Moisi Olivia-Debora
SE, 248-2

MyCustomTable

- Custom Angular Component -

1. Overview

MyCustomTable is a custom Angular component which enables showing different kinds of data in a paginated table. The column settings, the actions on each row (as buttons), the data types, column names and values can be customized which makes this component reusable. The second custom Angular component which is incorporated in the first one is ***AppPagination*** which allows setting the number of records per page in the table. Upon clicking on a certain page or on clicking the buttons "next" or "previous" the data in the table will be shown accordingly. To achieve the wanted behavior of the table we also created other custom components: pipes to format cell data and directives to apply styles.

2. Usage

The custom table component can be useful in many different cases. Almost each website requires at least one type of data table to show different kinds of data, for example an e-commerce site would require to show its customers their orders, the administrators of the store the orders, customers, products.

3. Getting started

For starting the development server of the project from the GitHub it should be firstly cloned into your local machine. After opening it you should start a terminal and run the command *"npm install"* so that each required node module will be installed. The next step is to run the command *"ng serve"* and then navigate to http://localhost:4200. Upon changing any of the source files the application will automatically perform a reload operation so that the changes are instantly reflected in the browser.

4. Description. Implementation

Adding the custom table component into your Angular application is very easy and straightforward. You should copy the directory found at /src/app/table and import ***CustomTableModule*** in your app.module.ts.

To add it to your template page use the app-custom-table for the custom tag, with the following inputs:

- ***title****: which represents the tables title*
- **records**: which represents the table data
- ***settings***: which represents configuration of each cell // Optional
- ***buttons***: which represents button configurations // Optional
- ***recordsPerPage:*** if not set, the default value is 5 // Optional

The interfaces for the ***ColumnSetting*** and ***ButtonSettings*** are shown in the table/models folder, files ButtonSettings.ts and ColumnSettings.ts files.

In the ngOnInit method we check if a buttons input is passed and then initialize it as an empty array if undefined. We perform the same check on the settings input, if its undefined we generate a default settings configuration.

In the ***custom-table template component*** we loop through the ColumnMap configuration and set the value of the <th> element to the be the value of the header property. We also show or hide the Action or Actions table head text based on the buttons input.

For the <tr> we loop through the records data, then for each record we loop through the ColumnMap config and set the value of the <td> element to the be the value of the primaryKey property. We also add the appStyleCell directive and the formatCell pipe to give us more flexibility. The <td> buttons are also populated based on the buttons input.

When a page is clicked we set the currentPage and get the appropriate records to show based on the active page and on the recordsPerPage values.

The table data can be formatted by using the custom defined pipe: ***formatCell***, by using one of the enum PipeFormat types from ColumnSettings.ts, which are:

- DEFAULT
- CURRENCY
- DATE
- PERCENTAGE

Firstly, we inject the CurrencyPipe and DatePipe into the component, then we

- check if the value is undefined, if its undefined we return a "-" in other to avoid empty spaces. Then we cater for the available format types specified in the model.ts file. If the format is DEFAULT, we check if the value is an ARRAY or an OBJECT then format accordingly. If the format is CURRENCY, we use the
- injected CurrencyPipe to transform the value and if the format is DATE, we use the injected DatePipe to transform the value.
- The CurrencyPipe and DatePipe injected into this pipe are also added to the providers property of the custom-table.module.ts file.

The ***appStyleCell*** directive can be seen in the table/style-cell.directive.ts file. It is used to apply given array of classes as styles for our elements. Here we expect two inputs appStyleCell which represents the cell value and the key which represents the cell's key. We also inject ElementRef and Renderer2 so we can update the view. Now that we have the inputs, we can set custom views based on incoming values.

The ***pagination component*** used by the custom table is accepting two inputs - TotalRecords and RecordsPerPage

- ***RecordsPerPage*** is used for the number of items we want to display in single page
- ***TotalRecords*** is used for the total number of items count which we want to display in pages; its value is the length of the table's records array

Also, we have created an ***EventEmitter*** which will emit the currently active page number to parent component whenever we will click on the page number or change it using “next” or “previous” buttons.

5. Integration. Demo

Full Name	Email	Phone Number	Loyal Customer	Amount Spent	Action
Bogdan Groza	bogdan@invalid.com	0744111222	Yes	$ 400.00	Info
Debora Moisi	debora@invalid.com	0788999666	Yes	$ 450,000.00	Info
Dorita Pustea	dorita@invalid.com	0766333777	No	$ 59,000.00	Info
Hannah Illyes	hannah@invalid.com	0799666333	No	$ 200.00	Info
Foltut Ana	ana@invalid.com	0732145678	Yes	$ 120,000.00	Info

« Previous | **1** | 2 | Next »

In order to have this table we have set the following:
We first have to download the table directory and include it in our Angular project. In the module we want to use the table we import the CustomTableModule and add it to the imports array of the module file.
We define the settings and records for our data in the component class. We also have to import ***ButtonSettings, ColumnSettings*** and ***PipeFormat.***

```
import ButtonSettings from './table/models/ButtonSettings';
import ColumnSetting, { PipeFormat } from './table/models/ColumnSettings';
```

For each column we have set a primaryKey which is unique. Also, the ***header*** takes in the displayed name for the column. The ***format*** is optional, but we can format different types of data: date, percentage, currency.

```
customerTableSettings: ColumnSetting[] = [
  {
    primaryKey: 'name',
    header: 'Full Name',
  },
  {
    primaryKey: 'email',
    header: 'Email',
  },
  {
    primaryKey: 'phone',
    header: 'Phone Number',
  },
  {
    primaryKey: 'loyaltyReward',
    header: 'Loyal Customer',
  },
  {
    primaryKey: 'totalSpent',
    header: 'Amount Spent',
    format: PipeFormat.CURRENCY,
  },
];
```

Regarding the actions, we can have multiple buttons, but we chose to have a single info button that displays a sweet alert with an info message when clicked. We passed the function to do it via ***func***, the classes for the button via ***class***, the displayed text in ***title*** and the parameters used in the function.

```
customerButtonSettings: ButtonSettings[] = [
  {
    title: 'Info',
    class: ['btn', 'btn-info'],
    params: ['name', 'totalSpent', 'email', 'phone'],
    func: (name: string, totalSpent: number, email: string, phone: string) => {
      Swal.fire("Customer info", `Customer name: ${name} spent $${totalSpent}. Email: ${email}. Phone: ${phone}`, 'info');
    },
  },
];
```

As said, we can display custom data in our tables. We chose to define an interface for the customers and also we present you some records.

```
interface Customer {
  name: string;
  email: string;
  phone: string;
  loyaltyReward: boolean;
  totalSpent: number;
  id?: number;
}
```

```
customers: Customer[] = [
  {
    name: 'Bogdan Groza',
    email: 'bogdan@invalid.com',
    phone: '0744111222',
    loyaltyReward: true,
    totalSpent: 400,
  },
  {
    name: 'Debora Moisi',
    email: 'debora@invalid.com',
    phone: '0788999666',
    loyaltyReward: true,
    totalSpent: 450000,
  },
```

In the template we used the custom tag with the properties declared in the class component as inputs.

```
<app-custom-table
  [title]="'Customers'"
  [records]="customers"
  [settings]="customerTableSettings"
  [buttons]="customerButtonSettings"
  [recordsPerPage]=5
></app-custom-table>
```