МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Введение в нереляционные системы управления базами данных»

Тема: Сервис учёта семейных финансов

Студентка гр. 9303 _____ Зарезина Е.А.

Студент гр. 9303 _____ Павлов Д.Р

Студентка гр. 9303 _____ Хафаева Н.Л.

Преподаватель _____ Заславский М.М.

Санкт-Петербург
2022

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ (КУРСОВОЙ ПРОЕКТ)

Студенты:

Зарезина Е.А.

Хафаева Н.Л.

Павлов Д.Р

Группа 9303

Тема работы: Сервис учёта семейных финансов

Исходные данные:

Необходимо реализовать веб-сервис учёта финансов при помощи СУБД MongoDB

Содержание пояснительной записки:

«Содержание»

«Введение»

«Качественные требования к решению»

«Сценарии использования»

«Модель данных»

«Разработанное приложение»

«Заключение»

«Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 05.09.2022

Дата сдачи реферата: 16.12.2022

Дата защиты реферата: 16.12.2022

Студентка гр. 9303	_____	Зарезина Е.А.
Студент гр. 9303	_____	Павлов Д.Р
Студентка гр. 9303	_____	Хафаева Н.Л.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

Был разработан веб-сервис учёта финансов. При разработке использовалась СУБД MongoDB, для подключения СУБД использовалось Pymongo. Бэкенд приложения написан на языке программирования Python с использованием СУБД Flask.

SUMMARY

A web-based financial accounting service was developed. MongoDB DBMS was used during development, Pymongo was used to connect the DBMS. The backend of the application is written in the Python programming language using the Flask DBMS.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы — создать веб приложение для учёта семейных финансов.

Задачи:

1. Сформулировать основные сценарии использования и составить макет
2. Разработать модель данных
3. Разработать прототип «Хранение и представление»
4. Разработать прототип «Анализ»

1. КАЧЕСТВЕННЫЕ ТРЕБОВАНИЯ К РЕШЕНИЮ

Текущие требования к решению:

1. Реализовать страницу логина и добавить пользователей и информацию о финансах

2. Просмотр базы данных с помощью таблиц

3. Добавление новых пользователей и изменение информации о существующих

4. Фильтрация данных

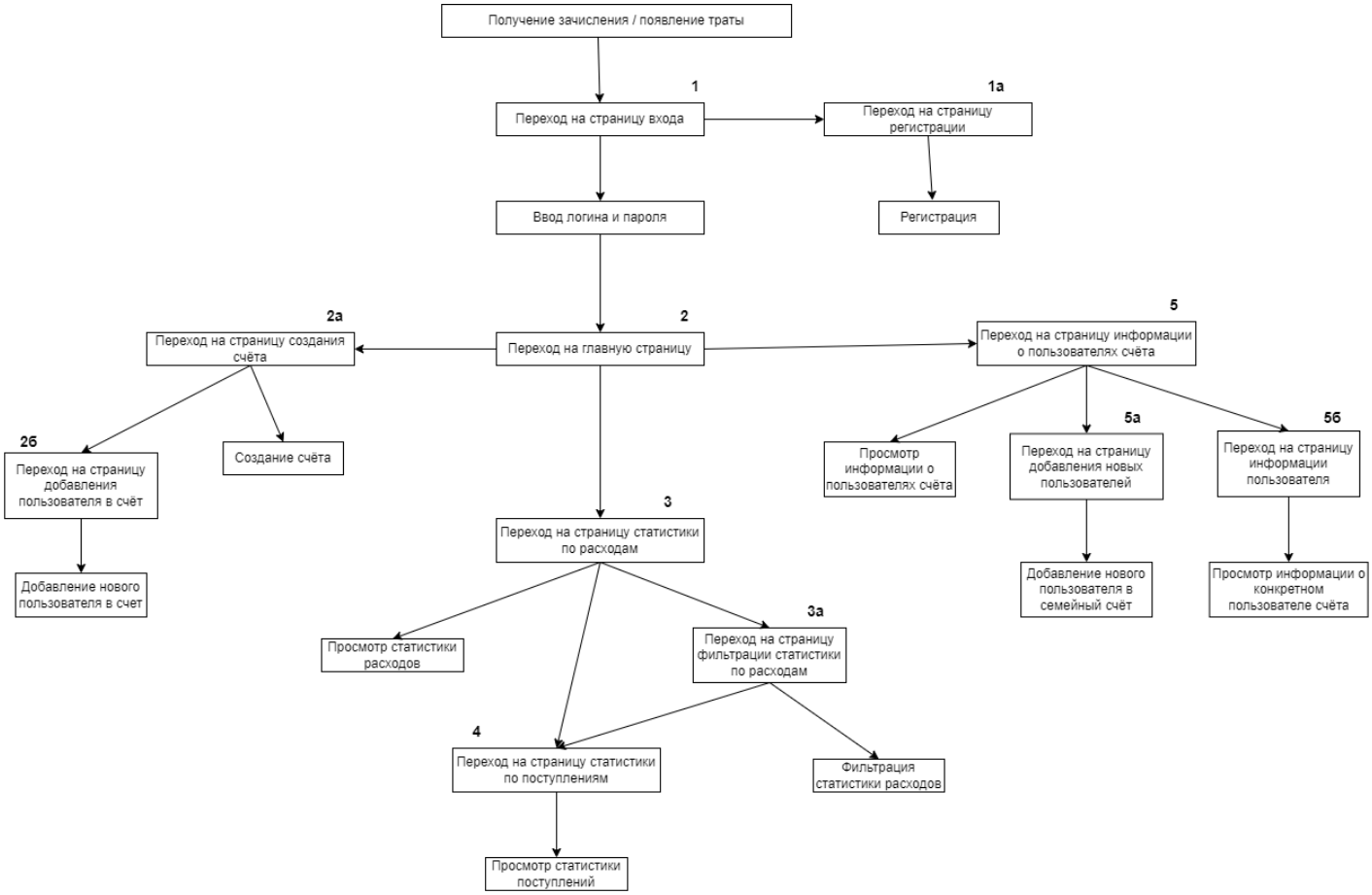
5. Приложение разворачивается через docker run --name mongodb -d -p 27017:27017 mongo

2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

2.1. Макет пользовательского интерфейса

Макет представлен в приложении А.

