

# Write Applications Fast Using Ignite UI Grid

WRITE WEB APPLICATIONS FASTER with Ignite UI. You can use the Ignite UI library to help quickly solve complex LOB requirements in HTML5, jQuery, Angular, React, or ASP.NET MVC. Use the Ignite UI library to add a fast, responsive grid with many features (like pagination, sorting, search, virtualization etc.). It takes just a few minutes using a few lines of code. Ignite UI has many controls, data visualizations charts, and framework elements that are simple to configure and customize. The ease of Ignite UI control configurations and customizations allows you to create a web application quickly.

In addition to seamlessly rendering large sets of data, the Ignite UI Grid features many valuable tools, such as filtering, paging, and sorting. You can learn more about Ignite UI features at <http://www.igniteui.com>; you can also learn more about Angular in *Angular Essentials*, a free eBook published by Infragistics.

## Lesson Objectives

---

1. Add Ignite UI grid
2. Configure grid columns

For more information on the controls used in this lesson, see <http://infragistics.com/products/igniteui/grids/data-grid>.

At the end of this lesson, you will have a working grid configured for columns in an Angular application.

You can learn more about Ignite UI Angular 2 here: <https://github.com/igniteui/igniteui-angular2>.

## Setting up the Project

You may download the starter project for this lesson [by clicking here](#). (You can also download the final project [by clicking here](#).)

After downloading the project, navigate to the directory and run the commands below:

```
npm install
npm start
```

You have executed the npm install command to install all dependencies, using the npm start command to run the Angular application. If the project setup is correct, you will have a running Angular application as shown in the image below:



### STEP 1: Import and Declare the Component

To work with Ignite UI Angular components, you must import and declare them in the module. For example, to use the igGrid component in an Angular application, import and declare the IgGridComponent in the application module.

In the project, navigate to the Finance App folder and then to the app folder. Open the file app.module.ts, and add the import statements below, just after the existing import statements.

```
import {IgGridComponent} from 'igniteui-angular2';
import {GridComponent} from './grid.component';
```

After importing the required components, you must declare them in the application module. Add IgGridComponent and GridComponent in the AppModule's declaration array. Modify @NgModule decorator in app.module.ts as shown below:

```
@NgModule({
  imports: [BrowserModule,HttpModule],
  declarations: [AppComponent,
    IgZoombarComponent,
    IgDataChartComponent,
    PriceChartComponent,
    InfoComponent,
    IndicatorChartComponent,
    VolumeChartComponent,
    IgGridComponent, GridComponent],
  providers: [AppService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

You have added `IgGridComponent` and `GridComponent` in the declaration array of `AppModule` module. We will examine other added components and properties (like providers) in subsequent lessons.

### STEP 2: Create a Data Source

You need data to bind to the grid. This data can be a JavaScript array or a JSON object array and can be local or provided by a REST service.

Ideally, you should create a function to return data in an Angular service so you can use the data function in multiple components. However, for this lesson, there is already a function called `getData` in `GridComponent` class. This function returns a JSON object array.

In the app folder, open the file `grid.component.ts` and find the `getData()` function. In later lessons, you will learn how to create a grid that uses data from the REST services.

### STEP 3: Get data

To use data returned from the `getData()` function, call the function inside Angular `ngOnInit()` life cycle hook and assign a returned value to the `GridComponent` property.

Learn more about Angular Life Cycle hooks here: <https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html>

In the app folder, open the file `grid.component.ts` and modify the `ngOnInit()` function as shown in the listing below:

```
ngOnInit(){
  this.stocks = this.getData();
}
```

---

#### STEP 4: Create a Grid

---

The Ignite UI Grid component can be used like any other component. In the app folder, open the file `grid.component.html`, and add the code as shown in the below listing:

```
<ig-grid widgetId="grid1" [dataSource]="stocks" [autoGenerateColumns]="true">
```

#### STEP 5: Use in an Application

---

To use the GridComponent in an application: in the app folder, open the `app.component.html` file and add the code below just at the end of all of the markup. Add it below the `<br/>` element.

```
<grid></grid>
```

Navigate to the application, scroll down, and, at the bottom of the page, you will find the grid added as shown in the image below:

Close	Date	High	Low	Open	Volume
22.34	Thu Jan 19 2017	22.6294	22.24	22.567584	25815932
22.288	Wed Jan 18 2017	22.48	22.048092	22.39343	42058903
21.9729	Tue Jan 17 2017	22.3991	21.96	22.109	32174501
22.12	Mon Jan 16 2017	22.40	21.901	22.2264	24456118
21.7834	Sun Jan 15 2017	22.1709	21.77	21.99839	23880247
21.6749	Thu Jan 12 2017	22.045677	21.575	22.044	21213381
21.5788	Wed Jan 11 2017	21.8817	21.476614	21.666788	29849803
21.3825	Tue Jan 10 2017	21.41007	21.3528	21.405875	19956106
21.32	Mon Jan 09 2017	21.5251	21.2728	21.332464	25611918
21.417	Sun Jan 08 2017	21.71433	21.2	21.5138	40663789

#### STEP 6: Configure Columns of the Grid

---

In Step 4, you created a grid by setting the `autoGenerateColumns` property to `true`. You didn't need to configure the columns of the grid and they were generated automatically.

In many cases you may need to configure columns manually. You can configure columns and other features such as paging, sorting, and filtering of the grid in the component class.

To configure columns: in the app folder, open `grid.component.ts` file, and update `ngOnInit()` function in `grid.component.ts` file with the listing below:

```
ngOnInit() {  
  this.stocks = this.getData();  
  this.gridId = "Grid1";  
}
```

```

this.gridOptions = {
  dataSource: this.stocks,
  autoGenerateColumns: false,
  columns: [
    { headerText: "CLOSE", key: "Close", dataType: "number" },
    { headerText: "DATE", key: "Date", dataType: "string" },
    { headerText: "HIGH", key: "High", dataType: "number" },
    { headerText: "LOW", key: "Low", dataType: "number" },
    { headerText: "OPEN", key: "Open", dataType: "number" },
    { headerText: "VOLUME", key: "Volume", dataType: "number" }
  ]
}
}
}

```

### STEP 7: Modify The Grid With Configured Columns

Ignite UI grid options and widgetId properties are enabled for two-way data binding, so any changes in the source will be reflected on the grid. To set options and widgetId properties: in the app folder, open the file grid.component.html, and modify it as shown in the below listing:

```
<ig-grid [(options)]="gridOptions" [(widgetId)]="gridId">
```

Navigate to the application and scroll to the bottom of the page to find the grid added as shown below:

CLOSE	DATE	HIGH	LOW	OPEN	VOLUME
22.34	Thu Jan 19 2017	22.6294	22.24	22.567584	25815932
22.288	Wed Jan 18 2017	22.48	22.048092	22.39343	42058903
21.9729	Tue Jan 17 2017	22.3991	21.96	22.109	32174501
22.12	Mon Jan 16 2017	22.4	21.901	22.2264	24456118
21.7834	Sun Jan 15 2017	22.1709	21.77	21.99839	23880247
21.6749	Thu Jan 12 2017	22.045677	21.575	22.044	21213381
21.5788	Wed Jan 11 2017	21.8817	21.476614	21.666788	29849803
21.3825	Tue Jan 10 2017	21.41007	21.3528	21.405875	19956106
21.32	Mon Jan 09 2017	21.5251	21.2728	21.332464	25611918
21.417	Sun Jan 08 2017	21.71433	21.2	21.5138	40663789

## Conclusion

Ignite UI can help you write web applications more quickly. In addition to Angular, Ignite UI may be used with React, AngularJS, jQuery, and ASP.NET MVC.

# Write Applications Fast Using Ignite UI Data Charts

Write web applications and solve complex LOB requirements more quickly with Ignite UI. The Ignite UI library (with HTML5, jQuery, Angular, React, or ASP.NET MVC) can add complex and dynamic charts to your web application quickly with a few lines of code.

Different types of charts are available in the Ignite UI:

- **Data Chart:** Display data on x-axis and y-axis as bars, lines, areas etc.
- **Pie Chart:** Display data in a circle, divided into sectors that each represent a proportion of the total data.
- **Doughnut Chart:** Display data in a circle, with more than one data series.

There are approximately 50 types of data charts available in Ignite UI. Learn more about Ignite UI data charts here: <http://www.igniteui.com/data-chart/overview>; you can also learn more about Angular in *Angular Essentials*, a free eBook published by Infragistics.

## Lesson Objectives

---

1. Add Ignite UI DataChart
2. Configure data charts for axes, data sources, and series
3. Configure data charts for various series types.

For more information on the controls used in this lesson, see <http://www.infragistics.com/products/igniteui/charts/data-chart>.

At the end of the lesson, you will have a working data chart configured for different types of series in an Angular application.

Learn more about Ignite UI Angular 2 here: <https://github.com/igniteui/igniteui-angular2>

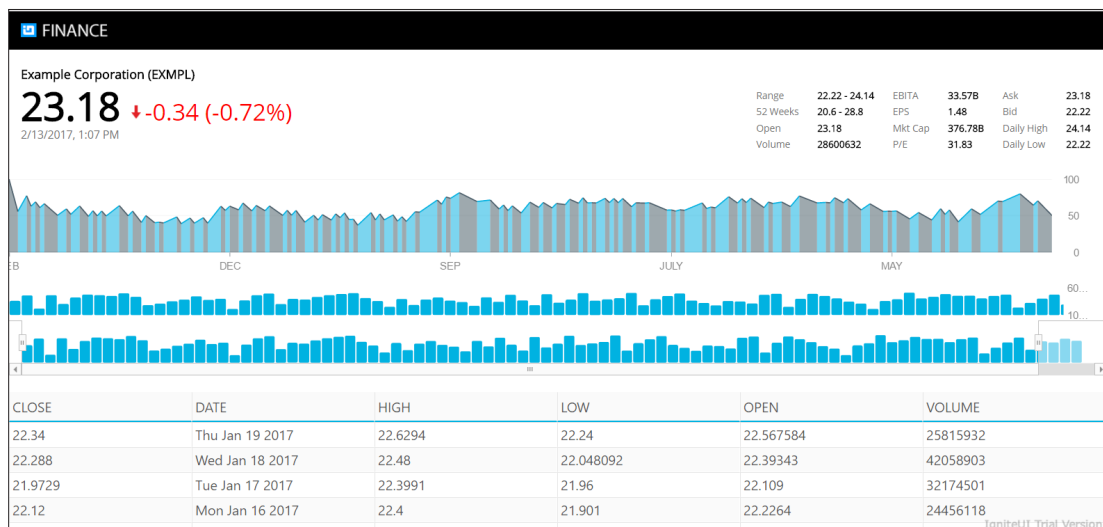
## Setting up the Project

Download the starter project for this lesson *by clicking here*. (You can also download the final project *by clicking here*.)

After you download the project, navigate to the Finance App directory and run the commands below:

```
npm install
npm start
```

You have executed the npm install command to install all dependencies and are using the npm start command to run the Angular application. If the project setup is correct, you will have a running Angular application as shown below. If you receive an error while running the application, stop and run the npm start command again.



### STEP 1: Import and Declare the Component

To work with Ignite UI Angular components, you must import and declare them in the module. For example, to use the `igDataChart` component in an Angular application, import and declare `IgDataChartComponent` in the application module.

In the project, navigate to the Finance App folder, and then the app folder. Open the file `app.module.ts`, and you will find that `igDataChartComponent` has been added. Add the import statements below, after the existing import statements.

```
import {PriceChartComponent} from './charts/pricechart.component';
```

---

After importing the required components, you must declare them in the application module. Add PriceChartComponent in the AppModule's declaration array. Modify @NgModule decorator in app.module.ts as shown below:

```
@NgModule({
  imports: [BrowserModule, HttpClientModule],
  declarations: [AppComponent,
    IgZoombarComponent,
    IgDataChartComponent,
    InfoComponent,
    IndicatorChartComponent,
    VolumeChartComponent,
    IgGridComponent,
    GridComponent,
    PriceChartComponent],
  providers: [AppService],
  bootstrap: [AppComponent]
})
```

You've now added PriceChartComponent in the declaration array of AppModule module. Other added components and other properties like providers will be outlined in subsequent lessons.

### **STEP 2:** Create Data Source

---

The data needed to bind the data chart can be a JavaScript array or a JSON object array and can be local or be may be provided by a REST service.

Ideally, you should create a function to return data in the Angular service so data may function in multiple components. However, for this lesson, there is already a function called getData in PriceChartComponent class, which returns a JSON object array. In the app\charts folder, open the file pricechart.component.ts and find the getData() function. In future lessons, you will learn to create a grid which uses data from the REST services.

### **STEP 3:** Get Data

---

To use data returned from getData() function, call the function inside the Angular ngOnInit() life cycle hook and assign returned value to PriceChartComponent property.

Learn more about Angular Life Cycle hooks here: <https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html>

In the app\charts folder, open the file pricechart.component.ts and modify the ngOnInit() function as shown in the listing below:



```
ngOnInit() {
  this.stocks = this.getData();
}
```

#### STEP 4: Configure Axes

To create a data chart, you must configure the chart options. Usually, chart options consist of three main properties:

1. X and Y axis
2. Data source
3. Series

Aside from these properties, other important properties are height, width, title, etc.

To configure axes, open the `pricechart.component.ts` file and directly after the `ngOnInit()` function, add the `getPriceChartAxes()` function as shown listed below :

```
getPriceChartAxes() {
  return [
    {
      name: "xAxis",
      type: "categoryX",
      label: "Date"
    },
    {
      name: "yAxis",
      type: "numericY",
      labelLocation: "outsideRight",
      labelExtent: 40
    }
  ];
}
```

In the above listing:

- X-axis type and Y-axis type are useful to display financial, scatter, or category price series. Other possible values are category, numericAngle, categoryDateTimeX, categoryAngle etc.

You can learn about these values and types of charts here: <http://www.igniteui.com/help/igdatachart-series-types>

- Y-axis labelExtent value is set to 40, which specifies the size of the area dedicated to the labels or how far label would be from the axis.

#### STEP 5: Configure Series

An Ignite UI data chart can have any number of series, but it must have at least one series. To add a series in a data chart in the app\charts folder, open pricechart.component.ts file and directly after the getPriceChartAxes() function, add the getPriceChartSeries() function as shown in the listing below:

```
getPriceChartSeries() {  
    return [  
        {  
            name: "stockSeries",  
            type: "splineArea",  
            title: "Price Data",  
            isHighlightingEnabled: true,  
            isTransitionInEnabled: true,  
            xAxis: "xAxis",  
            yAxis: "yAxis",  
            valueMemberPath: "High",  
            showTooltip: true,  
            Outline: "#00AADE"  
        }  
    ];  
}
```

In the above listing:

- Series type value is set to splineArea to create Spline Area series. If you want to create a Line series, set the value of type to "line". IgniteUI provides more than 25 possible series types for the data chart including area, bar, and column.
- As a series valueMemberPath, you need to set property from the data array to be displayed in the chart. Here you are setting "High" property from the data source will be rendered in the data chart series.
- Series isTransitionInEnabled value is set to true to enable animation when data source is assigned.

#### STEP 6: Configure Chart Option

You have configured axis and series. Next, configure a chart option. In chart option, you set all other important properties of a data chart.

Learn more about chart properties here: <http://www.igniteui.com/data-chart/overview>

To configure a data chart, in app\charts folder, open pricechart.component.ts file and directly after getPriceChartSeries () function, add getPriceChartOption() function as shown in the listing below:

```
getPriceChartOptions() {
  return {
    axes: this.getPriceChartAxes(),
    series: this.getPriceChartSeries(),
    windowResponse: "deferred",
    horizontalZoomable: true,
    width: "100%",
    height: this.desiredHeight,
    leftMargin: 0,
    rightMargin: 30,
    windowRectMinWidth: 0.05,
    syncChannel: "channel1",
    synchronizeVertically: false,
    synchronizeHorizontally: false
  };
}
```

In the above listing:

- Chart's syncChannel property is set so the chart can be synced with other charts of the application to intimate functionalities of other controls such as ZoomBar. Charts synced in same channel can use single zoom bar for zoom in and zoom out functionalities.
- Chart's windowResponse property is set to "deferred" so the chart view update will defer until after the user action is complete. Another possible value is "immediate."

#### **STEP 7:** Initialize Chart Option and Data Source

To initialize chart option and data source, in pricechart.component.ts file, modify ngOnInit() function as shown in the listing below:

```
ngOnInit() {
  this.stocks = this.getData();
  this.desiredHeight = 0.22 * (window.screen.height) + "px";
  this.chartOptions = this.getPriceChartOptions();
}
```

## STEP 8: Create Chart

To create a chart, open `pricechart.component.html` file and the add markup given below:

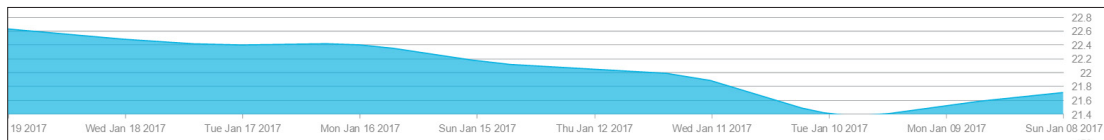
```
<ig-data-chart [(options)]="chartOptions" [(dataSource)]="stocks" widgetId="price-chart"></ig-data-chart>
```

## STEP 9: Use in Application

To use `PriceChartComponent` in an application, in the app folder, open `app.component.html` file and add below code just after the `<info-screen>` element and before `<indicatorchart>` element.

```
<pricechart></pricechart>
```

Navigate to the application, scroll down, and at the bottom of the page, you will find chart added as shown in the image below:



## Conclusion

Ignite UI is useful in writing web applications quickly. In addition to Angular, you can use Ignite UI in React, AngularJS, jQuery, and ASP.NET MVC. In this lesson, you learned how to use Ignite UI Data Charts in an Angular application.

# Sort, Filter, and Page Fast With Ignite UI Grid

Ignite UI enables you to write web applications faster. You can use Ignite UI library with HTML5, jQuery, Angular, React, or ASP.NET MVC. It helps you to solve complex LOB requirements faster. The Ignite UI library makes it possible for you to quickly and efficiently add a fast, responsive grid with features like pagination, sorting, search, virtualization, and more.

In addition to seamlessly rendering large sets of data, IgniteUI Grid is loaded with many other features, such as filtering, paging, and sorting. You can learn more about Ignite UI features at <http://www.igniteui.com>; you can also learn more about Angular in *Angular Essentials*, a free eBook published by Infragistics.

In this lesson, you will learn how to configure various important features of Ignite UI Grid.

## Lesson Objective

---

1. Enable sorting on the grid
2. Enable filters on the grid
3. Enable paging on the grid

For more information on the controls used in this lesson, see <http://infragistics.com/products/igniteui/grids/data-grid>.

At the end of this lesson, you will have an Ignite UI Grid configured for basic features such as sorting, filtering, and pagination in an Angular application.

You can learn more about Ignite UI Angular 2 here: <https://github.com/igniteui/igniteui-angular2>.

## Setting up the Project

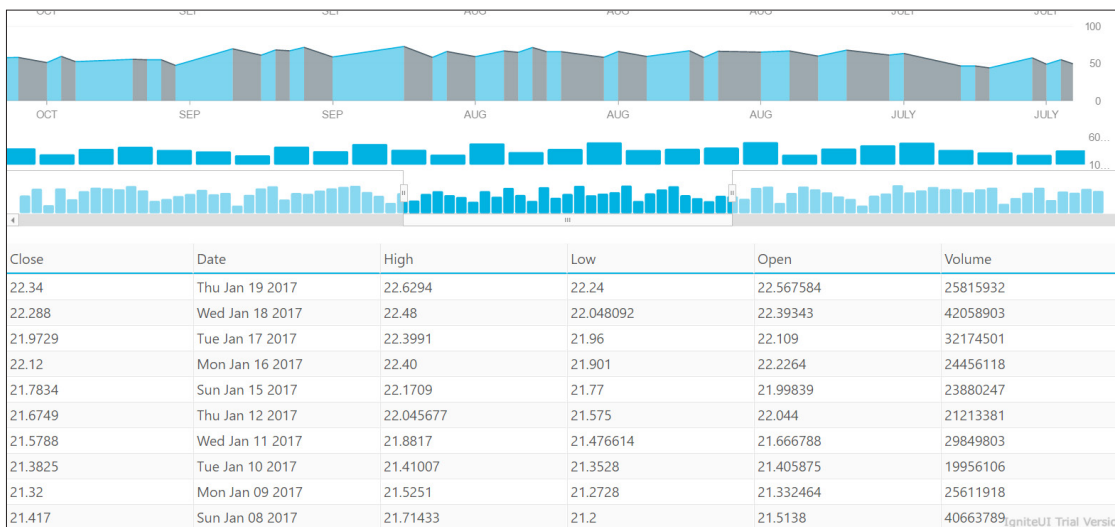
You can download the starting project for this lesson [by clicking here](#). (You can also download the final project [by clicking here](#).)

After downloading the project, navigate to Finance App directory and run the commands below:

```
npm install
npm start
```

You executed npm install command to install all dependencies, and to use the npm start command to run the Angular application. If the project setup is correct, you will have a running Angular application as shown in the image below.

Scroll to bottom of the application and you will find the Ignite UI Grid. At the end of the lesson, this grid will be configured with sorting, paging, and filter features.



### STEP 1: Enable Sorting

You can enable sorting on Ignite UI Grid by adding a feature with the name "Sorting" in the grid. Ignite UI Grid supports local and remote sorting.

To enable local sorting, create an object with the following properties and add it to the features property of the grid option:

- **name** : set to Sorting
- **type** : set to local

To do this, in the App folder, open the file `grid.component.ts` and add below `getGridFeatures()` function just after the `getData()` function.

```
getGridFeatures()
{
  return [
    {
      name: "Sorting",
      type: "local"
    }
  ];
}
```

Next, add sorting features to the grid options. For that in the `this.gridOptions`, add a new property called `feature` and set its value to `this.getGridFeatures()`. Updated `ngOnInit()` function in the `grid.component.ts` file will look like as shown in the code listed below:

```
ngOnInit() {
  this.stocks = this.getData();
  this.gridId = "grid1"
  this.gridOptions = {
    dataSource: this.stocks,
    autoGenerateColumns: true,
    features: this.getGridFeatures()
  }
}
```

Navigate to the application, scroll down, and at the bottom of the page, you will find the grid added as shown below:

Close	Date	High	Low	Open	Volume
21.3825	Tue Jan 10 2017	21.41007	21.3528	21.405875	19956106
21.6749	Thu Jan 12 2017	22.045677	21.575	22.044	21213381
21.7834	Sun Jan 15 2017	22.1709	21.77	21.99839	23880247
22.12	Mon Jan 16 2017	22.40	21.901	22.2264	24456118
21.32	Mon Jan 09 2017	21.5251	21.2728	21.332464	25611918
22.34	Thu Jan 19 2017	22.6294	22.24	22.567584	25815932
21.5788	Wed Jan 11 2017	21.8817	21.476614	21.666788	29849803
21.9729	Tue Jan 17 2017	22.3991	21.96	22.109	32174501
21.417	Sun Jan 08 2017	21.71433	21.2	21.5138	40663789
22.288	Wed Jan 18 2017	22.48	22.048092	22.39343	42058903

Click on any of the columns and you will find that the grid is sorted for that particular column as shown in the image above. In addition, you will notice that sorted column headers have sorting indicators applied, so sortable columns are distinguished visually

---

from the rest of the columns in the grid. Ignite UI also supports sorting on multiple columns.

You have configured sorting locally. Ignite UI also supports remote sorting. Learn more here: <http://www.igniteui.com/grid/sorting-remote>

## STEP 2: Enable Paging

---

You can enable paging on Ignite UI grid by adding a feature named "Paging" in the grid. Ignite UI Grid supports local and remote pagination.

To enable local paging, create an object with following properties and add to the features property of the grid option:

- **name** : set to Paging
- **type** : set to local
- **pageSize** : 5

You can also set show/hide size drop-down or show/hide paging buttons, etc. Therefore, Paging feature object would look like below

```
{
  name: "Paging",
  type: "local",
  pageSize: 5
}
```

Add above object in features object array. To do that, open the file grid.component.ts and modify getGridFeatures() function such that it returns both paging and sorting features. After adding paging feature, getGridFeatures() function will look like below,

```
getGridFeatures() {
  return [
    {
      name: "Sorting",
      type: "local"
    },
    {
      name: "Paging",
      type: "local",
      pageSize: 5
    }
  ];
}
```



## Sort, Filter, and Page Fast With Ignite UI Grid

To test configured grid with paging: navigate to the application, scroll down, and at the bottom of the page you will find the grid added as shown in the image below:

Close	Date	High	Low	Open	Volume
21.32	Mon Jan 09 2017	21.5251	21.2728	21.332464	25611918
21.3825	Tue Jan 10 2017	21.41007	21.3528	21.405875	19956106
21.417	Sun Jan 08 2017	21.71433	21.2	21.5138	40663789
21.5788	Wed Jan 11 2017	21.8817	21.476614	21.666788	29849803
21.7834	Sun Jan 15 2017	22.1709	21.77	21.99839	23880247

1 - 5 of 10 records

Ignite UI also supports remote paging. Learn more here: <http://www.igniteui.com/help/iggrid-paging#remote>

### STEP 3: Enable Filtering

You can enable filter on Ignite UI grid by adding a feature named "Filtering" in the grid.

To enable filtering, create an object with following properties and add to the features property of the grid option:

- **name** : set to Filtering
- **allowFiltering** : set to true
- **caseSensitive**: set to false/true

Filtering feature object would look like below

```
{
  name: "Filtering",
  allowFiltering: true,
  caseSensitive: false
}
```

Add above object in features object array. To do that, open the file grid.component.ts and modify getGridFeatures() function such that it returns paging, sorting and filtering features. After adding filtering feature, getGridFeatures() function will look like below,

```
getGridFeatures() {
  return [
    {
      name: "Sorting",
      type: "local"
    },
    {
      name: "Paging",
```

```

    type: "local",
    pageSize: 5
  },
  {
    name: "Filtering",
    allowFiltering: true,
    caseSensitive: false
  }
];
}

```

To test the configured grid with filtering, navigate to the application, scroll down, and at the bottom of the page, you will find the grid added as shown below:

Close	Date	High	Low	Open	Volume
21.3825	Tue Jan 10 2017	21.41007	21.3528	21.405875	19956106
21.6749	Thu Jan 12 2017	22.045677	21.575	22.044	21213381
21.7834	Sun Jan 15 2017	22.1709	21.77	21.99839	23880247
22.12	Mon Jan 16 2017	22.40	21.901	22.2264	24456118
21.32	Mon Jan 09 2017	21.5251	21.2728	21.332464	25611918

1 - 5 of 10 records

As you will find the grid configured with filtering, paging, and sorting.

## Conclusion

Ignite UI makes it possible to write your web applications faster. In addition to Angular, Ignite UI can be used in React, AngularJS, jQuery, and ASP.NET MVC. In this lesson, you learned to configure a grid for basic features like paging, sorting, and filtering.

# Run Fast Using Virtualization in Ignite UI Grids

## Why Virtualization?

---

Virtualization is a valuable tool when displaying large sets of records to end users. A virtualized grid can bind to and support a data source of thousands of records, while providing a responsive experience to the end user using a rapid scroll of the grid.

The Ignite UI igGrid support two types of virtualization

1. Continuous Virtualization
2. Fixed Virtualization

In fixed virtualization, only the visible rows are rendered in the grid; in continuous virtualization, a pre-defined number of rows are rendered in the grid. The Ignite UI grid can be configured for column virtualization, row virtualization, or both. In the row virtualization, data row will be virtualized; in columns virtualization, columns of data source will be virtualized. You may choose to enable column virtualization when you have large number of columns in a data source.

## Lesson Objectives

---

1. Configure grid for fixed virtualization
2. Configure grid for continuous virtualization

For more information on the controls used in this lesson, see <http://infragistics.com/products/igniteui/grids/data-grid>.

At the end of the lesson, you will have a working grid configured for virtualization in an Angular application. You can learn more about Ignite UI Angular 2 here: <https://github.com/igniteui/igniteui-angular2>; you can also learn more about Angular in *Angular Essentials*, a free eBook published by Infragistics.

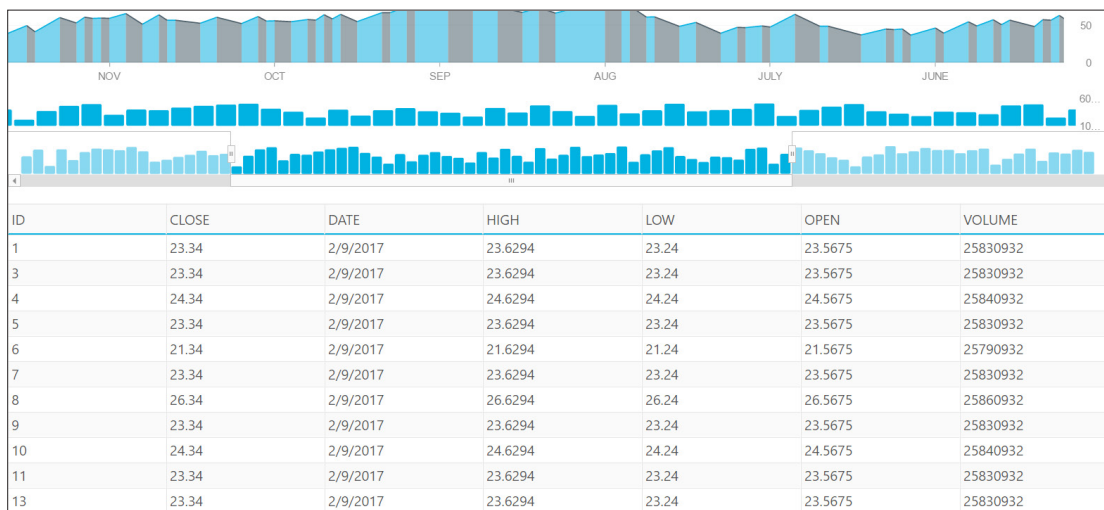
## Setting up the Project

You can download the starting project for this lesson *by clicking here*. (You can also download the final project *by clicking here*.)

After downloading the project, navigate to run the commands below:

