Data Grid

6.1 Data Binding

6.1.1 Simple Array

Data binding can be done by using a simple array.With the help of ArrayStore that helps for all operations like get, insert, remove, update, etc.

Data Grid

```
let commentsDataSource = new DevExpress.data.DataSource({
    store: {
      type: "array",
      data: commentsData,
      key: "id",
    },
  });

  $("#dataBindingDG").dxDataGrid({
    dataSource: commentsDataSource,
    keyExpr: "id",
  });
```

Data

```
let commentsData = [
  {
    userId: 1,
    id: 1,
    title:
      "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
    body: "quia et suscipit\suscipit recusando consequuntur expedia et cum\reprehenderit molestiae ut ut quan totam\nostrum rerum est autem sunt rem eveniet arquitecto",
  },
  {
    userId: 1,
    id: 2,
    title: "qui est esse",
    body: "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla",
  },
  {
    userId: 1,
    id: 3,
    title: "ea molestias quasi exercitationem repellat qui ipsa sit aut",
```

```
    body: "et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut ad\voluptatem
doloribus vel accusantium quis pariatur\nmolestiae porro eius odio et labore et velit aut",
  },
  {
    userId: 1,
    id: 4,
    title: "eum et est occaecat",
    body: "ullam et saepe reiciendis voluptatem adipisci\nsit amet autem assumenda
provident rerum culpa\nquis hic commodi nesciunt rem tenetur doloremque ipsam iure\nquis
sunt voluptatem rerum illo velit",
  },
  {
    userId: 1,
    id: 5,
    title: "nesciunt quas odio",
    body: "repudiandae veniam quaerat sunt sed\nalias aut fugiat sit autem sed
est\voluptatem omnis possimus esse voluptatibus quis\nest aut tenetur dolor neque",
  },
];
```

6.1.2 Using Ajax Request

Data binding can also be bind to the data grid using jQuery ajax requests with the help of CustomStore or any other Data Layer stores.

DataGrid

```
  let commentsDataSource = new DevExpress.data.CustomStore({
    load: function () {
      return $.ajax({
        type: "GET",
        url: "https://jsonplaceholder.typicode.com/posts",
        dataType: "json",
        success: function (response) {
          return response;
        },
        error: function (error) {
          return error;
        },
      });
    },
    errorHandler: function (error) {
      console.log("Error", error);
    },
  });

  $("#dataBindingDG").dxDataGrid({
```

```
    dataSource: commentsDataSource,
  });
```

6.2. Paging & Scrolling

Paging helps the server to send the data into pages for better storage and time complexity. If the server sends all data to the user then that data user doesn't want all data it needs only a little starting data so paging helps the server to reduce extra work. Scrolling allows a user to browse data left outside the current viewport. Scrolling can be done in 3 ways :- standard, infinite, virtual. useNative property is used to enable the scrollByThumb, showScrollBar, etc. property. If you want to enable scrollByThumb then false the useNative.

6.2.1 Record Paging

Record paging can be done using the help of pager and paging properties of the data grid.

```
paging: {
        enabled: true,
        //pageIndex: 0,
        pageSize: 10,
},

pager: {
        // displayMode: "full", // compact
        visible: true,
        showPageSizeSelector: true,
        allowedPageSizes: [10, 20, 50],
        showInfo: true,
        showNavigationButtons: true,
        infoText: "Page #{0} - Total: {1} Pages. ({2} items)",
},
```

6.2.2. Virtual Scrolling

Rows are loaded when they get into the viewport and removed once they leave it. If the rows take time to be loaded and rendered, they display gray boxes. Rendering optimization can reduce rendering time and remove gray boxes. In this mode, users can move to any page instantly.

```
scrolling: {
     mode: 'virtual',
},
```

6.2.3 Infinite Scrolling

Each next page is loaded once the scrollbar reaches the end of its scale. In this mode, users scroll data gradually from the first to the last page.

```
scrolling: {
```

```
    mode: 'infinite',
},
```

6.3 Editing

Editing allows the user to add, delete, update operations on the data grid. There are many modes that you can use for editing.

