт</button>

</div>

<hr/>

<div class="mb-2">

<button type="button" id="ExcelBut" class="btn btn-lg btn-warning btn-block">Excel отчеты</button>

</div>

<div id="ExcelDiv" style="display: none">

<div class="mb-2" >

<button style="width:100%" class="btn btn-lg btn-warning btn-block" type="submit" asp-controller="Home" asp-action="CreateExcelReport">Создать отчёт по переводам (EXCEL)</button>

</div>

<div class="mb-2">

<button style="width:100%" class="btn btn-lg btn-warning btn-block" type="submit" asp-controller="Home" asp-action="CreateCreditingExcelReport">Создать отчёт по пополнениям (EXCEL)</button>

</div>

<div class="mb-2">

<button style="width:100%" class="btn btn-lg btn-warning btn-block" type="submit" asp-controller="Home" asp-action="CreateDebitingExcelReport">Создать отчёт по снятиям (EXCEL)</button>

</div>

</div>

<div class="mb-2">

<button type="button" id="WordBut" class="btn btn-lg btn-warning btn-block">Word отчеты</button>

</div>

<div id="WordDiv" style="display: none">

<div class="mb-2">

<button style="width:100%" class="btn btn-lg btn-warning btn-block" type="submit" asp-controller="Home" asp-action="CreateWordReport">Создать отчёт по переводам (WORD)</button>

</div>

<div class="mb-2">

<button style="width:100%" class="btn btn-lg btn-warning btn-block" type="submit" asp-controller="Home" asp-action="CreateCreditingWordReport">Создать отчёт по пополнениям (WORD)</button>

</div>

<div class="mb-2">

<button style="width:100%" class="btn btn-lg btn-warning btn-block" type="submit" asp-controller="Home" asp-action="CreateDebitingWordReport">Создать отчёт по снятиям (WORD)</button>

</div>

</div>

</form>

</div>

</div>

<div class="aos-init aos-animate col-md" sf-type="container" sf-label="Column" sf-anim-delay="2" data-aos="fade-down" data-aos-delay="500" sf-uid="8">

<div sf-type="container" sf-label="Container" class="py-15 h-100 bg-bg-2" sf-uid="9">

<table class="table">

<thead>

<tr>

<th>

Номер карты

</th>

<th>

Тип операции

</th>

<th>

Сумма

</th>

<th>

Статус

</th>

<th>

Дата открытия

</th>

<th>

Дата закрытия

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Operations)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.CardNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.TypeOperation)

</td>

<td>

@Html.DisplayFor(modelItem => item.Sum)

</td>

<td>

@item.Status.ToString().Replace("\_", " ");

</td>

<td>

@Html.DisplayFor(modelItem => item.DateOpen)

</td>

<td>

@Html.DisplayFor(modelItem => item.DateClose)

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

<script>

document.getElementById('ExcelBut').addEventListener('click', event => {

if (document.getElementById("ExcelDiv").style.display == "none") {

document.getElementById("ExcelDiv").style.display = "block";

}

else {

document.getElementById("ExcelDiv").style.display = "none";

}

});

document.getElementById('WordBut').addEventListener('click', event => {

if (document.getElementById("WordDiv").style.display == "none") {

document.getElementById("WordDiv").style.display = "block";

}

else {

document.getElementById("WordDiv").style.display = "none";

}

});

</script>

Проект BankCashierApp

Program.cs

using BankCashierApp;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews();

var app = builder.Build();

APICashier.Connect(builder.Configuration);

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();

appsettings.json

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"IPAddress": "http://localhost:5164/"

}

APICashier.cs

using BankContracts.ViewModels;

using BankContracts.ViewModels.Cashier.ViewModels;

using BankContracts.ViewModels.Client.ViewModels;

using Newtonsoft.Json;

using System.Net.Http.Headers;

using System.Text;

namespace BankCashierApp

{

public class APICashier

{

private static readonly HttpClient \_client = new();

public static string ErrorMessage = string.Empty;

//Вью-модели, необходимые для дальнейшего генерирования запросов к апишке

public static CashierViewModel? Cashier { get; set; } = null;

public static CreditingViewModel? Crediting { get; set; } = null;

public static DebitingViewModel? Debiting { get; set; } = null;

public static AccountViewModel? Account { get; set; } = null;

public static CardViewModel? Card { get; set; } = null;

public static void Connect(IConfiguration configuration)

{

\_client.BaseAddress = new Uri(configuration["IPAddress"]);

\_client.DefaultRequestHeaders.Accept.Clear();

\_client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

}

public static void SetErrorMessage(string error)

{

ErrorMessage = error;

}

//Get-запрос

public static T? GetRequest<T>(string requestUrl)

{

var response = \_client.GetAsync(requestUrl);

var result = response.Result.Content.ReadAsStringAsync().Result;

if (response.Result.IsSuccessStatusCode)

{

return JsonConvert.DeserializeObject<T>(result);

}

else

{

throw new Exception(result);

}

}

//Post-запрос

public static void PostRequest<T>(string requestUrl, T model)

{

var json = JsonConvert.SerializeObject(model);

var data = new StringContent(json, Encoding.UTF8, "application/json");

var response = \_client.PostAsync(requestUrl, data);

var result = response.Result.Content.ReadAsStringAsync().Result;

if (!response.Result.IsSuccessStatusCode)

{

throw new Exception(result);

}

}

//Post-запрос для получения данных

public static T? PostRequestReport<T, U>(string requestUrl, U model)

{

var json = JsonConvert.SerializeObject(model);

var data = new StringContent(json, Encoding.UTF8, "application/json");

var response = \_client.PostAsync(requestUrl, data);

var result = response.Result.Content.ReadAsStringAsync().Result;

if (response.Result.IsSuccessStatusCode)

{

return JsonConvert.DeserializeObject<T>(result);

}

else

{

throw new Exception(result);

}

}

}

}

appsettings.Development.json

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

}

}

Папка Controllers

HomeController.cs

using BankCashierApp.Models;

using BankContracts.BindingModels.Cashier;

using BankContracts.BindingModels.Reports;

using BankContracts.ViewModels.Cashier.Diagram;

using BankContracts.ViewModels.Cashier.ViewModels;

using BankContracts.ViewModels.Client.ViewModels;

using BankContracts.ViewModels.Reports.Cashier;

using BankContracts.ViewModels.Reports;

using BankDataModels.Enums;

using Microsoft.AspNetCore.Mvc;

using System.Diagnostics;

namespace BankCashierApp.Controllers

{

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger)

{

\_logger = logger;

}

#region Загрузка главной страницы

//вытаскивает через API клиента Get-запросом список его собственных заказов

[HttpGet]

public IActionResult Index()

{

if (APICashier.Cashier == null)

{

return Redirect("~/Home/Enter");

