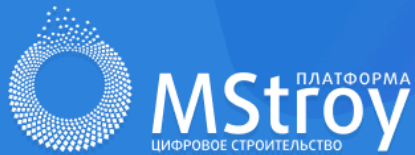


Тестовое задание MStroy Frontend



Разработчик Frontend developer / Vue.js

Есть массив объектов, которые имеют поля `id` и `parent`, через которые их можно связать в дерево и некоторые произвольные поля. `id` может быть как числом, так и строкой. Порядок `id` не гарантируется, они не должны иметь отношения к порядковым номерам, изначально отсутствует какой либо принцип сортировки:

```
const items = [  
  { id: 1, parent: null, label: 'Айтем 1' },  
  
  { id: '2', parent: 1, label: 'Айтем 2' },  
  { id: 3, parent: 1, label: 'Айтем 3' },  
  
  { id: 4, parent: '2', label: 'Айтем 4' },  
  { id: 5, parent: '2', label: 'Айтем 5' },  
  { id: 6, parent: '2', label: 'Айтем 6' },  
  
  { id: 7, parent: 4, label: 'Айтем 7' },  
  { id: 8, parent: 4, label: 'Айтем 8' }  
];
```

Нужно написать класс `TreeStore`, который принимает в конструктор массив этих объектов и реализует следующие методы:

☰ МЕТОДЫ:

- `getAll()` - Должен возвращать изначальный массив элементов.
- `getItem(id)` - Принимает `id` элемента и возвращает сам объект элемента.

- `getChildren(id)` - Принимает `id` элемента и возвращает массив элементов, являющихся дочерними для того элемента, чей `id` получен в аргументе. Если у элемента нет дочерних, то должен возвращаться пустой массив.
- `getAllChildren(id)` - Принимает `id` элемента и возвращает массив элементов, являющихся прямыми дочерними элементами того, чей `id` получен в аргументе + если у них в свою очередь есть еще дочерние элементы, они все тоже будут включены в результат и так до самого глубокого уровня.
- `getAllParents(id)` - Принимает `id` элемента и возвращает массив из цепочки родительских элементов, начиная от самого элемента, чей `id` был передан в аргументе и до корневого элемента, т.е. должен получиться путь элемента вверх дерева через цепочку родителей к корню дерева. В результате `getAllParents` ПОРЯДОК ЭЛЕМЕНТОВ ВАЖЕН!
- `addItem({...})` - Принимает объект нового элемента и добавляет его в общую структуру хранилища.
- `removeItem(id)` - Принимает `id` элемента и удаляет соответствующий элемент и все его дочерние элементы из хранилища.
- `updateItem({...})` - Принимает объект обновленного айтема и актуализирует этот айтем в хранилище.

Для визуализации и взаимодействия с этим классом нужно создать vue-компонент, выводящий элементы хранилища в таблицу на основе библиотеки [AgGrid](#), которая предоставляет уже готовый компонент `<ag-grid-vue/>` для использования с фреймворком Vue.

Айтемы хранилища должны быть представлены в виде строк таблицы. Если у айтема есть дочерние элементы, то такая строка должна быть разворачиваемой. По наличию дочерних элементов должно происходить определение категории строки(столбец **Категория**) - либо **Группа**, либо **Элемент**.

INFO:

Для группировки строк в таблице потребуется официальный плагин AgGrid из пакета Enterprise. Он доступен на npm бесплатно для ознакомительных целей.

Пример отображения в полностью развернутом виде в режиме просмотра:

Режим: просмотр

№ п\п	Категория	Наименование
1	▼ Группа	Айтем 1
2	▼ Группа	Айтем 2
3	▼ Группа	Айтем 4
4	Элемент	Айтем 7
5	Элемент	Айтем 8
6	Элемент	Айтем 5
7	Элемент	Айтем 6
8	Элемент	Айтем 3



❗ КОМПОНЕНТ С ТАБЛИЦЕЙ ДОЛЖЕН ИМЕТЬ 2 РЕЖИМА:

















- Режим просмотра.
- Режим редактирования.

Для переключения режимов над таблицей должна быть соответствующая кнопка. В режиме редактирования должны появляться возможности добавлять и удалять элементы в таблицу(хранилище) и редактировать наименования айтемов в таблице. После нажатия на кнопку "+" на строке, для нее должен добавиться дочерний элемент(строка), ячейку с наименованием которого можно будет отредактировать. После нажатия на кнопку "x" на строке, эта строка и ее дочерние строки должны быть удалены из таблицы(хранилища).

При любых изменениях данных в таблице(хранилище), сами изменения должны регистрироваться для возможности их отменять(повторять). Для перехода по истории изменений в режиме редактирования над таблицей должны отображаться две кнопки со стрелками. Добавление, удаление, переименование элементов можно будет отменять нажатиями на стрелку "назад", а нажатие на стрелку "вперед" будет повторять отмененные ранее изменения.

Пример отображения в режиме редактирования:

Режим: редактирование  

№ п\п	Категория		Наименование
1	▼ Группа	 	Айтем 1
2	▼ Группа	 	Айтем 2
3	▼ Группа	 	Айтем 4
4	Элемент	 	Айтем 7
5	Элемент	 	Айтем 8
6	Элемент	 	Айтем 5
7	Элемент	 	Айтем 6
8	Элемент	 	Айтем 3

ТРЕБОВАНИЯ:

- Решение должно быть оформлено в репозиторий и содержать файлы `package.json` и `package-lock.json` для возможности установить зависимости при проверке.
- Максимальное быстроедействие, следовательно, минимальное количество обходов массива при операциях.
- Класс `TreeStore` должен быть написан на TypeScript и хорошо поддаваться unit-тестированию. При проверке задания этот класс будет импортирован и пропущен через автоматические тесты на большом количестве элементов с замерами времени выполнения методов. Элементы массива будут другими, но их интерфейс будет соответствовать приведенным элементам в этом задании.
- При любых изменениях данных таблица не должна "прыгать" и не должна терять свое текущее состояние раскрытых строк.

БУДЕТ ПЛЮСОМ:

- Написание тестов для класса `TreeStore` и для vue-компонентов, создаваемых в рамках этого тестового задания.
- Написание vue-компонентов с использованием TypeScript.

БУДЕТ ОЦЕНИВАТЬСЯ:

- Полнота реализации этого тестового задания.
- Время выполнения методов класса-хранилища TreeStore.
- Соответствие приведенному дизайну.
- Чистота кода.
- Качество покрытия кода типизацией.
- Структура репозитория и оформление коммитов.
- Качество покрытия кода тестами.