МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«**КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ ИМЕНИ В.И. ВЕРНАДСКОГО**»

(ФГАОУ ВО «КФУ им. В.И. Вернадского»)

Физико-технический институт

(наименование института/филиала)

ОТЧЕТ

по *учебной, проектно-технологической*

*(вид, тип практики)*

обучающегося \_\_\_\_\_\_\_\_\_\_\_*Гоголева Виктора Григорьевича*\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество)

Физико-технический институт

Кафедра компьютерной инженерии и моделирования

Направление подготовки/специальность 09.03.01 «*Информатика и вычислительная техника»*

Направленность \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Курс *3* Группа *ИВТ-б-о-222* Форма обучения *очная*

|  |  |  |
| --- | --- | --- |
| Руководитель практической подготовки от ФГАОУ ВО «КФУ им. В. И. Вернадского» | Доцент \_\_\_\_\_\_\_\_\_\_\_\_\_ должность подпись | Лосев М.Ю.  (ФИО руководителя) |
|  |  |  |
|  |  |  |
|  |  |  |

Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_\_

ОГЛАВЛЕНИЕ

[ВЕДЕНИЕ 3](#_Toc202110037)

[РАЗДЕЛ 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 4](#_Toc202110038)

[1.1 Теоретические сведения 4](#_Toc202110039)

[1.1.1 REST API 4](#_Toc202110040)

[1.1.2 MongoDB 6](#_Toc202110041)

[1.1.3 Swagger 6](#_Toc202110042)

[1.1.4 NodeJS 7](#_Toc202110043)

[1.2 Проектирование БД 9](#_Toc202110044)

[1.2.1 Физическая диаграмма (диаграмма классов) 9](#_Toc202110045)

[1.2.2 ER-диаграмма 10](#_Toc202110046)

[РАЗДЕЛ 2. ПРАКТИЧЕСКАЯ ЧАСТЬ 11](#_Toc202110047)

[2.1 Создание Базы Данных 11](#_Toc202110048)

[2.2. Создание таблиц и заполнение таблиц начальными полями. 12](#_Toc202110049)

[2.3 Проверка созданных таблиц и корректности полей в них 14](#_Toc202110050)

[2.4 Написание программы для работы с БД 18](#_Toc202110051)

[2.5 CRUD операции к таблицам БД и проверка работоспособности решения (Postman, MongoDB Compass) 23](#_Toc202110052)

[2.6 Создание собственных запросов к таблицам БД через MQL (MongoDB Query Language) 28](#_Toc202110053)

[2.7 Реализация пользовательского интерфейса 30](#_Toc202110054)

[ЗАКЛЮЧЕНИЕ 38](#_Toc202110055)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 39](#_Toc202110056)

[ПРИЛОЖЕНИЕ 40](#_Toc202110057)

# ВЕДЕНИЕ

Учебная практика по специальности технического профиля предусматривает закрепление и углубление знаний, полученных обучающимися в процессе теоретического обучения, приобретение ими необходимых умений практической работы по избранной специальности, овладение навыками профессиональной деятельности.

В современном мире банковские системы требуют высокой надежности, производительности и удобства взаимодействия с данными. Одним из ключевых аспектов работы коммерческого банка является учет кассовых операций, который включает в себя обработку транзакций, управление счетами клиентов и контроль со стороны операторов (кассиров).

Целью данной работы является разработка приложения для обеспечения пользователя актуальной информацией из БД учета кассовых операций коммерческого банка.

Так как, согласно задания практики, технический стек свободен, было принято решение использовать современные технологии и стандарты для реализации, такие как MongoDB (NoSQL-база данных), Node.js (серверная платформа) и REST API (архитектурный стиль взаимодействия клиента и сервера).

Задачи работы:

* изучение принципов работы NoSQL-баз данных, в частности MongoDB;
* проектирование структуры базы данных для учета клиентов, счетов, транзакций и кассиров;
* разработка REST API для выполнения CRUD-операций с данными.
* документирование API с использованием инструмента Swagger;
* реализация пользовательского интерфейса, для удобного взаимодействия с ИС;
* тестирование работоспособности приложения с помощью Postman и MongoDB Compass.

# РАЗДЕЛ 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## Теоретические сведения

### REST API

REST (Representational State Transfer) – архитектурный стиль для создания веб-сервисов. API (Application Programming Interface) – способ взаимодействия между клиентом и сервером.

Основные принципы:

* Stateless – сервер не хранит состояние клиента (каждый запрос независим);
* Ресурсы (URI) – данные доступны по уникальным URL (/users, /posts);
* Идемпотентность - свойство, которое говорит, что повторный идентичный запрос, сделанный один или несколько раз подряд, имеет один и тот же эффект не меняя состояние сервера. Корректно реализованные методы PUT, GET, DELETE являются идемпотентыми, но не метод POST;
* Версионирование API – посредством использования v1/v2/v3 stages;
* Документирование API – посредством использования инструментов OpenAPI,Swagger;
* Кеширование даных -–может быть реализовано на стороне клиента(localstorage,sessionstorage), так и на стороне сервера(redis,memcache).[3]

HTTP-методы – определяют действие:

* GET – получить данные (например, список пользователей).
* POST – создать новый ресурс (регистрация пользователя).
* PUT / PATCH – обновить данные (изменить имя пользователя).
* DELETE – удалить ресурс (удалить пользователя).
* Форматы данных – обычно JSON (реже XML).[9]

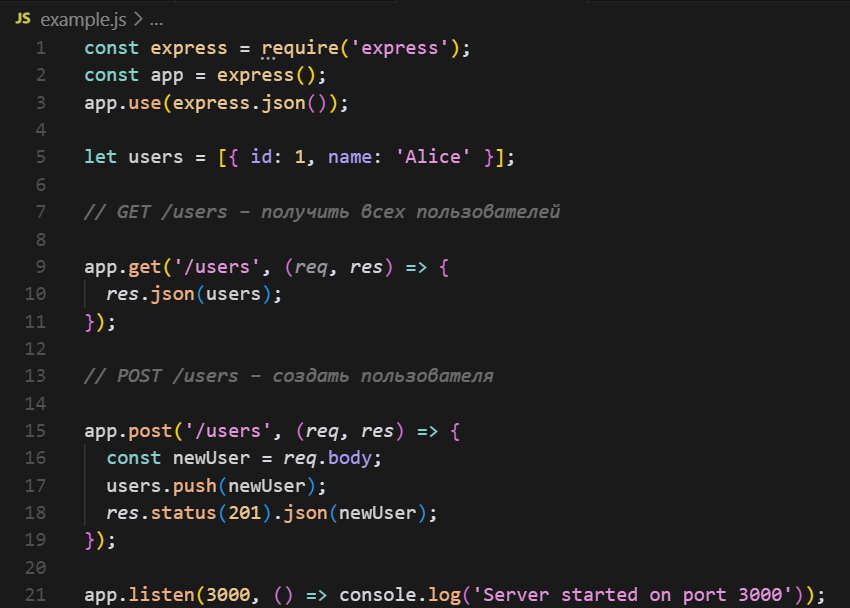
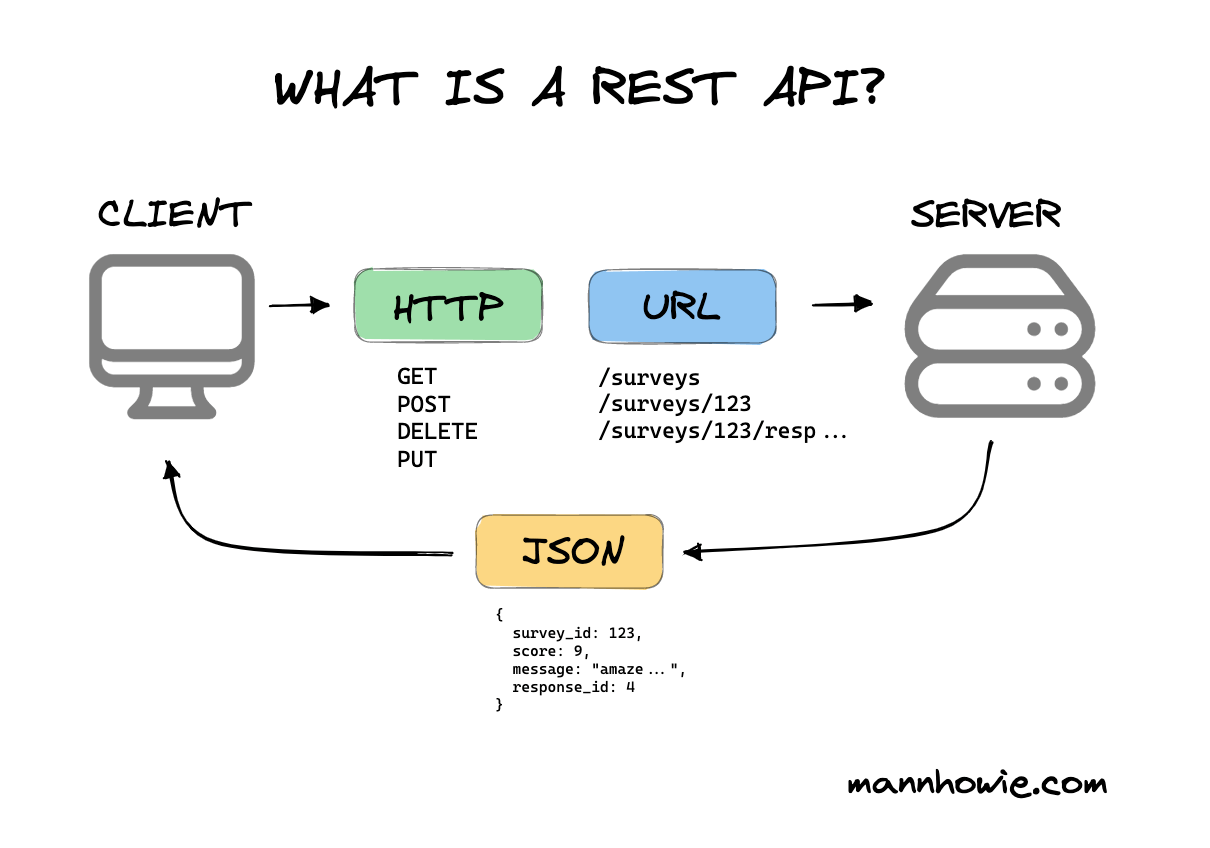
****Рисунок 1 – базовая схема реализации клиент-серверного взаимодействия посредством REST

Рисунок 2 – пример простейшего REST API сервиса

### MongoDB

MongoDB – представляет собой ведущую NoSQL-базу данных, которая кардинально отличается от традиционных реляционных систем. Её архитектура основана на документной модели, где данные хранятся в виде BSON-документов (бинарный JSON), что обеспечивает естественное отображение структур данных в современном программировании.[4]

Ключевые особенности:

* гибкая схема: в отличие от SQL-баз, MongoDB не требует строгого определения структуры таблиц заранее. Документы в одной коллекции могут иметь разную структуру, что особенно полезно при итеративной разработке;
* масштабируемость: поддержка горизонтального масштабирования через шардинг позволяет распределять данные по нескольким серверам. Репликация обеспечивает отказоустойчивость и высокую доступность;
* богатый язык запросов: поддерживает сложные запросы, включая поиск по полям, диапазонам, регулярным выражениям. Имеет мощные операторы агрегации для аналитики;
* индексация: поддерживает различные типы индексов (включая геопространственные и текстовые), что значительно ускоряет поиск.[6]

В качестве СУБД рекомендуется использовать – веб-версию в браузере(Atlas), или MongoDB Compass, однако также исправно работает с многофункциональными СУБД по типу DBeaver.[2]

### Swagger

OpenAPI (ранее Swagger) - это не просто инструмент документации, а полноценная экосистема для проектирования, разработки и сопровождения API. Спецификация OpenAPI стала отраслевым стандартом для REST API.

Направлен на описание и документирование моделей API сервисов в формате YAML/JSON, упрощения взаимодействия клиентской и серверной части ИС.

Возможности:

* жизненный цикл API: Поддержка всего цикла от проектирования (Swagger Editor) до тестирования (Swagger UI) и мониторинга;
* генерация кода: Возможность автоматической генерации клиентских и серверных заглушек на различных языках;
* валидация: Встроенные механизмы проверки соответствия API его спецификации;
* безопасность: Поддержка описания схем аутентификации (OAuth2, API Keys,JWT).[5]

### 1.1.4 NodeJS

Node.js – это серверная платформа для выполнения JavaScript вне браузера.