```
npm install
npm start
```

You have executed the `npm install` command to install all dependencies and the `npm start` command to run the Angular application. If the project setup is correct, you will have a running Angular application with a grid as shown below. If you receive an error while running the application, stop and run the `npm start` command again.



The starter project includes a grid that was created with a large set of data. Since virtualization is not yet enabled on the grid, the grid is taking some time to render all of the records. In addition, it is rendering all rows at once. For 5,000 rows, the grid is creating 5,000 row elements on the DOM, which causes the application to run more slowly and less efficiently. To run the application faster, despite the very large set of data, you need to configure virtualization on the grid.

The starter project of this lesson contains code to work with REST API in an Angular application to create a large data set. To work with REST API and server communication, Angular provides an `http` class.

Learn more about the `http` class and server communication in Angular here: <https://angular.io/docs/ts/latest/guide/server-communication.html>

**STEP 1:** Enabling Fixed Virtualization

To enable virtualization, you must set these three properties of the Ignite UI grid.

1. `virtualizationMode`
2. `virtualization`
3. `height`

You must set the `height` property of the grid to enable virtualization. If the `height` property is not set, and `virtualization` is `true`, Ignite UI will throw an error.

To enable both row and column virtualization, set the value of the `virtualization` property to `true`. The `virtualization` property can also be set to a numeric value so whenever a number of records in the data source is more than the specified number, virtualization will be enabled.

To enable fixed virtualization, set the following properties of the grid:

- Set `virtualization` property to `"true"`
- Set `virtualizationMode` property to `"fixed"`
- Set `height` property to some pixel value (here it will be set to `"300px"`)
- To configure all of these properties of the grid, in the app folder: open the `grid.component.ts` file and update the `getGridOptions()` function as shown in the highlighted listing below. You are adding three more properties to existing grid options.

```
getGridOptions() {
  return {
    width: "100%",
    autoGenerateColumns: false,
    height: "300px",
    virtualization: true,
    virtualizationMode: "fixed",
    columns: [
      { headerText: "ID", key: "Id", dataType: "string",width:"10%" },
      { headerText: "CLOSE", key: "Close", dataType: "number",width:"15%" },
      { headerText: "DATE", key: "Date", dataType: "string",width:"15%" },
      { headerText: "HIGH", key: "High", dataType: "number",width:"15%" },
      { headerText: "LOW", key: "Low", dataType: "number",width:"15%" },
      { headerText: "OPEN", key: "Open", dataType: "number",width:"15%" },
      { headerText: "VOLUME", key: "Volume", dataType: "number",width:"15%"}
    ]
  };
}
```

To test the fixed virtualization: navigate to the application, scroll down, and you will find the grid configured with fixed virtualization added as shown in the image below:

ID	CLOSE	DATE	HIGH	LOW	OPEN	VOLUME
1	23.34	3/1/2017	23.6294	23.24	23.5675	25830932
3	23.34	3/1/2017	23.6294	23.24	23.5675	25830932
4	24.34	3/1/2017	24.6294	24.24	24.5675	25840932
5	23.34	3/1/2017	23.6294	23.24	23.5675	25830932
6	21.34	3/1/2017	21.6294	21.24	21.5675	25790932
7	23.34	3/1/2017	23.6294	23.24	23.5675	25830932
8	26.34	3/1/2017	26.6294	26.24	26.5675	25860932
9	23.34	3/1/2017	23.6294	23.24	23.5675	25830932

## STEP 2: Enabling Continuous Virtualization

To enable continuous virtualization, set the following properties of the grid:

- Set `rowVirtualization` property to "true"
- Set `virtualizationMode` property to "continuous"
- Set `height` property to some pixel value (here it will be set to "300px")

To configure all of these properties of the grid, in the app folder: open the `grid.component.ts` file and update the `getGridOptions()` function as shown in the highlighted listing below

```
getGridOptions() {  
  return {  
    width: "100%",  
    autoGenerateColumns: false,  
    height: "300px",  
    rowVirtualization: true,  
    virtualizationMode: "continuous",  
    columns: [  
      { headerText: "ID", key: "Id", dataType: "string" },  
      { headerText: "CLOSE", key: "Close", dataType: "number" },  
      { headerText: "DATE", key: "Date", dataType: "string" },  
      { headerText: "HIGH", key: "High", dataType: "number" },  
      { headerText: "LOW", key: "Low", dataType: "number" },  
      { headerText: "OPEN", key: "Open", dataType: "number" },  
      { headerText: "VOLUME", key: "Volume", dataType: "number" }  
    ]  
  };  
}
```

## Run Fast Using Virtualization in Ignite UI Grids

To test the continuous virtualization: navigate to the application, scroll down, and you will find the grid configured with continuous virtualization added as shown in the image below:

ID	CLOSE	DATE	HIGH	LOW	OPEN	VOLUME
409	23.34	2/9/2017	23.6294	23.24	23.5675	25830932
411	23.34	2/9/2017	23.6294	23.24	23.5675	25830932
412	24.34	2/9/2017	24.6294	24.24	24.5675	25840932
413	23.34	2/9/2017	23.6294	23.24	23.5675	25830932
414	26.34	2/9/2017	26.6294	26.24	26.5675	25860932
415	23.34	2/9/2017	23.6294	23.24	23.5675	25830932
416	21.34	2/9/2017	21.6294	21.24	21.5675	25790932
417	23.34	2/9/2017	23.6294	23.24	23.5675	25830932

In continuous virtualization, only a portion of the total rows in the data source are rendered in the DOM. As the user scrolls up and down on the grid, the virtualization feature determines if the current rows are sufficient to display the next/previous portion of rows. If new rows are required, the current portion of rows is deleted and the new portion of rows is created.

### Conclusion

In any functional LOB application, you must render thousands of records in a grid. When an application is rendering thousands of records, the grid should be responsive during a rapid scroll of the grid. Achieve this by enabling the virtualization feature on the grid. In this lesson, you learned about configuring a grid for fixed and continuous virtualization.

# Run Fast with Large Sets of Data in Ignite UI Data Charts

Ignite UI Data Charts can render thousands of data points very smoothly and are fastest when rendering large sets of data.

## Lesson Objectives

---

1. Configure a data chart to work with REST API
2. Create a data chart with large set of data

For more information on the controls used in this lesson, see <http://www.infragistics.com/products/igniteui/charts/data-chart>.

At the end of this lesson, you will have a data chart configured to work with large sets of data in an Angular application. You will see that even with a large amount of data, the chart renders quickly and zooming in and out of the chart is fluid and responsive.

Learn more about Ignite UI Angular 2 here: <https://github.com/igniteui/igniteui-angular2>; you can also learn more about Angular in *Angular Essentials*, a free eBook published by Infragistics.

## Setting up the Project

---

You can download the starting project for this lesson [by clicking here](#). (You can also download the final project [by clicking here](#).)

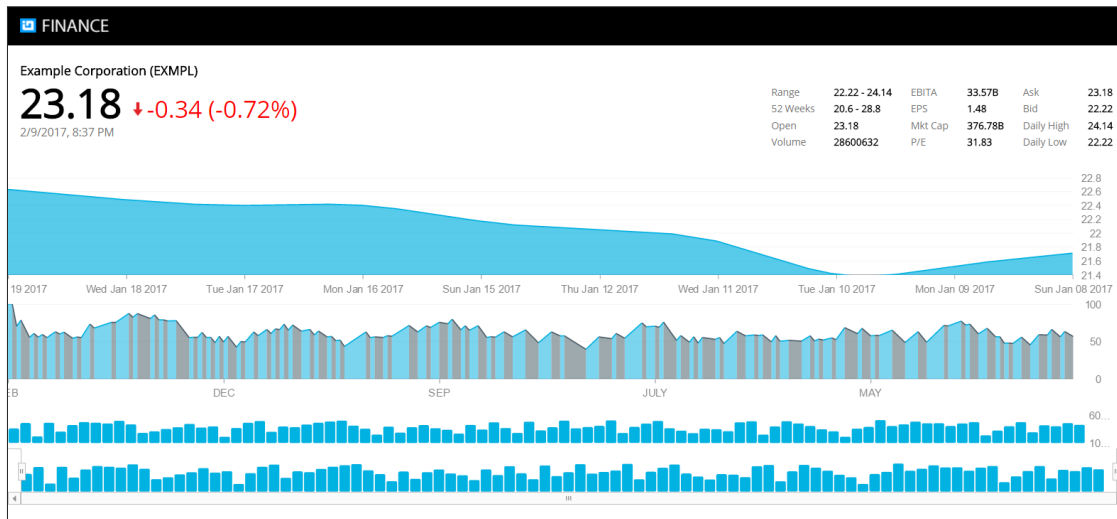
After you download the project, navigate to the `directory` and run the commands below:

```
npm install  
npm start
```



## Run Fast with Large Sets of Data in Ignite UI Data Charts

You have executed the npm install command to install all dependencies and the npm start command to run the Angular application. If the project is setup correctly, you will have a running Angular application as shown in the image below. If you receive an error while running the application, stop and run the npm start command again.



Right now, the first chart of the application is configured to work with a small data set. It is rendering a data source with ten rows and is functioning smoothly so you can zoom in and zoom out on the chart. Ignite UI data charts are created to work with large *and* small data sets. Whether you are rendering 10 data points or 1000 data points, Ignite UI data charts will behave in the same smooth, seamless manner to help to run the application faster.

To see it in action, modify the chart to work with a large data set returning from a REST API.

### STEP 1: Get Data in a Component

Currently the chart is configured to work with a small data set, which is configured in the first line of code in the `ngOnInit()` function. To get a large data set in the `PriceChartComponent`, you must use `AppService` in the component. To do so, in the app folder, open the `pricechart.component.ts` file, navigate to the `ngOnInit()` function, and (in the function) delete the first line of code and make a call to `appService.getStocks()` method. Replace only the first line of code as shown below and leave other codes of `ngOnInit()` to function as they are.