2.2. Сценарии использования



Сценарий использования - “Регистрация”:

Действующее лицо: Пользователь;

Предусловие: Открыта страница входа в приложение

Основной сценарий:

1. Перейти на страницу регистрации(1а), нажав на кнопку «Регистрация»
2. В открывшемся поле ввести имя пользователя, его дату рождения, а также логин и пароль
3. Нажать на кнопку «Зарегистрироваться»

Сценарий использования - “Вход в приложение”:

Действующее лицо: Пользователь;

Предусловие: Открыта страница входа в приложение

Основной сценарий:

1. Ввести логин пользователя и пароль
2. Нажать на кнопку «Вход»

Сценарий использования - “Создание счёта”:

Действующее лицо: Пользователь;

Предусловие: Открыта главная страница приложения;

Основной сценарий:

3. Перейти на страницу создания счёта(2а), нажав на кнопку «Создать счёт»
4. В открывшемся поле ввести название счёта, а также логин пользователя.
5. Нажать на кнопку «Создать»

Сценарий использования - “Добавление нового пользователя в счёт”:

Действующее лицо: Пользователь;

Предусловие: Открыта страница создания счёта (2а);

Основной сценарий:

1. Перейти на страницу добавления пользователя в счёт(2б), нажав на кнопку «Добавить пользователя»
2. В открывшемся поле ввести название счёта, а также логины пользователей.
3. Нажать на кнопку «Создать»

Сценарий использования - “Добавление нового пользователя в счёт”:

Действующее лицо: Пользователь;

Предусловие: Открыта страница создания счёта (2а);

Основной сценарий:

1. Перейти на страницу добавления пользователя в счёт(2б), нажав на кнопку «Добавить пользователя»
2. В открывшемся поле ввести название счёта, а также логины пользователей.
3. Нажать на кнопку «Создать»

Сценарий использования - “Просмотр статистики расходов”:

Действующее лицо: Пользователь;

Предусловие: Открыта главная страница приложения;

Основной сценарий:

1. Перейти на страницу статистики по расходам(3), нажав на кнопку «Все операции»
2. В открывшемся поле указать период статистики, тип операций, а также при необходимости дополнительную информацию.
3. В появившейся странице статистики по расходам (3а/3б в зависимости от параметров фильтрации) просмотреть статистику в отфильтрованной таблице.

Сценарий использования - “Просмотр информации о транзакции”:

Действующее лицо: Пользователь;

Предусловие: Открыта страница статистики по расходам (3);

Основной сценарий:

1. Перейти на страницу информации о транзакции(3с), нажав строку с информацией о транзакции в таблице.
2. В открывшемся поле просмотреть необходимую информацию.

Сценарий использования –
“Просмотр информации о пользователе счёта”:

Действующее лицо: Пользователь;

Предусловие: Открыта главная страница приложения

Основной сценарий:

1. Перейти на страницу информации о пользователях счёта (5), нажав на кнопку «Общий счёт»
2. В открывшемся поле выбрать пользователя, нажав на его имя, и перейти на страницу информации о конкретном пользователе (5а)
3. Просмотреть информацию

Сценарий использования - “Добавление нового пользователя в семейный счёт”:

Действующее лицо: Пользователь;

Предусловие: Открыта страница информации о пользователях счёта (5)

Основной сценарий:

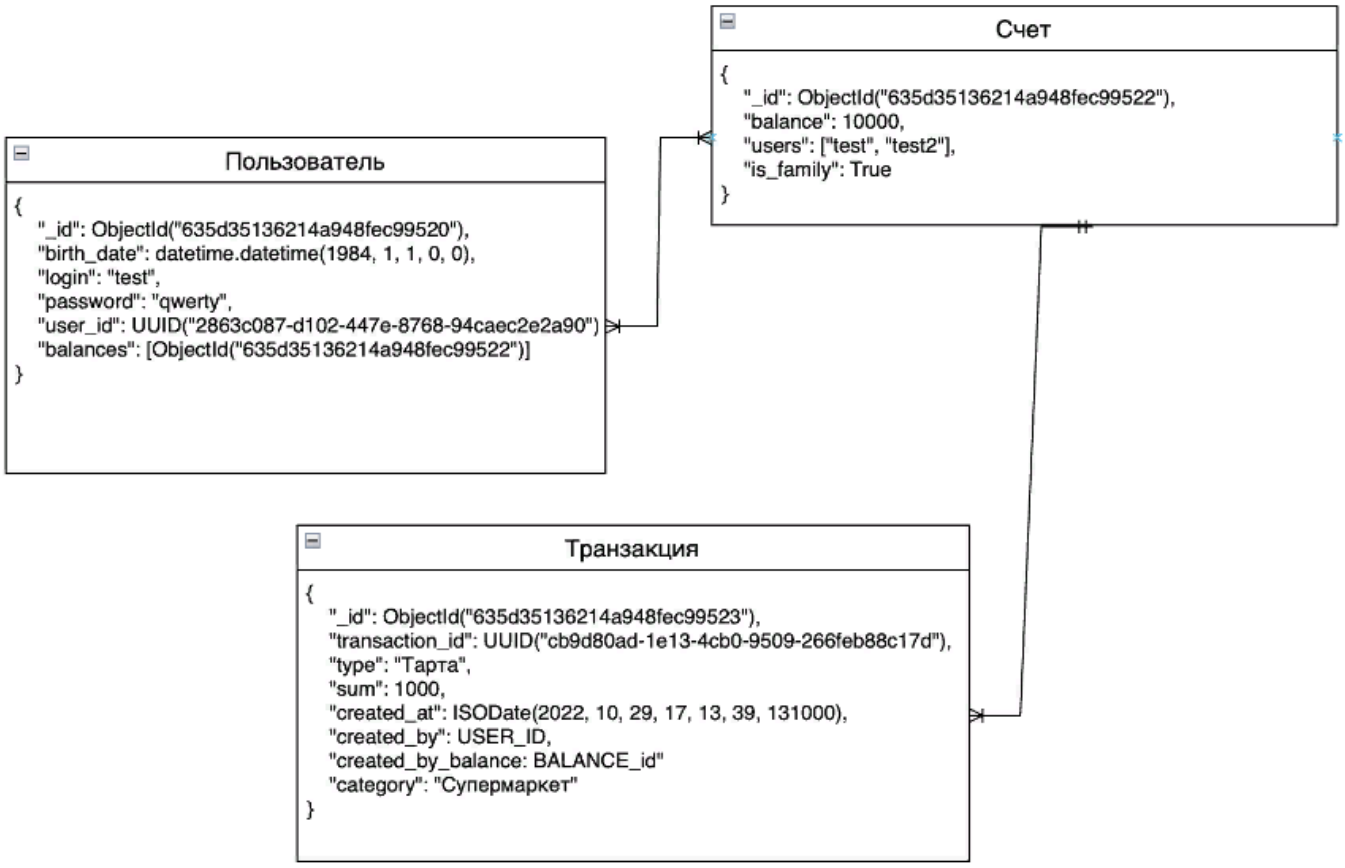
1. Перейти на страницу добавления нового пользователя в семейный счёт(5б), нажав на кнопку «Добавить пользователя»
2. В открывшемся поле ввести логин пользователя
3. Нажать на кнопку «Добавить»

