Tabulator

Tabulator 테이블 구현 가이드

📌 시작하기 전에

- 자바스크립트를 몰라도 괜찮습니다
- 아래 코드를 **전체 복사** 후 필요한 부분만 수정하세요
  - (가장 하단의 `clearInterval(this.myInterval)` 은 destroy tab에)
  - REST API와 DB Query의 코드가 조금 차이가 있습니다.
  - DBQuery는 응답을 Format하는 함수가 추가되어 있습니다.
- `this.`는 각 컴포넌트를 구분하는 표시이므로 그대로 두세요

🔧 전체 코드 구조 (REST API)

```
// 1. 기본 함수 정의
this.createTable = fx.curry((element, option) => new Tabulator(element, opti
on));

this.settingData = fx.curry((table, response) => (table.setData(response.dat
a)))

this.getTableData = async apiArguments => {
    const { response } = await WKit.fetchData(...Object.values(apiArgument
s)).catch(console.error);
    return response;
};

// 2. 테이블 생성 및 설정
const tableElement = this.element.querySelector('#tabulator');

// 🟡 테이블 옵션 (필요시 수정)
const tableOption = {
    autoColumns: true,          // 자동으로 컬럼 생성
```

```
    layout: 'fitDataFill',      // 화면에 꽉 차게
    resizableColumnFit: true,    // 컬럼 너비 조절 가능
    resizableRows: true,         // 행 높이 조절 가능
};

this.myTable = this.createTable(tableElement, tableOption);

// 🔴 API 연결 정보 (수정 필요)
const apiArguments = {
    page: this.page,
    datasetName: 'api_tabulator',    // ← 여러분의 API 이름으로 변경
    params: { port: 3000, rows: 10 } // ← 필요한 파라미터로 변경
}

// 3. 데이터 로드 함수
const loadTable = () => fx.go(
    this.getTableData(apiArguments),
    this.settingData(this.myTable)
)

loadTable();

// 🟡 자동 새로고침 설정 (필요시 수정)
this.REFRESH_INTERVAL = 5000;  // 5초마다 새로고침
this.myInterval = setInterval(loadTable, this.REFRESH_INTERVAL);

// 4. 정리 (destroy 섹션)
clearInterval(this.myInterval);
```

🔧 전체 코드 구조 (DB Query)

```
this.createTable = fx.curry((element, option) => new Tabulator(element, option));

this.settingData = fx.curry((table, response) => (table.setData(response.data)))
```

```
this.getTableData = async apiArguments => {
    const { response } = await WKit.fetchData(...Object.values(apiArgument
s)).catch(console.error);
    return formatTableData(response);
};


function formatTableData(response) {
    const data = Array.isArray(response) ? response : (response.rows || []);
    if (!data || data.length === 0) {
        return {
            data: [],
            totalRows: 0,
            columns: [],
            metadata: {
                generated: new Date().toISOString(),
                requestedRows: 0,
                requestedColumns: 0
            }
        };
    }
    const columns = Object.keys(data[0]);
    return {
        data: data,
        totalRows: data.length,
        columns: columns,
        metadata: {
            generated: new Date().toISOString(),
            requestedRows: data.length,
            requestedColumns: columns.length
        }
    };
}


// 2. 테이블 생성 및 설정
const tableElement = this.element.querySelector('#tabulator');
```

```
// 🟡 테이블 옵션 (필요시 수정)
const tableOption = {
    autoColumns: true,          // 자동으로 컬럼 생성
    layout: 'fitDataFill',      // 화면에 꽉 차게
    resizableColumnFit: true,   // 컬럼 너비 조절 가능
    resizableRows: true,        // 행 높이 조절 가능
};

this.myTable = this.createTable(tableElement, tableOption);

// 🔴 API 연결 정보 (수정 필요)
const apiArguments = {
    page: this.page,
    datasetName: 'api_tabulator_query',    // ← 여러분의 API 이름으로 변경
    params: { rows: "10" } // ← 필요한 파라미터로 변경, query는 value를 "" 문자
열로
}

// 3. 데이터 로드 함수
const loadTable = () => fx.go(
    this.getTableData(apiArguments),
    this.settingData(this.myTable)
)

loadTable();

// 🟡 자동 새로고침 설정 (필요시 수정)
this.REFRESH_INTERVAL = 5000;  // 5초마다 새로고침
this.myInterval = setInterval(loadTable, this.REFRESH_INTERVAL);


// 4. 정리 (destroy 섹션)
clearInterval(this.myInterval);
```