}

return View(APICashier.GetRequest<List<AccountViewModel>>($"/api/Account/GetAllAccounts"));

}

#endregion

#region Обновление данных пользователя

//изменемение ланных Get-ом

[HttpGet]

public IActionResult Privacy()

{

if (APICashier.Cashier == null)

{

return Redirect("~/Home/Enter");

}

return View(APICashier.Cashier);

}

//изменение данных Post-ом

[HttpPost]

public void Privacy(string login, string password, string name, string surname, string patronymic, string telephone)

{

if (APICashier.Cashier == null)

{

throw new Exception("Вы как сюда попали? Суда вход только авторизованным");

}

if (string.IsNullOrEmpty(login) || string.IsNullOrEmpty(password) || string.IsNullOrEmpty(name)

|| string.IsNullOrEmpty(surname) || string.IsNullOrEmpty(patronymic)

|| string.IsNullOrEmpty(telephone))

{

throw new Exception("Введите логин, пароль, ФИО и телефон");

}

APICashier.PostRequest("/api/Cashier/UpdateData", new CashierBindingModel

{

Id = APICashier.Cashier.Id,

Name = name,

Surname = surname,

Patronymic = patronymic,

Telephone = telephone,

Email = login,

Password = password

});

APICashier.Cashier.Name = name;

APICashier.Cashier.Surname = surname;

APICashier.Cashier.Patronymic = patronymic;

APICashier.Cashier.Email = login;

APICashier.Cashier.Password = password;

APICashier.Cashier.Telephone = telephone;

Response.Redirect("Enter");

}

[HttpGet]

public IActionResult Login()

{

return View();

}

[HttpPost]

public IActionResult Login(string login, string password)

{

if (string.IsNullOrEmpty(login) || string.IsNullOrEmpty(password))

{

APICashier.SetErrorMessage("Необходимо заполнить оба поля");

return Redirect("ErrorPage");

}

APICashier.Cashier = APICashier.GetRequest<CashierViewModel>($"api/Cashier/Login?login={login}&password={password}");

if (APICashier.Cashier == null)

{

APICashier.SetErrorMessage("Неверный логин или пароль");

return Redirect("ErrorPage");

}

return Redirect("Enter");

}

[HttpPost]

public IActionResult Logout()

{

APICashier.Cashier = null;

return Redirect("Enter");

}

#endregion

#region Вывод ошибок

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel

{

RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier

});

}

[HttpGet]

public IActionResult ErrorPage()

{

return View();

}

#endregion

#region Вход в приложение

//просто открытие вьюхи

[HttpGet]

public IActionResult Enter()

{

return View();

}

//отсылаем указанные данные на проверку

[HttpPost]

public IActionResult Enter(string login, string password)

{

if (string.IsNullOrEmpty(login) || string.IsNullOrEmpty(password))

{

APICashier.SetErrorMessage("Введите логин и пароль");

return Redirect("ErrorPage");

}

APICashier.Cashier = APICashier.GetRequest<CashierViewModel>($"/api/Cashier/Login?login={login}&password={password}");

if (APICashier.Cashier == null)

{

APICashier.SetErrorMessage("Неверный логин или пароль");

return Redirect("ErrorPage");

}

return Redirect("Index");

}

#endregion

#region Регистрация

//просто открытие вьюхи

[HttpGet]

public IActionResult Register()

{

return View();

}

//Post-запрос по созданию нового пользователя

[HttpPost]

public void Register(string login, string password, string name, string surname, string patronymic, string telephone)

{

if (string.IsNullOrEmpty(login) || string.IsNullOrEmpty(password) || string.IsNullOrEmpty(name)

|| string.IsNullOrEmpty(surname) || string.IsNullOrEmpty(patronymic) || string.IsNullOrEmpty(telephone))

{

throw new Exception("Введите логин, пароль, ФИО и телефон");

}

APICashier.PostRequest("/api/Cashier/Register", new CashierBindingModel

{

Name = name,

Surname = surname,

Patronymic = patronymic,

Email = login,

Password = password,

Telephone = telephone

});

//переход на вкладку "Enter", чтобы пользователь сразу смог зайти

Response.Redirect("Enter");

return;

}

#endregion

#region Открытие нового счёта

//открытие счёта. Получаем и передаём список изделий во вьюху?

[HttpGet]

public IActionResult CreateAccount()

{

if (APICashier.Cashier == null)

{

return Redirect("~/Home/Enter");

}

//запрашиваем список в формате вспомогательной вьюшки из-за работы select в asp net

ViewBag.Clients = APICashier.GetRequest<List<ClientViewModel>>($"/api/Client/GetAllClients").Select(x => new ClientSelectViewModel

{

Id = x.Id,

FullName = x.Surname + " " + x.Name + " " + x.Patronymic

}).ToList();

return View();

}

//создание заказа Post-запросом

[HttpPost]

public IActionResult CreateAccount(int clientId, string accountNumber, string password, int balance)

{

if (APICashier.Cashier == null)

{

APICashier.SetErrorMessage("Необходимо авторизоваться");

return Redirect("ErrorPage");

}

if (clientId <= 0 || string.IsNullOrEmpty(accountNumber) || string.IsNullOrEmpty(password) || balance < 0)

{

APICashier.SetErrorMessage("Проверьте корректность вводимых данных");

return Redirect("ErrorPage");

}

APICashier.PostRequest("/api/Account/Register", new AccountBindingModel

{

CashierId = APICashier.Cashier.Id,

ClientId = clientId,

AccountNumber = accountNumber,

PasswordAccount = password,

Balance = balance,

DateOpen = DateTime.Now

});

return Redirect("Index");

}

#endregion

#region Работа с заявками на зачисление

//для страницы "Заявки на зачисление"

[HttpGet]

public IActionResult Crediting()

{

if (APICashier.Cashier == null)

{

APICashier.SetErrorMessage("Необходимо авторизоваться");

Response.Redirect("ErrorPage");

}

return View(APICashier.GetRequest<List<CreditingViewModel>>($"/api/Account/FindOpenCrediting"));

}

//открытие вьюхи одобрения заявки на зачисление

[HttpGet]

public IActionResult CloseCrediting()

{

if (APICashier.Cashier == null)

{

return Redirect("~/Home/Enter");

}

ViewBag.Creditings = APICashier.GetRequest<List<CreditingViewModel>>("/api/Account/FindOpenCrediting");

ViewBag.Accounts = APICashier.GetRequest<List<AccountViewModel>>("/api/Account/GetAllAccounts");

return View();

}

//одобрения заявки на зачисление Post-запросом

[HttpPost]

public IActionResult CloseCrediting(int creditingId, int accountPayeeId)

{

if (APICashier.Cashier == null)

{

APICashier.SetErrorMessage("Необходимо авторизоваться");

return Redirect("ErrorPage");