```
ngOnInit() {
  this._appService.getStocks()
    .subscribe(
      stocks => this.stocks = stocks,
```

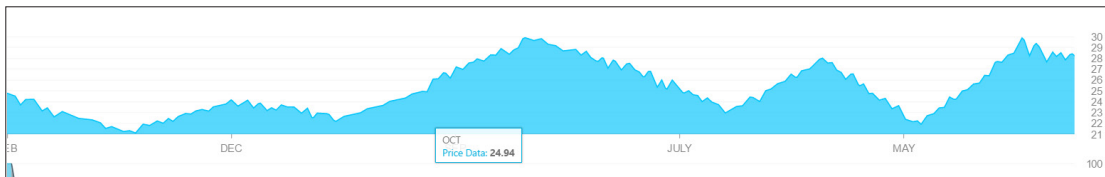
```
error => this.errorMessage = <any>error);
this.desiredHeight = 0.22 * (window.screen.height) + "px";
this.chartOptions = this.getPriceChartOptions();
}
```

Right now, you are fetching data from AppService in the PriceChartComponent. The AppService getStocks() method is fetching data from REST API, which has more than 200 data points. Essentially, you have reconfigured the chart to work with a large data set.

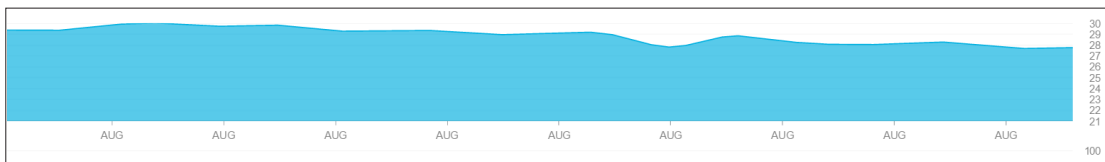
## STEP 2: Run The Application

Navigate to the application and you will see that the Ignite UI data chart is rendering a large set of data very quickly and smoothly.

Now there are more than 200 data points rendered in the chart.



You can zoom out to a particular data point and the IgniteUI data chart will render in the same way and help the application run faster.



## Conclusion

Ignite UI can be very useful in writing faster web applications. In addition to Angular, you can use Ignite UI in React, AngularJS, jQuery, and ASP.NET MVC. In this lesson, you learned how to use Ignite UI data charts with large sets of data in an Angular application. Various functionalities of Ignite UI data charts, such as zoom in and zoom out, work seamlessly with both small and large data sets.

# Zoom Fast with Ignite UI Zoombar

Ignite UI provides a Zoombar control to zoom range-enabled controls like data charts. Use Zoombar to zoom in on a widget in a resizable zoom-range window. Zoombar includes a horizontal scroll bar that can zoom either the whole range or a particular section of the chart. Zoombar works as a stand-alone control.

Learn more about other Ignite UI features here: <http://www.igniteui.com>

In this lesson, you will learn to configure Ignite UI Zoombar with a data chart.

## Lesson Objective

---

1. Add Zoombar
2. Configure Zoombar with a Ignite UI data chart.

For more information on the controls used in this lesson, see <http://www.infragistics.com/products/igniteUI/other-charts/zoombar>.

At the end of this lesson, you will have an Ignite UI data chart that is configured with Ignite UI Zoombar in an Angular application.

Learn more about Ignite UI Angular 2 here: <https://github.com/igniteUI/igniteui-angular2>; you can also learn more about Angular in *Angular Essentials*, a free eBook published by Infragistics.

## Setting up the Project

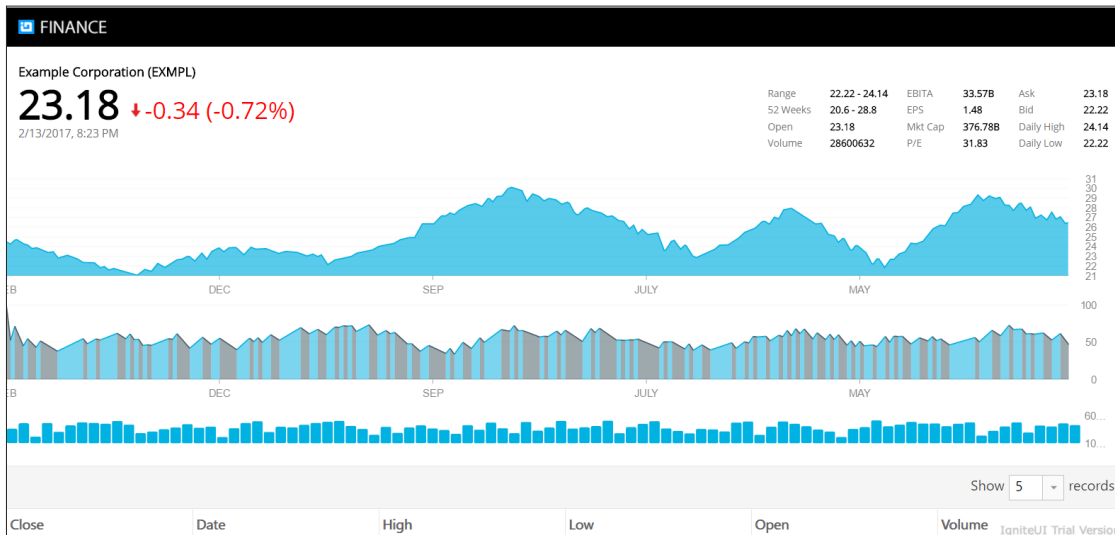
---

You can download the starting project for this lesson *by clicking here*. (You can also download the final project *by clicking here*.)

Next, navigate to the Finance App directory and run the commands below:

```
npm install  
npm start
```

You have executed the `npm install` command to install all dependencies, and the `npm start` command to run the Angular application. If the project is setup correctly, you will have a running Angular application as shown in the image below. In addition, while working through the lesson, if you receive an error while running the application, stop and run the `npm start` command again.



## STEP 1: Import and Declare the Component

To work with Ignite UI Angular components, you must import and declare them in the module. For example, to use the `igGrid` component in an Angular application, import and declare the `IgGridComponent` in the application module.

Navigate to the Finance App folder and then the app folder. Open the file `app.module.ts`, and add below import statements, just after all of the existing import statements:

```
import { IgZoombarComponent } from 'igniteui-angular2';
```

After importing the required components, you must declare them in the application module. Add `IgZoombarComponent` in the `AppModule`'s declaration array. Modify `@NgModule` decorator in `app.module.ts` as shown below:

```
@NgModule({
  imports: [BrowserModule, HttpClientModule],
  declarations: [AppComponent,
    IgDataChartComponent,
    InfoComponent,
    IndicatorChartComponent,
    VolumeChartComponent,
```

```

    IgGridComponent,
    GridComponent,
    PriceChartComponent,
    IgZoombarComponent,
  ],
  providers: [AppService],
  bootstrap: [AppComponent]
})

```

You have added `IgZoombarComponent` in the declaration array of the `AppModule` module. Other added components and other properties, like providers, will be reviewed in subsequent lessons.

### STEP 2: Add Zoombar

To work with Ignite UI Zoombar, you must first add the Zoombar component. In the `app\charts` folder, open the `volumechart.component.html` file and add the `ig-zoombar` control as shown below, just after the `ig-data-chart` control:

```
<ig-zoombar [(options)]="zoombarOptions" widgetId="zoombar"></ig-zoombar>
```

### STEP 3: Add Zoombar Options Property

In the Zoombar option, you can attach a chart to the Zoombar. To configure the Zoombar option, create a property in the `VolumeChartComponent` class. In the `app\charts` folder, open the `volumechart.component.ts` file and, just above the constructor, add the property listed below:

```
private zoombarOptions: IgZoombar;
```

### STEP 4: Attach Chart to Zoombar

To attach a chart widget to Zoombar, you must set the target property value of Zoombar options to the ID of the chart widget. In the `app\charts` folder, open the `volumechart.component.ts` file and the code below just after the `this.chartOptions` assignment in the `ngOnInit()` function:

```

this.zoombarOptions = {
  target: "#volumechart"
};

```

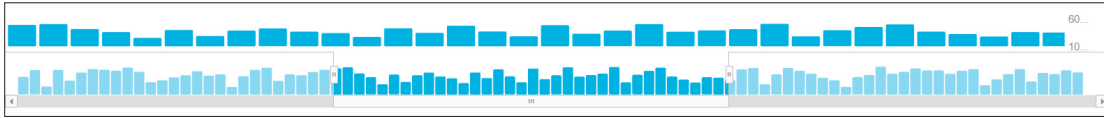
In the above listing, `volumechart` is the ID of the data chart widget.

---

## STEP 5: Run the Application

---

Navigate to application, scroll down, and at the bottom of the page, you will find Zoombar added as shown in the image below:



The chart has been cloned in the Zoombar and, by using the horizontal scroll bar, you can zoom the chart. You will find animation while zooming in and out is very fast and smooth. Regardless of how many data points are rendered in the chart, Ignite UI Zoombar will zoom in on a particular section very quickly and with smooth animation.

## Conclusion

Ignite UI can be very helpful in writing and running web applications more quickly. In addition to Angular, you may use Ignite UI in React, AngularJS, jQuery, and ASP.NET MVC. In this lesson, you learned how to configure Ignite UI Zoombar in an Angular application.

# Ignite UI With Different Package Managers

Ignite UI works with popular package managers to manage the dependencies of the project. The most popular package managers are:

- NPM
- Yarn
- JSPM

## STEP 1: Working With NPM

---

In previous lessons, you have used NPM to work with Ignite UI controls. To see in how NPM works, download the *starter project* (you'll also find the final version of the project *here*), open the terminal, and run the command listed below:

```
npm install
```

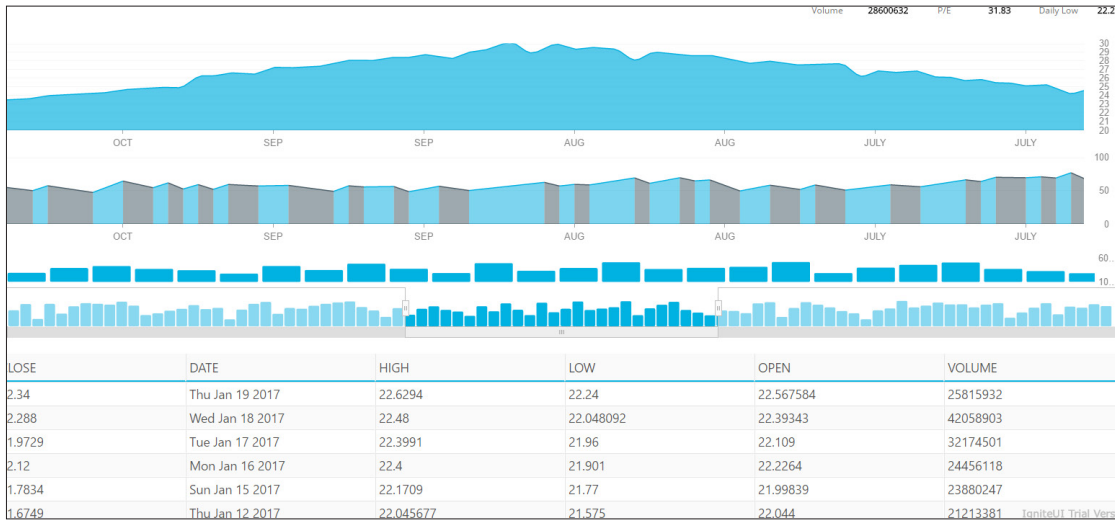
NPM install commands install the dependencies. It reads package.json file to install all the dependencies. to work with NPM, you must have NodeJS installed. If you do not have NodeJS installed, you can install it from <https://nodejs.org/en/>. After running NPM install command, you will find folder node\_modules added in your project. This folder contains all the libraries installed using command NPM install.

If you are working on an existing project, then you may install the individual package in the project. To install the Ignite UI package individually, execute the following command:

```
npm install --save-dev igniteui-angular2
```

To run the application, execute the command below:

```
npm start
```



## STEP 2: Working with Yarn

NPM is one of the most popular package managers, but it has some shortcomings, including:

- Nested dependencies, which causes a long file path on Windows
- NPM does only sequential installation, so one package must be completely installed before moving to the install of the next package
- It can only install from the NPMSJS package and it does not have offline installation.

Yarn solves these problems. Yarn is a fast, reliable, and secure package manager. It takes packages from npmjs or bower registry. Although Yarn has advantages over NPM, NPM is still widely used and is the most popular package manager.

You can learn more about Yarn on their github page here: <https://github.com/Yarnpkg/Yarn>.

Like NPM, Yarn also reads package.json files of your project to install dependencies.

To work with Yarn, download the start project and open it in the terminal.

If you do not have Yarn installed on your machine, you must install it with the command below:

```
npm install -g yarn
```



**NOTE:** If you are using Apple's Mac OS, you may receive permission errors when you try to install global packages with NPM. If this happens, try:

```
sudo npm install -g yarn
```

After installing Yarn, you can use Yarn to install dependencies in your project. Like npm, Yarn also reads package.json to install dependencies. To install dependencies, run the command below:

```
yarn install
```

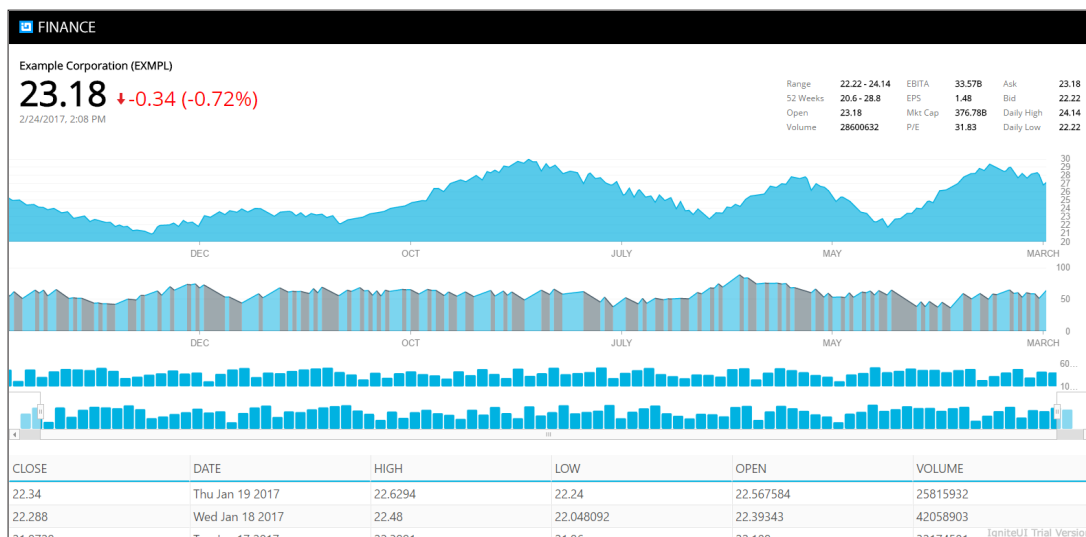
If you are working on an existing project, you may also install an individual package in the project. To install the Ignite UI package, execute the command below:

```
yarn add -dev igniteui-angular2
```

After successful installation, you will find a node\_modules folder added in the project. To run the application, execute the command below:

```
yarn start
```

If everything is correct, the above command should start the application and you will have a running application as shown below:



---

## Working With Dynamic Module Loaders

---

SystemJS is a module loader that can import a module at run-time and is built on the top of the ES6 module loader. It can transpile ES6 code or TypeScript. SystemJS can work with many types of modules formats such as AMD and CommonJS. SystemJS module loader can also work with Ignite UI modules and support it.

In the downloaded project, open the System.config.js file and you will find mapping for Ignite UI Angular 2 as shown in the listing below:

```
'igniteui-angular2': 'npm:igniteui-angular2'
```

In addition, you can find the Ignite UI package for loading as shown below:

```
'igniteui-angular2': {  
  main: 'index.js',  
  defaultExtension: 'js'  
}
```

Due to packaging and mapping information in system.config.js, when the Angular application needs Ignite UI modules, it will be dynamically loaded by SystemJS in the application.

## Conclusion

In web development, adding references of libraries has come a long way. It began with adding references manually in the project, then using Content Delivery Networks (CDN) to add references, and then various package managers such as bower, NPM, and Yarn came to existence. Ignite UI can be used with previous ways of managing packages like CDN or can be used with modern package managers such as NPM and Yarn. In addition to package managers, Ignite UI can be used with popular module loaders like SystemJS.

# Write React JS Apps with Ignite UI

Ignite UI fully supports modern web development. In addition to Angular, you can use Ignite UI library in React. This lesson will demonstrate how to use Ignite UI grid in a React application.

## Lesson Objective

---

- Add Ignite UI grid in ReactJS
- Configure columns of the grid.

For more information on the controls used in this lesson, see <http://infragistics.com/products/igniteui/grids/data-grid>.

## Setting up the Project

---

You can download the starting project for this lesson [by clicking here](#). (You can also download the final project [by clicking here](#).)

This project is already configured to work with ReactJS and Ignite UI and all references have been added to the project. You can learn more about using Ignite UI in ReactJS project here:

[http://www.infragistics.com/community/blogs/igniteui\\_team/archive/2016/11/04/how-to-use-ignite-ui-components-with-react.aspx](http://www.infragistics.com/community/blogs/igniteui_team/archive/2016/11/04/how-to-use-ignite-ui-components-with-react.aspx)

In the [starting project for this lesson](#), in addition to the React and Ignite UI libraries, you will find following files.

- **index.html** : contains application markup and references
- **index.js** : contains react code
- **data.js**: contains data to be used as data source of the Ignite UI grid.

In the project, data.js contains data to be rendered in the Ignite UI grid. File Index.js contains the App component. In index.js, you can find component class created as shown in the listing below:

```
var App = React.createClass(  
  {  
    getInitialState: function()  
    {  
      return{  
      }  
    },  
    render: function()  
    {  
      return(  
        <h2>Ignite UI Grid will be rendered here</h2>  
      )  
    }  
  });  
ReactDOM.render(  
  <App />,  
  document.getElementById("app")  
);
```

The above App component class contains two functions :

- The getInitialState() function simply returns an Object of initial state
- The render() function returns the description of what you want to render. In the next steps, we will render Ignite UI grid in the render function of the App component.

You can learn more about React.createClass API here: <https://facebook.github.io/react/docs/react-api.html>

On the index.html, as shown in the listing below, you will find that index.js has been referenced as babel script:

```
<div id="app">
  <script type="text/babel" src="index.js">
  </script>
</div>
```

In index.html, you will also find references of React, jquery, and Ignite UI libraries.

After downloading the project, navigate to the directory and run the commands below:

```
npm install
npm start
```

You have executed the npm install command to install lite server (web server) dependencies, using the npm start command to run the React application. If everything is correct, you will find a React application running in the browser as shown below:



### STEP 1 : Initialize Initial State

To initialize the grid, you may want to set values for various properties of grid such as datasource, width, row styles, etc. You can set these grid properties in the getInitialState() function. Open index.js file, and modify the getInitialState() function with the code below.