Основные моменты:

* работает на движке V8 (Google Chrome);
* aсинхронная, событийно-ориентированная архитектура (non-blocking I/O).
* используется для бэкенд-разработки, API, микросервисов, чат-ботов;
* имеет огромное сообщество и экосистему (npm – крупнейший пакетный менеджер);
* открывает возможности нативно работать с сокетами, файловой системой; [1,10]

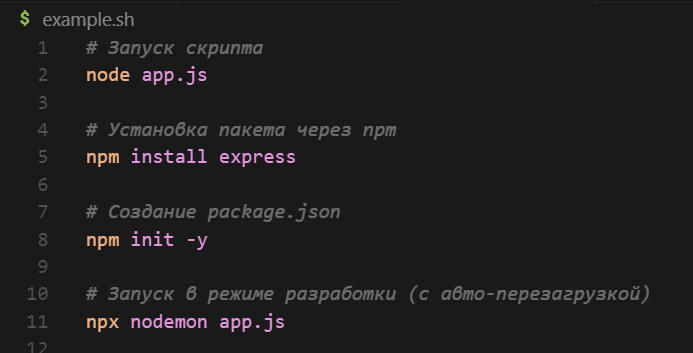


Рисунок 3 – основные команды для работы с Node JS



Рисунок 4 – пример простейшего сервера на Node JS

## 1.2 Проектирование БД

### 1.2.1 Физическая диаграмма (диаграмма классов)

Эта диаграмма ближе к физической реализации в MongoDB, показывая "коллекции" (в виде классов) и их поля.

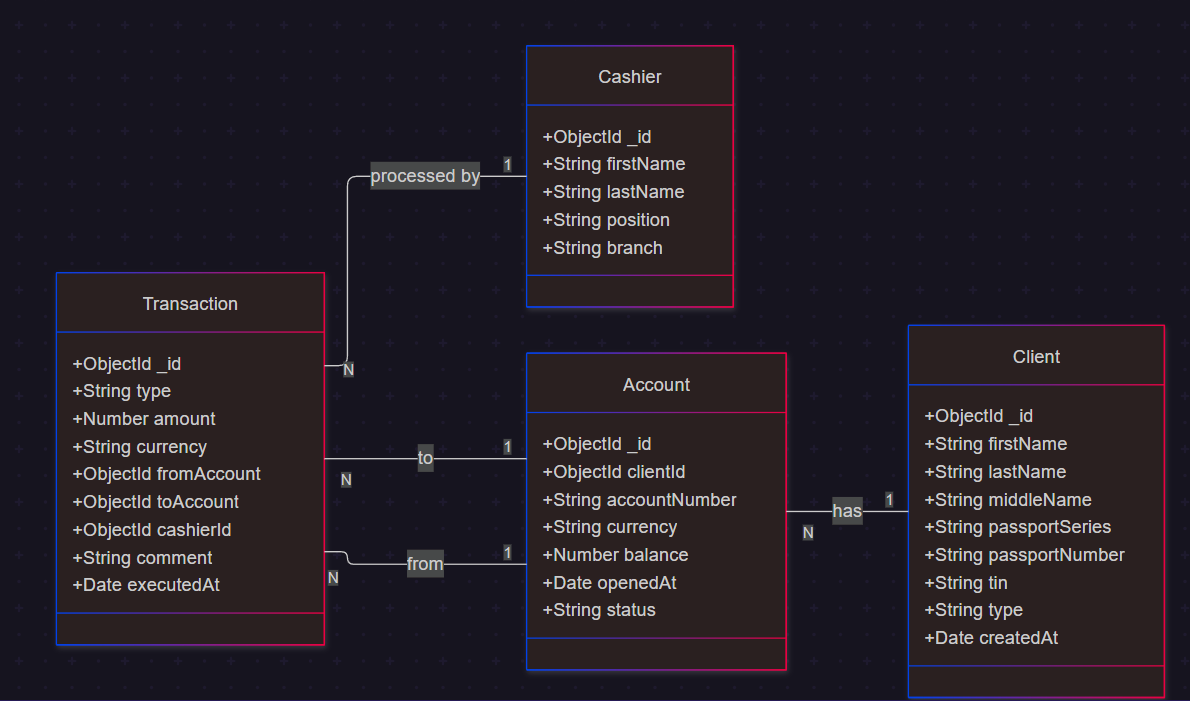


Диаграмма 1 - Физическая диаграмма (диаграмма классов) исходя из технического задания

Показывает, как именно база данных будет физически устроена. Она детализирует каждую "коллекцию" (аналог таблицы в NoSQL), перечисляя все её поля, их типы данных (String, ObjectId, Number) и ключи (PK — Primary Key, FK — Foreign Key).

Это уже технический чертеж для разработчиков. По нему создаются модели в коде и настраивается база данных. Она устраняет двусмысленность и служит точной спецификацией для реализации.[8]

### 1.2.2 ER-диаграмма

Данный тип диаграмм показывает сущности и их взаимодействия на концептуальном уровне. Показывает основные сущности вашей системы (Клиент, Счёт, Транзакция) и логические связи между ними. Например, один Клиент может иметь много Счетов. Это взгляд на данные с высоты птичьего полета. Нужна для проектирования и анализа структуры данных на концептуальном уровне. Она помогает понять бизнес-логику и убедиться, что все данные правильно взаимосвязаны, еще до написания кода. [11]

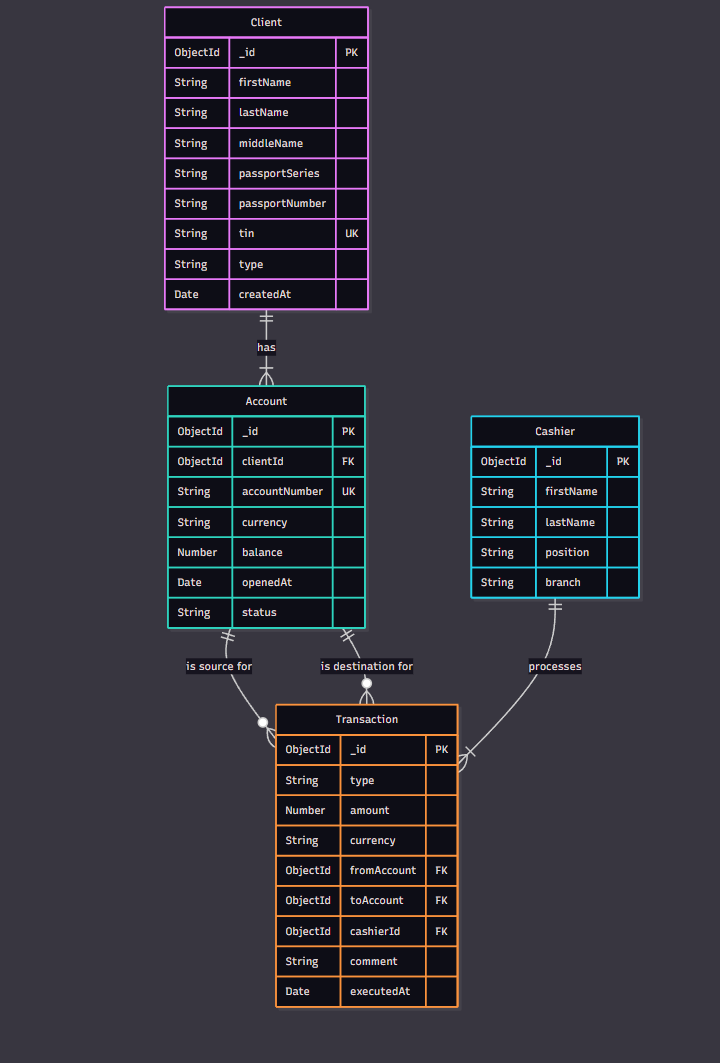


Диаграмма 2 - Физическая диаграмма (диаграмма классов) исходя из технического задания

# РАЗДЕЛ 2. ПРАКТИЧЕСКАЯ ЧАСТЬ

### 2.1 Создание Базы Данных

Для реализации технического задания по практике, первым делом была создана База Данных, для удобства я использовал образ mongo:latest и запускал базу в контейнере, благодаря использования volume – таблицы и записи в базе будут сохранятся при остановке/перезапуске контейнера.

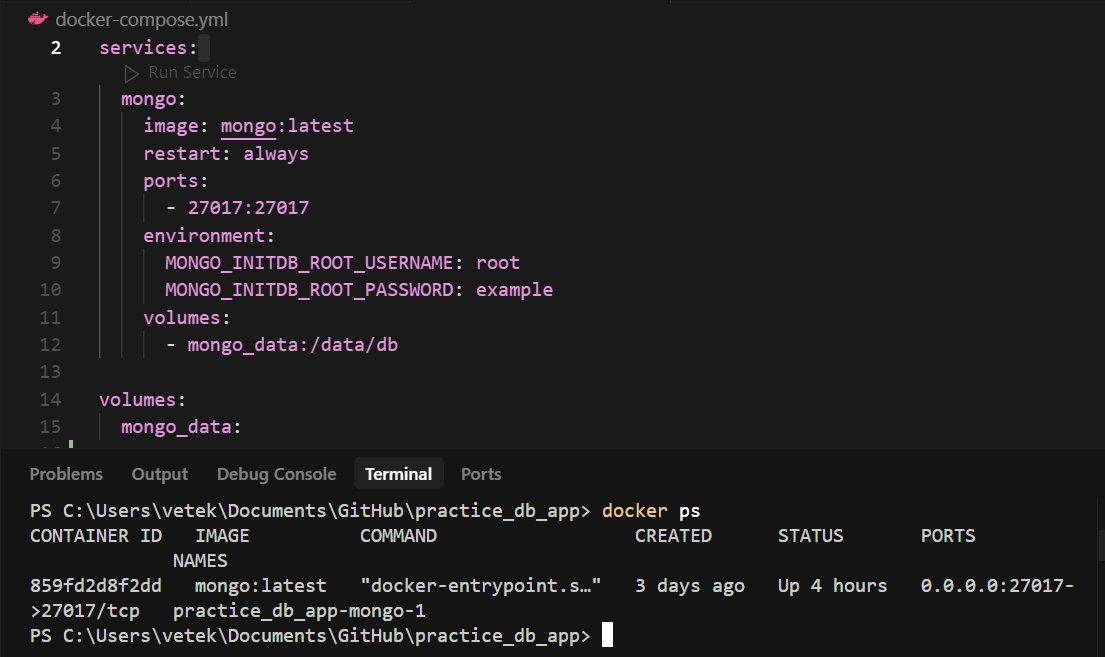
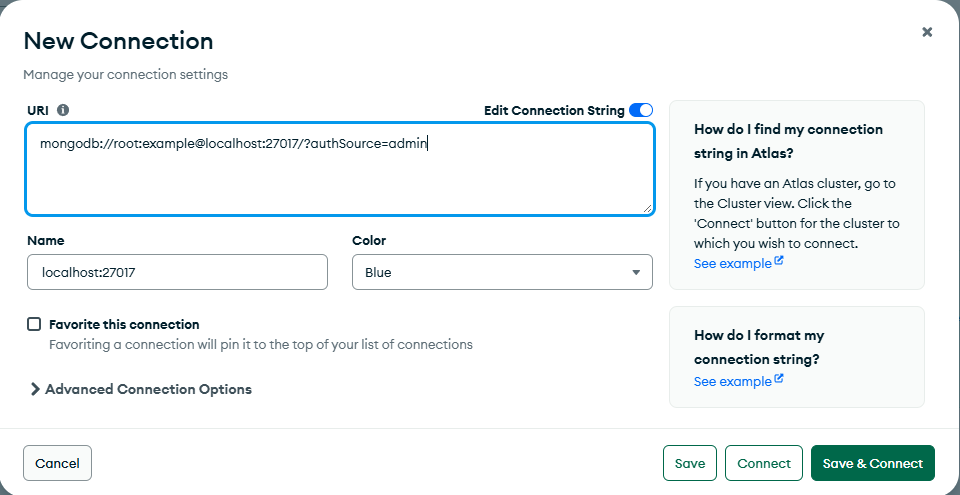


Рисунок 5 – docker-compose файл с конфигурацией БД mongoDB

После поднятия контейнера БД будет доступна на порту 21017 на всех сетевых интерфейсах системы.

Рисунок 6 – проверка подключения к БД через СУБД MongoDB Compass

### 2.2. Создание таблиц и заполнение таблиц начальными полями.

Для наполнения таблиц тестовыми значениями был написан и запущен скрипт – seed.js. [листинг кода в Приложении 1]

Основные этапы:

1. Подключение к MongoDB

* использует строку подключения из .env (переменная MONGO\_URI).

1. Очистка старых данных

* удаляет все записи из коллекций: клиентов, счетов, транзакций и кассиров.

1. Генерация кассиров

* создаёт 5 случайных кассиров с помощью библиотеки faker.

1. Генерация клиентов и счетов

* создаёт 20 клиентов с фейковыми паспортными данными;
* для каждого клиента создаёт банковский счёт с валютой, балансом и статусом.

1. Генерация транзакций

* создаёт 50 транзакций разных типов: пополнение (deposit), снятие (withdrawal), перевод (transfer);
* для переводов выбирает разные счета отправителя и получателя;
* каждая транзакция привязана к случайному кассиру.