}

if (creditingId <= 0 || accountPayeeId <= 0)

{

APICashier.SetErrorMessage("Проверьте коректность передавваемых значений");

return Redirect("ErrorPage");

}

//получаем необходимые данные для запроса

APICashier.Crediting = APICashier.GetRequest<CreditingViewModel>($"/api/Account/FindCrediting?id={creditingId}");

APICashier.Card = APICashier.GetRequest<CardViewModel>($"/api/Card/FindCard?id={APICashier.Crediting.CardId}");

APICashier.PostRequest("/api/Account/CloseCrediting", new MoneyTransferBindingModel

{

CashierId = APICashier.Cashier.Id,

CreditingId = creditingId,

Sum = APICashier.Crediting.Sum,

AccountPayeeId = accountPayeeId

});

//очистка данных

APICashier.Crediting = null;

APICashier.Card = null;

return Redirect("Crediting");

}

#endregion

#region Работа с заявками на снятие

//для страницы "Заявки на снятие"

[HttpGet]

public IActionResult Debiting()

{

if (APICashier.Cashier == null)

{

APICashier.SetErrorMessage("Необходимо авторизоваться");

Response.Redirect("ErrorPage");

}

return View(APICashier.GetRequest<List<DebitingViewModel>>($"/api/Account/FindOpenDebiting"));

}

//открытие вьюхи одобрения заявки на снятие

[HttpGet]

public IActionResult CloseDebiting()

{

if (APICashier.Cashier == null)

{

APICashier.SetErrorMessage("Необходимо авторизоваться");

Response.Redirect("ErrorPage");

}

ViewBag.Debitings = APICashier.GetRequest<List<DebitingViewModel>>("/api/Account/FindOpenDebiting");

ViewBag.Accounts = APICashier.GetRequest<List<AccountViewModel>>("/api/Account/GetAllAccounts");

return View();

}

//одобрения заявки на снятие Post-запросом

[HttpPost]

public IActionResult CloseDebiting(int debitingId, int accountId)

{

if (APICashier.Cashier == null)

{

APICashier.SetErrorMessage("Необходимо авторизоваться");

return Redirect("ErrorPage");

}

if (debitingId <= 0 || accountId <= 0)

{

APICashier.SetErrorMessage("Проверьте корректность передаваемых значений");

return Redirect("ErrorPage");

}

//получаем необходимые данные для запроса

APICashier.Debiting = APICashier.GetRequest<DebitingViewModel>($"/api/Account/FindDebiting?id={debitingId}");

APICashier.Card = APICashier.GetRequest<CardViewModel>($"/api/Card/FindCard?id={APICashier.Debiting.CardId}");

APICashier.PostRequest("/api/Account/CloseDebiting", new CashWithdrawalBindingModel

{

CashierId = APICashier.Cashier.Id,

DebitingId = debitingId,

Sum = APICashier.Debiting.Sum,

AccountId = accountId

});

APICashier.Debiting = null;

APICashier.Card = null;

return Redirect("Debiting");

}

#endregion

//получение номера запрашиваемого счёта для снятия - для работы с начислениями и списаниями

[HttpPost]

public string GetAccountNumber(int id)

{

if (APICashier.Cashier == null)

{

APICashier.SetErrorMessage("Необходимо авторизоваться");

Response.Redirect("ErrorPage");

}

APICashier.Debiting = APICashier.GetRequest<DebitingViewModel>($"/api/Account/FindDebiting?id={id}");

APICashier.Crediting = APICashier.GetRequest<CreditingViewModel>($"/api/Account/FindCrediting?id={id}");

if (APICashier.Debiting == null)

{

APICashier.Card = APICashier.GetRequest<CardViewModel>($"/api/Card/FindCard?id={APICashier.Crediting.CardId}");

}

else

{

APICashier.Card = APICashier.GetRequest<CardViewModel>($"/api/Card/FindCard?id={APICashier.Debiting.CardId}");

}

APICashier.Account = APICashier.GetRequest<AccountViewModel>($"/api/Account/GetAccount?accountId={APICashier.Card.AccountId}");

string AccountNumber = APICashier.Account.AccountNumber;

APICashier.Debiting = null;

APICashier.Card = null;

APICashier.Account = null;

return AccountNumber;

}

#region Работа с переводом со счёта на счёт

[HttpGet]

public IActionResult MoneyTransfers()

{

if (APICashier.Cashier == null)

{

return Redirect("~/Home/Enter");

}

ViewBag.Accounts = APICashier.GetRequest<List<AccountViewModel>>("/api/Account/GetAllAccounts");

return View();

}

[HttpPost]

public IActionResult MoneyTransfers(int accountSenderId, int accountPayeeId, int sumMoneyTransfer)

{

if (APICashier.Cashier == null)

{

APICashier.SetErrorMessage("Необходимо авторизоваться");

return Redirect("ErrorPage");

}

if (accountPayeeId <= 0 || accountSenderId <= 0 || sumMoneyTransfer <= 0)

{

APICashier.SetErrorMessage("Проверьте корректность передаваемых значений");

return Redirect("ErrorPage");

}

APICashier.PostRequest("/api/Account/CloseCrediting", new MoneyTransferBindingModel

{

CashierId = APICashier.Cashier.Id,

Sum = sumMoneyTransfer,

AccountPayeeId = accountPayeeId,

AccountSenderId = accountSenderId

});

return Redirect("Index");

}

#endregion

#region Отчёт с выборкой по счетам

[HttpGet]

public IActionResult ReportWithAccounts()

{

if (APICashier.Cashier == null)

{

return Redirect("~/Home/Enter");

}

ViewBag.Accounts = APICashier.GetRequest<List<AccountViewModel>>("/api/Account/GetAllAccounts");

return View(new List<ReportCashierAccountsViewModel>());

}

//создание excel отчёта у касира

[HttpPost]

public IActionResult CreateCashierExcelReport(string accountId)

{

if (APICashier.Cashier == null)

{

APICashier.SetErrorMessage("Необходимо авторизоваться");

return Redirect("ErrorPage");

}

if (string.IsNullOrEmpty(accountId))

{

APICashier.SetErrorMessage("Необходимо выбрать счёт для создания отчёта");

return Redirect("ErrorPage");

}

APICashier.PostRequest("api/Report/CreateExcelCashier", new ReportSupportBindingModel()

{

AccountId = int.Parse(accountId),

Email = APICashier.Cashier.Email

});

return Redirect("ReportSuccess");

}

//создание excel отчёта у касира

[HttpPost]

public IActionResult CreateCashierWordReport(string accountId)

{

if (APICashier.Cashier == null)

{

APICashier.SetErrorMessage("Необходимо авторизоваться");

return Redirect("ErrorPage");

}

if (string.IsNullOrEmpty(accountId))