2.3. Преобладающие операции.

Преобладать будут операции чтения

3. МОДЕЛЬ ДАННЫХ

3.1. Нереляционная модель данных



Список сущностей модели:

1. Пользователь (202 байта) :
   - _id: ObjectId - идентификатор (12 байт)
   - name: string - имя (50 байт)
   - surname: string - фамилия (50 байт)
   - login: string - логин (50 байт)
   - password: string - хеш пароля (32 байт)
   - birth_date: datetime - дата рождения (8 байт)
   - balances: ObjectId[] - счета пользователя (12 * B байт)
2. Транзакция (126 байт) :
   - _id: ObjectId - идентификатор (12 байт)

- type: string - тип транзакции (7 байт)
- sum: int - сумма транзакции (8 байт)
- created_at: datetime - дата создания транзакции (8 байт)
- created_by: string - логин пользователя, который совершил транзакцию (50 байт)
- created_by_balance_id: ObjectId - идентификатор баланса (12 байт)
- category: string - категория транзакции (11 байт)
- name: string - название операции (имя получателя) (30 байт)

3. Счет (28 + 12 * U байт) :
   - _id: ObjectId - идентификатор (12 байт)
   - users: ObjectId[] - группа пользователей семейного финанса (12 * U байт)
   - balance: Decimal128 - баланс счета (16 байт)
   - is_family: boolean - является ли счет семейным (8 байт)
4. Коллекции
   - Пользователь (User)
   - Счет (Balance)
   - Операция (Transaction)

Оценка удельного объёма информации:

Пусть T – количество транзакций, U – среднее количество пользователей, B - количество счетов

С определенной долей условности можно считать, что T = 200, U = 50, B = 75.

Документ в коллекции User:

- 190+12B байт («чистый»)
- 202+12B байт (фактический)

Документ в коллекции Balance:

- 24+12U байт («чистый»)
- 36+12U байт (фактический)

Документ в коллекции Transaction:

- 126 байт («чистый»)
- 138 байт (фактический)

«Чистый» объём данных равен (190U + 12B) + (24+12U) + 126T = 11024+126T

Фактический объём данных равен (202U + 12B) + (36+12U) + 138T = 11636+138T

Избыточность модели данных равна (11636+138T)/(11024+126T)

При увеличении количества пользователей, счетов, количества транзакций рост модели будет линейным

Запросы к модели с помощью которых реализуются сценарии использования

- Запрос (Регистрация пользователя):

```
def user_register(self, *args, **kwargs):
    return self.user.insert_one(kwargs.get("user"))
```

```
use_case.user_register(user={
        "name": "Test",
        "surname": "Testoviy",
        "login": "test_user_3",
        "password": hashlib.md5(str("test_password_3").encode()).hexdigest(),
        "birth_date": datetime.datetime(1985, 1, 1)
    })
```

Пользователь сохранился в коллекцию

Кол-во запросов: 1

Кол-во коллекций: 1

- Запрос (Получение юзера)

```
def user_retrieve(self, pk, *args, **kwargs):
```

```
    return self.user.find_one({"_id": ObjectId(pk)})


use_case.user_retrieve("635e603bb0f25b62f95c4276")
```

Ответ:

```
{
    '_id': ObjectId('635e603bb0f25b62f95c4276'),
    'birth_date': datetime.datetime(1984, 1, 1, 0, 0),
    'login': 'test_user',
    'name': 'Dmitry',
    'password': '16ec1ebb01fe02ded9b7d5447d3dfc65',
    'surname': 'Pavlov'
}
```

Кол-во запросов: 1

Кол-во коллекций: 1

- Запрос (Создание семейного баланса)

```
def create_family_balance(self, user_id, *args, **kwargs):
    return self.family.insert_one({
        "balance": Decimal128(str(0)),
        "users": [ObjectId(user_id)],
        "is_family": True
    })


use_case.create_family_balance("635e603bb0f25b62f95c4276")
```

Создался семейный баланс

Кол-во запросов: 1

Кол-во коллекций: 1

- Запрос (Добавить пользователя в семейный баланс):

```
def add_user_to_family_balance(self, user_login, family_id):
    user_id = self.user.find_one({"login": user_login}).get("_id")
    return self.family.update_one({"_id": ObjectId(family_id)},
                                  {"$push": {"users": ObjectId(user_id)}})


use_case.add_user_to_family_balance("test_user_1", "635e67d3fb176336eee4262a")
```

