

Je souhaite créer un composant vuejs 2 permettant d'intégrer datatables.net dans une page vuejs 2 avec typescript

ChatGPT said:

Parfait ! Je peux t'aider à créer un composant Vue.js 2 en TypeScript qui intègre DataTables.net.

## Contexte rapide

- Vue 2 + TypeScript
  - DataTables.net (plugin jQuery)
  - Le composant doit initialiser DataTables sur un tableau HTML
- 

## Étapes principales

1. Installer les dépendances nécessaires (jQuery, DataTables, types)
  2. Créer un composant Vue 2 en TypeScript
  3. Initialiser DataTables dans le lifecycle hook `mounted`
  4. Nettoyer dans `beforeDestroy`
- 

## 1. Installer les dépendances

Dans ton projet Vue 2, lance :

```
npm install jquery datatables.net datatables.net-dt --save
npm install --save-dev @types/jquery @types/datatables.net
```

---

## 2. Exemple de composant Vue 2 + TypeScript avec DataTables.net

```
<template>
  <div>
    <table ref="datatable" class="display" style="width:100%">
      <thead>
        <tr>
          <th>Nom</th>
          <th>Position</th>
          <th>Bureau</th>
          <th>Âge</th>
          <th>Date de début</th>
          <th>Salaire</th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="(item, index) in data" :key="index">
          <td>{{ item.name }}</td>
          <td>{{ item.position }}</td>
```

```

        <td>{{ item.office }}</td>
        <td>{{ item.age }}</td>
        <td>{{ item.startDate }}</td>
        <td>{{ item.salary }}</td>
    </tr>
</tbody>
</table>
</div>
</template>

```

```

<script lang="ts">
import Vue from 'vue';
import $ from 'jquery';
import 'datatables.net';
import 'datatables.net-dt/css/jquery.dataTables.css';

```

```

interface DataItem {
  name: string;
  position: string;
  office: string;
  age: number;
  startDate: string;
  salary: string;
}

```

```

export default Vue.extend({
  name: 'DataTableComponent',
  data() {
    return {
      data: [
        {
          name: 'Tiger Nixon',
          position: 'System Architect',
          office: 'Edinburgh',
          age: 61,
          startDate: '2011/04/25',
          salary: '$320,800',
        },
        {
          name: 'Garrett Winters',
          position: 'Accountant',
          office: 'Tokyo',
          age: 63,
          startDate: '2011/07/25',
          salary: '$170,750',
        },
        // ... plus de données
      ] as DataItem[],
      dataTableInstance: null as any | null,
    };
  },
  mounted() {
    this.$nextTick(() => {
      this.dataTableInstance = ($(this.$refs.datatable) as any).DataTable({
        // options DataTables si besoin
      });
    });
  },
  beforeDestroy() {

```

```

        if (this.dataTableInstance) {
            this.dataTableInstance.destroy();
            this.dataTableInstance = null;
        }
    },
});
</script>

<style scoped>
/* Optionnel, pour ajuster le style DataTables */
</style>

```

---

## Explications

- `ref="datatable"` pour cibler la table dans le DOM.
  - `this.dataTableInstance = $(...).DataTable()` initialise DataTables.
  - `destroy()` pour bien nettoyer à la destruction du composant.
  - Utilisation de `lang="ts"` pour TypeScript dans `<script>`.
  - Le tableau est simple, avec des données en dur, tu peux bien sûr passer les données en props si tu veux.
- 

Veux-tu que je te prépare un exemple avec des données passées en props ? Ou un exemple avec plus d'options DataTables (pagination, recherche, etc.) ?

**You said:**

je veux un composant générique

**ChatGPT said:**

Ok, un composant générique DataTable pour Vue 2 + TypeScript, qui :

- Prend en prop les données (tableau d'objets)
- Prend en prop les colonnes (pour définir l'en-tête et les clés)
- Initialise DataTables avec les options passées en prop

Je te fais ça.