{

APICashier.SetErrorMessage("Необходимо выбрать счёт для создания отчёта");

return Redirect("ErrorPage");

}

APICashier.PostRequest("api/Report/CreateWordCashier", new ReportSupportBindingModel()

{

AccountId = int.Parse(accountId),

Email = APICashier.Cashier.Email

});

return Redirect("ReportSuccess");

}

[HttpPost]

public IActionResult ReportWithAccounts(string accountId)

{

if (APICashier.Cashier == null)

{

APICashier.SetErrorMessage("Необходимо авторизоваться");

return Redirect("ErrorPage");

}

if (string.IsNullOrEmpty(accountId))

{

APICashier.SetErrorMessage("Необходимо выбрать счёт для создания отчёта");

return Redirect("ErrorPage");

}

ViewBag.Accounts = APICashier.GetRequest<List<AccountViewModel>>("/api/Account/GetAllAccounts");

var cashWithdrawals = APICashier.GetRequest<List<CashWithdrawalViewModel>>("api/Account/FindAllCashWithdrawal").Where(x => x.AccountId == int.Parse(accountId))

.Select(x => new ReportCashierAccountsViewModel

{

CashierSurname = x.SurmaneCashier,

Sum = x.Sum,

AccountSenderNumber = x.AccountNumber,

DateOperation = x.DateOperation,

typeOperation = TypeOperationEnum.Снятие

});

var moneyTransfers = APICashier.GetRequest<List<MoneyTransferViewModel>>("/api/Account/FindAllMoneyTransfer").Where(x => (x.AccountPayeeId == int.Parse(accountId) || x.AccountSenderId == int.Parse(accountId)))

.Select(x => new ReportCashierAccountsViewModel

{

CashierSurname = x.CashierSurname,

Sum = x.Sum,

AccountPayeeNumber = x.AccountPayeeNumber,

AccountSenderNumber = x.AccountSenderNumber != null ? x.AccountSenderNumber : "---",

DateOperation = x.DateOperation,

typeOperation = x.AccountSenderId.HasValue ? TypeOperationEnum.Перевод : TypeOperationEnum.Пополнение

});

return View(cashWithdrawals.Concat(moneyTransfers).OrderBy(x => x.DateOperation).ToList());

}

#endregion

#region Получение отчёта PDF

[HttpGet]

public IActionResult CreateReport()

{

if (APICashier.Cashier == null)

{

return Redirect("~/Home/Enter");

}

//запрашиваем список в формате вспомогательной вьюшки из-за работы select в asp net

ViewBag.Clients = APICashier.GetRequest<List<ClientViewModel>>($"/api/Client/GetAllClients").Select(x => new ClientSelectViewModel

{

Id = x.Id,

FullName = x.Surname + " " + x.Name + " " + x.Patronymic

}).ToList();

return View();

}

[HttpPost]

public IActionResult CreateReport(int clientId, DateTime dateFrom, DateTime dateTo)

{

if (APICashier.Cashier == null)

{

APICashier.SetErrorMessage("Необходимо авторизоваться");

return Redirect("ErrorPage");

}

if (clientId <= 0 || dateFrom == dateTo || dateFrom > dateTo || dateFrom.Year == 0001 || dateTo.Year == 0001)

{

APICashier.SetErrorMessage("Пожалуйста, установите корректные границы периода для отчёта и выберите счёт");

return Redirect("ErrorPage");

}

//запрашиваем список в формате вспомогательной вьюшки из-за работы select в asp net

ViewBag.Clients = APICashier.GetRequest<List<ClientViewModel>>($"/api/Client/GetAllClients").Select(x => new ClientSelectViewModel

{

Id = x.Id,

FullName = x.Surname + " " + x.Name + " " + x.Patronymic

}).ToList();

return View(APICashier.PostRequestReport<ReportCashierViewModelForHTML, ReportSupportBindingModel>("api/Report/CreateCashierReport", new ReportSupportBindingModel()

{

ClientId = clientId,

Email = APICashier.Cashier.Email,

DateFrom = dateFrom,

DateTo = dateTo

}));

}

#endregion

#region Диаграмма

[HttpGet]

public IActionResult Diagram()

{

if (APICashier.Cashier == null)

{

return Redirect("~/Home/Enter");

}

ViewBag.Accounts = APICashier.GetRequest<List<AccountViewModel>>($"api/Account/GetAllAccounts");

return View();

}

[HttpPost]

public IActionResult Diagram(int accountId)

{

if (APICashier.Cashier == null)

{

return Redirect("~/Home/Enter");

}

if (accountId <= 0)

{

APICashier.SetErrorMessage("Для построения диаграммы необходимо выбрать счёт");

return Redirect("ErrorPage");

}

ViewBag.Accounts = APICashier.GetRequest<List<AccountViewModel>>($"api/Account/GetAllAccounts");

return View(new CashierDiagramViewModel()

{

DiagramName = "Hello World",

Elements = APICashier.GetRequest<List<CashierDiagramElementsViewModel>>($"api/Account/getAccountMonthResult?cardId={accountId}")

});

}

#endregion

[HttpGet]

public IActionResult ReportSuccess()

{

if (APICashier.Cashier == null)

{

APICashier.SetErrorMessage("Необходимо авторизоваться");

Response.Redirect("ErrorPage");

}

return View();

}

}

}

Папка Models

ErrorViewModel.cs

namespace BankCashierApp.Models

{

public class ErrorViewModel

{

public string? RequestId { get; set; }

public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);

}

}

Папка Properties

launchSettings.json

{

"iisSettings": {

"windowsAuthentication": false,

"anonymousAuthentication": true,

"iisExpress": {

"applicationUrl": "http://localhost:26194",

"sslPort": 44342

}

},

"profiles": {

"BankCashierApp": {

"commandName": "Project",

"dotnetRunMessages": true,

"launchBrowser": true,

"applicationUrl": "https://localhost:7021;http://localhost:5213",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

},

"IIS Express": {

"commandName": "IISExpress",

"launchBrowser": true,

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

}

}

}

Папка Views

Папка Shared

\_Layout.cshtml

@using BankCashierApp

@{

bool authenticated = APICashier.Cashier != null;

}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>@ViewData["Title"]</title>

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />

<link rel="stylesheet" href="~/css/site.css" />

<script src="~/lib/jquery/dist/jquery.min.js"></script>

<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

<style>

body {

margin-bottom: 60px;

background-color: #ececec; /\* Измененный цвет фона \*/

color: #333; /\* Основной цвет текста \*/

}

.nav-main a {

position: relative;

color: #333;

cursor: pointer;

line-height: 1.5;

text-decoration: none;

padding: 10px;

transition: color 0.3s ease-out;

}

.nav-main a:after {

display: block;

position: absolute;

left: 0;

bottom: -5px;

width: 0;

height: 2px;

background-color: #333;

content: "";

transition: width 0.3s ease-out;