Пользователь с конкретным логином добавился в семейный счет

Кол-во запросов: 2

Кол-во коллекций: 2

- Запрос (Получить семейный счет для конкретного юзера)

```
def get_family_balance_for_user(self, user_id, *args, **kwargs):
    return self.family.find_one({
        "users": ObjectId(user_id),
        "is_family": True
    })
use_case.get_family_balance_for_user("635e603bb0f25b62f95c4276")
```

Ответ:

```
{
  '_id': ObjectId('635e67d3fb176336eee4262a'),
  'balance': Decimal128('0'),
  'users': [
    ObjectId('635e603bb0f25b62f95c4276'),
    ObjectId('635e79baa7b68a00821b18ea')
  ]
}
```

Кол-во запросов: 1

Кол-во коллекций: 1

- Запрос (Создание транзакции):

```
def create_transaction(self, *args, **kwargs):
    return self.transaction.insert_one(kwargs.get('transaction'))

use_case.create_transaction(transaction={
            "type": "Поплнение",
            "sum": Decimal128(str(10000)),
            "created_at": datetime.datetime.now(),
            "created_by": ObjectId("635e79baa7b68a00821b18ea"),
            "created_by_balance": ObjectId("945e79baa7b68a00823b18ea") # id balance
            "name": "Семейный счет",
            "category": "Переводы"
        })
```

Кол-во запросов: 1

Кол-во коллекций: 1

- Запрос (Получение списка транзакций):

```
def list_transactions(self, user_id,
                      date_from=None,
                      date_to=None,
                      user_filter=None,
                      category_filter=None,
                      search_filter=None):
    family = self.get_family_balance_for_user(user_id)
    query = dict()

    if user_filter:
        query["created_by"] = ObjectId(user_filter)
    else:
        query["created_by"] = {"$in": family.get("users")}
```

```
        query["created_at"] = dict()
        if date_from:
            query["created_at"]["$gte"] = date_from

        if date_to:
            query["created_at"]["$lte"] = date_to

        if category_filter:
            query["category"] = category_filter

        if search_filter:
            query["name"] = {"$regex": search_filter}

        transactions = self.transaction.find(query)
        return [transaction for transaction in transactions]
    use_case.list_transactions("635e603bb0f25b62f95c4276",
date_from=datetime.datetime(2022, 10, 30))
```

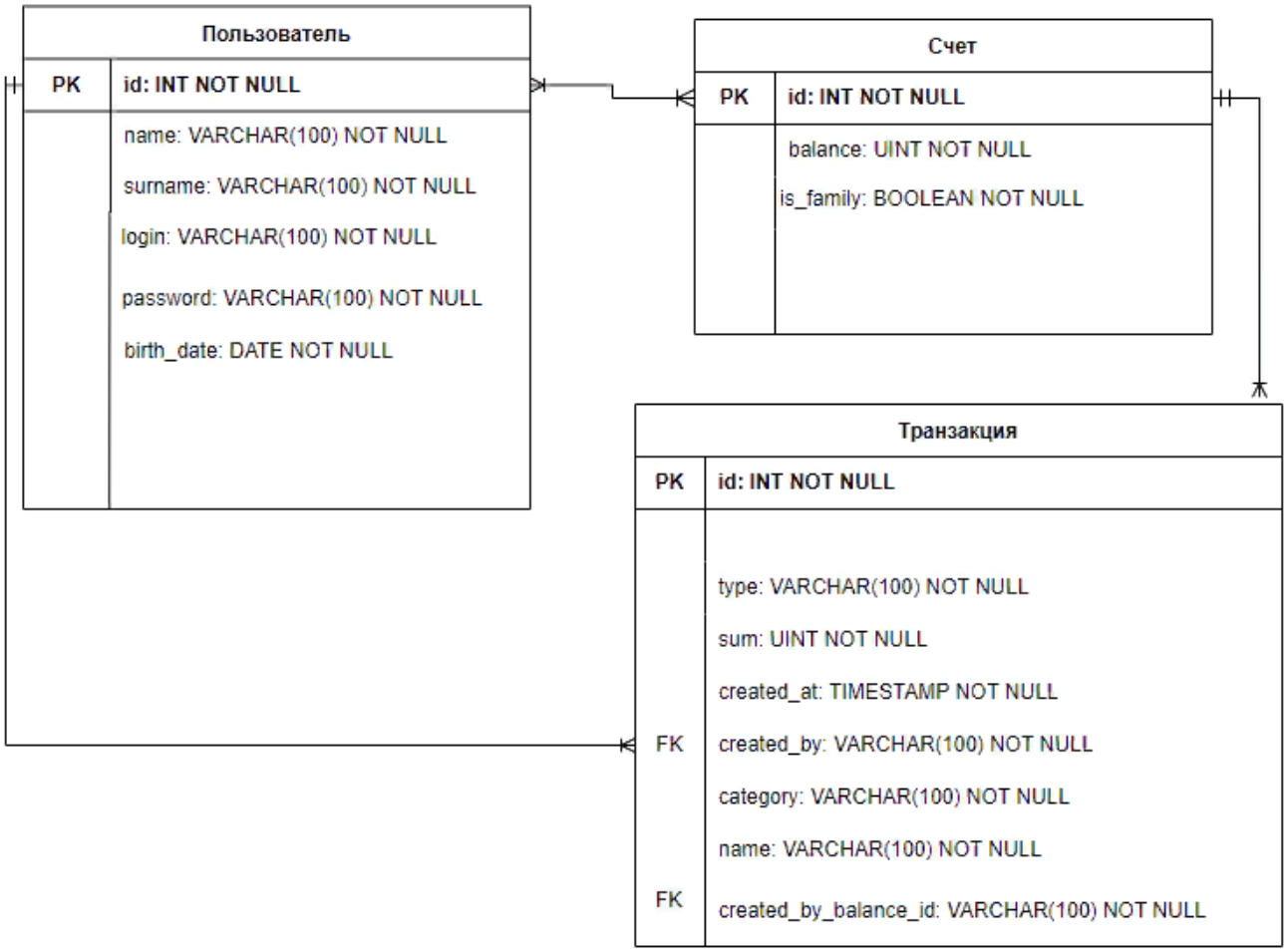
Ответ (в данном случае получаем транзакции сделанные с 30ого числа этого месяца):

```
    [
      {
        '_id': ObjectId('635e75a41f63a3c08c55663e'),
        'category': 'Переводы',
        'created_at': datetime.datetime(2022, 10, 30, 16, 1, 24, 911000),
        'created_by': ObjectId('635e603bb0f25b62f95c4276'),
        "created_by_balance": ObjectId("945e79baa7b68a00823b18ea"),
        'name': 'Семейный счет',
        'sum': Decimal128('1000'),
        'type': 'Пополнение'},
```

```
  {
   '_id': ObjectId('635e818be6055af3fa828277'),
   'category': 'Переводы',
   'created_at': datetime.datetime(2022, 10, 30, 16, 52, 11, 685000),
   'created_by': ObjectId('635e79baa7b68a00821b18ea'),
   "created_by_balance": ObjectId("945e79baa7b68a00823b18ea"),
   'name': 'Семейный счет',
   'sum': Decimal128('10000'),
   'type': 'Пополнение'},
  {
   '_id': ObjectId('635e81a034c857e7f6287e4b'),
   'category': 'Переводы',
   'created_at': datetime.datetime(2022, 10, 30, 16, 52, 32, 201000),
   'created_by': ObjectId('635e79baa7b68a00821b18ea'),
   "created_by_balance": ObjectId("945e79baa7b68a00823b18ea"),
   'name': 'Семейный счет',
   'sum': Decimal128('10000'),
   'type': 'Пополнение'},
  {
   '_id': ObjectId('635e98c9de1c4a03abdc389a'),
   'category': 'Переводы',
   'created_at': datetime.datetime(2022, 10, 30, 18, 31, 21, 19000),
   'created_by': ObjectId('635e79baa7b68a00821b18ea'),
   "created_by_balance": ObjectId("945e79baa7b68a00823b18ea"),
   'name': 'Семейный счет',
   'sum': Decimal128('10000'),
   'type': 'Пополнение'
  }
]
Кол-во запросов: 2
```