```
getInitialState: function()
{
  return{
    data: stocks,
    gridWidth: "100%",
    alternateRowStyles: true
  }
},
```

You are creating properties to set the grid's width, row style, and data source. In addition, you will find there is already an array called "stocks" in the application.

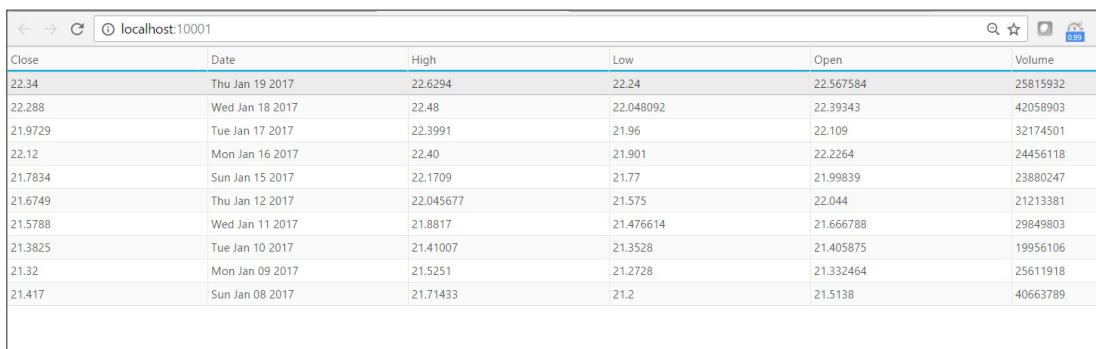
## STEP 2: Render the Grid

To render the grid, you must return it from the render() function of the component class. To return IgGrid, open index.js and modify the render function as shown below:

```
render: function()
{
  return(
    <div>
      <IgGrid id="grid1"
        autoGenerateColumns={true}
        dataSource={this.state.data}
        width={this.state.gridWidth}
        alternateRowStyles={this.state.alternateRowStyles} />
    </div>
  )
}
```

You are setting dataSource, width, and alternateRowStyles properties with the properties of the object returned from the getInitialState() function.

Navigate to the application to find the RecatJS application running with the Ignite UI grid as shown in the image below:



The screenshot shows a web browser window with the address bar displaying "localhost:10001". The browser content is a table with the following data:

Close	Date	High	Low	Open	Volume
22.34	Thu Jan 19 2017	22.6294	22.24	22.567584	25815932
22.288	Wed Jan 18 2017	22.48	22.048092	22.39343	42058903
21.9729	Tue Jan 17 2017	22.3991	21.96	22.109	32174501
22.12	Mon Jan 16 2017	22.40	21.901	22.2264	24456118
21.7834	Sun Jan 15 2017	22.1709	21.77	21.99839	23880247
21.6749	Thu Jan 12 2017	22.045677	21.575	22.044	21213381
21.5788	Wed Jan 11 2017	21.8817	21.476614	21.666788	29849803
21.3825	Tue Jan 10 2017	21.41007	21.3528	21.405875	19956106
21.32	Mon Jan 09 2017	21.5251	21.2728	21.332464	25611918
21.417	Sun Jan 08 2017	21.71433	21.2	21.5138	40663789

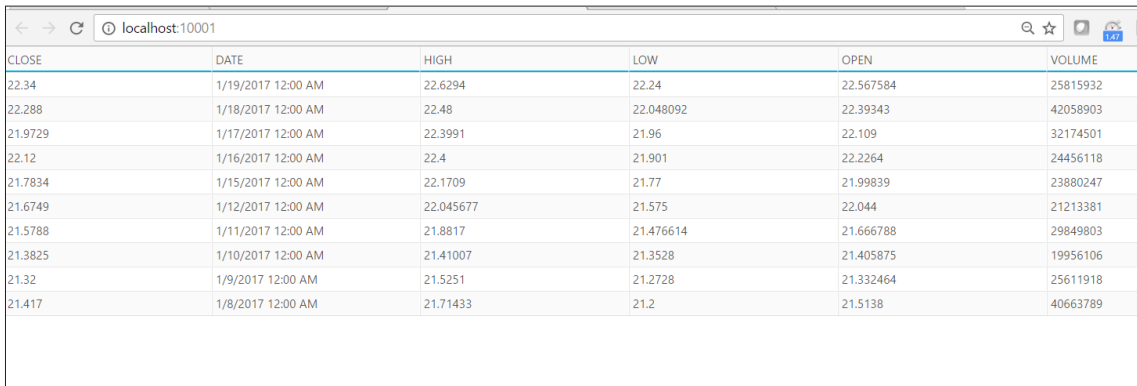
## STEP 3 CONFIGURE COLUMNS OF THE GRID

In the previous step, you set autoGenerateColumns to true in order to create a grid. You can also configure selected columns from the data set to display. To do so, you must configure columns for the Ignite UI Grid by setting the autoGenerateColumns

property to false and adding the columns property in the grid. Modify the IgGrid in the render() function as shown in the listing below:

```
<IgGrid id="grid1"
  autoGenerateColumns={false}
  dataSource={this.state.data}
  width={this.state.gridWidth}
  alternateRowStyles={this.state.alternateRowStyles}
  columns=[
    { headerText: "CLOSE", key: "Close", dataType: "number" },
    { headerText: "DATE", key: "Date", dataType: "date", format: "dateTime" },
    { headerText: "HIGH", key: "High", dataType: "number" },
    { headerText: "LOW", key: "Low", dataType: "number"},
    { headerText: "OPEN", key: "Open", dataType: "number"},
    { headerText: "VOLUME", key: "Volume", dataType: "number"},
  ]
/>
```

Navigate to the application and you will find that a grid has been configured with the columns.



CLOSE	DATE	HIGH	LOW	OPEN	VOLUME
22.34	1/19/2017 12:00 AM	22.6294	22.24	22.567584	25815932
22.288	1/18/2017 12:00 AM	22.48	22.048092	22.39343	42058903
21.9729	1/17/2017 12:00 AM	22.3991	21.96	22.109	32174501
22.12	1/16/2017 12:00 AM	22.4	21.901	22.2264	24456118
21.7834	1/15/2017 12:00 AM	22.1709	21.77	21.99839	23880247
21.6749	1/12/2017 12:00 AM	22.045677	21.575	22.044	21213381
21.5788	1/11/2017 12:00 AM	21.8817	21.476614	21.666788	29849803
21.3825	1/10/2017 12:00 AM	21.41007	21.3528	21.405875	19956106
21.32	1/9/2017 12:00 AM	21.5251	21.2728	21.332464	25611918
21.417	1/8/2017 12:00 AM	21.71433	21.2	21.5138	40663789

## Conclusion

React is quickly becoming a very popular option for building client-side JavaScript applications. Enterprises are already looking at using React of their Line of Business applications. Ignite UI supports modern web development and its controls can be used with modern web development framework such as React

# Look Great With IgniteUI Themes

As a developer, you want to ensure that your application looks good and works on all types of devices, including desktops, tablets, and mobile devices. Modern web applications should be responsive and touch-enabled, but will require a lot of CSS/SASS /LESS in your application. As a developer, you may not be skilled in CSS or have the time to learn it for use in your application. IgniteUI can help by providing various themes, which can be used as they are in your application or you can use the IgniteUI Theme Generator to create themes as required by your application.

Provided themes:

- Infragistics theme
- Metro theme
- iOS theme
- Default bootstrap theme
- Superhero bootstrap theme
- Yeti bootstrap theme
- Flatly bootstrap theme

In addition to these themes, you can use the IgniteUI Bootstrap Theme Generator to create your own theme. Learn more about IgniteUI Theme Generator at <http://www.igniteui.com/bootstrap-theme-generator/Help>; you can also learn more about Angular in *Angular Essentials*, a free eBook published by Infragistics.

## Setting Up The Project

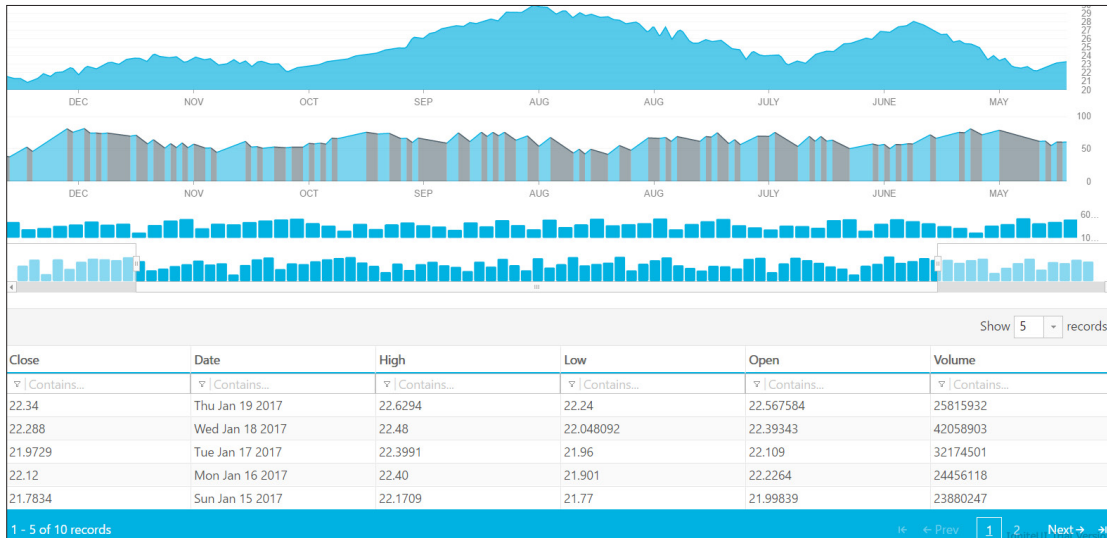
You can download the starting project for this lesson *by clicking here*. (You can also download the final project *by clicking here*.)

After downloading the project, navigate to the directory and run the commands below:

```
npm install  
npm start
```



You have executed the `npm install` command to install all dependencies and have used the `npm start` command to run the Angular application. If the project setup is correct, you will have a running Angular application as shown in the image below:



The application is currently using the IgniteUI metro theme. In the project, open the `index.html` file, navigate to line number 9 to 10, or look for the CSS references in the head section. You will find that the application is referring to the metro theme from the IgniteUI CDN as shown in the listing below.