- allowAdding
- allowDeleting
- allowUpdating
- confirmDelete
- editColumnName
- editRowKey
- form
- mode
- popup
- refreshMode
- selectTextOnEditStart
- startEditAction
- texts
- useIcons

Demo

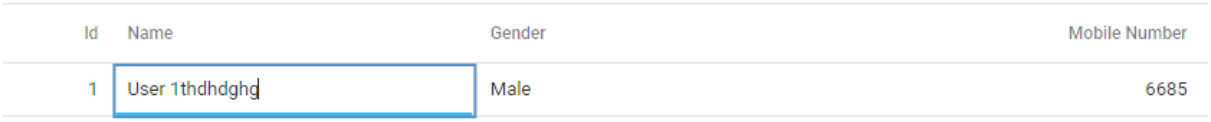
```
editing: {
      allowAdding: true,
      allowDeleting: true,
      allowUpdating: true,
      confirmDelete: true,
      mode: "popup",
      popup: {
        title: "Student Form",
        showTitle: true,
      },
      form: {
        items: [
          {
            itemType: "group",
            caption: "Personal Data",
            items: ["U01F02", "U01F03", "U01F06"],
          },
          {
            itemType: "group",
            caption: "Contact Details",
            items: ["U01F04", "U01F05"],
          },
        ],
```

```
  },
  useIcons: true,
},
```

6.3.1 Row Editing & Editing Events

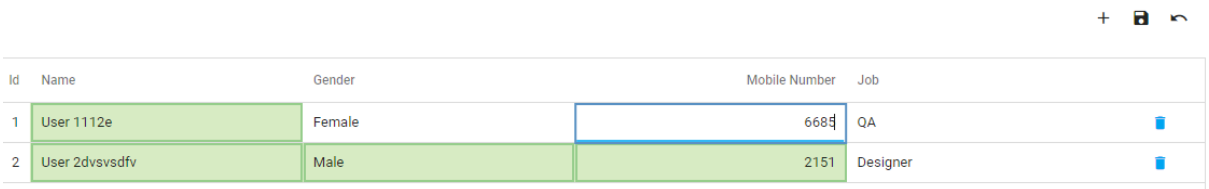
In Row editing a user edits one row at a time. The UI component saves changes when the row leaves the editing state.

- Editing Events
  - rowInserted
  - rowInserting
  - rowPrepared
  - rowRemoved
  - rowRemoving
  - rowUpdated
  - rowUpdating
  - rowValidating



6.3.2 Batch editing

A user edits data cell by cell. The UI component does not save changes until a user clicks the global "Save" button.



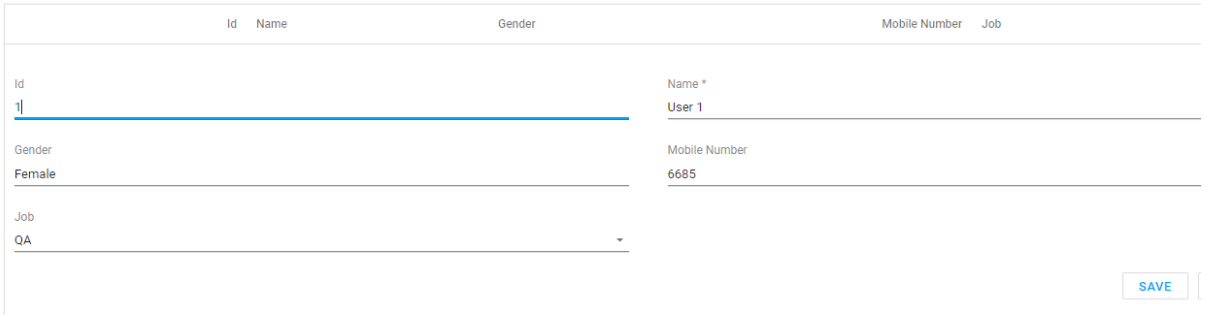
6.3.3 Cell Editing

Differs from the batch mode in that the UI component saves changes when the cell leaves the editing state.