1. Завершение работы

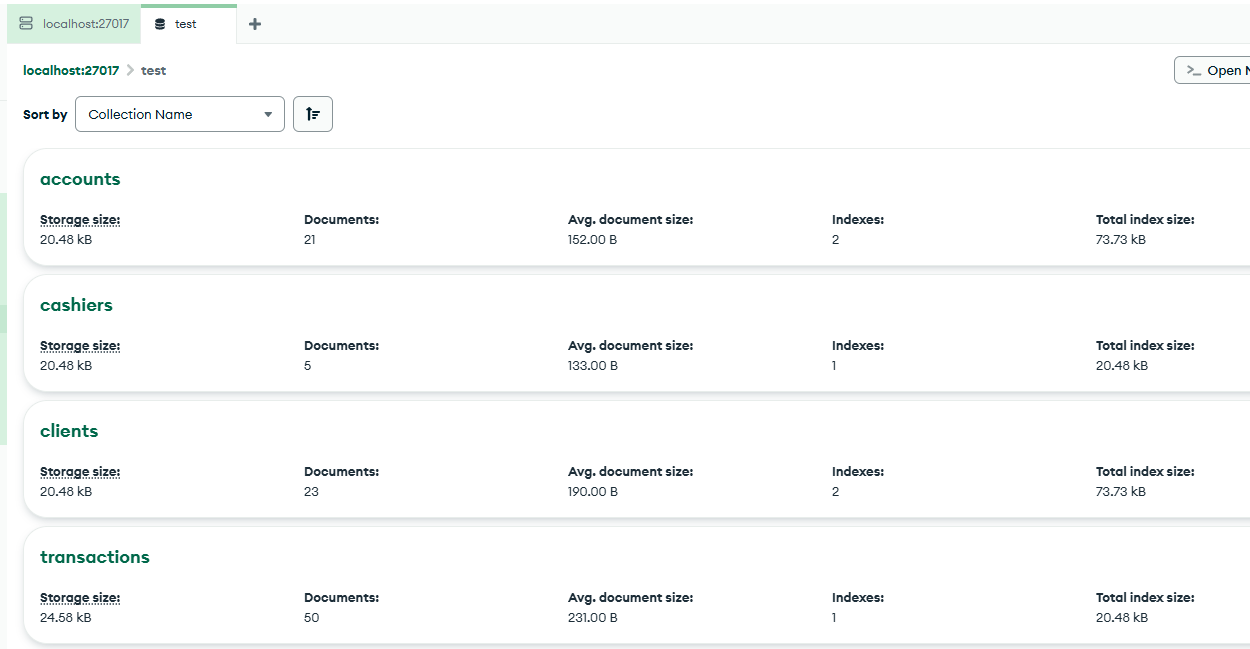
* после генерации данных скрипт отключается от базы.

Рисунок 7 – проверка что БД test и 4 таблицы в ней успешно создались через MongoDB Compass

### 2.3 Проверка созданных таблиц и корректности полей в них

Было выделено 4 основные таблицы в качестве сущностей БД:

* accounts – сущность банковского счета пользователя в ИС банка;
* clients – сущность пользователя(клиента) в ИС банка;
* transactions – сущность каждой операций по аккаунту пользователя;
* cashiers – сущность кассира(оператора), обрабатывающего транзакцию в ИС банка.

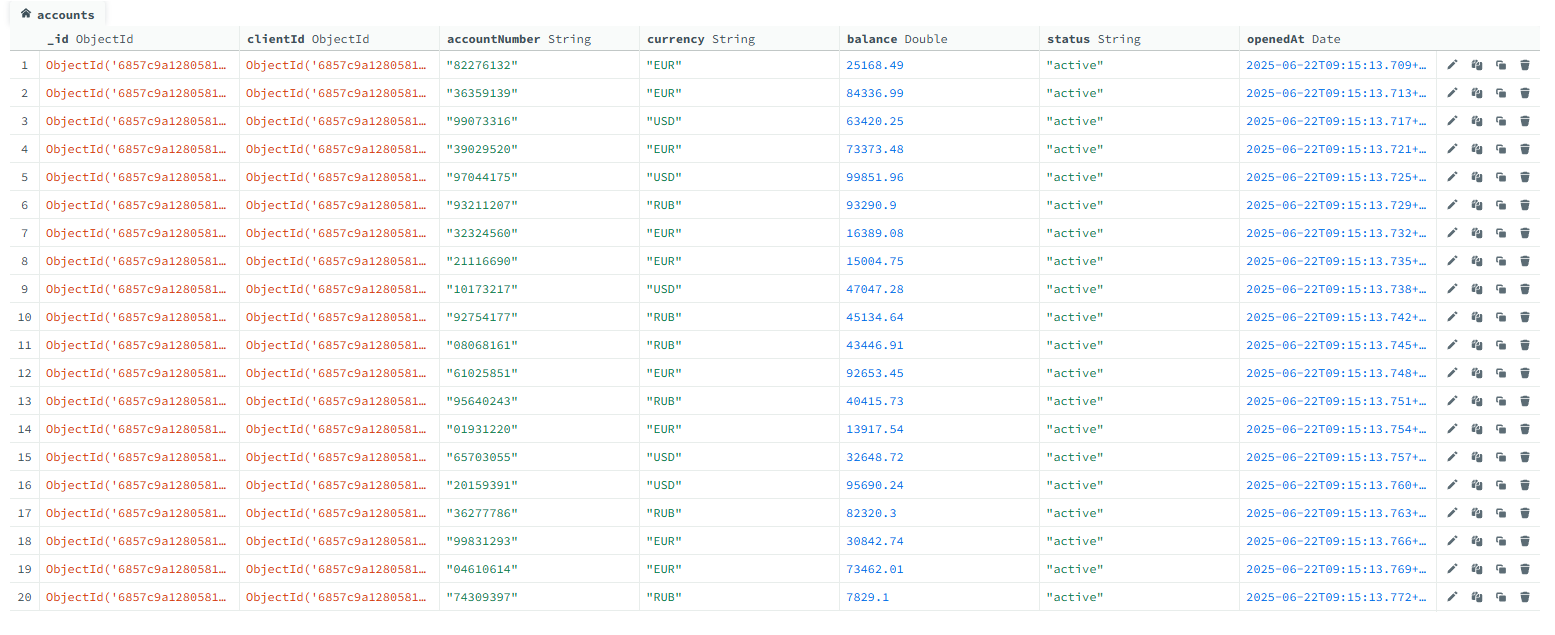
Содержит 20 начальных записей, таблица выглядит следующим образом:

Рисунок 8 – таблица accounts

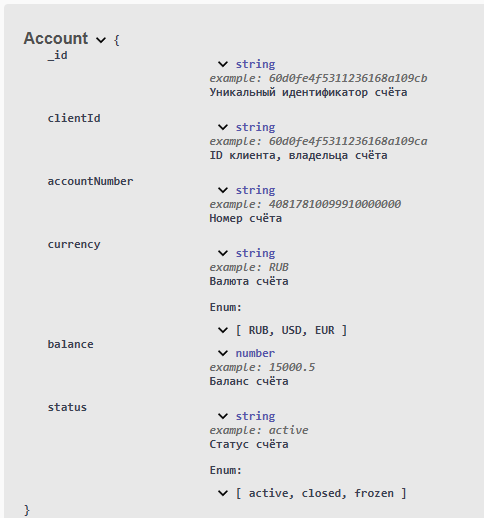


Рисунок 9 – схема данных таблицы accounts



Рисунок 10 – таблица clients



Рисунок 11 – схема данных таблицы clients

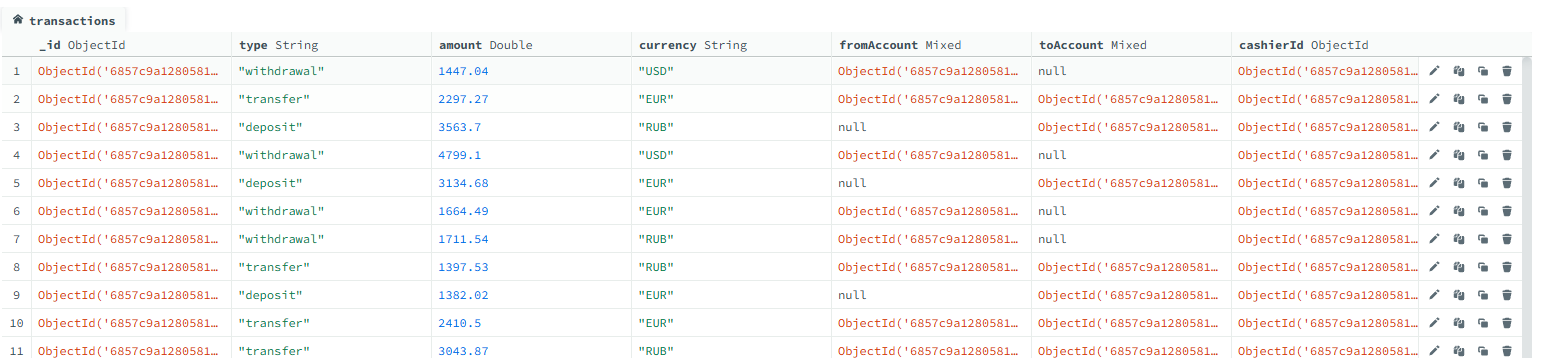


Рисунок 12 – таблица transactions



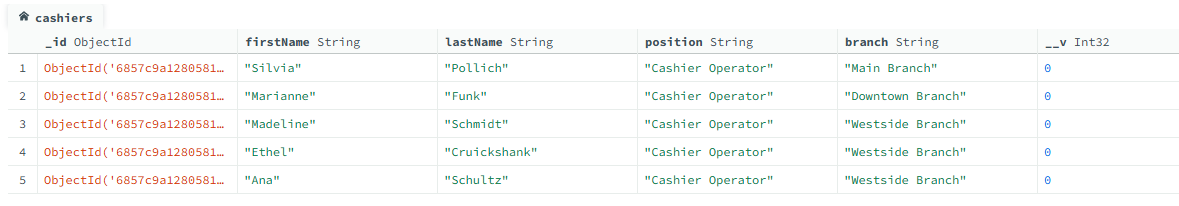
Рисунок 13 – схема данных таблицы transactions

Рисунок 14 – таблица cashiers



Рисунок 15 – схема данных таблицы cashiers

### 2.4 Написание программы для работы с БД

Программная часть серверного приложения была написана на ExpressJS и Mongoose (ODM) для удобной работы в стиле ООП с mongoDB.

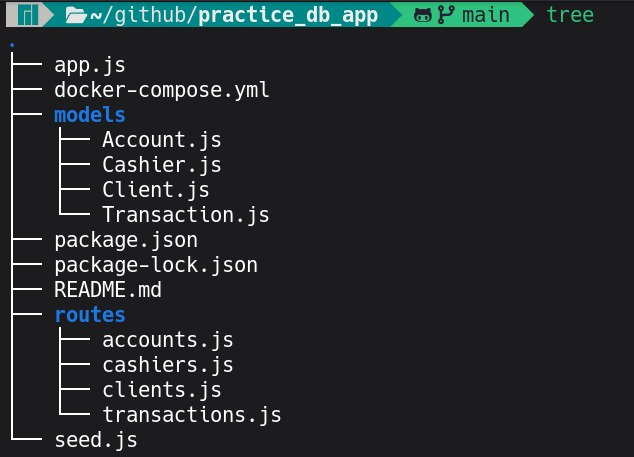


Рисунок 16 – общая структура проекта

- app.js – основной файл приложения, в нем происходит инициализация Express-сервера, подключение к базе данных и настройка маршрутов для обработки endpoints;

- docker-compose.yml – конфигурация контейнера для запуска образа с mongoDB;

- package.json – основной файл npm-пакета .Содержит информацию о проекте, его зависимостях, скриптах для запуска и сборки;

- package-lock.json - aвтоматически генерируемый файл, фиксирующий точные версии установленных npm-зависимостей;

- README.md - документация по проекту: как запускать, описание функционала, требования и т.д;

- seed.js - скрипт для заполнения базы данных начальными (тестовыми) данными;

-swagger.yaml - описание API в формате Swagger/OpenAPI. Используется для автогенерации документации и тестирования API;

- models/ - Содержит модели данных для работы с базой данных (обычно через ORM, например, Sequelize или Mongoose):

* account.js - модель для работы с аккаунтами (счетами) пользователей;
* cashier.js - модель для кассиров (например, сотрудники банка или магазина);
* client.js - модель для клиентов (пользователей, владельцев счетов);
* transaction.js - модель для транзакций (операций между счетами, пополнения, снятия и т.д.).

- routes/ - содержит файлы с маршрутами (endpoint'ами) для Express-приложения:

* accounts.js - маршруты для работы с аккаунтами (создание, просмотр, изменение, удаление счетов);
* cashiers.js - маршруты для работы с кассирами;
* clients.js - маршруты для работы с клиентами;
* transactions.js - маршруты для работы с транзакциями.