```
<link href="http://cdn-na.infragistics.com/igniteui/latest/css/themes/metro/infragistics.theme.css" rel="stylesheet" />
<link href="http://cdn-na.infragistics.com/igniteui/latest/css/structure/infragistics.css" rel="stylesheet" />
```

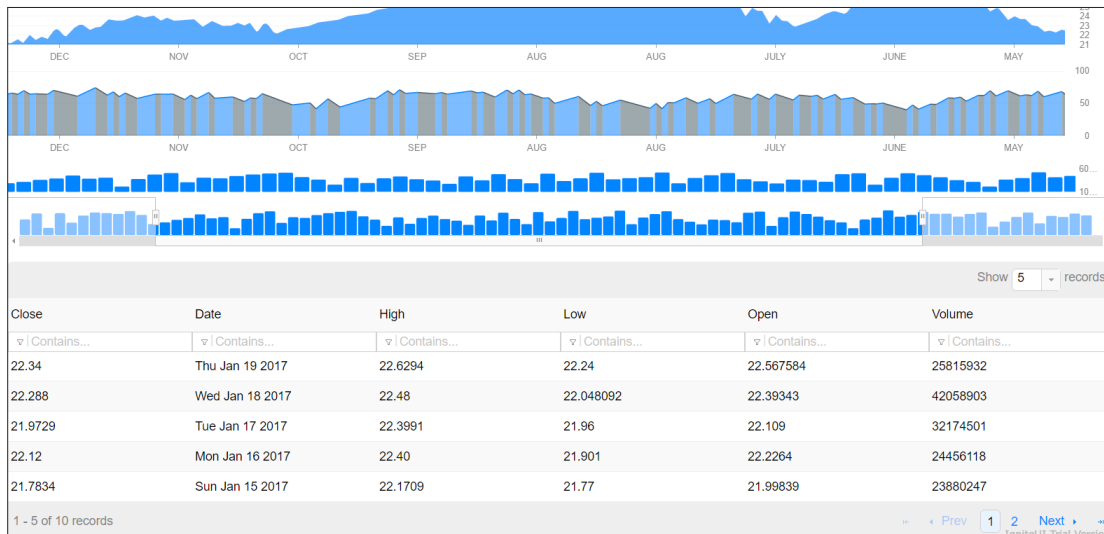
To work with any theme, you need a reference of `Infragistics.css` besides the theme reference.

### STEP 1: Changing to iOS Theme

Change the existing IgniteUI themes easily by switching to the desired theme reference. To change the theme from the metro theme to the iOS theme, leave the reference of `infragistics.css` as it is and modify the IgniteUI theme reference in the `index.html` head section as shown in the listing below.

```
<link href="http://cdn-na.infragistics.com/igniteui/latest/css/themes/ios/infragistics.theme.css" rel="stylesheet" />
```

Navigate to the application and you will find that all of the controls have been changed to the iOS theme. You may notice that the grid's look and the navigation button's design have been changed.

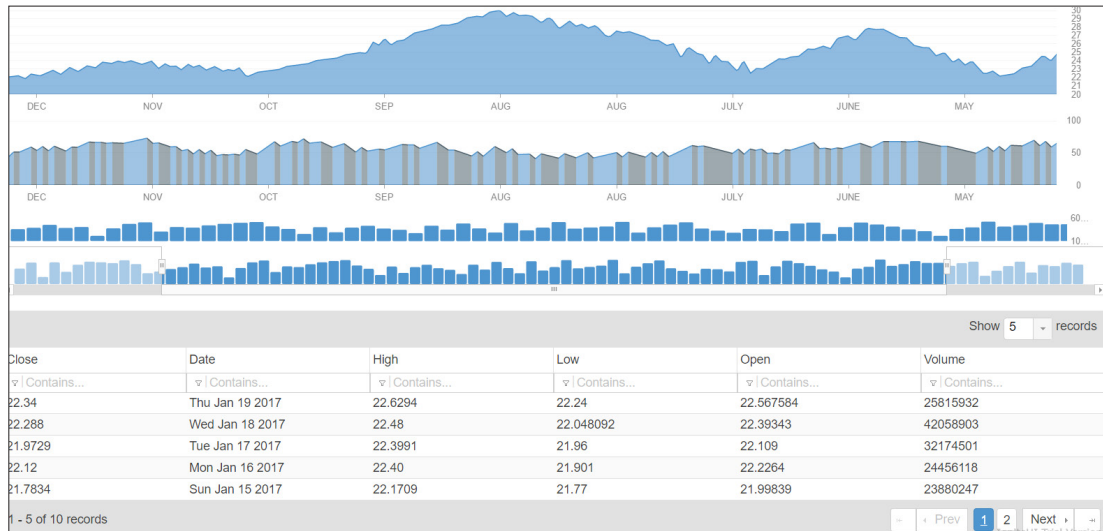


## STEP 2: Changing to Default Bootstrap Theme

IgniteUI provides a default bootstrap theme. To change the theme to the bootstrap theme, leave the reference of `infragistics.css` as it is and modify the IgniteUI theme reference in the `index.html` head section as shown in the listing below.

```
<link href="http://cdn-na.infragistics.com/igniteui/latest/css/themes/bootstrap/infragistics.theme.css" rel="stylesheet" />
```

Navigate to the application and you will find that all of the controls have been changed to the default basic bootstrap theme. The grid's look and the navigation button's design have been changed to the bootstrap theme.



### STEP 3: Using Your Own Bootstrap Theme

IgniteUI helps you to create your own bootstrap-based theme. Simply upload a `variables.less` file in IgniteUI bootstrap theme generator and download the theme (combination of LESS, Compiled CSS and images) to use in your application.

Learn more about IgniteUI theme generator here: <http://www.igniteui.com/bootstrap-theme-generator/Help>. IgniteUI theme generator helps you in two possible ways:

1. To customize existing IgniteUI themes
2. To create new bootstrap theme using the `variables.less` file.

In previous steps you have used themes provided by IgniteUI. To use your own bootstrap theme, follow the steps as below. Note that, for this lesson, you do not have to perform these steps, as a bootstrap-based theme has been added in the project.

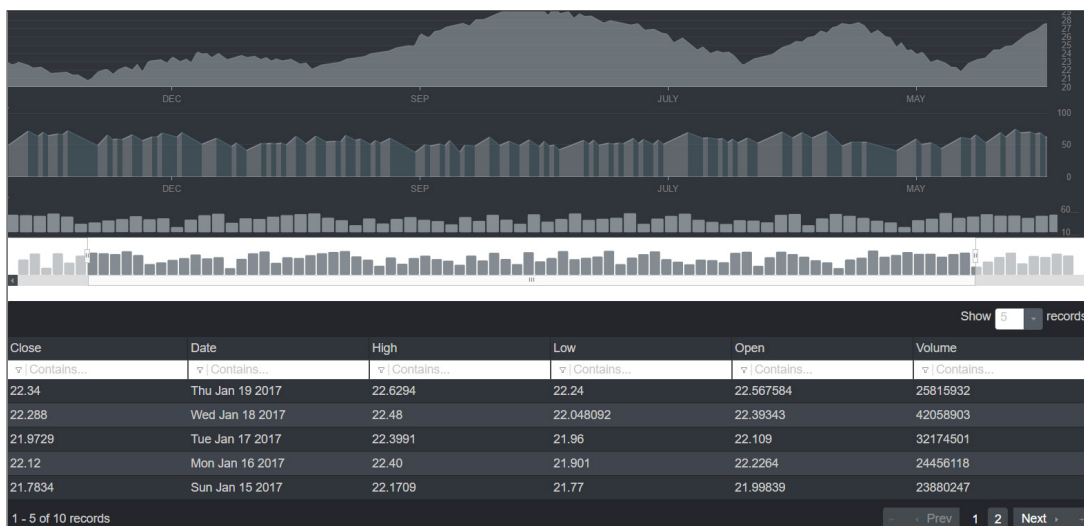
1. You can create your own `variables.less` file or use one of the bootstrap themes from <http://bootswatch.com>. To use a theme from bootswatch, select the theme and download the `variables.less` file.
2. Upload `variables.less` file here: <http://www.igniteui.com/bootstrap-theme-generator/Theme/Upload>.
3. Download the theme and unzip it.
4. Save the downloaded theme in your application project.

The project contains a `CSS` folder, which contains a theme generated by the IgniteUI theme builder. To use this theme: in the head section of `index.html` add a reference of the theme and bootstrap as shown in the listing below. Delete reference of metro

theme (line number 9) and add the below references just before the ./css/structure/infragistics.css reference.

```
<link href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css"
rel="stylesheet" type="text/css" />
<link href="css/themes/infragistics.theme.css" rel="stylesheet" type="text/css" />
```

Navigate to the application and you will find that application is using new theme.



## Conclusion

In addition to the themes provided by Infragistics, you can use your own themes or jQueryUI Theme Roller. Learn more here: [http://www.infragistics.com/help/deployment-guide-styling-and-theming#\\_Styling\\_and\\_Theming\\_Infragistics](http://www.infragistics.com/help/deployment-guide-styling-and-theming#_Styling_and_Theming_Infragistics)

Infragistics supports the latest designs available in the modern web development and allows you to write web applications faster.