Id	Name	Gender	Mobile Number
1	User 1	Female	6685
2	User 2fbdvb	Male	9493
3	User 3bffb	Female	545645
4	User 4	Female	1180

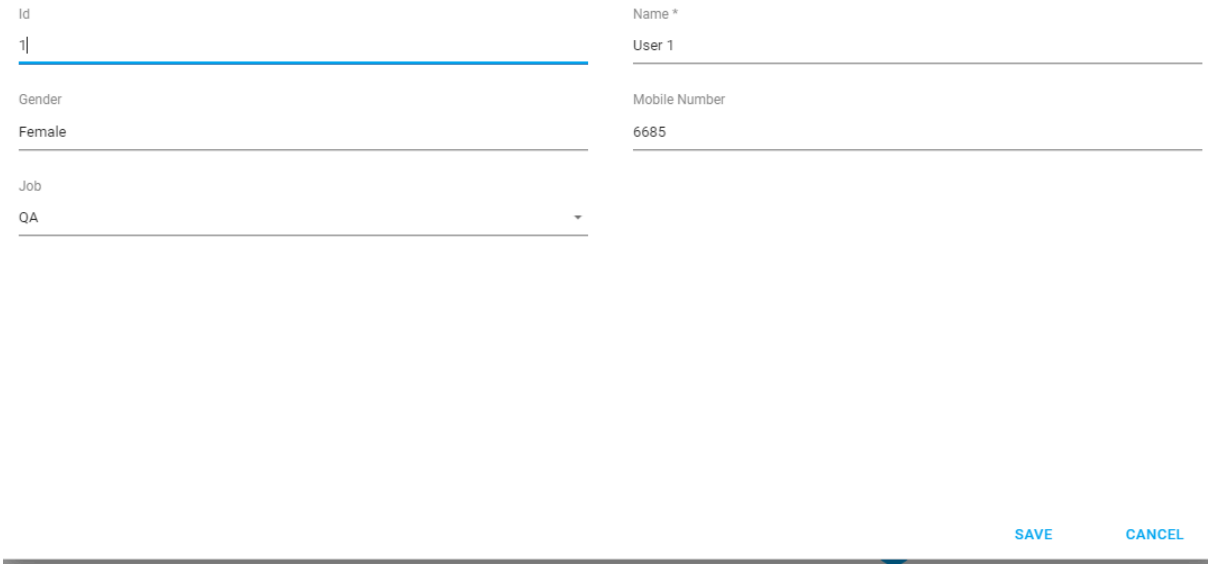
6.3.4 Form Editing

On entering the editing state, a row becomes a form with editable fields. The UI component saves changes after a user clicks the "Save" button.



6.3.5 Popup Editing

Differs from the form mode in that the form with editable fields is placed in a popup window.



6.3.6 Data Validation

Data Validation is applying validationRules on the column of the dataGrid and it validates data according to the rules.

```
columns: [
      {
        dataField: "id",
        caption: "Id",
      },
      {
        dataField: "name",
        caption: "Name",
        validationRules: [
          {
            type: "required",
```

```
        },
      ],
    },
    {
      dataField: "gender",
      caption: "Gender",
      validationRules: [
        {
          type: "custom",
          validationCallback: function (e) {
            return genderArr.includes(e.value);
          },
        },
      ],
    },
    {
      dataField: "number",
      caption: "Mobile Number",
    },
    {
      dataField: "jobId",
      caption: "Job",
      },
    },
  ],
```

6.3.7 Cascading Lookups

A lookup column restricts the set of values that can be chosen when a user edits or filters the column. In a lookup column, each cell is a drop-down menu. You can use a lookup column when you need to substitute displayed values with required values.

```
{
      dataField: "jobId",
      caption: "Job",
      lookup: {
        dataSource: jobCustomStore,
        valueExpr: "id",
        displayExpr: "jobName",
       },
},
```

6.4 Grouping

Grouping allows the user to group the data according to the same values for better finding operations. In Datagrid grouping can be enabled using the grouping attribute. Grouping property have following properties

- allowCollapsing
- autoExpandAll
- contextMenuEnabled
- expandMode :- 'buttonClick', 'rowClick' (mobile devices)
- texts- specify the group text for the grouping use in the context menu.