Как это работает вместе:

1. app.js запускает сервер, подключает модели и маршруты;
2. models/ определяет структуру данных и взаимодействие с базой;
3. routes/ реализует API для работы с этими данными.

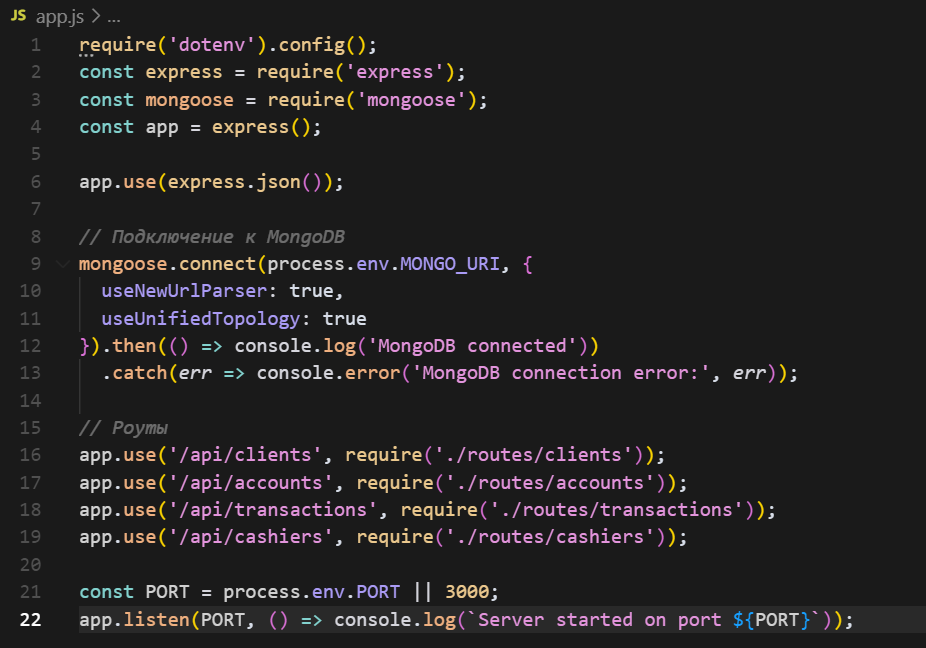
Рассмотрим реализацию app.js, а также для примера схему Транзакций и реализацию endpoint для нее.

Рисунок 17 – листинг основного скрипта приложения ./app.js

В файле ./app.js:

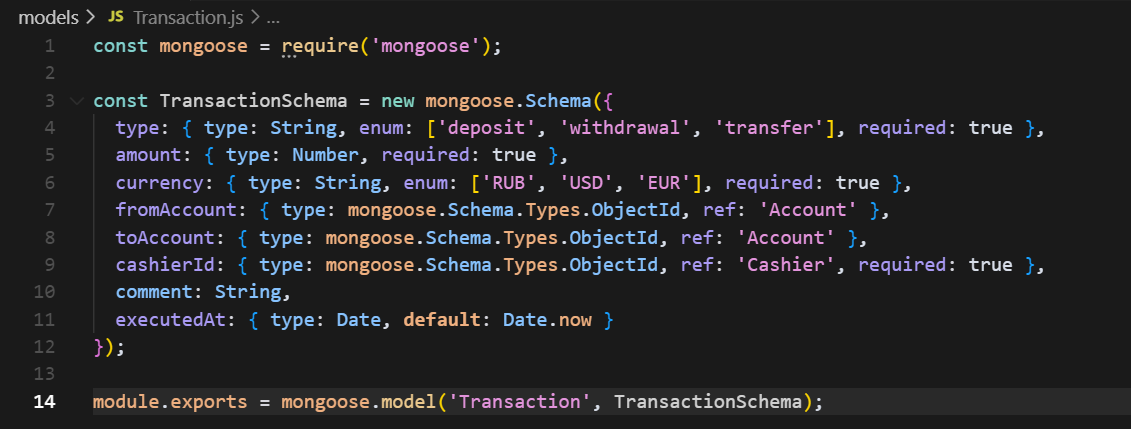
1. загрузка переменных окружения из файла для подключения к БД;
2. импорт библиотек mongoose и express;
3. создается приложение(app), на базе фреймворка express;
4. посредством модуля mongoose (ODM для NodeJS) идет подключение приложением к БД;
5. к приложению подключаются «ручки» (endpoind) и указывается путь до файлов с их реализацией в проекте;
6. приложение запускается на локальном сокете.

Рисунок 18 – листинг модели данных для таблицы Transactions (файл models/Transaction.js)

В файле models/Transaction.js:

1. импортируется библиотека mongoose для работы с MongoDB;
2. описывается схема (структура) данных для транзакции: тип, сумма, валюта, счета отправителя и получателя, кассир, комментарий, дата выполнения;
3. экспортируется модель Transaction для использования в других частях приложения.

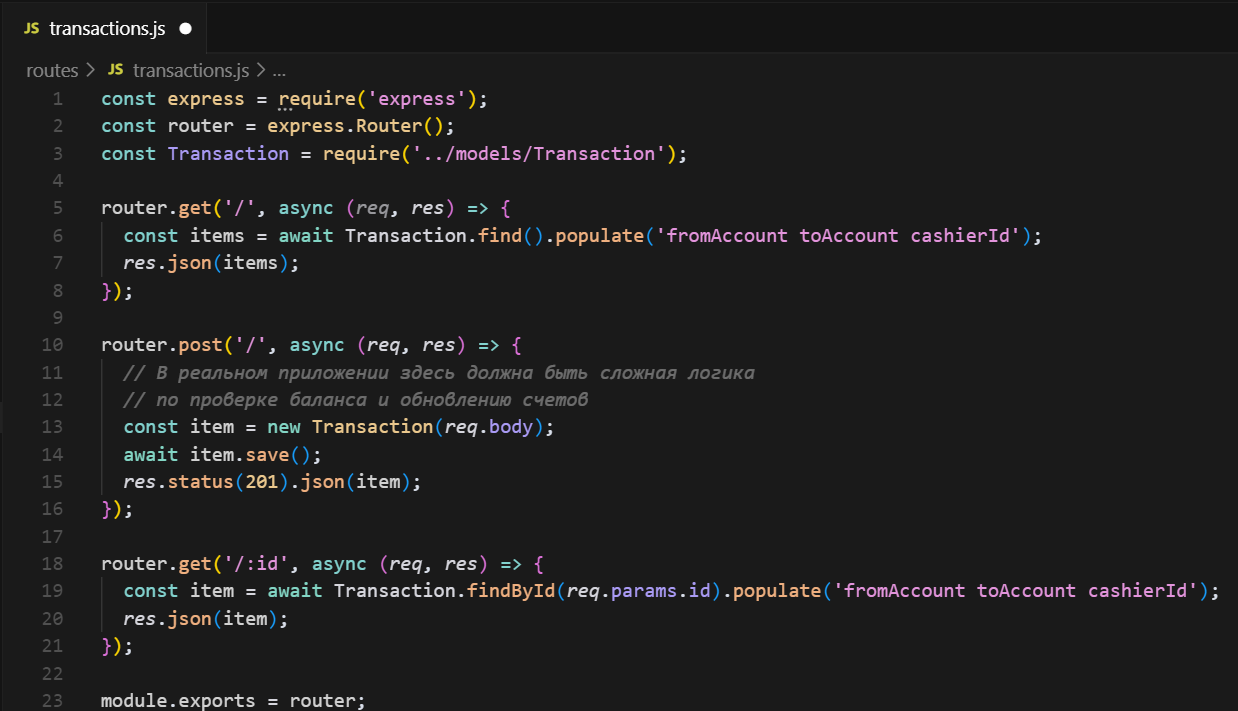
****

Рисунок 19 – листинг endpoints для схемы данных Transactions

(файл routes/transactions.js)

В файле routes/transactions.js:

1. импортируется express и создаётся router для маршрутов;
2. импортируется модель Transaction;
3. описывается GET-роут по адресу / — возвращает список всех транзакций с деталями счетов и кассира;
4. описывается POST-роут по адресу / — создаёт новую транзакцию на основе данных из запроса;
5. описывается GET-роут по адресу /:id — возвращает одну транзакцию по её id с деталями счетов и кассира;
6. экспортируется router для подключения в основное приложение.

C аналогичным листингом остальных файлов, а именно схемы коллекций(таблиц) для БД и реализация endpoint для них, вы можете ознакомиться подробнее в Приложении к работе.

### 2.5 CRUD операции к таблицам БД и проверка работоспособности решения (Postman, MongoDB Compass)

Для проверки созданных endpoint к таблицам БД, я использовал утилиту Postman. Рассмотрим основные запросы и результаты их выполнения.

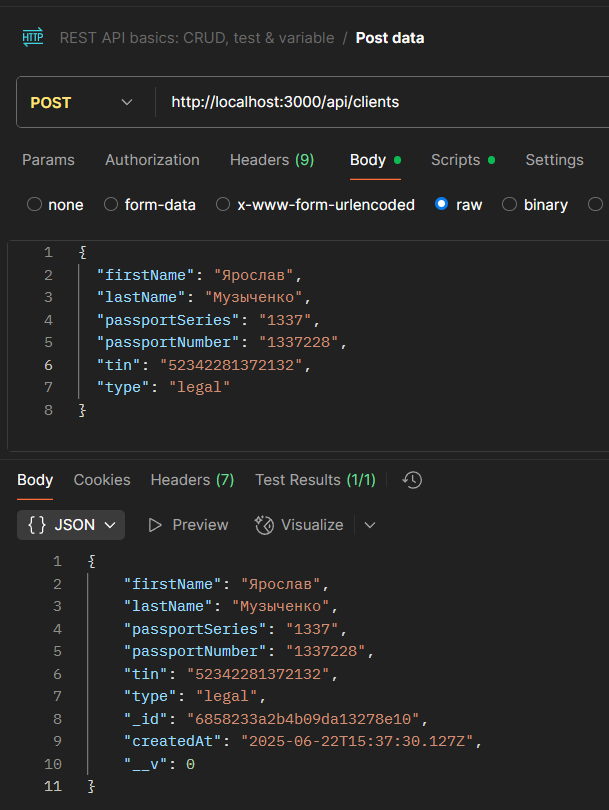


Рисунок 20 – POST запрос на endpoint /clients – создание нового пользователя

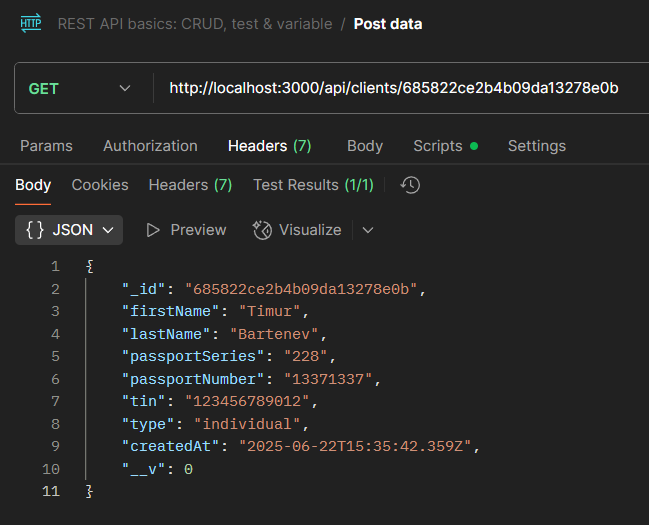


Рисунок 21 – GET запрос на endpoint /clients/{id} – получение конкретного пользователя по его ID

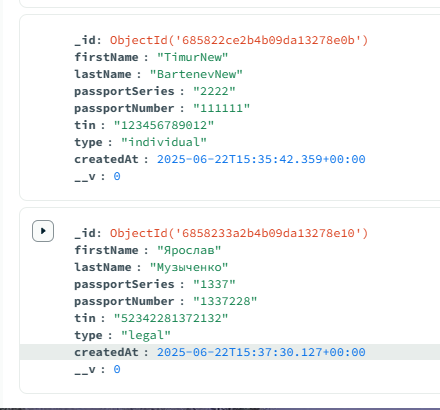


Рисунок 22 – проверка что действия над записями и информацией в записях таблиц действительно корректно применяются в БД

