УДК 004

ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ NEST.JS ДЛЯ АВТОМАТИЗАЦИИ ПРОЦЕССА УЧЁТА ФИНАНСОВЫХ ОПЕРАЦИЙ

Кенн П.Ю., студент группы ИТб-182, Гладкова М.Ю., студентка группы ИТб-182, Гладков И.А., студент группы ИТб-182, IV курс
Кузбасский государственный технический университет имени Т. Ф. Горбачева
г. Кемерово

В современном мире, когда на смену бумажным деньгам, приходят банковские карты, люди сталкиваются с проблемой учета своих доходов и расходов. Если раньше было видно и можно было посчитать, на что именно тратятся деньги, то сейчас из-за того, что большинство расчетов происходит в электронном виде, человек теряет счет деньгам. Существуют разные способы учета финансовых операций:

- ручной – учет финансов в бумажном виде, анализ осуществляется без использования программного обеспечения. Данный способ имеет следующие недостатки – большие временные затраты, большая вероятность ошибок при расчетах.
- полу автоматизированный – учет финансов с частичным использованием специализированного программного обеспечения, систем онлайн-банкинга. Зачастую такие системы производят расчет неверно и совершают логические ошибки, например, учитывают, как расходы движения средств между счетами, что приводит к ситуации, когда расходы, рассчитанные за период в системе, в несколько раз превышают реальные доходы, что невозможно. Существенным недостатком является невозможность объединять информацию о счетах в разных банках в одной системе.
- полностью автоматизированный – информационная система обладает функционалом, достаточным для динамического анализа текущего состояния счетов разных банках. На данный момент существует несколько реализаций подобных систем, но как правило их функционал очень урезан. Главный недостаток данных систем – финансирование за счет торговли личными данными пользователей.

В данной статье рассмотрено приложение для учета расходов и доходов человека, что относится к полностью автоматизированным способам контроля бюджета.

Рассмотрим функционал, требуемый для данного приложения:

- контроль доходов и расходов;
- контроль финансовых сбережений;

- обсуждение своих финансовых сбережений с другими пользователями системы (чат или форум).

При реализации подобного функционала могут возникнуть следующие проблемы:
1) требуется быстрая обработка данных;
2) требуется быстрый обмен информацией между компонентами системы;
3) требуется надежное хранение, поддержка конфиденциальности, защита от утечки.

Для разработки серверной части приложения был выбран фреймворк Nest.js по следующим причинам:

- поддерживает модульную организацию проектов, которые строятся в соответствии с принадлежностью каждой логической части проекта (модуля) к определенной предметной области (может быть использовано при реализации трех отдельных модулей описываемой системы);
- фреймворк позволяет создавать расширяемые программные решения, где нет сильной связи между компонентами;
- Nest поддерживает два типа связи — TCP и Redis pub/sub, что позволяет производить обмен информации между модулями не через http;
- асинхронный метод, который используется в Node.js, максимально использует однопоточную обработку, что сокращает время отклика в несколько раз, а это необходимо при реализации анализа доходов и расходов пользователя, функционала чата;
- полная поддержка JSON, это особенно необходимо для RESTful API, так как при накоплении большого объема данных с сообщениями пользователей, они будут сохраняться в NoSQL базу данных.

Для обмена информацией между компонентами системы был выбран формат JSON, так как:

- JSON — компактный формат, с ним большие объемы данных быстро обмениваются между браузером и веб-сервером;
- с JSON-файлами можно работать не только методами JavaScript, почти у всех языков есть инструменты для чтения и генерации данных JSON;
- хранение и экспорт данных в JSON поддерживают современные реляционные базы данных, такие как PostgreSQL и MySQL.

Рассмотрим реализацию серверной части приложения с помощью описанных технологий на примере процесса контроля доходов и расходов. Диаграмма деятельности данного процесса представлена на рисунке 1.