---

## Composant Vue 2 générique DataTable avec TypeScript

```

<template>
  <div>

```

```

<table ref="datatable" class="display" style="width:100%">
  <thead>
    <tr>
      <th v-for="(col, index) in columns" :key="index">{{ col.label }}</th>
    </tr>
  </thead>
  <tbody>
    <tr v-for="(row, rowIndex) in data" :key="rowIndex">
      <td v-for="(col, colIndex) in columns" :key="colIndex">
        {{ row[col.field] }}
      </td>
    </tr>
  </tbody>
</table>
</div>
</template>

<script lang="ts">
import Vue from 'vue';
import $ from 'jquery';
import 'datatables.net';
import 'datatables.net-dt/css/jquery.dataTables.css';

interface Column {
  label: string; // Nom affiché dans l'entête
  field: string; // Clé dans l'objet data
}

export default Vue.extend({
  name: 'GenericDataTable',
  props: {
    data: {
      type: Array as () => Array<Record<string, any>>,
      required: true,
    },
    columns: {
      type: Array as () => Column[],
      required: true,
    },
    options: {
      type: Object,
      required: false,
      default: () => ({}),
    },
  },
  data() {
    return {
      dataTableInstance: null as any | null,
    };
  },
  watch: {
    data: {
      handler() {
        this.reloadTable();
      },
      deep: true,
    },
  },
  mounted() {

```

```

        this.initDataTable();
    },
    beforeDestroy() {
        this.destroyDataTable();
    },
    methods: {
        initDataTable() {
            this.$nextTick(() => {
                this.dataTableInstance = ($(this.$refs.datatable) as
any).DataTable(this.options);
            });
        },
        reloadTable() {
            if (this.dataTableInstance) {
                this.dataTableInstance.clear();
                this.dataTableInstance.rows.add(this.data);
                this.dataTableInstance.draw();
            }
        },
        destroyDataTable() {
            if (this.dataTableInstance) {
                this.dataTableInstance.destroy();
                this.dataTableInstance = null;
            }
        },
    },
});
</script>

<style scoped>
/* Optionnel */
</style>

```

---

## Utilisation

```

<template>
  <GenericDataTable
    :data="tableData"
    :columns="tableColumns"
    :options="{ paging: true, searching: true }"
  />
</template>

<script lang="ts">
import Vue from 'vue';
import GenericDataTable from './components/GenericDataTable.vue';

export default Vue.extend({
  components: { GenericDataTable },
  data() {
    return {
      tableColumns: [
        { label: 'Nom', field: 'name' },
        { label: 'Poste', field: 'position' },
        { label: 'Bureau', field: 'office' },
        { label: 'Âge', field: 'age' },
      ],
    };
  },
});

```

```
        tableData: [
          { name: 'Tiger Nixon', position: 'System Architect', office: 'Edinburgh',
age: 61 },
          { name: 'Garrett Winters', position: 'Accountant', office: 'Tokyo', age: 63
},
        ],
      },
    });
</script>
```

---

## Notes

- La prop `data` est un tableau d'objets.
  - La prop `columns` définit la colonne : label (affiché) et field (clé dans l'objet).
  - La prop `options` permet de passer des options DataTables natives.
  - Le watch sur `data` permet de recharger le tableau si les données changent.
- 

Veux-tu que je t'aide à gérer les colonnes triables ou des colonnes avec du rendu personnalisé ?

**You said:**

peut-on ajouter la pagination et le nombre de lignes par page ?

**ChatGPT said:**

Oui, absolument ! DataTables gère la pagination et le choix du nombre de lignes par page nativement via ses options.

Dans le composant générique, tu passes ça dans la prop `options`. Par défaut, DataTables affiche une pagination et une liste déroulante pour sélectionner le nombre de lignes (length menu).

---

## Comment activer pagination et nombre de lignes par page

Dans ta prop `options`, tu peux spécifier par exemple :

```
{
  paging: true, // active la pagination (true par défaut)
  lengthMenu: [5, 10, 25, 50], // options pour le nombre de lignes affichées
  pageSize: 10, // nombre de lignes par défaut affichées
}
```

---

## Exemple complet avec pagination et longueur de page

```
<GenericDataTable
  :data="tableData"
```

```
:columns="tableColumns"
:options="{ paging: true, lengthMenu: [5, 10, 25, 50], pageLength: 10 }"
/>
```

---

## Petit rappel sur le composant

Le composant utilise cette option quand il initialise DataTables :

```
this.dataTableInstance = ($(this.$refs.datatable) as any).DataTable(this.options);
```

Donc, tout ce que tu passes dans `options` sera appliqué à DataTables.