Group Panel
It is placed above datagrid for better grouping experience of the data grid. It has following properties :-

- allowColumnDragging
- emptyPanelText
- visible

6.5 Filtering

Filtering allows the user to get the data according to the filter value and conditions. Filters are different for number, string, date, etc. Filtering can be enabled in different ways.

Filter Builder

Opens a popup that user can create a user according condition based filter for the datagrid. And it can be configured in this way.

```
filterBuilder: {
      groupOperationDescriptions: {
        and: "&",
      },
    },
    filterBuilderPopup: {
      position: {
        of: window,
        at: "top",
        my: "top",
        offset: { y: 10 },
      },
      width: "500px",
      height: "300px",
    },
```

Header Filter

Header filter enables the header filter property. By using that user can filter the data using header row.

```
headerFilter: {
      visible: true,
      allowSearch: true,
```

```
    searchTimeout: 0,
  },
```

Filter Panel

Filter panel position is below the datagrid and above the pager. It helps the user to show filterBuilder for handling multiple filter conditions in one place. Filter panel can be enabled by this way.

```
filterPanel: {
    visible: true,
    filterEnabled: true,
    texts: {
      createFilter: "Create Custom Filter",
    },
  },
```

Filter Row

Filter row is the first row after which the header row configures the filter expression of value for the given columns according to the condition specified by the user. For enabling filterRow enable the visible property of the filterRow.

```
filterRow: {
    visible: true,
    applyFilter: "auto",
  },
```

6.6 Sorting

Sorting helps the user to sort the data alphabetically, numeric, and date wise ascending or descending order. Sorting has two types: single and multiple. And if you need to disabled sorting for the specific column then use allowSorting: false. Sorting can be configured in this way.

```
sorting: {
    mode: "multiple",
    showSortIndexes: true,
  },
```

6.7 Selection

Selection is used for exporting the data which rows are selected and all rows.

```
selection: {
    mode: "multiple",
```

```
  selectAllMode: "allPages",
  showCheckBoxesMode: "always",
},
```

- clearSelection() - Clears selection of all rows on all pages.

6.8 Columns

An array of grid columns that helps to show the data to the user.

6.8.1 Column Customization

Column can be customized using the columns[] in that it takes the column object in that you configure it. Column have this properties for the customization

- alignment
- allowEditing
- allowExporting
- allowFiltering
- allowFixing
- allowGrouping
- allowHeaderFiltering
- allowHiding
- allowReordering
- allowResizing
- allowSearch
- allowSorting
- autoExpandGroup
- buttons
- calculateCellValue
- calculateDisplayValue
- calculateFilterExpression
- calculateGroupValue
- calculateSortValue
- caption
- cellTemplate
- columns
- cssClass
- customizeText
- dataField
- dataType
- editCellTemplate
- editorOptions
- encodeHtml
- falseText
- filterOperations
- filterType
- filterValue
- filterValues

- fixed
- fixedPosition
- format
- formItem
- groupCellTemplate
- groupIndex
- headerCellTemplate
- headerFilter
- hidingPriority
- isBand
- lookup
- minWidth
- name
- ownerBand
- renderAsync
- selectedFilterOperation
- setCellValue
- showEditorAlways
- showInColumnChooser
- showWhenGrouped
- sortIndex
- sortingMethod
- sortOrder
- trueText
- type
- validationRules
- visible
- visibleIndex
- width

6.8.2 Column based on a data source

If you don’t specify the columns field in the data grid it automatically set the data grid columns using the response it gets using data source.

