Персональный проект

Название: Транзакционный модуль

Техническое задание

Необходимо реализовать модуль управления транзакциями.

`Транзакция` — это класс, который реализует механизм последовательного выполнения нескольких связанных шагов **`сценария`**.

Также этот механизм включает в себя возможность делать **`откат`**.

`Откат` — механизм по восстановлению данных для каждого шага **`сценария`**.

`Откат` происходит в случае возникновения ошибки во время выполнения любого шага **`сценария`**.

Механизм **`отката`** также поддерживает последовательное выполнение.

При переходе в режим **`отката`** транзакция последовательно выполнит набор заранее определённой логики для полного восстановления данных, до состояния, зафиксированного до начала выполнения **`сценария`**.

`Сценарий` — это массив, в котором каждый элемент является объектом.

Каждый объект описывает шаг сценария.

Например, сценарий похода в магазин за покупками состоит из следующих шагов:

1. Одеться;
2. Выйти из дому;
3. Прийти в магазин;
4. Совершить покупки;
5. Вернуться домой.

`Транзакция`

Обладает следующими свойствами и методами:

- `store` — ассоциативный массив `Map` , в котором каждое свойство хранит в себе данные, необходимые для выполнения следующего шага сценария;
- `logs` — массив, который содержит в себе объекты, где каждый объект описывает результат выполнения или отката каждого шага сценария;
- `dispatch` — асинхронный метод, который в качестве аргумента принимает **`сценарий`** и реализует логику его выполнения;
- `rollback` — асинхронный метод, который реализует логику отката **`сценария`**.

`Сценарий`

Каждый шаг **`сценария`** содержит четыре **обязательных** свойства:

- `call` — метод для выполнения шага;
- `restore` — метод для восстановления шага;
- `index` — свойство, обозначающее порядковый номер шага;
- `meta` — свойство, с описанием шага. Это объект с двумя вложенными свойствами `title` и `description`. Оба этих свойства являются обязательными.

Статусы выполнения транзакции

`Успешно выполнена` — когда все шаги сценария выполнились без ошибок.

`Успешно восстановлена` — когда на каком-то шаге возникла ошибка, что привело к остановке выполнения сценария, и успешному завершению механизма **`отката`**.

`Неуспешно восстановлена` — когда во время выполнения механизма **`отката`** возникла ошибка.

Основной функционал транзакционного модуля

- Создание новой **`транзакции`** на основании бизнес-сценария;
- Запуск транзакции производится с помощью метода **`dispatch()`**;
- Восстановление состояния **`транзакции`** до изначального состояния на случай возникновения ошибки во время выполнения **`сценария`**;
- Логирование всех действий и всех ошибок;
- Доступность данных на каждом шаге с помощью свойства `store`;
- Результатом выполнения транзакции должно быть выведение в консоль:
  - Ассоциативного массива `store`;
  - Массива `logs`;
  - А также **`статуса выполнения транзакции`**;

Подсказки:

- **`Сценарием`** может быть любой бизнес-скрипт. Например, скрипт трансфера средств с банковского счёта с долларами на банковский счёт с евро:
  1. Получить данные об изначальном состоянии обоих аккаунтов на случай необходимости выполнения логики **`отката`**;
  2. Снять с баланса долларового счёта средства и заморозить его (в период заморозки никакие операции со счётом совершать нельзя);
  3. Сконвертировать снятые средства в нужную валюту (евро);
  4. Полученный объем средств в нужной валюте отправить на другой счёт и заморозить его (в период заморозки никакие операции со счётом совершать нельзя);
  5. Разморозить счёт с евро;
  6. Разморозить счёт с долларами.

Пример использования

```
const scenario = [
    {
        index: 1,
        meta: {
            title: 'Collect backup information.',
            description: 'Collects pieces of data that required for restore
scenario',
        },
        async call(store, logs) {
            // Логика выполнения шага
        },
        async restore(store, logs) {
            // Логика отката шага
        },
    },
    {
      index: 2,
      meta: {
        title: 'Withdraw funds from source account.',
        description:
        'Takes off funds from source account and freezes it until entire
scenario ends successfully or unsuccessfully.',
      },
      async call(store, logs) {
        // Логика выполнения шага
      },
      async restore(store, logs) {
        // Логика отката шага
      },
    },
];

const transaction = new Transaction();

(async() => {
    try {
        await transaction.dispatch(scenario);

        const { store, logs, status } = transaction;
        log(store);
        log(logs);
        log(status);
    } catch (error) {
        // Send email about broken transaction
    }
})();
```

Структура массива `logs`

```
[
    {
        index: 1,
        meta: {
            title: 'Read popular customers'
            description: 'This action is responsible for reading the most
popular customers'
        },
        stepResult: {},
        error: null
    },
    {
        index: 2,
        meta: {
            title: 'Add customer'
            description: 'This action will add some customer'
        },
        NextStep: null,
        error: {
            name: 'TypeError',
            message: 'name is not a function',
            stack: 'Stack trace'
        }
    }
]
```