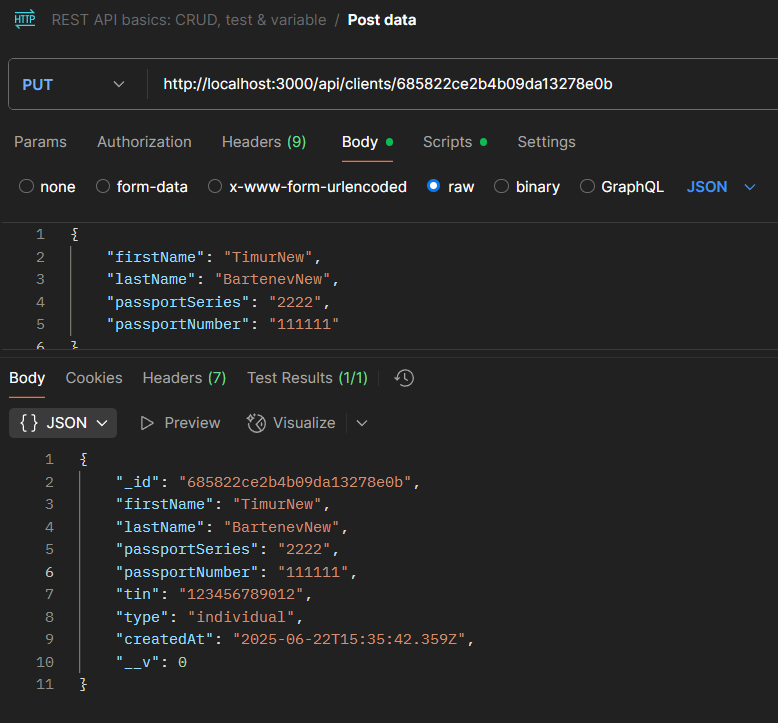


Рисунок 23 – PUT запрос на endpoint /clients/{id} – обновление информации о пользователе банка по его ID

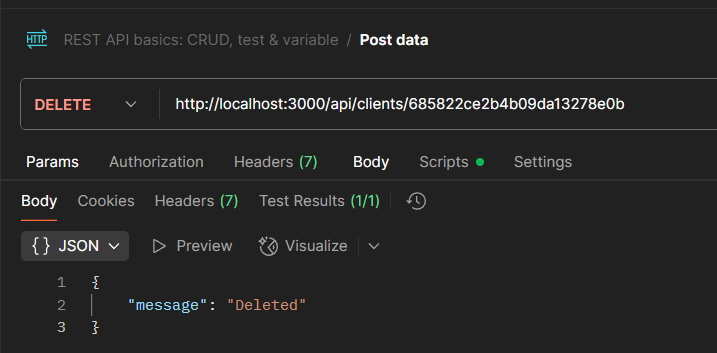


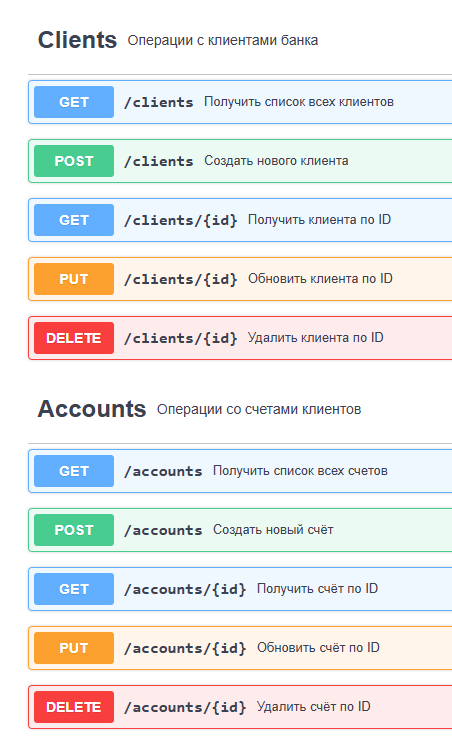
Рисунок 24 – DELETE запрос на endpoint /clients/{id} – удаление записи о пользователе из таблицы clients по его ID

Рисунок 25 – swagger спецификация endpoint к сущностям БД

В целом для остальных endpoints реализованы похожие базовые CRUD операции, там, где это возможно, вот как выглядит список возможных операций в swagger-документации, вы можете ознакомиться на рисунках 23 и 24.

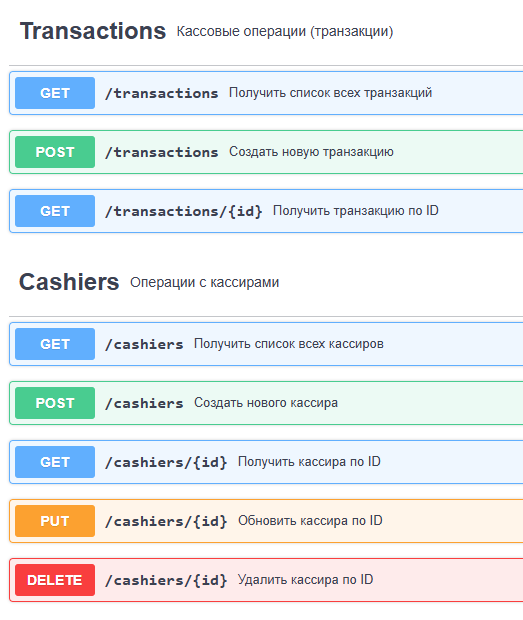


Рисунок 26 – swagger документация endpoint к сущностям БД

### 2.6 Создание собственных запросов к таблицам БД через MQL (MongoDB Query Language)

Для каждой коллекции(таблицы) реализовал несколько проверочных и полезных запросов для выборки и последующего анализа.

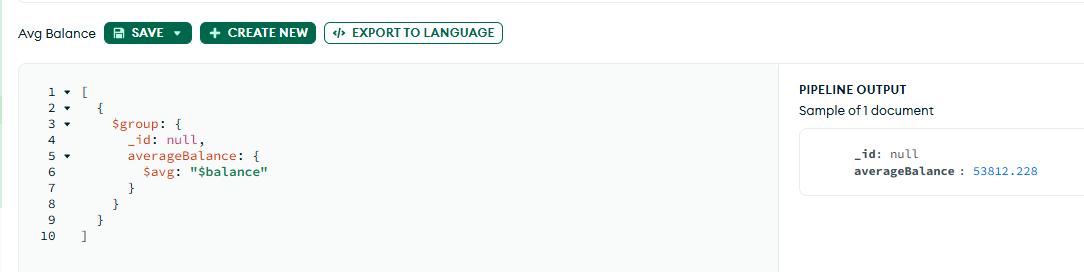


Рисунок 27 – вывод среднего кол-ва средств на балансе аккаунтов (во всех валютах)

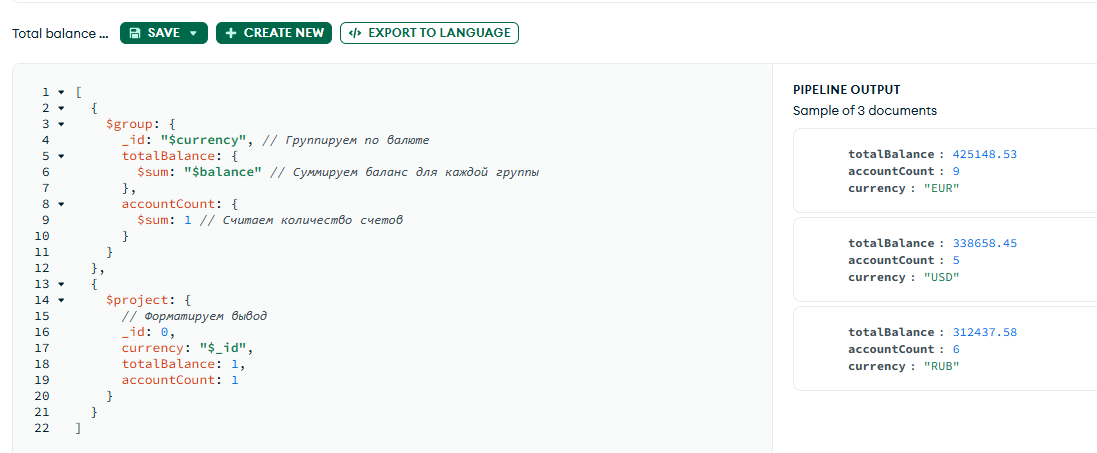


Рисунок 28 – вывод общего кол-ва средств на счетах пользователей с разделением по валютным единицам

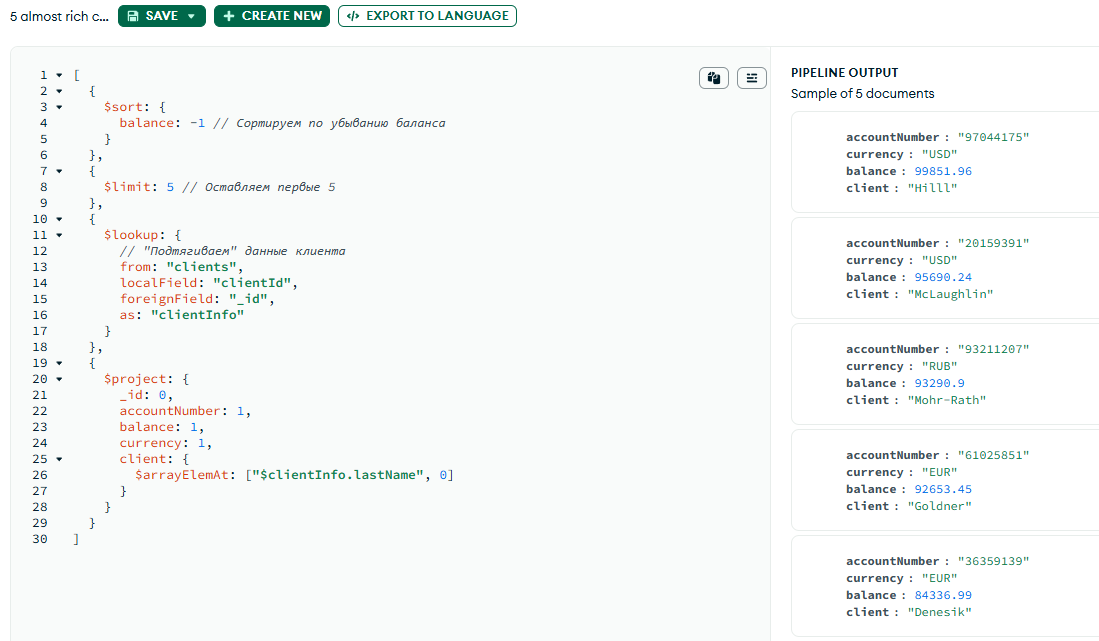


Рисунок 29 – вывод пяти самых богатых клиентов по всем валютам

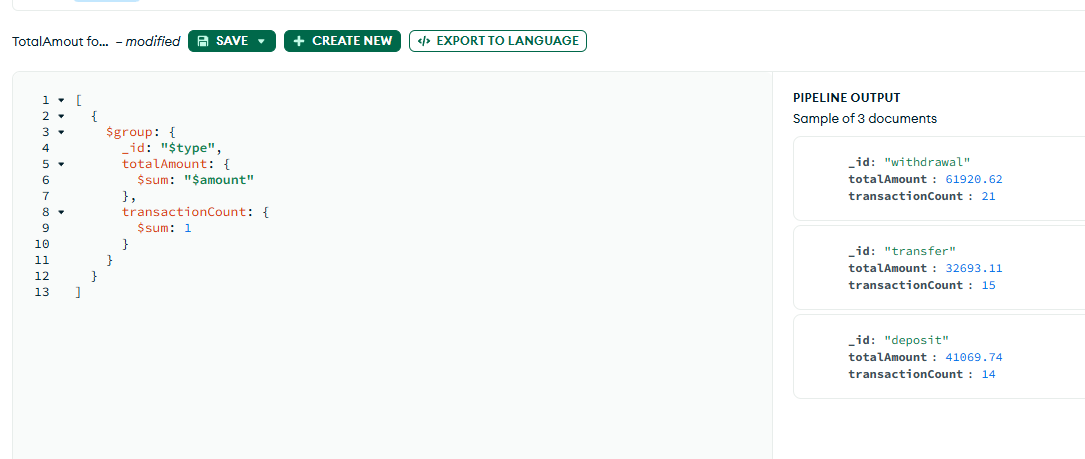


Рисунок 30 – вывод средней суммы для каждого типа перевода

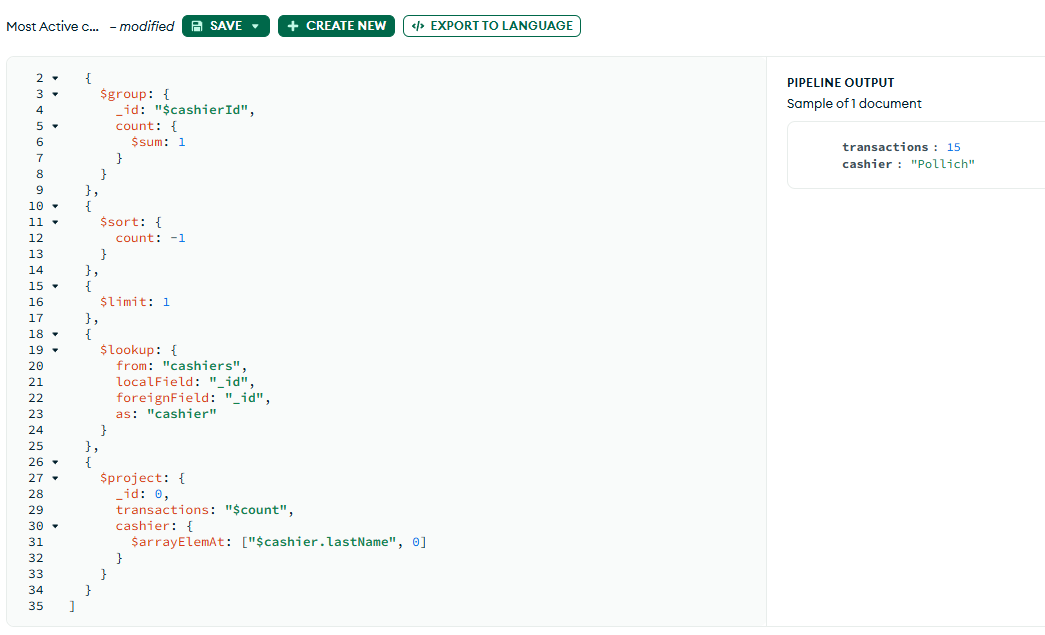


Рисунок 31 – вывод информации о самом эффективном кассире (провел больше всего транзакций)

### 2.7 Реализация пользовательского интерфейса

Для пользователя был реализован UI в браузере. Интерфейс был реализован без использования фреймворков и библиотек, использовался классический HTML5 + CSS + JS(ES6), для иконок/шрифтов использовался сервис – Font Awesome. Позиционирование элементов выполнено на Flexbox.