---

## Bonus : personnaliser la pagination ou le texte ?

Tu peux ajouter d'autres options DataTables dans `options` comme `language` pour personnaliser les labels :

```
{
  language: {
    paginate: {
      previous: "Précédent",
      next: "Suivant"
    },
    lengthMenu: "Afficher _MENU_ entrées",
    info: "Affichage de _START_ à _END_ sur _TOTAL_ entrées"
  }
}
```

---

Veux-tu que je t'intègre ces options directement dans le composant avec un exemple complet prêt à l'emploi ?

**You said:**

oui

**ChatGPT said:**

Parfait ! Voici un composant Vue 2 + TypeScript générique **avec pagination, choix du nombre de lignes par page, et texte en français** personnalisé via les options DataTables.

---

## Composant `GenericDataTable.vue` complet

```
<template>
  <div>
    <table ref="datatable" class="display" style="width:100%">
      <thead>
        <tr>
          <th v-for="(col, index) in columns" :key="index">{{ col.label }}</th>
```

```

        </tr>
    </thead>
    <tbody>
        <tr v-for="(row, rowIndex) in data" :key="rowIndex">
            <td v-for="(col, colIndex) in columns" :key="colIndex">
                {{ row[col.field] }}
            </td>
        </tr>
    </tbody>
</table>
</div>
</template>

```

```

<script lang="ts">
import Vue from 'vue';
import $ from 'jquery';
import 'datatables.net';
import 'datatables.net-dt/css/jquery.dataTables.css';

```

```

interface Column {
    label: string;
    field: string;
}

```

```

export default Vue.extend({
    name: 'GenericDataTable',
    props: {
        data: {
            type: Array as () => Array<Record<string, any>>,
            required: true,
        },
        columns: {
            type: Array as () => Column[],
            required: true,
        },
        options: {
            type: Object,
            required: false,
            default: () => ({}),
        },
    },
    data() {
        return {
            dataTableInstance: null as any | null,
        };
    },
    watch: {
        data: {
            handler() {
                this.reloadTable();
            },
            deep: true,
        },
    },
    mounted() {
        this.initDataTable();
    },
    beforeDestroy() {
        this.destroyDataTable();
    },

```



```

    },
    methods: {
        initDataTable() {
            this.$nextTick(() => {
                const defaultOptions = {
                    paging: true,
                    lengthMenu: [5, 10, 25, 50],
                    pageLength: 10,
                    language: {
                        paginate: {
                            previous: "Précédent",
                            next: "Suivant",
                        },
                        lengthMenu: "Afficher _MENU_ entrées",
                        info: "Affichage de _START_ à _END_ sur _TOTAL_ entrées",
                        infoEmpty: "Aucune entrée disponible",
                        infoFiltered: "(filtré à partir de _MAX_ entrées au total)",
                        search: "Recherche :",
                        zeroRecords: "Aucun résultat trouvé",
                        emptyTable: "Aucune donnée disponible dans le tableau",
                    },
                    ...this.options, // merge avec options passées en prop
                };

                this.dataTableInstance = ($(this.$refs.datatable) as
any).DataTable(defaultOptions);
            });
        },
        reloadTable() {
            if (this.dataTableInstance) {
                this.dataTableInstance.clear();
                this.dataTableInstance.rows.add(this.data);
                this.dataTableInstance.draw();
            }
        },
        destroyDataTable() {
            if (this.dataTableInstance) {
                this.dataTableInstance.destroy();
                this.dataTableInstance = null;
            }
        },
    },
});
</script>

<style scoped>
/* Optionnel, tu peux personnaliser le style ici */
</style>

```

---

## Exemple d'utilisation

```

<template>
  <GenericDataTable
    :data="tableData"
    :columns="tableColumns"
    :options="{ pageLength: 5 }"