Кол-во коллекций: 2

3.2. Аналог модели данных для SQL СУБД



Список сущностей модели:

1. Пользователь (288 байт) :
    - id: INT - идентификатор (8 байт)
    - login: VARCHAR(100) - логин (100 байт)
    - password: VARCHAR(100) - пароль (100 байт)
    - birth_date: DATE - дата рождения в формате YYYY-MM-DD (8 байт)
2. Транзакция (336 байт) :
    - id: INT - идентификатор (8 байт)

- type: VARCHAR(100) - тип транзакции (100 байт)
- sum: UINT - сумма транзакции (4 байт)
- created_at: TIMESTAMP - дата создания транзакции в формате YYYY-MM-DDThh:mm:ssZ (8 байт)
- category: VARCHAR(100) - категория транзакции (100 байт)
- name: VARCHAR(100) - название операции (имя получателя) (100 байт)
- created_by: INT - идентификатор пользователя, который совершил транзакцию (8 байт)
- created_by_balance_id: INT - идентификатор счета пользователя, который совершил транзакцию (8 байт)

3. Семейный счет (28 байт) :
    - id: INT - идентификатор (8 байт)
    - balance: UINT - баланс счета (16 байт)
    - is_family: BOLLEAN - переменная, определяющая является ли счет семейным (4 байт)
    - Счета пользователей (16 байт)
    - user_id: INT - идентификатор пользователя (8 байт)
    - bank_account_id: INT - идентификатор счета (8 байт)
4. Коллекции
    - Пользователь (user_account)
    - Счет (bank_account)
    - Операция (transaction)
    - Счета пользователей (user_accounts)

Оценка удельного объёма информации:

Пусть T – количество транзакций, U – среднее количество пользователей, B - количество счетов

С определенной долей условности можно считать, что T = 200, U = 50, B = 75.

Строка в таблице user_account:

- 288 байт ("чистый")
- 288 байт (фактический)

Строка в таблице bank_account:

- 28 байт ("чистый")
- 28 байт (фактический)

Строка в таблице transaction:

- 336 байт ("чистый")
- 360 байт (фактический)

Строка в таблице user_accounts:

- 16 байт ("чистый")
- 28 байт (фактический)

«Чистый» объём данных равен 180U + 28 + 336T + 16 = 14444 + 336T

Фактический объём данных равен 288U + 28 + 360T + 28 = 14456 + 360T

Избыточность модели данных равна (14456 + 360T)/(14444 + 336T)

При увеличении количества пользователей, счетов, количества транзакций рост модели будет линейным

3.3. Сравнение моделей

Пусть T = 200, U = 50, B = 75

NoSQL:

- Чистый: 36224 байт
- Фактический: 39363 байт
- Избыточность: 1.087

SQL:

- Чистый: 81644 байт

- Фактический: 86456 байт
- Избыточность: 1.059

3.4. Вывод

Хранение данных при использовании нереляционной модели БД значительно сокращает количество занимаемого места, нежели использование реляционной БД.