}

.nav-main a:hover, .nav-main a:focus {

color: #007bff; /\* Цвет при наведении \*/

}

.nav-main a:hover:after,

.nav-main a:focus:after {

width: 100%;

}

.nav-main .dropdown:hover .dropdown-menu {

display: block;

margin-top: 0;

}

.footer {

position: fixed;

left: 0;

bottom: 0;

width: 100%;

height: 80px;

background-color: #333;

color: #fff;

}

.form-signin {

width: 100%;

max-width: 400px;

padding: 20px;

margin: auto;

background-color: #fff;

border-radius: 10px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

.form-signin .form-control {

position: relative;

box-sizing: border-box;

height: auto;

padding: 10px;

font-size: 16px;

}

.form-signin .form-control:focus {

z-index: 2;

}

.form-signin input[type="email"] {

margin-bottom: -1px;

border-bottom-right-radius: 0;

border-bottom-left-radius: 0;

}

.form-signin input[type="password"] {

margin-bottom: 10px;

border-top-left-radius: 0;

border-top-right-radius: 0;

}

table.table tbody tr td,

table.table thead tr th,

table.table thead {

border-left: solid;

border-right: solid;

border-width: 2px;

border-color: #333;

}

table {

vertical-align: middle;

}

header {

background-color: #007bff; /\* Цвет фона шапки \*/

color: #fff; /\* Цвет текста шапки \*/

}

header a {

color: #fff; /\* Цвет текста ссылок в шапке \*/

}

header a:hover {

color: #ffd700; /\* Цвет текста ссылок при наведении \*/

}

.btn-custom {

background-color: #007bff; /\* Пользовательский цвет кнопки \*/

color: #fff; /\* Цвет текста кнопки \*/

border: none;

}

.btn-custom:hover {

background-color: #0056b3; /\* Цвет кнопки при наведении \*/

}

</style>

</head>

<body class="MyBody">

<header class="d-flex flex-wrap align-items-center justify-content-between py-3 px-4 mb-4 border-bottom">

<div class="col-md-3 mb-2 mb-md-0">

<a asp-controller="Home" asp-action="Enter" class="d-inline-flex link-body-emphasis text-decoration-none">

<span class="fs-4">Банк "Вы банкрот"</span>

</a>

</div>

<ul class="nav col-12 col-md-auto mb-2 justify-content-center mb-md-0 nav-main">

<li><a class="nav-link px-2" asparea="" asp-controller="Home" asp-action="Index">Счета</a></li>

<li class="dropdown">

<a href="#" class="nav-link px-2">Операции</a>

<ul class="dropdown-menu">

<li><a class="dropdown-item" asp-controller="Home" asp-action="Debiting">Заявки на снятие</a></li>

<li><a class="dropdown-item" asp-controller="Home" asp-action="Crediting">Заявки на начисление</a></li>

<li><a class="dropdown-item" asp-controller="Home" asp-action="MoneyTransfers">Заявки на перевод</a></li>

</ul>

</li>

<li class="dropdown">

<a href="#" class="nav-link px-2">Отчеты</a>

<ul class="dropdown-menu">

<li><a class="dropdown-item" asp-controller="Home" asp-action="ReportWithAccounts">Отчёт по аккаунтам</a></li>

<li><a class="dropdown-item" asp-controller="Home" asp-action="CreateReport">Отчёт за период</a></li>

<li><a class="dropdown-item" asp-controller="Home" asp-action="Diagram">Диаграмма</a></li>

</ul>

</li>

</ul>

@{

if (APICashier.Cashier == null)

{

<div class="col-md-3 text-end">

<a class="btn btn-custom me-2" asp-controller="Home" asp-action="Login">Войти</a>

<a class="btn btn-custom" asp-controller="Home" asp-action="Register">Регистрация</a>

</div>

}

else

{

<div class="col-md-3 text-end">

<a class="btn btn-custom me-2" id="exit" name="exit" asp-controller="Home" asp-action="Privacy">@APICashier.Cashier.Surname @APICashier.Cashier.Name</a>

</div>

}

}

</header>

<div class="container">

<main role="main" class="pb-3">

@RenderBody()

</main>

</div>

<footer class="border-top footer">

<div class="container text-center">

&copy; 2024 - BankCashierApp

</div>

</footer>

<script src="~/lib/jquery/dist/jquery.min.js"></script>

<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

<script src="~/js/site.js" asp-append-version="true"></script>

@await RenderSectionAsync("Scripts", required: false)

</body>

</html>

Папка Home

CloseCrediting.cshtml

@{

ViewData["Title"] = "Одобрение зачислений";

}

<div class="text-center">

<h2 class="display-4">Зачисление</h2>

</div>

<form method="post">

<div class="row mb-2">

<div class="col-4">Номер запроса на зачисление:</div>

<div class="col-8">

<select id="creditigId" name="creditingId" class="form-control" asp-items="@(new SelectList( @ViewBag.Creditings, "Id", "Id"))">

<option disabled selected>Выберите запрос</option>

</select>

</div>

</div>

<div class="row mb-2">

<div class="col-4">Запрашиваемый счёт для зачисления:</div>

<div class="col-8">

<input type="text" id="accountNumber" class="form-control" name="accountNumber" readonly />

</div>

</div>

<div class="row mb-2">

<div class="col-4">Номер счёта для зачисления:</div>

<div class="col-8">

<select id="accountPayeeId" name="accountPayeeId" class="form-control" asp-items="@(new SelectList( @ViewBag.Accounts, "Id", "AccountNumber"))" required>

<option disabled selected>Выберите счёт</option>

</select>

</div>

</div>

<div class="row mb-2">

<input type="submit" style="width: 100%" value="Начислить" class="btn btn-dark" />

</div>

</form>

<!-- подгрузка номера запрашиваемого клиентом счёта в реальном времени -->

<script>

$('#creditigId').on('change', function () {

check();

});

function check() {

var creditigId = $('#creditigId').val();

$.ajax({

method: "Post",

url: "/Home/GetAccountNumber",

data: { id: creditigId },

success: function (result) {

$("#accountNumber").val(result);

}

});

}

</script>

CloseDebiting.cshtml

@{

ViewData["Title"] = "Одобрение зачислений";

}

<div class="text-center">

<h2 class="display-4">Снятие</h2>

</div>

<form method="post">

<div class="row mb-4">

<div class="col-4">Номер запроса на снятие:</div>

<div class="col-8">

<select id="debitingId" name="debitingId" class="form-control" asp-items="@(new SelectList( @ViewBag.Debitings, "Id", "Id"))">

<option disabled selected>Выберите запрос</option>

</select>

</div>

</div>

<div class="row mb-4">

<div class="col-4">Запрашиваемый счёт для перевода:</div>

<div class="col-8">

<input type="text" class="form-control" id="accountNumber" name="accountNumber" readonly />