📊 데이터 구조 이해하기

API 응답 구조:


```
{
  "data": [
    {
      "id": 1,
      "name": "Employee 1",
      "age": 36,
      "department": "Finance",
      "salary": 66624,
      "email": "employee1@company.com",
      "status": "Active",
      "joinDate": "2022-08-09"
    },
    {
      "id": 2,
      "name": "Employee 2",
      "age": 61,
      "department": "Sales",
      "salary": 81704,
      "email": "employee2@company.com",
      "status": "Active",
      "joinDate": "2023-10-23"
    }
  ],
  "totalRows": 2,
  "columns": [
    "id",
    "name",
    "age",
    "department",
    "salary",
    "email",
    "status",
    "joinDate"
  ],
  "metadata": {
    "generated": "2025-08-12T02:50:40.908Z",
```

```
  "requestedRows": 2,
  "requestedColumns": 5
 }
}
```

DB Query 응답 구조:

```
[
  {
   "id": 1,
   "name": "Employee 1",
   "age": 31,
   "department": "Marketing",
   "salary": 30793
  },
  {
   "id": 2,
   "name": "Employee 2",
   "age": 44,
   "department": "Finance",
   "salary": 72449
  }
 ]
```

중요 포인트:

- 현재 예제는 `data` 속성 안에 배열이 있는 구조입니다
- `response.data`를 테이블에 전달하고 있습니다
- 각 객체의 속성(id, name, age 등)이 자동으로 컬럼이 됩니다

💡 **데이터 구조가 다른 경우:**

```
// 만약 API가 바로 배열을 반환한다면:
// [{ id: 1, name: "Employee 1" }, { id: 2, name: "Employee 2" }]

this.settingData = fx.curry((table, response) => (
  table.setData(response)  // response.data가 아닌 response 사용
))
```

```
// 만약 다른 속성명을 사용한다면:
// { "employees": [...], "total": 2 }

this.settingData = fx.curry((table, response) => (
    table.setData(response.employees)  // response.employees 사용
))

---

## 🎯 데이터 흐름 설명

### `loadTable` 함수의 동작:
```javascript
const loadTable = () => fx.go(
    this.getTableData(apiArguments),  // 1단계: API 호출
    this.settingData(this.myTable)    // 2단계: 테이블에 표시
)
```

단계별 설명:

1. `this.getTableData(apiArguments)`
   - API를 호출하여 데이터를 가져옵니다
   - 결과: `{ data: [...], totalRows: 5 }` 형태의 response 객체
2. `this.settingData(this.myTable)`
   - response를 받아서 `response.data`를 추출
   - `table.setData(response.data)`로 테이블에 표시
   - **setData()는 배열을 받아 테이블을 그립니다**

데이터 전달 과정:

```
API 호출 → response 객체 → response.data 추출 → setData() → 테이블 표시
```

✏️ 수정 가능한 부분

1. API 정보 변경 (🔴 필수)
```

```
const apiArguments = {
    page: this.page,
    datasetName: 'employee_list',     // API 이름 변경
    params: {
        port: 3000,
        rows: 20,                     // 20개 행 요청
        department: 'Sales'           // 추가 파라미터
    }
}
```

2. 테이블 옵션 변경 (🟡 선택)

```
const tableOption = {
    autoColumns: true,
    layout: 'fitColumns',      // fitDataFill → fitColumns
    pagination: true,          // 페이지네이션 추가
    paginationSize: 15         // 한 페이지당 15개
};
```

3. 새로고침 주기 변경 (🟡 선택)

```
this.REFRESH_INTERVAL = 10000;  // 10초로 변경
// 또는 자동 새로고침을 원하지 않으면 setInterval 라인 삭제
```

🔍 Preview 섹션 이해

Preview는 개발 중 미리보기용입니다:

```
// 샘플 응답 데이터
const response = {
  "data": [
    { "id": 1, "name": "Employee 1", "department": "Marketing" },
    { "id": 2, "name": "Employee 2", "department": "Finance" }
  ]
}
```

```
// 미리보기 테이블
this.previewTable = new Tabulator("#" + this.id + " #tabulator", {
  data: response.data,  // response.data를 직접 사용
  autoColumns: true,
  layout: 'fitData'
});
```

⚠️ 자주 발생하는 문제

테이블이 안 보일 때

- HTML에 `<div id="tabulator"></div>` 확인
- 브라우저 콘솔(F12)에서 에러 메시지 확인

데이터가 안 나올 때

- `datasetName`이 정확한지 확인
- **`setData()`에 배열이 제대로 전달되는지 확인**
  - 브라우저 콘솔에서 `console.log(response)` 찍어보기
  - 배열이 어디 있는지 확인 후 `settingData` 함수 수정
- 배열 안에 객체들 `[{}, {}, {}]`이 있는지 확인

✅ 체크리스트

- ☐ HTML에 `<div id="tabulator"></div>` 존재
- ☐ `datasetName`을 내 API 이름으로 변경
- ☐ `params`를 필요한 값으로 수정
- ☐ **`setData()`에 배열 형태의 데이터가 전달되는지 확인**
  - 예: `[{...}, {...}, {...}]` 형태
  - API 구조에 따라 `response`, `response.data`, `response.items` 등으로 조정