```

```

/>
</template>

<script lang="ts">
import Vue from 'vue';
import GenericDataTable from './components/GenericDataTable.vue';

export default Vue.extend({
  components: { GenericDataTable },
  data() {
    return {
      tableColumns: [
        { label: 'Nom', field: 'name' },
        { label: 'Poste', field: 'position' },
        { label: 'Bureau', field: 'office' },
        { label: 'Âge', field: 'age' },
      ],
      tableData: [
        { name: 'Tiger Nixon', position: 'System Architect', office: 'Edinburgh',
age: 61 },
        { name: 'Garrett Winters', position: 'Accountant', office: 'Tokyo', age: 63
},
        { name: 'Ashton Cox', position: 'Junior Technical Author', office: 'San
Francisco', age: 66 },
        { name: 'Cedric Kelly', position: 'Senior Javascript Developer', office:
'Edinburgh', age: 22 },
        { name: 'Airi Satou', position: 'Accountant', office: 'Tokyo', age: 33 },
        { name: 'Brielle Williamson', position: 'Integration Specialist', office:
'New York', age: 61 },
        // plus de données ici...
      ],
    };
  },
});
</script>

```

---

## Résumé

- Pagination activée par défaut (`paging: true`)
  - Choix du nombre de lignes affichées via `lengthMenu` (5,10,25,50)
  - Texte en français personnalisé dans l'option `language`
  - Tu peux toujours passer des options supplémentaires dans la prop `options` (ex: désactiver recherche, etc.)
- 

Si tu veux, je peux aussi t'aider à ajouter la gestion des colonnes triables, du rendu personnalisé dans une cellule, ou encore l'intégration avec des données asynchrones.

N'hésite pas !

**You said:**

on peut ajouter la taille des colonnes ?

**ChatGPT said:**

Oui, on peut tout à fait définir la largeur des colonnes dans DataTables. Deux manières classiques :

1. **CSS** : appliquer des styles CSS (width) aux `<th>` ou `<td>`.
  2. **Option DataTables `columns.width`** : tu peux définir un tableau d'objets `columns` dans l'option DataTables, chaque objet pouvant contenir une propriété `width`.
- 

## Intégrer la taille des colonnes dans ton composant générique

On va enrichir la définition des colonnes pour ajouter une propriété optionnelle `width` (ex : "100px" ou "20%").

Ensuite, dans l'option DataTables, on va passer cette info dans la clé `columns` (attention, ce n'est pas la même que la prop `columns` vue par Vue).

---

## Modifications à faire

### 1. Modifier l'interface `Column` dans ton composant

```
interface Column {
  label: string;
  field: string;
  width?: string; // largeur optionnelle (ex: "100px", "20%")
}
```

### 2. Dans la méthode `initDataTable()`, préparer l'option `columns` pour DataTables

```
initDataTable() {
  this.$nextTick(() => {
    // Construire l'option "columns" pour DataTables, en gardant la largeur si
    définie
    const dtColumns = this.columns.map(col => {
      const colOption: any = {};
      if (col.width) {
        colOption.width = col.width;
      }
      return colOption;
    });

    const defaultOptions = {
      paging: true,
      lengthMenu: [5, 10, 25, 50],
      pageLength: 10,
      language: {
        paginate: {
          previous: "Précédent",
          next: "Suivant",

```

```

    },
    lengthMenu: "Afficher _MENU_ entrées",
    info: "Affichage de _START_ à _END_ sur _TOTAL_ entrées",
    infoEmpty: "Aucune entrée disponible",
    infoFiltered: "(filtré à partir de _MAX_ entrées au total)",
    search: "Recherche :",
    zeroRecords: "Aucun résultat trouvé",
    emptyTable: "Aucune donnée disponible dans le tableau",
  },
  columns: dtColumns,
  ...this.options,
};

this.dataTableInstance = ($(this.$refs.datatable) as
any).DataTable(defaultOptions);
});
}

```

---

## Exemple d'utilisation avec largeur des colonnes

```

tableColumns: [
  { label: 'Nom', field: 'name', width: '150px' },
  { label: 'Poste', field: 'position', width: '200px' },
  { label: 'Bureau', field: 'office', width: '120px' },
  { label: 'Âge', field: 'age', width: '80px' },
],

```

---

## Note importante

- DataTables applique souvent la largeur via le tableau `columns` (option JS) plutôt que via CSS inline dans le `<th>`.
- Si tu veux forcer la largeur côté CSS (dans `<th>`), tu peux aussi ajouter dans le template un `:style="{ width: col.width }"` sur les `<th>`.