Наименьшее количество запросов: SQL

Наименьшая избыточность: SQL

Наименьшее количество таблиц: NoSQL

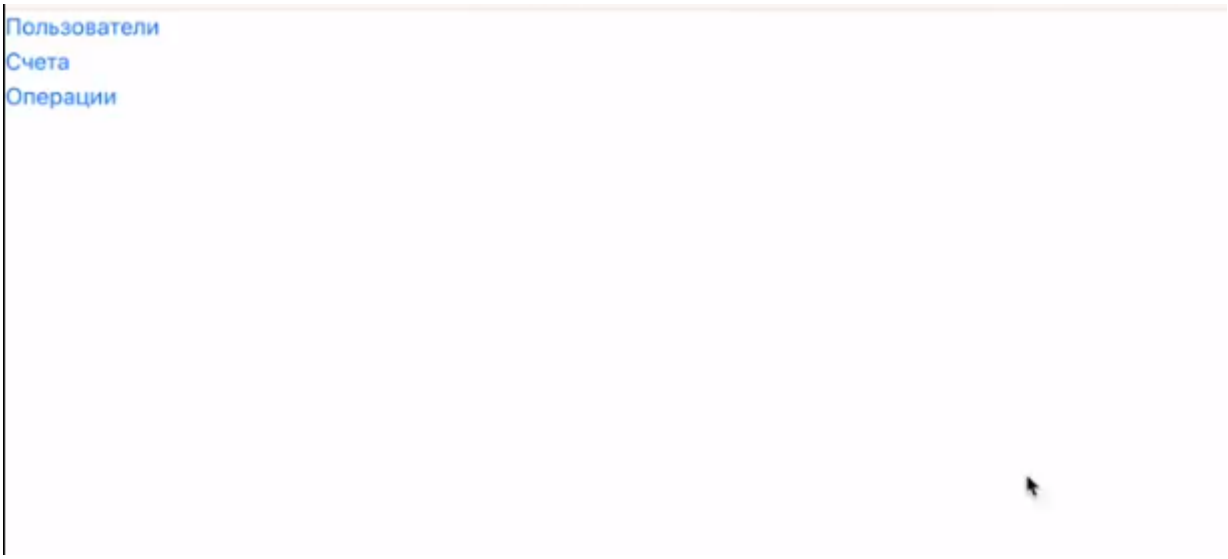
В данном конкретном случае наиболее выгодным решением является использование SQL СУБД.

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

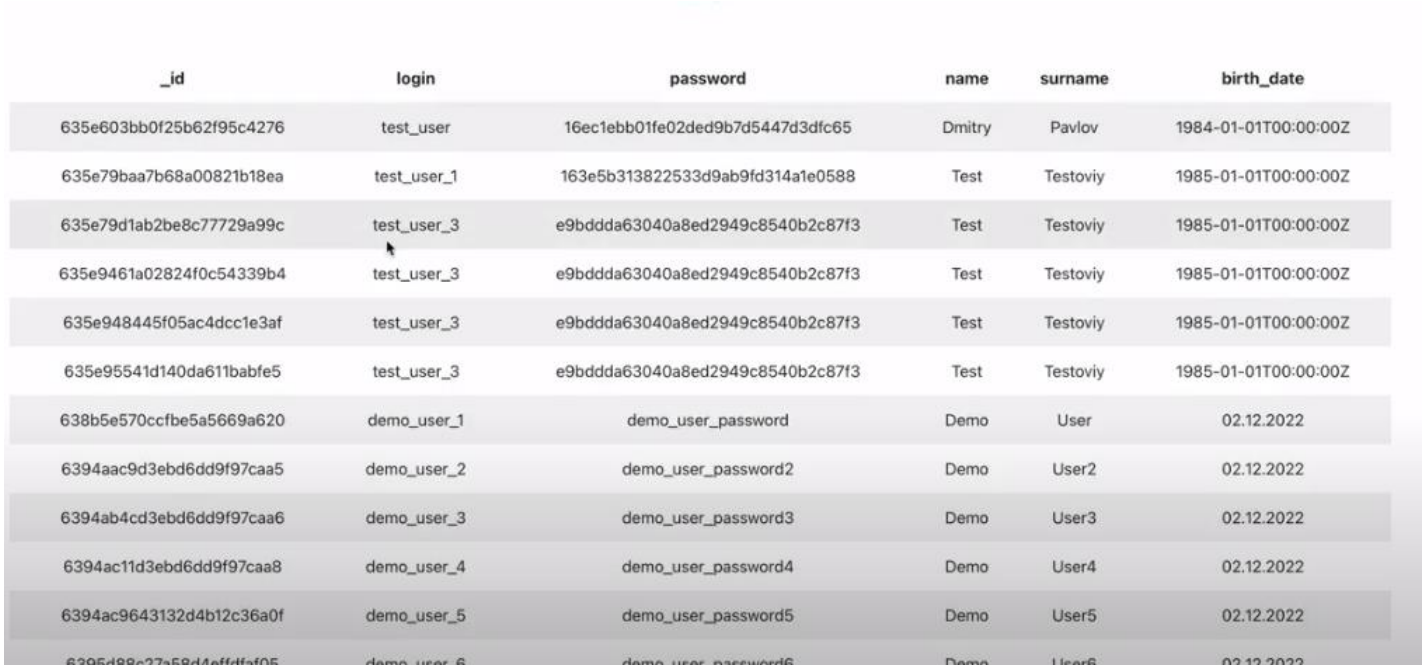
4.1. Схема экранов приложения

Страница регистрации:

Вход в приложение:

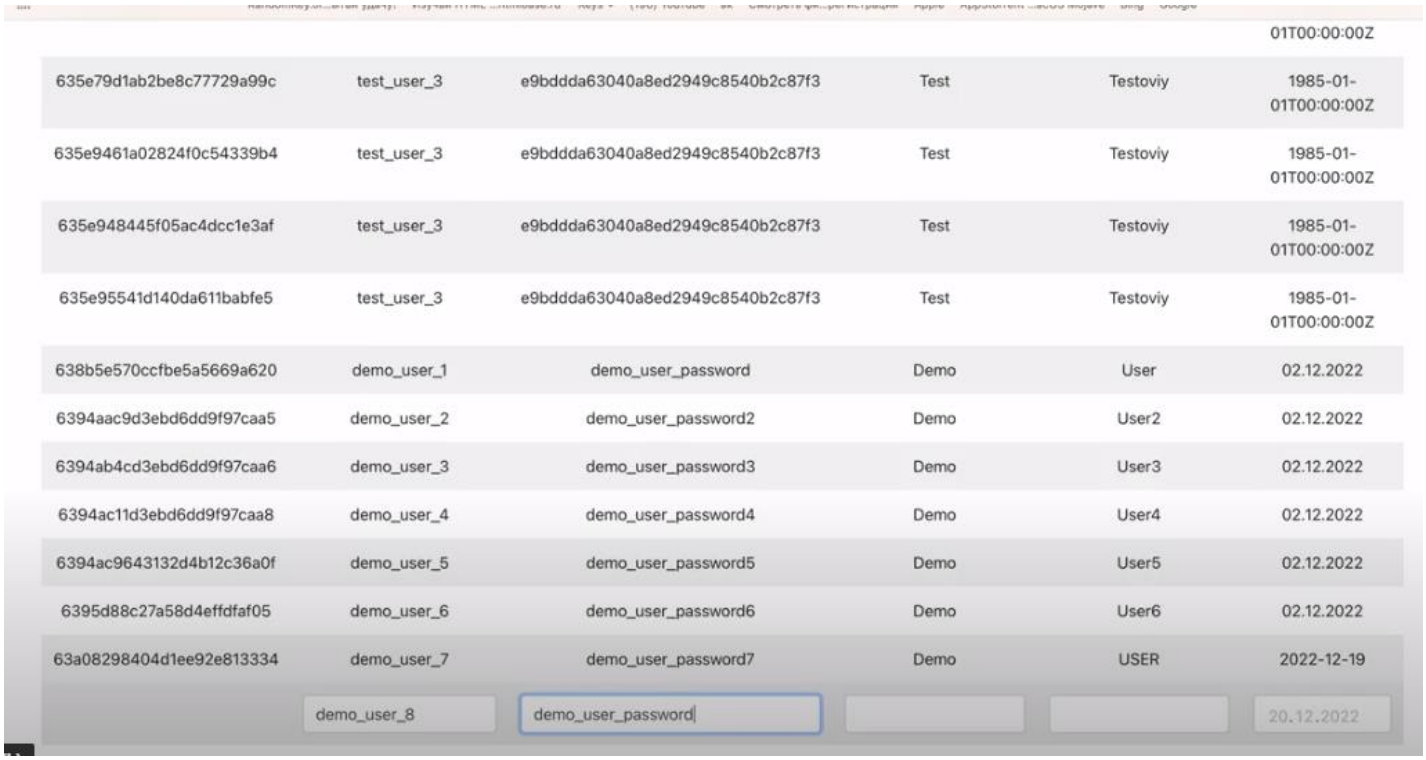


Информация о пользователях:



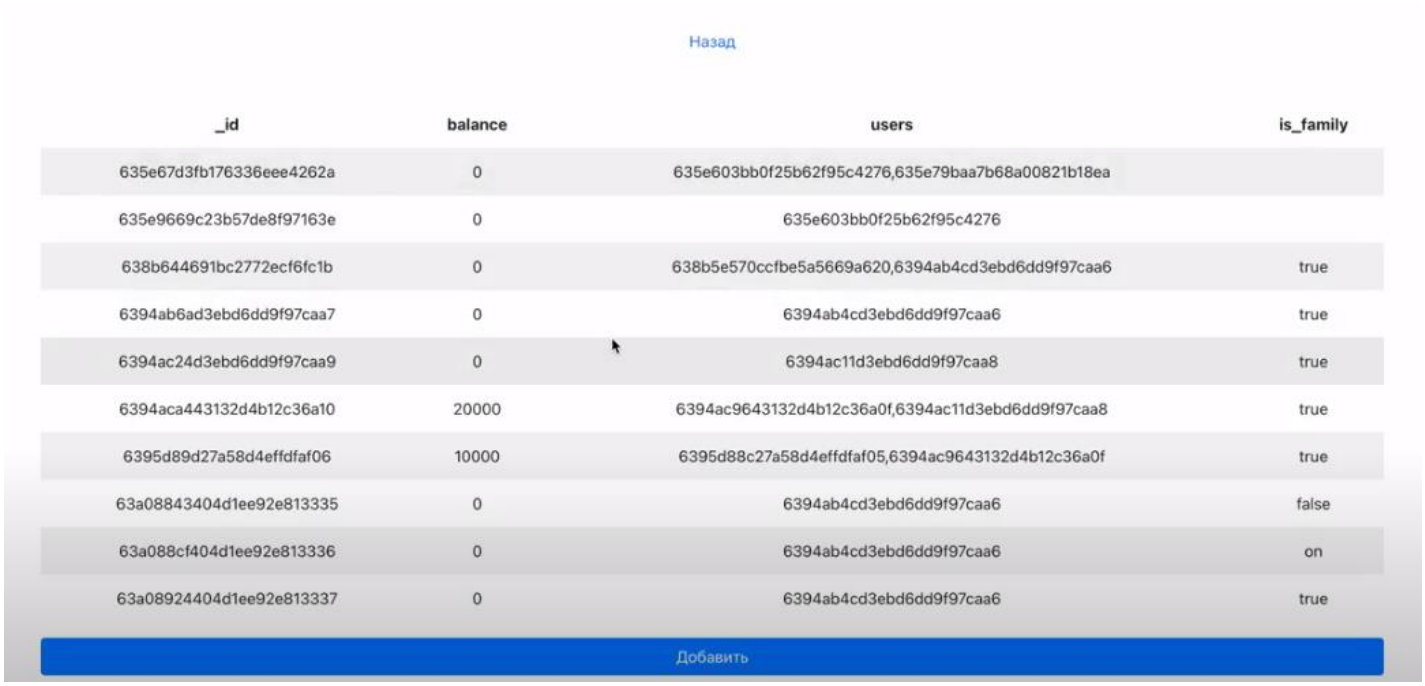
_id	login	password	name	surname	birth_date
635e603bb0f25b62f95c4276	test_user	16ec1ebb01fe02ded9b7d5447d3dfc65	Dmitry	Pavlov	1984-01-01T00:00:00Z
635e79baa7b68a00821b18ea	test_user_1	163e5b313822533d9ab9fd314a1e0588	Test	Testoviy	1985-01-01T00:00:00Z
635e79d1ab2be8c77729a99c	test_user_3	e9bddda63040a8ed2949c8540b2c87f3	Test	Testoviy	1985-01-01T00:00:00Z
635e9461a02824f0c54339b4	test_user_3	e9bddda63040a8ed2949c8540b2c87f3	Test	Testoviy	1985-01-01T00:00:00Z
635e948445f05ac4dcc1e3af	test_user_3	e9bddda63040a8ed2949c8540b2c87f3	Test	Testoviy	1985-01-01T00:00:00Z
635e95541d140da611babfe5	test_user_3	e9bddda63040a8ed2949c8540b2c87f3	Test	Testoviy	1985-01-01T00:00:00Z
638b5e570ccfbe5a5669a620	demo_user_1	demo_user_password	Demo	User	02.12.2022
6394aac9d3ebd6dd9f97caa5	demo_user_2	demo_user_password2	Demo	User2	02.12.2022
6394ab4cd3ebd6dd9f97caa6	demo_user_3	demo_user_password3	Demo	User3	02.12.2022
6394ac11d3ebd6dd9f97caa8	demo_user_4	demo_user_password4	Demo	User4	02.12.2022
6394ac9643132d4b12c36a0f	demo_user_5	demo_user_password5	Demo	User5	02.12.2022