6.8.3 Multi Row Headers

Multi row headers can be created or implemented by specifying the columns property in other columns.

```
{
        caption: "Name",
        columns: [
         {
           dataField: "firstName",
           caption: "First Name",
           dataType: "string",
```

```
      width: 100,
      allowResizing: false,
      allowReordering: false,
      customizeText: function (cellInfo) {
        return cellInfo.value + " $";
      },
    },
    {
      dataField: "lastName",
      caption: "Last Name",
      visible: true,
      minWidth: 50,
    },
  ],
},
```

6.8.4 Column Resizing

Column resizing can be easily done using the properties of the datagrid. Just enable the allowColumnResizing property to true value then it will resize the columns. And it can be done in two ways:- nextColumn, and widget. If nextColumn customResizingMode is enabled then it will change the size of the next column or neighbor column. In widget mode all the UI component's column size changed.
And for the resizing disabled of the specific column choose allowResizing: false in that column properties.

6.8.5 Command Column Customization

The DataGrid provides the following command columns:

Adaptive column
Contains ellipsis buttons that expand/collapse adaptive detail rows. Appears if columnHidingEnabled is true or hidingPriority is set for at least one column and only when certain columns do not fit into the screen or container size.

Selection column
Contains checkboxes that select rows. Appears when selection.mode is "multiple" and showCheckBoxesMode is not "none".

Group expand column
Contains arrow buttons that expand/collapse groups.

Detail expand column
Contains arrow buttons that expand/collapse detail sections.

Button column (custom command column)

Contains buttons that execute custom actions. See Create a Column with Custom Buttons.

Edit column
A Button column pre-populated with edit commands. See Customize the Edit Column.

Drag Column
Contains drag icons. Appears when a column's type is "drag", and the allowReordering and showDragIcons properties of the rowDragging object are true.

Command Column can be configure in this way:-

```
 columns: [{
        type: "selection",
        cellTemplate: function ($cellElement, cellInfo) {
            // Render custom cell content here
        },
        headerCellTemplate: function ($headerElement, headerInfo) {
            // Render custom header content here
        }
    }]
```

Customize the buttons

```
columns: [{
        type: "buttons",
        buttons: [{
            name: "save",
            cssClass: "my-class"
        }, "edit", "delete"]
    }]
```

Add Custom Button

```
columns: [{
        type: "buttons",
        buttons: ["edit", "delete", {
            text: "My Command",
            icon: "/url/to/my/icon.ico",
            hint: "My Command",
            onClick: function (e) {
                // Execute your command here
            }
        }]
    }]
```

6.9 State Persistence

State persistence is a way to store the all sorting, filter, paging and all configuration to localstorage, session storage, or any other server side. It can be achieved by using this way

```
stateStoring: {
      enabled: true,
      type: "localStorage", // sessionStorage
      storageKey: "storage",
      savingTimout: 200,
    },
```

And if you are using custom then you need to implement the customLoad and customSave method.

6.10 Appearance

Appearance of a data grid can be done by 4 properties

- showBorders
- showColumnLines
- showRowLines
- rowAlternationEnabled

6.11 Template

6.11.1 Column Template

Column template can be done by specifying the properties and format etc on the columns[] property.

```
{
    dataField: "id",
    dataType: "numer",
    caption: "Id",
  },
  {
    dataField: "profilepic",
    caption: "Profile Pic",
    dataType: "string",
    cellTemplate(container, option) {
      const { value } = option;
      $("<div>")
        .append(
          $("<img>", { src: value, style: "hieght: 50px; width: 50px" })
        )
        .appendTo(container);
    },
  },
  {
```

```
    dataField: "name",
    dataType: "string",
    caption: "Name",
  },
  {
    dataField: "gender",
    dataType: "string",
    caption: "Gender",
  },
```