</div>

</div>

<div class="row mb-4">

<div class="col-4">Номер счёта для снятия:</div>

<div class="col-8">

<select id="accountId" name="accountId" class="form-control" asp-items="@(new SelectList( @ViewBag.Accounts, "Id", "AccountNumber"))" required>

<option disabled selected>Выберите счёт</option>

</select>

</div>

</div>

<div class="row mb-4">

<input type="submit" style="width: 100%" value="Снять" class="btn btn-dark" />

</div>

</form>

<!-- подгрузка номера запрашиваемого клиентом счёта в реальном времени -->

<script>

$('#debitingId').on('change', function () {

check();

});

function check() {

var debitingId = $('#debitingId').val();

$.ajax({

method: "Post",

url: "/Home/GetAccountNumber",

data: { id: debitingId },

success: function (result) {

$("#accountNumber").val(result);

}

});

}

</script>

CreateAccount.cshtml

@{

ViewData["Title"] = "Открытие счёта";

}

<div class="text-center">

<h2 class="display-4">Открытие счёта</h2>

</div>

<form method="post">

<div class="row mb-2">

<div class="col-4">Клиент:</div>

<div class="col-8">

<select id="client" name="clientId" class="form-control" asp-items="@(new SelectList( @ViewBag.Clients, "Id", "FullName"))"></select>

</div>

</div>

<div class="row mb-2">

<div class="col-4">Номер счёта:</div>

<div class="col-8">

<input type="text" id="accountNumber" class="form-control" name="accountNumber" required/>

</div>

</div>

<div class="row mb-2">

<div class="col-4">Пароль:</div>

<div class="col-8">

<input type="text" id="password" class="form-control" name="password" required />

</div>

</div>

<div class="row mb-2">

<div class="col-4">Баланс:</div>

<div class="col-8">

<input type="number" id="balance" class="form-control" name="balance" value=0 required min=0/>

</div>

</div>

<div class="row mb-2">

<div class="col-8"></div>

<div class="col-4">

<input type="submit" value="Создать" class="form-control" class="btn btn-dark" />

</div>

</div>

</form>

<script>

function createNum(len) {

chrs = '0123456789';

var str = '';

for (var i = 0; i < len; i++) {

var pos = Math.floor(Math.random() \* chrs.length);

str += chrs.substring(pos, pos + 1);

}

return str;

}

document.getElementById("accountNumber").value = createNum(16);

document.getElementById("password").value = createNum(4);

</script>

CreateReport.cshtml

@using BankContracts.ViewModels;

@using BankContracts.ViewModels.Reports

@model ReportCashierViewModelForHTML

@{

ViewData["Title"] = "Создание отчёта";

}

<div class="text-center">

<h2 class="display-4">Отчёт по счетам за выбранный период</h2>

</div>

<form method="post">

<div class="row mb-2">

<div class="col-4">Клиент:</div>

<div class="col-8">

<select id="clientId" name="clientId" class="form-control" asp-items="@(new SelectList( @ViewBag.Clients, "Id", "FullName"))"></select>

</div>

</div>

<div class="row mb-2">

<div class="col-4">Дата начала периода:</div>

<div class="col-8">

<input id="dateFrom" name="dateFrom" class="form-control" type="date" required/>

</div>

</div>

<div class="row mb-2">

<div class="col-4">Дата конца периода:</div>

<div class="col-8">

<input id="dateTo" name="dateTo" class="form-control" type="date" required />

</div>

</div>

<div class="row mb-2">

<input type="submit" style="width:100%" value="Сформировать отчёт" class="btn btn-dark" />

</div>

<hr class="mt-5 mb-3" />

@if (Model != null)

{

<div class="row p-3 text-center">

<h3>Отчет отправлен на почту @APICashier.Cashier.Email</h3>

<hr class="mt-5 mb-3" />

<h4>Отчёт по выдаче наличных со счёта</h4>

<table class="table">

<thead>

<tr>

<th>

Номер операции

</th>

<th>

Номер счёта

</th>

<th>

Сумма операции

</th>

<th>

Дата операции

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.ReportCashWithdrawal)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.OperationId)

</td>

<td>

@Html.DisplayFor(modelItem => item.AccountPayeeNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.SumOperation)

</td>

<td>

@Html.DisplayFor(modelItem => item.DateComplite)

</td>

</tr>

}

</tbody>

</table>

</div>

<hr class="my-12" />

<div class="row p-3 text-center">

<h4>Отчёт по денежным переводам между счетами</h4>

<table class="table">

<thead>

<tr>

<th>

Номер операции

</th>

<th>

Номер счёта отправителя

</th>

<th>

Номер счёта получаетля

</th>

<th>

Сумма операции

</th>

<th>

Дата операции

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.ReportMoneyTransfer)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.OperationId)

</td>

<td>

@Html.DisplayFor(modelItem => item.AccountSenderNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.AccountPayeeNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.SumOperation)

</td>

<td>

@Html.DisplayFor(modelItem => item.DateComplite)

</td>

</tr>

}

</tbody>

</table>

</div>

}

</form>

<!--@{

ViewData["Title"] = "Создание отчёта";

}

<div class="text-center">

<h2 class="display-4">Создание отчёта</h2>

</div>

<form method="post">

<div class="container px-4">

<div class="row gx-5">

<div class="col">

<div class="p-3">С</div>

</div>

<div class="col">

<div class="p-3 btn-group">

<button class="btn btn-secondary btn-sm dropdown-toggle" type="button" data-bs-toggle="dropdown" aria-expanded="false" padding="10">

Начало периода

</button>

<ul class="dropdown-menu">

...

</ul>

</div>

</div>

<div class="col">

<div class="p-3">по</div>

</div>

<div class="col">

<div class="p-3 btn-group">

<button class="btn btn-secondary btn-sm dropdown-toggle" type="button" data-bs-toggle="dropdown" aria-expanded="false" padding="10">

Конец периода

</button>

<ul class="dropdown-menu">

...