Exemple dans le template :

```

<th
  v-for="(col, index) in columns"
  :key="index"
  :style="{ width: col.width || 'auto' }"
>
  {{ col.label }}
</th>

```

---

Veux-tu que je t'intègre ces modifications dans le composant complet ?

**You said:**

je souhaite ajouter le type de pagination

ChatGPT said:

Bien sûr ! DataTables propose plusieurs types de pagination via l'option `pagingType`. Par exemple :

- `'simple'` : boutons **Précédent** et **Suivant**
  - `'simple_numbers'` : boutons Précédent/Suivant + numéros de page
  - `'full'` : boutons Premier, Précédent, Suivant, Dernier
  - `'full_numbers'` : boutons Premier, Précédent, numéros de page, Suivant, Dernier
  - `'numbers'` : seulement numéros de page
- 

## Comment l'ajouter dans ton composant générique

Il suffit de passer la propriété `pagingType` dans l'option DataTables.

Comme on transmet déjà une prop `options` qui merge avec les options par défaut, il suffit de préciser `pagingType` dans la prop `options` lors de l'utilisation du composant.

---

## Exemple d'utilisation avec `pagingType`

```
<GenericDataTable
  :data="tableData"
  :columns="tableColumns"
  :options="{ pagingType: 'full_numbers', pageLength: 10 }"
/>
```

---

## Pour intégrer par défaut dans le composant

Dans le composant, dans `initDataTable()`, tu peux ajouter à `defaultOptions` :

```
pagingType: 'simple_numbers',
```

Ce qui sera la pagination par défaut, mais l'utilisateur pourra override via la prop `options`.

---

## Résumé rapide

- Par défaut dans le composant :

```
const defaultOptions = {
  paging: true,
  pagingType: 'simple_numbers',
  lengthMenu: [5, 10, 25, 50],
  pageLength: 10,
  // autres options...
}
```

- L'utilisateur peut toujours passer `pagingType` dans la prop `options` pour changer la pagination.
- 

Veux-tu que je te mette à jour le composant complet avec cette option par défaut et un exemple d'utilisation ?

**You said:**

oui

**ChatGPT said:**

Voici la version complète mise à jour de ton composant Vue 2 + TypeScript, incluant :

- la pagination avec `pagingType` par défaut à `'simple_numbers'`
  - la gestion de la largeur des colonnes (`width`)
  - la pagination, le choix du nombre de lignes, et le texte en français
  - possibilité de surcharger les options via la prop `options`
- 

## GenericDataTable.vue

```
<template>
  <div>
    <table ref="datatable" class="display" style="width:100%">
      <thead>
        <tr>
          <th
            v-for="(col, index) in columns"
            :key="index"
            :style="{ width: col.width || 'auto' }"
          >
            {{ col.label }}
          </th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="(row, rowIndex) in data" :key="rowIndex">
          <td v-for="(col, colIndex) in columns" :key="colIndex">
            {{ row[col.field] }}
          </td>
        </tr>
      </tbody>
    </table>
  </div>
</template>

<script lang="ts">
import Vue from 'vue';
import $ from 'jquery';
import 'datatables.net';
import 'datatables.net-dt/css/jquery.dataTables.css';
```

```

interface Column {
  label: string;
  field: string;
  width?: string; // Exemple : "150px", "20%"
}

export default Vue.extend({
  name: 'GenericDataTable',
  props: {
    data: {
      type: Array as () => Array<Record<string, any>>,
      required: true,
    },
    columns: {
      type: Array as () => Column[],
      required: true,
    },
    options: {
      type: Object,
      required: false,
      default: () => ({}),
    },
  },
  data() {
    return {
      dataTableInstance: null as any | null,
    };
  },
  watch: {
    data: {
      handler() {
        this.reloadTable();
      },
      deep: true,
    },
  },
  mounted() {
    this.initDataTable();
  },
  beforeDestroy() {
    this.destroyDataTable();
  },
  methods: {
    initDataTable() {
      this.$nextTick(() => {
        // Construire les colonnes pour DataTables en récupérant la largeur
        const dtColumns = this.columns.map(col => {
          const colOption: any = {};
          if (col.width) {
            colOption.width = col.width;
          }
          return colOption;
        });

        const defaultOptions = {
          paging: true,
          pagingType: 'simple_numbers', // pagination avec numéros +
          précédent/suivant