Для обращения к БД – использовался AJAX подход, который в данный момент является стандартом, который не блокирует основной поток выполнения JS, и за счет worker-ов(fetch) позволяет выполнять асинхронные операции(такие как I/O,обращения к FS, БД, API и другим сущностям)

Схема запроса выглядит следующим образом:

Браузер → fetch('/api/clients') → Express Router → Mongoose → MongoDB

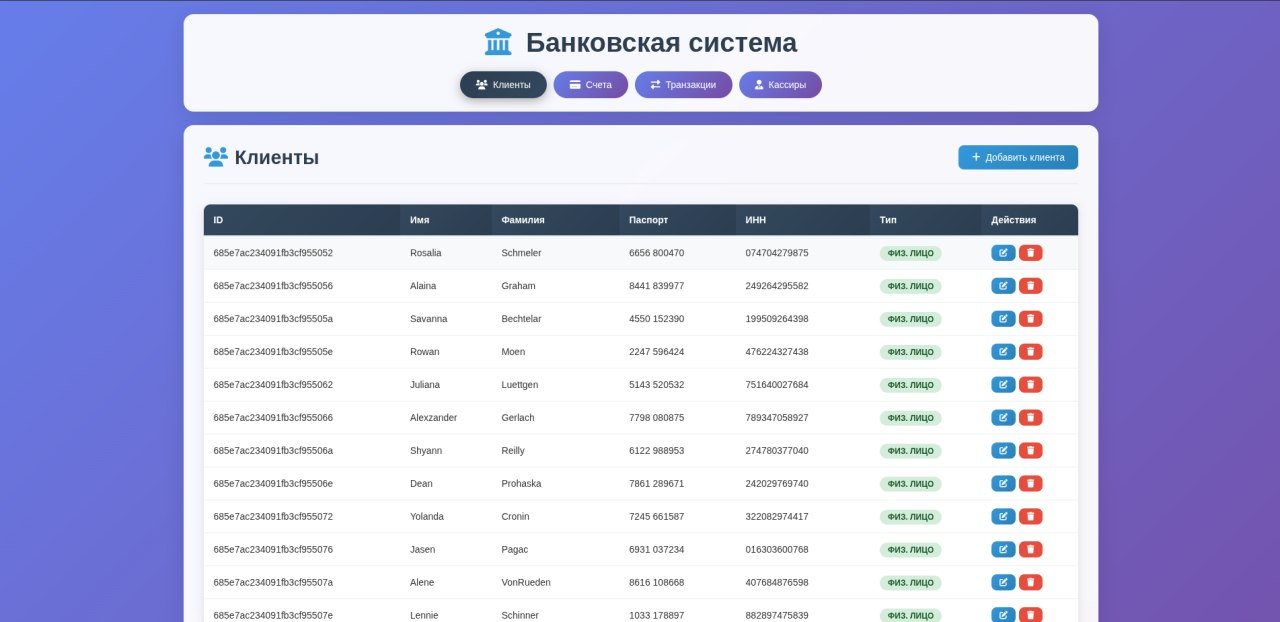


Рисунок 31 – страница для работы с клиентами

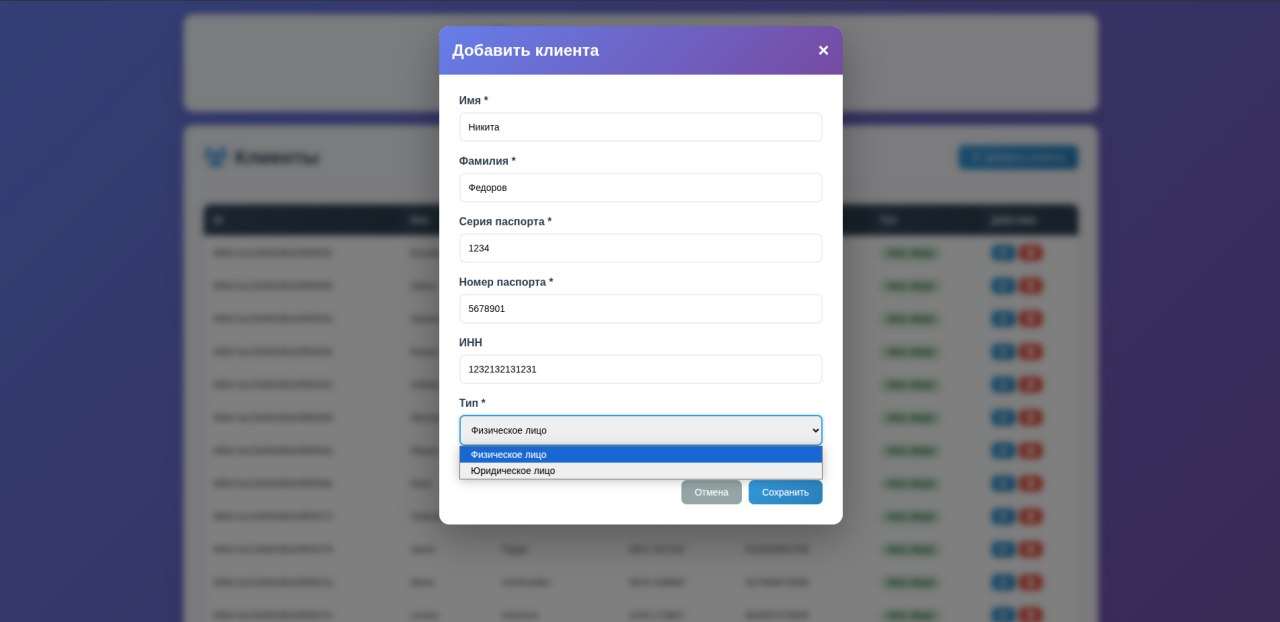


Рисунок 32 – модальное окно добавления клиента в базу

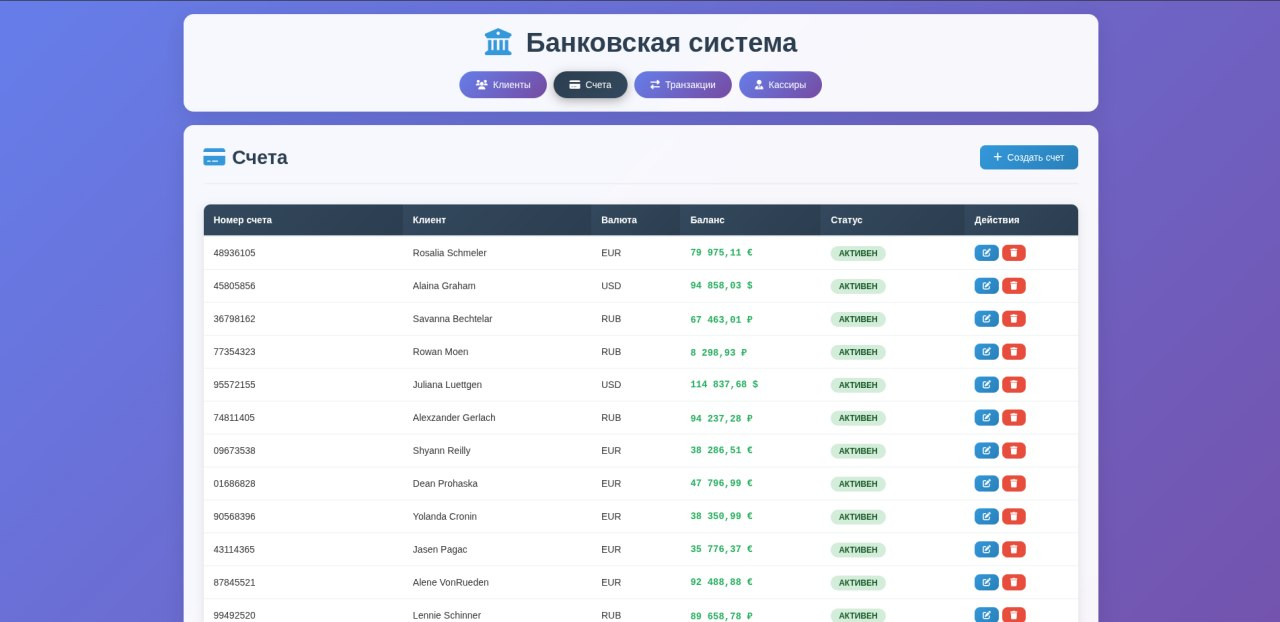