6.11.2 Row Template

Row template can be done by using the rowTemplate method of the grid. It takes two arguments. The first one is a container and the second one is an item that has the data.

```
rowTemplate(container, item) {
    const { data } = item;
    const markup =
      "<tr>" +
      `<td>${data.id}</td>` +
      `<td><img src='${data.profilepic}' style="height: 50px; with: 50px;" />
${data.name}</td>` +
      `<td>${data.gender}</td>` +
      `<td>${data.city}</td>` +
      `<td>${data.state}</td>` +
      "</tr>";
    container.append(markup);
  },
```

6.11.3 Cell Template

Cell template is generally used to modified the cell of the data grid according to the value of the column and we can add the chart, or any other things according to the data.

```
{
    dataField: "percentage",
    dataType: "number",
    caption: "Passing Percentage",
    cellTemplate(container, options) {
      let color = options.value * 10 > 33 ? "green" : "red";
      $("<div>")
        .dxBullet({
          value: options.value * 10,
          target: 33,
          color: color,
          startScaleValue: 0,
          endScaleValue: 100,
```

```
        showTarget: true,
        showZeroLevel: true,
        tooltip: {
          enabled: true,
          format: "fixedPoint",
          customizeTooltip: function (arg) {
            return {
              text:
                "Value: " + arg.value + "%\nTarget: " + arg.target + "%",
            };
          },
        },
      })
      .appendTo(container);
  },
},
```

6.11.4 Toolbar Customization

Toolbar can be customized on the onToolbarPreparing method that has arguments which contains the current toolbar options. And it can have a button, menu, search box, etc. and it can be configured in this way.

```
let toolbarItems = e.toolbarOptions.items;

    toolbarItems.push({
      location: "center",
      locateInMenu: "never",
      template() {
        return $(
          "<div class='toolbar-label'><b>Deep's Club</b> Products</div>"
        );
      },
    });

    toolbarItems.push({
      location: "before",
      widget: "dxButton",
      options: {
        icon: "refresh",
        onClick() {
          window.location.reload();
        },
        hint: "Refresh Page",
      },
    });
```

```
toolbarItems.push({
  location: "after",
  widget: "dxButton",
  locateInMenu: "always",
  options: {
    text: "Google",
    onClick() {
      window.location.assign("https://google.com");
    },
  },
});

toolbarItems.push({
  location: "before",
  widget: "dxButton",
  locateInMenu: "always",
  options: {
    text: "Youtube",
    onClick() {
      window.location.assign("https://youtube.com");
    },
  },
});
```

6.12 Summaries

Summaries help the user to find avg, max, min, etc aggregate values of the data. There are 3 types of summaries:- total, group and custom.

6.12.1 Grid Summaries

It can be configured using the totalItems[] property of the summary. In that it have following properties

- alignment
- column
- cssClass
- customizeText
- displayFormat
- name
- showInColumn
- skipEmptyValues
- summaryType
- valueFormat

```
totalItems: [
    {
      column: "ID",
      summaryType: "count",
```

```
    },
    {
      column: "SaleAmount",
      summaryType: "sum",
    },
    {
      column: "TotalAmount",
      summaryType: "max",
    },
  ],
```

6.12.2. Group Summary

Group summary helps the user to get the aggregate values according to the groups. It gives us the values of the group items. It can be configured by following properties

- alignByColumn
- column
- customizeText
- displayFormat
- name
- showInColumn
- showInGroupFooter
- skipEmptyValues
- summaryType
- valueFormat

6.12.3 Custom summaries

Custom summary is useful for the user to apply his custom logic. For implementing this type of summary use custom in summaryType and specify name property also.
Custom summary is calculated in calculateCustomSummary method.

```
groupItems: [
    {
      column: "CustomerStoreCity",
      summaryType: "custom",
      name: "customerStoreCity",
    },
  ],

calculateCustomSummary(options) {
    if (options.name == "customerStoreCity") {
      let count;
      switch (options.summaryProcess) {
        case "start":
          options.totalValue = 0;
          break;
```

```
        case "calculate":
          if (options.component.isRowSelected(options.value.ID)) {
            options.totalValue += calculateArea(options.value);
          }
          break;
        case "finalize":
          break;
      }
    }
  },
```

6.13 Master Detail

In DataGrid, a master-detail interface supplies a usual data row with an expandable section that contains the details on this data row. In that case, the data row is called "master row", while the section is called "detail section".
For implementing master-detail in the datagrid enabled the masterDetail property of the datagrid which have other properties.