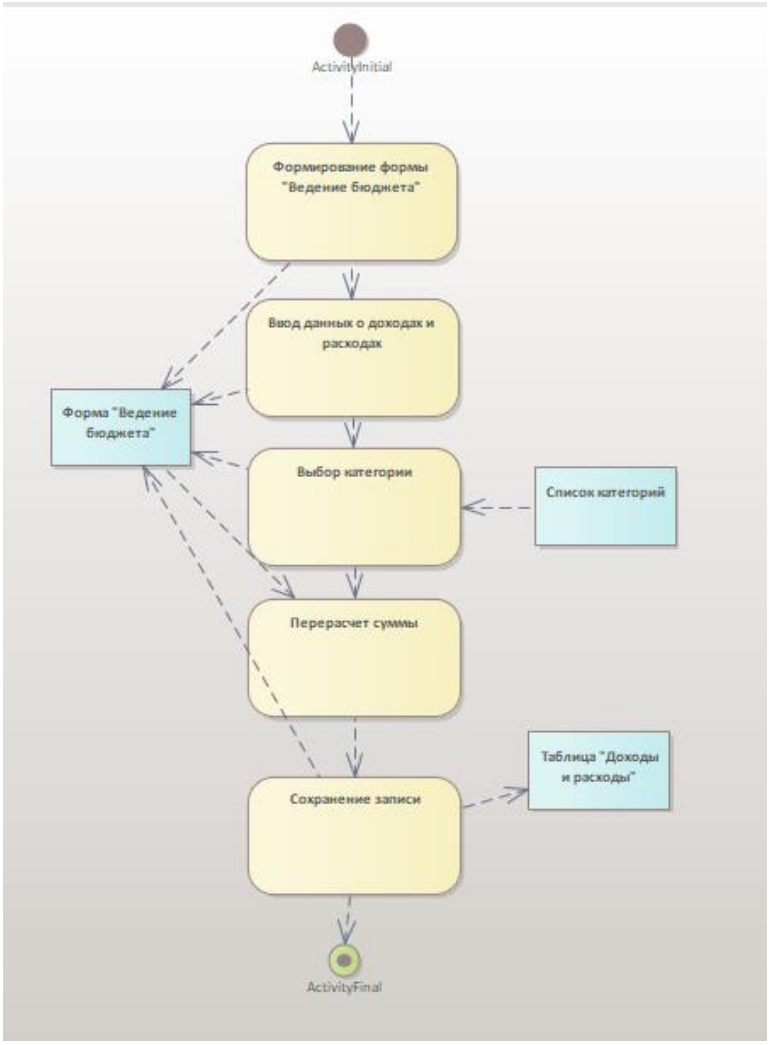

Рисунок 1 – Диаграмма деятельности процесса контроля расходов

Для отображения всех расходов необходимо реализовать формирование диаграммы. Для этого нужно сделать выборку из базы данных. У запроса должны быть параметры: период за который пользователь хочет получить выборку. Реализация данного алгоритма представлена на рисунке 3.

```
public async gant(id: number, time: {start: Date, end: Date}): Promise<any> {
    const result = new RegistrationRespModel();
    const gant = await this.category.query( query: `SELECT *,
        (SELECT SUM(value)
        FROM events
        WHERE events."categoryId" = categories.id
        AND (events.date
        BETWEEN (${time.start} AS quoted)
        AND (${time.end}) AS quoted) )
        AS value
        FROM categories
        WHERE categories."userId" = ${id}`);
    result.successStatus = true;
    result.message = 'Success';
    result.data = gant;
    return result;
}
```

Рисунок 3 – Листинг кода

Пример JSON ответа данного метода представлен на рисунке 4.


```
{
    "successStatus": true,
    "message": "Success",
    "data": [
        {
            "id": 8,
            "name": "Прочее",
            "limit": null,
            "color": "#faf3dd",
            "type": "0",
            "userId": 9,
            "value": "29433"
        },
        {
            "id": 2,
            "name": "Аптека",
            "limit": null,
            "color": "#ffa69e",
            "type": "0",
            "userId": 9,
            "value": "2293"
        },
        {
            "id": 10,
            "name": "Развлечение",
            "limit": null,
            "color": "#b8f2e6",
            "type": "0",
            "userId": 9,
            "value": "15674"
        },
```