Рисунок 33 – страница для работы со счетами клиентов

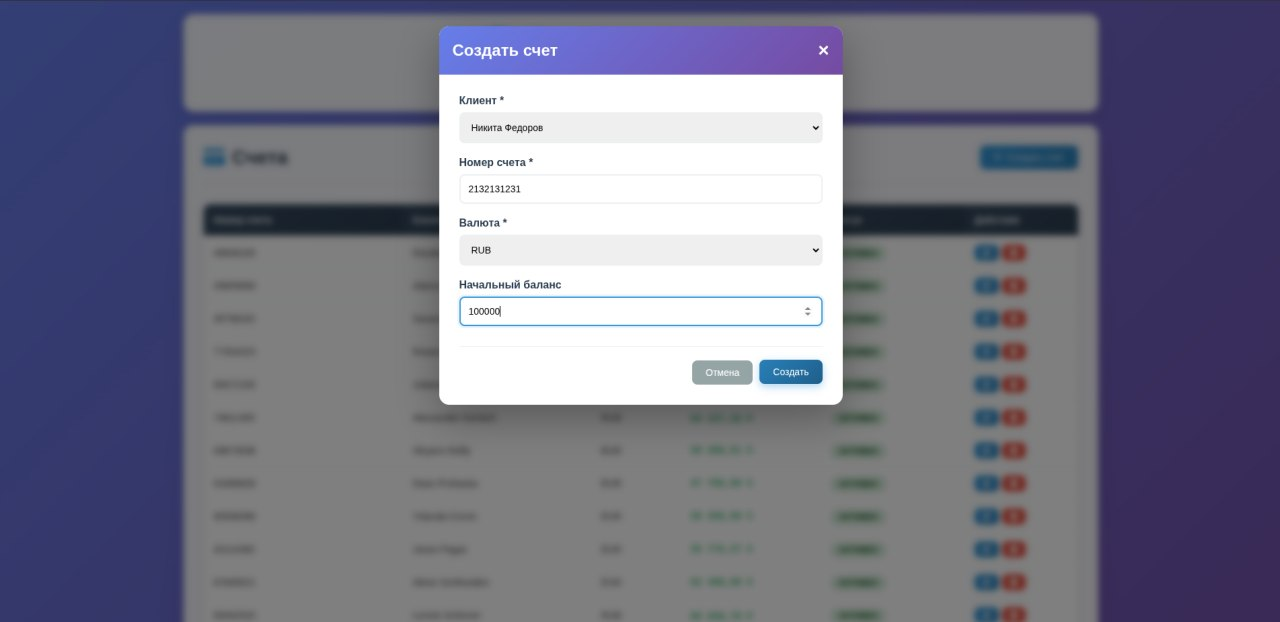


Рисунок 34 – модальное окно добавления счета для клиента

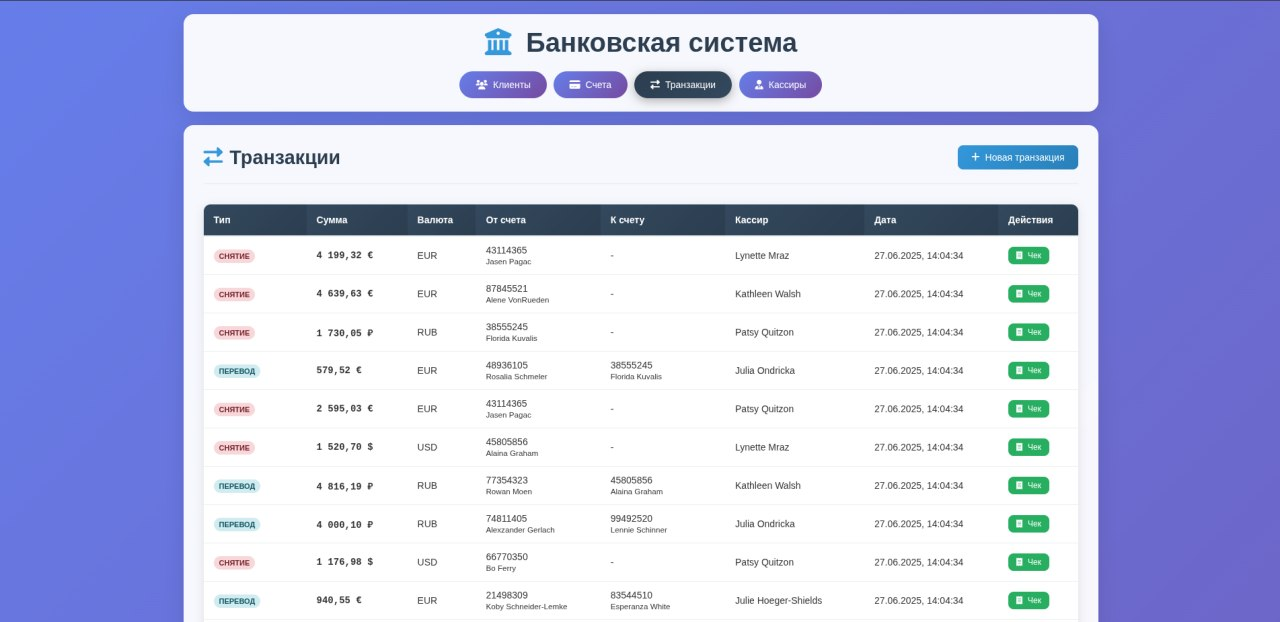


Рисунок 35 – страница для работы с транзакциями по счетам

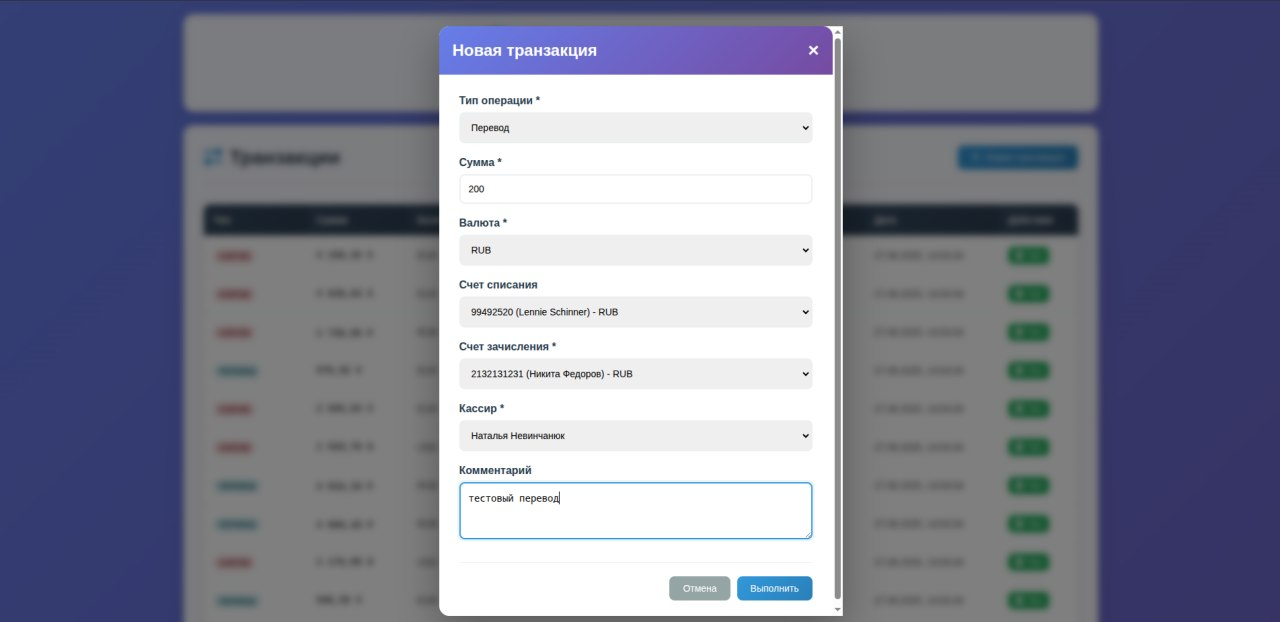


Рисунок 36 – модальное окно создания новой транзакции

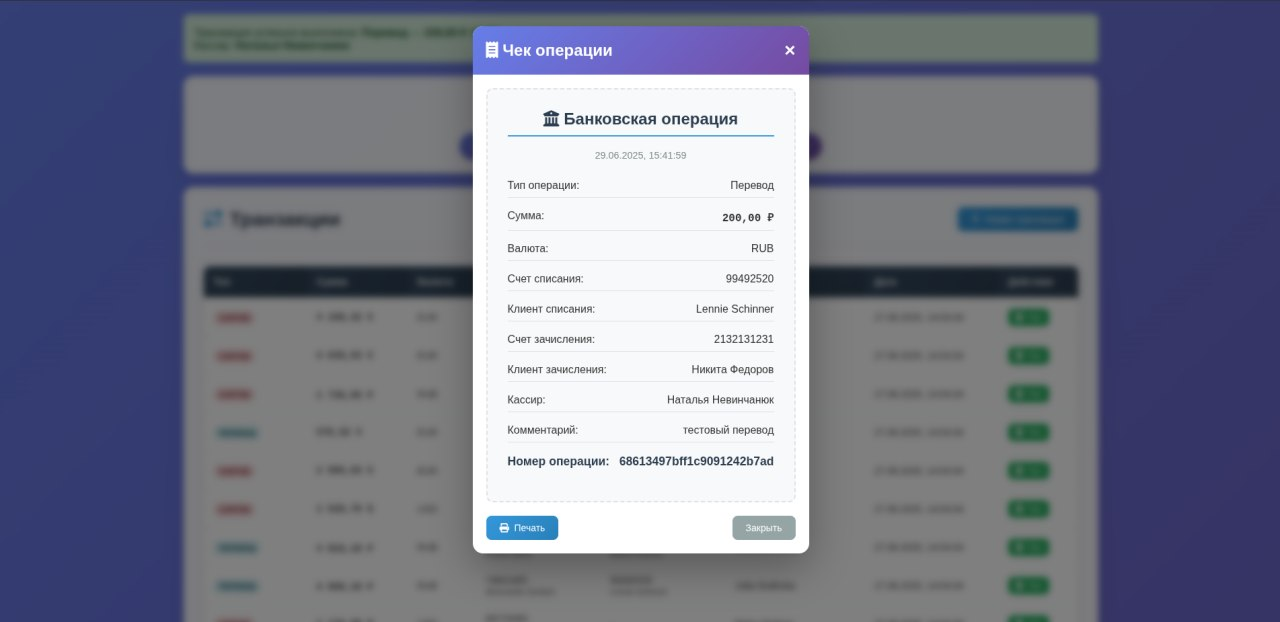


Рисунок 37 – формирование чека при успешной операции

При успешной операции для каждой транзакции формируется чек, он появляется сразу после формирования операции, или его можно открыть из таблицы для соответствующей записи.