- autoExpandAll - If this property's value is true then it opens all the detail section of the master
- Enabled: it enabled the master detail for the datagrid.
- Template method takes two arguments first one is container and second one is items. After that it gets the data and append that data to the container of the master.

```
masterDetail: {
    enabled: true,
    autoExpandAll: false,
    template(container, options) {
      const currentUserData = options.data;

      $("<div>")
        .addClass("master-detail-header")
        .text(`${currentUserData.name}'s Orders:- `)
        .appendTo(container);

      $("<div>")
        .dxDataGrid({
          dataSource: new DevExpress.data.CustomStore({
            load: function () {
              return $.ajax({
                type: "GET",
                url: `${ordersUrl}/${options.data.id}`,
                dataType: "json",
              });
            },
          }),
          key: "id",
```

```
        })
        .appendTo(container);
    },
```

6.14 Export

Exporting the data of the datagrid. It exports all the data and selected data. Exporting of the selected data can be done in two ways:- pdf and excel.

```
    export: {
      enabled: true,
      allowExportSelectedData: true,
    },
```

6.14.1 PDF Exporting

Pdf exporting can be done by adding this cdn to the head of the html page.

- <script src="https://cdnjs.cloudflare.com/ajax/libs/jszip/3.7.1/jszip.min.js"></script>
- <script src="https://cdnjs.cloudflare.com/ajax/libs/jspdf/2.5.1/jspdf.umd.min.js"></script>
- <script src="https://cdnjs.cloudflare.com/ajax/libs/jspdf-autotable/3.5.24/jspdf.plugin.autotable.min.js"></script>

For the pdf exporting can be done in this way.

```
window.jsPDF = window.jspdf.jsPDF;

onExporting: function (e) {
 const doc = new jsPDF();

  DevExpress.pdfExporter
    .exportDataGrid({
      jsPDFDocument: doc,
      component: e.component,
      indent: 5,
    })
    .then(() => {
      doc.save("StudentsMasterData.pdf");
    });
 },
```

6.14.2 Excel Exporting

For excel exporting add this cdn files :-

- <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-polyfill/7.4.0/polyfill.min.js"></script>
- <script src="https://cdnjs.cloudflare.com/ajax/libs/exceljs/4.1.1/exceljs.min.js"></script>
- <script src="https://cdnjs.cloudflare.com/ajax/libs/FileSaver.js/2.0.2/FileSaver.min.js"></script>

Excel exporting function :-

```
function excel(e) {
  var workbook = new ExcelJS.Workbook();
  var worksheet = workbook.addWorksheet("Main sheet");
  DevExpress.excelExporter
    .exportDataGrid({
      worksheet: worksheet,
      component: e.component,
      customizeCell: function (options) {
        var excelCell = options;
        excelCell.font = { name: "Arial", size: 12 };
        excelCell.alignment = { horizontal: "left" };
      },
    })
    .then(function () {
      workbook.xlsx.writeBuffer().then(function (buffer) {
        saveAs(
          new Blob([buffer], { type: "application/octet-stream" }),
          "StudentsMasterData.xlsx"
        );
      });
    });
  e.cancel = true;
}
```

6.15 Adaptability

Adaptability can be achieved by enabling the columnHidingEnabled property. Adaptability helps to show only data which is necessary when the user opens the website in mobile phone. It generally hides the column by checking the hidingPriority property of the columns.

```
    columnHidingEnabled: true,

{
  dataField: "U01F06",
  dataType: "number",
  caption: "City",
```

```
  alignment: "right",
  hidingPriority: 0,
  lookup: {
    allowClearing: true,
    dataSource: CityCustomStore(),
    valueExpr: "Y01F01",
    displayExpr: "Y01F02",
  },
},
```