Рисунок 4 – JSON ответ

Далее рассмотрим метод, который производит добавление новой операции в систему. Подразумевается, что при совершении новой операции, меняется текущий баланс пользователя, его средств в свободном доступе. Потенциальные проблемы, которые могут возникнуть при использование данного метода – это ошибки при выполнения какой-либо SQL операции, в таком случае нам нужно обернуть все SQL операции в транзакцию. Транзакция дает возможность откатить все SQL операции в случае ошибки одной из них. Реализация данного метода представлена на рисунке 5.

```
public async create(userId: number, event: EventRegType): Promise<any> {
    const result = new RegistrationRespModel();
    const newEvent = new EventClass();
    newEvent.category = event.categoryId;
    newEvent.mark = event.markId;
    newEvent.comment = event.comment;
    newEvent.value = event.value;
    newEvent.date = event.date ? event.date : new Date();
    // @ts-ignore
    const queryRunner = this.event.createQueryRunner();
    await queryRunner.startTransaction('REPEATABLE READ');
    // @ts-ignore
    await this.event.insert(newEvent);
    const user = await this.user.findOne(userId)
    const ok = await this.user.update(userId, partialEntity: {account: user.account + event.value});
    if (ok) {
        await queryRunner.commitTransaction();
    } else {
        await queryRunner.rollbackTransaction();
    }
    result.successStatus = true;
    result.message = 'Success';
    return result;
}
```

Рисунок 5 – Листинг кода

Интерфейс полученного приложения представлен на рисунке 6.

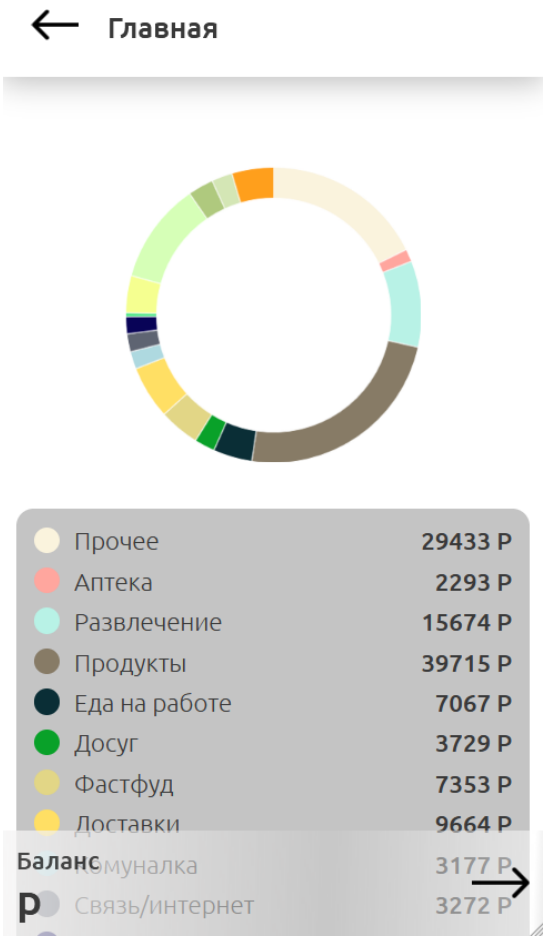

Рисунок 6 – Интерфейс

Таким образом, проблема разработки серверной части приложения может быть решена путем использования фреймворка Nest.js и формата обмена данными JSON.

Список литературы:

1. Open Source. Documentation of TypeORM [Электронный ресурс] URL: https://typeorm.io/ (дата обращения: 23.03.2022).
2. Kamil Mysliwiec. Documentation of Nest.js [Электронный ресурс] URL: https://docs.nestjs.com/ (дата обращения: 23.03.2022).
3. Documentation of React.js [Электронный ресурс] URL: https://reactjs.org/docs/getting-started.html (дата обращения: 24.03.2022).