</ul>

</div>

</div>

<div class="col">

<div class="p-3">

<button type="button" class="btn btn-primary з-3">Создать</button>

</div>

</div>

</div>

</div>

<div class="row">

<div class="col-4">Отчёт:</div>

</div>

</form> -->

Crediting.cshtml

@using BankContracts.ViewModels.Client.ViewModels

@model List<CreditingViewModel>

@{

ViewData["Title"] = "Заявки на зачисление";

}

<div class="text-center">

<h1 class="display-4">Заявки</h1>

</div>

<div class="text-center">

@{

if (Model == null)

{

<h3 class="display-4">Сначала авторизируйтесь</h3>

return;

}

<p>

<a asp-action="CloseCrediting">Одобрение заявки</a>

</p>

<table class="table">

<thead>

<tr>

<th>

Номер заявки

</th>

<th>

Номер карты

</th>

<th>

Сумма зачисления

</th>

<th>

Дата открытия заявки

</th>

<th>

Статус заявки

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Id)

</td>

<td>

@Html.DisplayFor(modelItem => item.CardNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.Sum)

</td>

<td>

@Html.DisplayFor(modelItem => item.DateOpen)

</td>

<td>

@Html.DisplayFor(modelItem => item.Status)

</td>

</tr>

}

</tbody>

</table>

}

</div>

Debiting.cshtml

@using BankContracts.ViewModels.Client.ViewModels

@model List<DebitingViewModel>

@{

ViewData["Title"] = "Заявки на снятия";

}

<div class="text-center">

<h1 class="display-4">Заявки</h1>

</div>

<div class="text-center">

@{

if (Model == null)

{

<h3 class="display-4">Сначала авторизируйтесь</h3>

return;

}

<p>

<a asp-action="CloseDebiting">Одобрение заявки</a>

</p>

<table class="table">

<thead>

<tr>

<th>

Номер заявки

</th>

<th>

Номер карты

</th>

<th>

Сумма снятия

</th>

<th>

Дата открытия заявки

</th>

<th>

Статус заявки

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Id)

</td>

<td>

@Html.DisplayFor(modelItem => item.CardNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.Sum)

</td>

<td>

@Html.DisplayFor(modelItem => item.DateOpen)

</td>

<td>

@Html.DisplayFor(modelItem => item.Status)

</td>

</tr>

}

</tbody>

</table>

}

</div>

Diagram.cshtml

@using BankContracts.ViewModels.Cashier.Diagram

@model CashierDiagramViewModel

@{

ViewData["Title"] = "Диаграмма";

}

<div class="text-center">

<h1 class="display-4">Диаграмма суммарных поступлений на счёт по месяцам</h1>

</div>

<form method="post">

<div class="row mb-2">

<div class="row">Номер счета:</div>

<select id="accountId" name="accountId" class="form-control" asp-items="@(new SelectList( @ViewBag.Accounts, "Id", "AccountNumber"))">

<option disabled selected>Выберите счёт</option>

</select>

</div>

<div class="row mb-2">

<input style="width:100%;" type="submit" value="Выбрать" class="btn btn-dark" />

</div>

</form>

@if (Model == null) return;

<div id="Diagrams" class="text-center">

<div id="@Model.DiagramName Diagram">

<canvas id="Chart"></canvas>

<div id="params">

@foreach (var info in Model.Elements) {

<input type="hidden" id="@info.Name" value="@info.Value" />

}

</div>

</div>

</div>

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<script>

const diagrams = document.getElementById('Diagrams').childNodes;

let diagram\_name = diagrams[1].id;

console.log(diagram\_name);

let diagram = document.getElementById(diagram\_name).childNodes;

console.log(diagram);

let labels = [];

let data = [];

document.getElementById('params').childNodes.forEach(element => {

if (element.id != undefined) {

labels.push(element.id);

}

});

document.getElementById('params').childNodes.forEach(element => {

if (element.id != undefined) {

data.push(Number(element.value));

}

});

new Chart(diagram.item(1), {

type: 'bar',

data: {

labels: labels,

datasets: [{

label: 'Денег в этом месяце',

data: data,

borderWidth: 1,

backgroundColor: 'rgb(33, 37, 41)'

}]

},

options: {

plugins: {

legend: {

display: false

}

},

scales: {

y: {

suggestedMin: Math.min(data) - Math.min(data) \* -0.1,

suggestedMax: Math.max(data) + Math.max(data) \* 0.1,

}

}

}

}

);

</script>

Enter.cshtml

@{

ViewData["Title"] = "Страница кассира";

}

<div class="text-center">

<h1 class="display-4">Страница кассира</h1>

</div>

<div class="text-center">

@{

<img src="https://sun9-23.userapi.com/impg/RgpJNh6qIEd2V0pMH7WFORJvhO1AGF6Rm7eLfg/97sHI\_zVlkk.jpg?size=807x688&quality=96&sign=17dd944d545af33ac0ece7c5af1a9904&c\_uniq\_tag=eFM1QnUaoeGpkGFaMZvVo9XpvQKNYZM8EYd9ZaGj5gw&type=album" style="width: 700px" />

if (APICashier.Cashier == null)

{

<h3 class="display-4">Сначала авторизируйтесь</h3>

return;

}

<h3 class="display-4">Здравствуйтe, @APICashier.Cashier.Name @APICashier.Cashier.Patronymic</h3>

}

</div>

ErrorPage.cshtml

@using BankCashierApp

@{

ViewData["Title"] = "Отправка отчета";

}

<div class="text-center p-5">

<h3 class="display-4">Упс, что-то пошло не так...</h3>

<h3 class="display-4">Ошибка: @APICashier.ErrorMessage</h3>

</div>

Index.cshtml

@using BankContracts.ViewModels.Cashier.ViewModels

@model List<AccountViewModel>

@{

ViewData["Title"] = "Cписок счетов";

}

<div class="text-center">

<h1 class="display-4">Счета</h1>

</div>

<div class="text-center">

@{

if (Model == null)

{

<h3 class="display-4">Сначала авторизируйтесь</h3>

return;

}

<p>

<a asp-action="CreateAccount">Открыть счёт</a>

</p>

<table class="table">

<thead>

<tr>

<th>

Номер счёта

</th>

<th>

Имя владельца

</th>

<th>

Отчество владельца

</th>

<th>

Балланс

</th>

<th>

Дата открытия

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.AccountNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.Patronymic)

</td>

<td>

@Html.DisplayFor(modelItem => item.Balance)

</td>

<td>

@Html.DisplayFor(modelItem => item.DateOpen)

</td>

</tr>

}

</tbody>

</table>

}

</div>

Login.cshtml

@{

ViewData["Title"] = "Авторизация";

}

<div class="text-center">

<h2 class="display-4">Вход в приложение</h2>

</div>

<form class="form-signin text-center" method="post">

<h1 class="h3 mb-3 font-weight-normal">Логин</h1>

<input type="email" id="login" name="login" class="form-control" placeholder="Почта" required autofocus>

<input type="password" id="password" name="password" class="form-control" placeholder="Пароль" required>

<button class="btn btn-lg btn-dark btn-block" type="submit" asp-controller="Home" asp-action="Login">Войти</button>

</form>

MoneyTransfers.cshtml

@{

ViewData["Title"] = "Переводы меджу счетами";

}

<div class="text-center">

<h2 class="display-4">Перевод между счетами</h2>

</div>

<form method="post">

<div class="row mb-2">

<div class="col-4">Номер счёта для снятия:</div>

<div class="col-8">

<select id="accountSenderId" name="accountSenderId" class="form-control" asp-items="@(new SelectList( @ViewBag.Accounts, "Id", "AccountNumber"))">