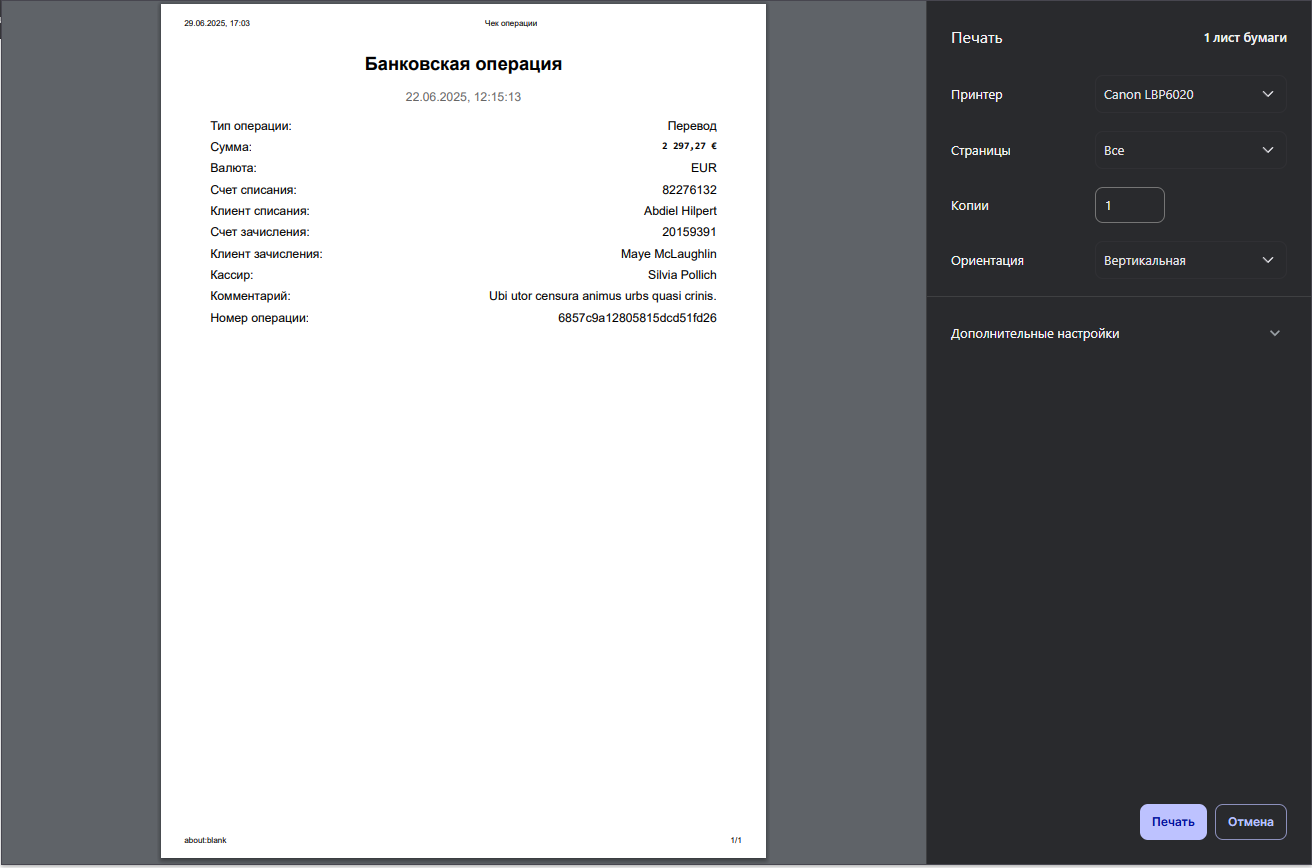


Рисунок 38 – страница печати чека транзакции



Рисунок 39 – страница для работы с кассирами банка

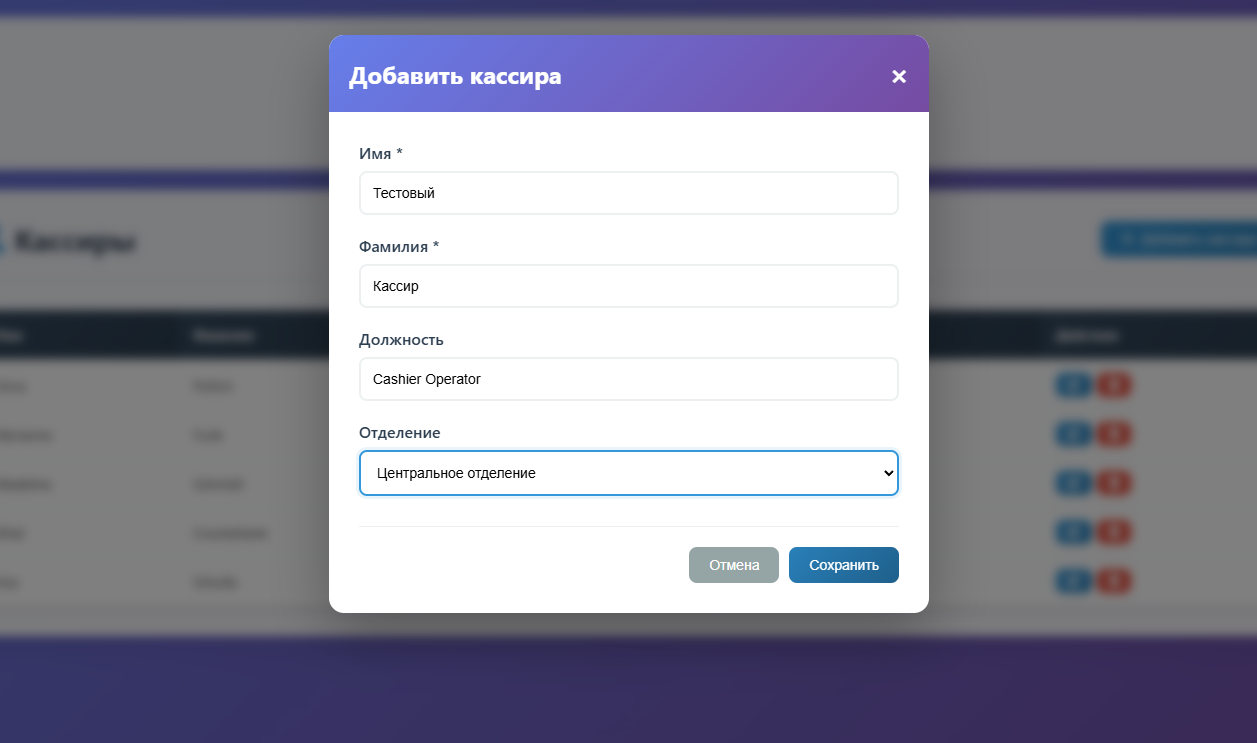


Рисунок 40 – модальное окно добавления нового кассира

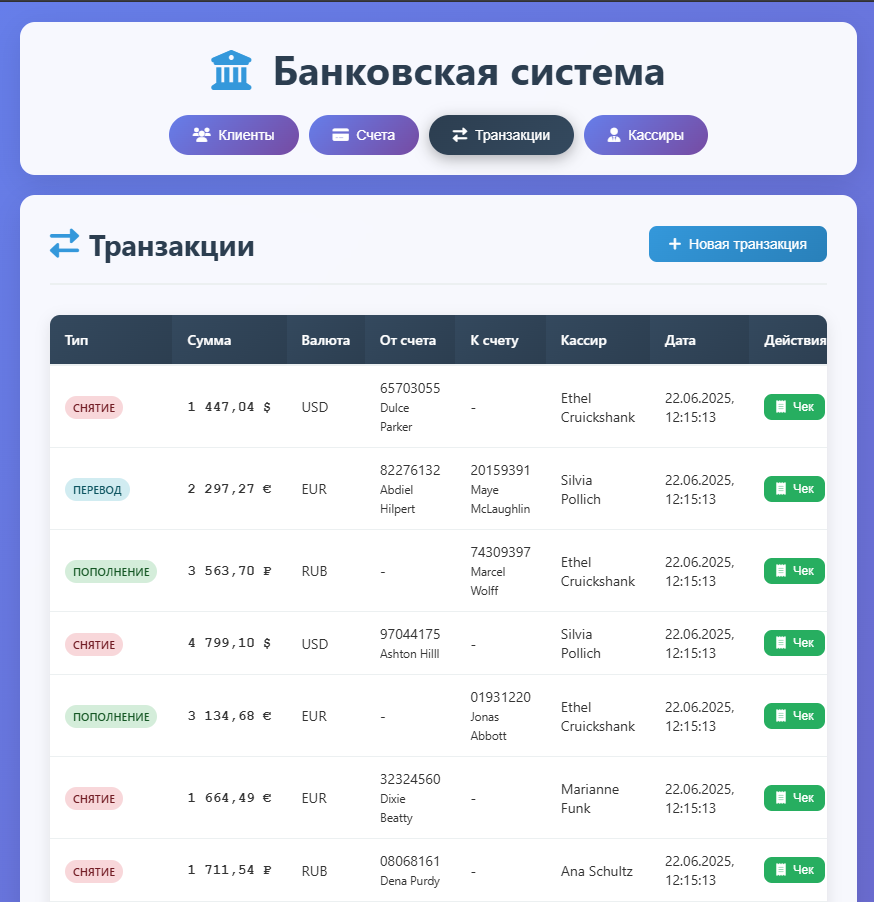


Рисунок 41 – адаптивная верстка для планшетов

Для удобства пользователей была реализована адаптивная верстка с использованием css @media-query, для каждой страницы интерфейса, с примерами вы можете ознакомится на рисунках 41 и 42, так же при успешных и ошибочных операциях были добавлены push alert вверху экрана.

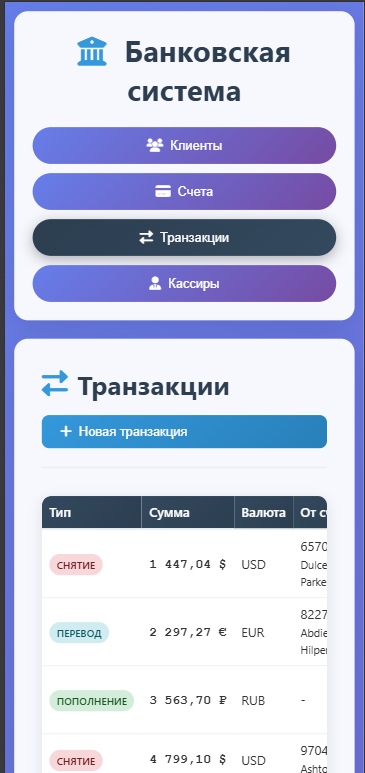


Рисунок 42 – адаптивная верстка для смартфонов

# ЗАКЛЮЧЕНИЕ

В рамках учебной практики было разработано приложение для обеспечения пользователя актуальной информацией из БД учета кассовых операций коммерческого банка.

Цель работы и соответствующие задачи для её реализации были выполнены в полном объеме.

Основные результаты работы:

* спроектирована база данных, состоящая из 4 коллекций (таблиц): Clients, Accounts, Transactions и Cashiers;
* реализованы CRUD-операции для каждой сущности с использованием Express.js и Mongoose;
* API задокументировано с помощью Swagger, что упрощает его тестирование и интеграцию;
* проведено тестирование функционала через Postman и MongoDB Compass;
* для удобства развертывания базы данных использован Docker;
* реализован простой и удобный UI.

Приложение успешно решает поставленную задачу — обеспечивает удобный и безопасный учет кассовых операций в банковской системе.

Использование NoSQL подхода (MongoDB) позволило гибко управлять данными, а REST API обеспечит удобное взаимодействие с клиентской частью.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Node.js Documentation // Node.js Official URL: https://nodejs.org/en/docs/ (дата обращения: 23.06.2025).
2. MongoDB Official Guide // MongoDB URL: https://www.mongodb.com/docs/ (дата обращения: 24.06.2025).
3. REST API Best Practices // RESTful API URL: https://restfulapi.net/ (дата обращения: 25.06.2025).
4. Mongoose Tutorial // GeeksforGeeks URL: https://www.geeksforgeeks.org/mongoose-tutorial/ (дата обращения: 23.06.2025).
5. Swagger Documentation // Swagger URL: https://swagger.io/docs/ (дата обращения: 26.06.2025).
6. NoSQL Databases Explained // MongoDB URL: https://www.mongodb.com/nosql-explained (дата обращения: 24.06.2025).
7. Express.js Framework Guide // Express.js URL: https://expressjs.com/ (дата обращения: 25.06.2025).
8. ER Diagrams in Database Design // Lucidchart URL: https://www.lucidchart.com/pages/er-diagrams (дата обращения: 23.06.2025).
9. Understanding RESTful APIs // MDN Web Docs URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview (дата обращения: 26.06.2025).
10. Node.js and MongoDB Integration // W3Schools URL: https://www.w3schools.com/nodejs/nodejs\_mongodb.asp (дата обращения: 24.06.2025).
11. Database Schema Design // IBM Developer URL: https://developer.ibm.com/articles/ (дата обращения: 25.06.2025).

# ПРИЛОЖЕНИЕ

const mongoose = require('mongoose');

const { faker } = require('@faker-js/faker');

require('dotenv').config();

const Client = require('./models/Client');

const Account = require('./models/Account');

const Transaction = require('./models/Transaction');

const Cashier = require('./models/Cashier');

async function **seed**() {

  await mongoose.connect(process.env.MONGO\_URI, {

    useNewUrlParser: true,

    useUnifiedTopology: true

  });

  console.log('Clearing old data...');

  await Client.deleteMany();

  await Account.deleteMany();

  await Transaction.deleteMany();

  await Cashier.deleteMany();

  console.log('Seeding new data...');

*// Кассиры*

  const cashiers = [];

  for (let i = 0; i < 5; i++) {

    cashiers.push(await Cashier.create({

      firstName: faker.person.firstName('female'),

      lastName: faker.person.lastName('female'),

      position: 'Cashier Operator',

      branch: faker.helpers.arrayElement(['Main Branch', 'Westside Branch', 'Downtown Branch'])

    }));

  }

*// Клиенты и их счета*

  const accounts = [];

  for (let i = 0; i < 20; i++) {

    const client = await Client.create({

      firstName: faker.person.firstName(),

      lastName: faker.person.lastName(),

      passportSeries: faker.string.numeric(4),

      passportNumber: faker.string.numeric(6),

      tin: faker.string.numeric(12),

      type: 'individual'

    });

    accounts.push(await Account.create({

      clientId: client.\_id,

      accountNumber: faker.finance.accountNumber(),

      currency: faker.helpers.arrayElement(['RUB', 'USD', 'EUR']),

      balance: faker.finance.amount({ min: 1000, max: 100000, dec: 2 }),

      status: 'active'

    }));

  }

*// Транзакции*

  for (let i = 0; i < 50; i++) {

    const type = faker.helpers.arrayElement(['deposit', 'withdrawal', 'transfer']);

    const fromAccount = faker.helpers.arrayElement(accounts);

    let toAccount = null;

    if (type === 'transfer') {

        do {

            toAccount = faker.helpers.arrayElement(accounts);

        } while (toAccount.\_id.equals(fromAccount.\_id));

    }

    await Transaction.create({

      type,

      amount: faker.finance.amount({ min: 100, max: 5000, dec: 2 }),

      currency: fromAccount.currency,

      fromAccount: (type === 'withdrawal' || type === 'transfer') ? fromAccount.\_id : null,

      toAccount: (type === 'deposit' || type === 'transfer') ? (type === 'deposit' ? fromAccount.\_id : toAccount.\_id) : null,

      cashierId: faker.helpers.arrayElement(cashiers).\_id,

      comment: faker.lorem.sentence()

    });

  }

  console.log('Test data seeded successfully!');

  await mongoose.disconnect();

}

seed().catch(*err* => {

  console.error(err);

  mongoose.disconnect();

});

Приложениe 1 – листинг программного кода файла seed.js ( скрипт генерирует начальные и тестовые данные в таблицы БД)

const mongoose = require('mongoose');

const CashierSchema = new mongoose.Schema({

  firstName: { type: String, required: true },

  lastName: { type: String, required: true },

  position: String,

  branch: String *// Отделение банка*

});

module.exports = mongoose.model('Cashier', CashierSchema);

Приложенииe 2 – листинг программного кода файла models/Сashier.js

(скрипт создает схему(модель) данных для документов(записей) в коллекции(таблице) Cashier

const mongoose = require('mongoose');

const AccountSchema = new mongoose.Schema({

  clientId: { type: mongoose.Schema.Types.ObjectId, ref: 'Client', required: true },

  accountNumber: { type: String, unique: true, required: true },

  currency: { type: String, enum: ['RUB', 'USD', 'EUR'], required: true },

  balance: { type: Number, default: 0 },

  openedAt: { type: Date, **default**: Date.now },

  status: { type: String, enum: ['active', 'closed', 'frozen'], default: 'active' }

});

module.exports = mongoose.model('Account', AccountSchema);

Приложенииe 3 – листинг программного кода файла models/Account.js

(скрипт создает схему(модель) данных для документов(записей) в коллекции(таблице) Account

const mongoose = require('mongoose');

const ClientSchema = new mongoose.Schema({

  firstName: { type: String, required: true },

  lastName: { type: String, required: true },

  middleName: String,

  passportSeries: { type: String, required: true },

  passportNumber: { type: String, required: true },

  tin: { type: String, unique: true }, *// ИНН*

  type: { type: String, enum: ['individual', 'legal'], required: true },

  createdAt: { type: Date, **default**: Date.now }

});

module.exports = mongoose.model('Client', ClientSchema);

Приложенииe 4 – листинг программного кода файла models/Client.js (скрипт создает схему(модель) данных для документов(записей) в коллекции(таблице) Client