```

```

lengthMenu: [5, 10, 25, 50],
pageLength: 10,
language: {
  paginate: {
    previous: "Précédent",
    next: "Suivant",
  },
  lengthMenu: "Afficher _MENU_ entrées",
  info: "Affichage de _START_ à _END_ sur _TOTAL_ entrées",
  infoEmpty: "Aucune entrée disponible",
  infoFiltered: "(filtré à partir de _MAX_ entrées au total)",
  search: "Recherche :",
  zeroRecords: "Aucun résultat trouvé",
  emptyTable: "Aucune donnée disponible dans le tableau",
},
columns: dtColumns,
...this.options, // surcharge avec options passées en prop
};

this.dataTableInstance = ($(this.$refs.datatable) as
any).DataTable(defaultOptions);
});
},
reloadTable() {
  if (this.dataTableInstance) {
    this.dataTableInstance.clear();
    this.dataTableInstance.rows.add(this.data);
    this.dataTableInstance.draw();
  }
},
destroyDataTable() {
  if (this.dataTableInstance) {
    this.dataTableInstance.destroy();
    this.dataTableInstance = null;
  }
},
},
});
</script>

<style scoped>
/* Personnalise ici si besoin */
</style>

```

---

## Exemple d'utilisation avec pagination complète et largeur

```

<template>
  <GenericDataTable
    :data="tableData"
    :columns="tableColumns"
    :options="{ pagingType: 'full_numbers', pageLength: 5 }"
  />
</template>

<script lang="ts">
import Vue from 'vue';

```



```

import GenericDataTable from './components/GenericDataTable.vue';

export default Vue.extend({
  components: { GenericDataTable },
  data() {
    return {
      tableColumns: [
        { label: 'Nom', field: 'name', width: '150px' },
        { label: 'Poste', field: 'position', width: '200px' },
        { label: 'Bureau', field: 'office', width: '120px' },
        { label: 'Âge', field: 'age', width: '80px' },
      ],
      tableData: [
        { name: 'Tiger Nixon', position: 'System Architect', office: 'Edinburgh',
age: 61 },
        { name: 'Garrett Winters', position: 'Accountant', office: 'Tokyo', age: 63
},
        { name: 'Ashton Cox', position: 'Junior Technical Author', office: 'San
Francisco', age: 66 },
        { name: 'Cedric Kelly', position: 'Senior Javascript Developer', office:
'Edinburgh', age: 22 },
        { name: 'Airi Satou', position: 'Accountant', office: 'Tokyo', age: 33 },
        { name: 'Brielle Williamson', position: 'Integration Specialist', office:
'New York', age: 61 },
        // plus de données...
      ],
    };
  },
});
</script>

```

---

## Tu peux modifier la pagination via la prop options :

- `pagingType`: 'simple', 'simple\_numbers', 'full', 'full\_numbers', 'numbers'
  - `pageLength`: nombre de lignes affichées par défaut
  - `lengthMenu`: tableau des choix du nombre de lignes (ex: [5, 10, 25, 50])
- 

Si tu veux je peux aussi t'aider à gérer :

- colonnes triables ou non
- rendu personnalisé dans une cellule (slot)
- chargement des données asynchrones (via AJAX)
- autres fonctionnalités DataTables

N'hésite pas !