<option disabled selected>Выберите счёт</option>

</select>

</div>

</div>

<div class="row mb-2">

<div class="col-4">Номер счёта для начисления:</div>

<div class="col-8">

<select id="accountPayeeId" name="accountPayeeId" class="form-control" asp-items="@(new SelectList( @ViewBag.Accounts, "Id", "AccountNumber"))">

<option disabled selected>Выберите счёт</option>

</select>

</div>

</div>

<div class="row mb-2">

<div class="col-4">Сумма перевода:</div>

<div class="col-8">

<input type="number" id="sumMoneyTransfer" class="form-control" name="sumMoneyTransfer" required min=1 value=1/>

</div>

</div>

<div class="row mb-2">

<input type="submit" style="width: 100%" value="Перевести" class="btn btn-dark" />

</div>

</form>

Privacy.cshtml

@using BankContracts.ViewModels

@using BankContracts.ViewModels.Cashier.ViewModels

@model CashierViewModel

@{

ViewData["Title"] = "Личный кабинет";

}

<div class="text-center">

<h2 class="display-4">Личные данные</h2>

</div>

<form method="post" class="form-signin">

<div class="row">

<div class="col-4">Логин:</div>

<input type="email" id="login" name="login" class="form-control" placeholder="Почта" value=@Html.DisplayFor(modelItem => Model.Email) required>

</div>

<div class="row">

<div class="col-4">Пароль:</div>

<input type="password" id="password" name="password" class="form-control" placeholder="Пароль" value=@Html.DisplayFor(modelItem => Model.Password) required>

</div>

<div class="row">

<div class="col-4">Имя:</div>

<input type="text" id="name" name="name" class="form-control" placeholder="Имя" value=@Html.DisplayFor(modelItem => Model.Name) required>

<div class="col-4">Фамилия:</div>

<input type="text" id="surname" name="surname" class="form-control" placeholder="Фамилия" value=@Html.DisplayFor(modelItem => Model.Surname) required>

<div class="col-4">Отчество:</div>

<input type="text" id="patronymic" name="patronymic" class="form-control" placeholder="Отчество" value=@Html.DisplayFor(modelItem => Model.Patronymic) required>

</div>

<div class="row mb-2">

<div class="col-4">Телефон:</div>

<input type="text" id="telephone" name="telephone" class="form-control" placeholder="Телефон" value=@Html.DisplayFor(modelItem => Model.Telephone) required>

</div>

<div class="row mb-2">

<button class="btn btn-lg btn-dark btn-block mb-2" type="submit" asp-controller="Home" asp-action="Privacy">Coхранить</button>

<button class="btn btn-lg btn-dark btn-block" type="submit" asp-controller="Home" asp-action="Logout">Выйти из аккаунта</button>

</div>

</form>

Register.cshtml

@{

ViewData["Title"] = "Регистрация";

}

<div class="text-center">

<h2 class="display-4">Регистрация</h2>

</div>

<form class="form-signin text-center" method="post">

<h1 class="h3 mb-3 font-weight-normal">Регистрация</h1>

<input type="email" id="login" name="login" class="form-control" placeholder="Почта" required>

<input type="password" id="password" name="password" class="form-control" placeholder="Пароль" required>

<input type="text" id="name" name="name" class="form-control" placeholder="Имя" required>

<input type="text" id="surname" name="surname" class="form-control" placeholder="Фамилия" required>

<input type="text" id="patronymic" name="patronymic" class="form-control" placeholder="Отчество" required>

<input type="text" id="telephone" name="telephone" class="form-control" placeholder="Телефон" required>

<button class="btn btn-lg btn-dark btn-block" type="submit" asp-controller="Home" asp-action="Register">Регистрация</button>

</form>

ReportSuccess.cshtml

@{

ViewData["Title"] = "Отправка отчета";

}

<div class="text-center">

@{

if (APICashier.Cashier == null)

{

<h3 class="display-4">Сначала авторизируйтесь</h3>

return;

}

}

<h3 class="display-4">Отчeт был отправлен на почту @APICashier.Cashier.Email</h3>

</div>

ReportWithAccounts.cshtml

@using BankContracts.ViewModels

@using BankContracts.ViewModels.Reports.Cashier

@model List<ReportCashierAccountsViewModel>?

@{

ViewData["Title"] = "Отчет по счетам";

}

<div class="text-center">

<h1 class="display-4">Отчет</h1>

</div>

<div class="container" sf-type="container" sf-label="Bootstrap Container" sf-uid="2">

<div class="row" sf-type="container" sf-label="Row" sf-uid="3">

<div class="mb-4 mb-md-0 aos-init aos-animate col-md-3" sf-type="container" sf-label="Column" sf-anim-delay="1.5" data-aos="fade-down" data-aos-delay="400" sf-uid="4">

<div sf-type="container" sf-label="Container" class="py-15 h-100 bg-bg-2" sf-uid="5">

<form method="post">

<select id="accountId" name="accountId" class="form-control mb-2" asp-items="@(new SelectList(ViewBag.Accounts, "Id", "AccountNumber"))">

<option disabled selected>Выберите счёт</option>

</select>

<div class="mb-2">

<button class="btn btn-lg btn-dark btn-block" style="width: 100%;" type="submit" asp-controller="Home" asp-action="ReportWithAccounts">Создать отчёт по аккаунтам</button>

</div>

<div class="mb-2">

<button class="btn btn-lg btn-dark btn-block" style="width: 100%;" type="submit" asp-controller="Home" asp-action="CreateCashierExcelReport">Создать отчёт по заявкам снятия (EXCEL)</button>

</div>

<div class="mb-2">

<button class="btn btn-lg btn-dark btn-block" style="width: 100%;" type="submit" asp-controller="Home" asp-action="CreateCashierWordReport">Создать отчёт по заявкам снятия (WORD</button>

</div>

</form>

</div>

</div>

<div class="aos-init aos-animate col-md" sf-type="container" sf-label="Column" sf-anim-delay="2" data-aos="fade-down" data-aos-delay="500" sf-uid="8">

<div sf-type="container" sf-label="Container" class="py-15 h-100 bg-bg-2" sf-uid="9">

<table class="table">

<thead>

<tr>

<th>

Тип операции

</th>

<th>

Кассир

</th>

<th>

Счет отправителя

</th>

<th>

Счет получателя

</th>

<th>

Сумма операции

</th>

<th>

Дата

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.typeOperation)

</td>

<td>

@Html.DisplayFor(modelItem => item.CashierSurname)

</td>

<td>

@Html.DisplayFor(modelItem => item.AccountSenderNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.AccountPayeeNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.Sum)

</td>

<td>

@Html.DisplayFor(modelItem => item.DateOperation)

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div></div>