Добавление нового пользователя:



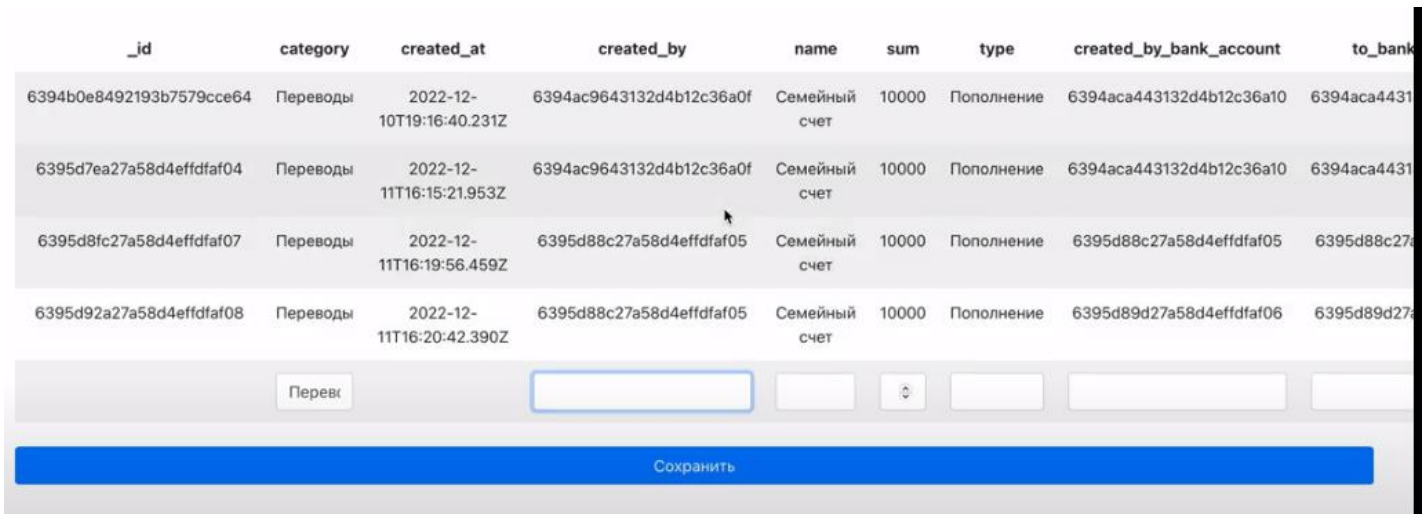
					01T00:00:00Z
635e79d1ab2be8c77729a99c	test_user_3	e9bddda63040a8ed2949c8540b2c87f3	Test	Testoviy	1985-01-01T00:00:00Z
635e9461a02824f0c54339b4	test_user_3	e9bddda63040a8ed2949c8540b2c87f3	Test	Testoviy	1985-01-01T00:00:00Z
635e948445f05ac4dcc1e3af	test_user_3	e9bddda63040a8ed2949c8540b2c87f3	Test	Testoviy	1985-01-01T00:00:00Z
635e95541d140da611babfe5	test_user_3	e9bddda63040a8ed2949c8540b2c87f3	Test	Testoviy	1985-01-01T00:00:00Z
638b5e570ccfbe5a5669a620	demo_user_1	demo_user_password	Demo	User	02.12.2022
6394aac9d3ebd6dd9f97caa5	demo_user_2	demo_user_password2	Demo	User2	02.12.2022
6394ab4cd3ebd6dd9f97caa6	demo_user_3	demo_user_password3	Demo	User3	02.12.2022
6394ac11d3ebd6dd9f97caa8	demo_user_4	demo_user_password4	Demo	User4	02.12.2022
6394ac9643132d4b12c36a0f	demo_user_5	demo_user_password5	Demo	User5	02.12.2022
6395d88c27a58d4effdfaf05	demo_user_6	demo_user_password6	Demo	User6	02.12.2022
63a08298404d1ee92e813334	demo_user_7	demo_user_password7	Demo	USER	2022-12-19
	demo_user_8	demo_user_password			20.12.2022

Информация о счетах:



Информация об операциях:

Добавление новой операции:



4.2. Использованные технологии

Для хранения данных была использована MongoDB, для создания фреймворка на бэкенде использовался Flask, для подключения базы данных Pymongo. Язык программирования Python.

ВЫВОДЫ

Достигнутые результаты:

В ходе работы было разработано веб-приложение для учёта семейных финансов, в котором возможен просмотр и добавление информации о пользователях, просмотр и добавление информации о счетах и транзакциях. Для хранения данных и управления ими использовалась СУБД MongoDB.

Будущее развитие решения:

Реализация всех сценариев использования и совершенствование интерфейса.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация MongoDB: https://www.mongodb.com/docs/
2. Документация PyMongo: https://www.mongodb.com/docs/drivers/pymongo/

ПРИЛОЖЕНИЕ А
МАКЕТ ДАННЫХ

Макет: