

COM661 Full Stack Strategies and Development

FE18. The Angular Ecosystem

Aims

- To summarise the structure of the Angular framework
- To describe the provision of 3rd party libraries
- To identify the AG Grid library to add data grid functionality
- To update the back-end API to provide a data feed for the grid component
- To build the basic AG Grid
- To configure the functionality and presentation of the grid
- To identify opportunities for further development

Table of Contents

18.1 ANGULAR FRAMEWORK	2
18.1.1 ANGULAR STRUCTURE	2
18.1.2 ANGULAR LIBRARIES	2
18.1.3 CASE STUDY – AG GRID	3
18.1.4 INSTALLATION AND PREPARATION	4
18.2 USING AG GRID	5
18.2.1 BUILDING THE WEB SERVICE	5
18.2.2 A BASIC GRID	6
18.3 CONFIGURING ADDITIONAL FUNCTIONALITY	11
18.3.1 FILTERING	11
18.3.2 COLUMN OPTIONS	14
18.3.3 PAGINATION	16
18.3.4 STYLING	18
18.3.5 OTHER EXTENSIONS	20
18.4 FURTHER INFORMATION	21

18.1 Angular Framework

The Angular framework has consistently been in widespread use since its first release in 2010 and has been in continuous development since then, with a release pattern that has now settled down to a new major release each year. Angular is maintained and developed by a single-focus team at Google, who devote significant resource to the project. Although the React framework maintains its overall lead in the most popular front-end web frameworks, Angular is still increasing in popularity among developers due to its suitability for rapid development, performance (especially in Single Page Apps), and code security.

18.1.1 Angular Structure

Angular applications are comprised of services and components, and it is important to understand the difference between these key elements. Services are typically used when you need to share data or functionality between different components in your application. For example, if you have multiple components that need to access the same data, you can create a service to provide that data and inject it into each component. Another common use case for services is when you need to interact with an API. You can create a service that makes HTTP requests to the API and provides the data to your components. Services are characterized as *singletons*, meaning that they are created once and used throughout the entire lifespan of the application.

Angular components define and manage the view-related logic for a piece of your application. They consist of a TypeScript class and an HTML template, and they can be styled with CSS. Components are created and destroyed as needed by the Angular framework, and they are typically used to compose the structure of your application. You can use components to create reusable UI elements, such as buttons and forms, or to define specific views for different parts of your application. Components represent visible content elements on the web page and are included in the HTML stream by use of a tag which mirrors the text of the component selector.

18.1.2 Angular Libraries

An Angular Library is a project that cannot be run on its own but needs to be incorporated into an application to extend Angular's base features. For example, to add reactive forms to an application, we add the library package using `ng add @angular/forms`, then import the `ReactiveFormsModule` from the `@angular/forms` library in our application code.

One of the strengths of Angular is the range of libraries available from 3rd party developers and many commercial software companies produce Angular Libraries that are made

available – often with a free tier of use for anyone, and a subscription tier that releases more advanced features. The production of libraries to enhance the features of Angular is encouraged by the Angular Core Team who make available detailed documentation on how libraries should be produced and released.

18.1.3 Case Study – AG Grid

In this practical, we will investigate the incorporation of an Angular Library into an application by examining **AG Grid**, which provides data grid functionality on a collection of data.

A Data Grid is a table-based approach to browsing a collection of data. It often includes filtering to select a subset of the elements to be displayed, pagination to control the display of large volumes of data, and sorting on one or more fields. **AG Grid** provides all of these and more, and its fully configurable and programmable structure makes it an ideal candidate for close investigation.

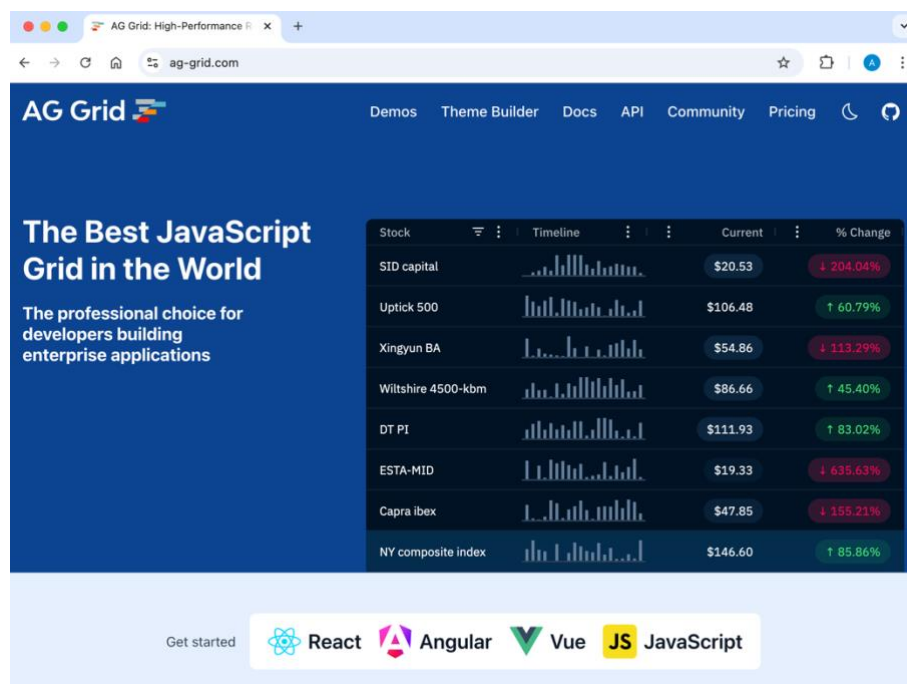


Figure 18.1 AG Grid Home Page

In the sections of this practical that follow, we will install **AG Grid** into an Angular application, configure it to display our collection of businesses, and demonstrate some of the additional functionality that can be incorporated.

18.1.4 Installation and Preparation

AG Grid is made available in the global **npm** library, so can be easily installed into our Angular applications. Our first task is to create a new Angular application in which to showcase AG Grid and then to install it.

First, we create a new Angular application by the following command. Note that we specify `C:\` as an example of the location where the new application might be created – you can create it at any suitable location on your system.

```
C:\> ng new datagrid
```

Next, we navigate into the newly created **datagrid** folder and add the AG Grid library into the new application by the following command. As usual, we append the `--save` flag to request that the file **package.json** is updated with the newly installed library.

```
C:\datagrid> npm install ag-grid-angular --save
```

Note: Choose here to build the demonstration application in a new Angular project rather than within our existing **BizFE** structure. If you prefer to integrate it within **BizFE**, you should create a new component and develop the following content there, rather than in **AppComponent**.

Do it now! Create the new Angular application and install the **ag-grid-angular** package into it, as described above

One of the features of the Data Grid is that it works on the entire collection of data, rather than a single page as is produced by our current back-end API. In order to feed data to the grid, we need to add a new endpoint to the API that returns the entire dataset – without any reference to page size or page number parameters. The code that follows provides this endpoint, which is essentially a copy of the existing code to fetch a collection of businesses, but at a new endpoint and returning all businesses in the MongoDB collection.

File: biz\app.py

```

...

@businesses_bp.route("/api/v1.0/allbusinesses",
                    methods=['GET'])
def get_all_businesses():
    data_to_return = []
    for business in businesses.find():
        business['_id'] = str(business['_id'])
        for review in business['reviews']:
            review['_id'] = str( review['_id'] )
        data_to_return.append(business)

    return make_response( jsonify( data_to_return ), 200 )

...

```

Do it now!	Add a new endpoint to the back-end API to retrieve the complete collection of businesses, as presented in the code box above.
-------------------	---

18.2 Using AG Grid

With the back-end facility to return the entire collection in place, we can now concentrate on the front-end app that will incorporate the AG Grid component.

18.2.1 Building the Web Service

The first task is to provide the Web Service that will fetch data from the back-end and return it to the front-end application. The Web Service requires only a single method **getBusinesses()** which makes an `http.get()` request to the new endpoint and returns the Observable from which the businesses collection data can be retrieved.

File: src/app/web.service.ts

```
import { HttpClient } from "@angular/common/http";
import { Injectable } from "@angular/core";

@Injectable()
export class WebService {

  constructor( private http: HttpClient) { }

  getBusinesses() {
    return this.http.get<any>(
      'http://localhost:5000/api/v1.0/allbusinesses');
  }
}
```

**Do it
now!**

Add the Web Service to the Angular application to make a GET request to the new endpoint, as demonstrated in the code box above.

18.2.2 A Basic Grid

As a first step in using the **AG Grid** library, we modify the App Component TypeScript file to import the required packages that have been made available by installing the **ag-grid-angular** library, as shown in the code box below.

File: src/app/app.component.ts

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { WebService } from '../web.service';
import { AgGridAngular } from 'ag-grid-angular';
import { ColDef } from 'ag-grid-community';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet, AgGridAngular],
  providers: [WebService],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'datagrid';
}
```

Next, we identify the fields to be presented as columns in the grid and declare a property **data** that will store the data to be retrieved from the back-end via the Web Service. Note that even though the API returns all fields in each document, the **headings** definition selects only those fields that are to be included in the grid – other fields retrieved from the back-end will be ignored.

File: src/app/app.component.ts

```
...

export class AppComponent {

    headings: ColDef[] = [
        { field: "name" },
        { field: "town" },
        { field: "rating" }
    ];

    data: any = [];
}
```

To populate the data grid, we first make the Web Service available by injecting it as a parameter to the component constructor. Then, we specify the **ngOnInit()** method that runs automatically when the Component is loaded and make a call to the Web Service **getBusinesses()** method that returns the complete collection. This request populates the **data** property of the grid without any modification required to the API response.

File: src/app/app.component.ts

```
...

export class AppComponent {

    ...

    constructor(private webService: WebService) {}

    ngOnInit() {
        this.webService.getBusinesses()
            .subscribe( (response) => {
                this.data = response;
            });
    }
}
```

Do it now! Complete the file ***app.component.ts*** using the code provided in the code boxes above.

With the TypeScript specification in place, we can move to the HTML and CSS elements of the App Component. First, in the HTML template, we replace the current contents of ***app.component.html*** with the **`<ag-grid-angular>`** tag presented in the following code box. Note now the **`rowData`** and **`columnDefs`** attributes of the **`ag-grid-angular`** element are bound to the previously established values of the component **`data`** and **`headings`** properties.

File: src/app/app.component.html

```
<ag-grid-angular
  [rowData]="data"
  [columnDefs]="headings" />
```

Do it now! Replace the default content of ***app.component.html*** with the code provided in the code box above.

Installing the **`ag-grid-angular`** library also makes a number of stylesheet files available within the **`node_modules`** folder for the component. We can include these in the application by adding their paths to the **`styles`** entry within ***angular.json***.

File: angular.json

```
...

"styles": [
  "src/styles.css",
  "./node_modules/ag-grid-community/styles/ag-grid.css",
  "./node_modules/ag-grid-community/styles/ag-theme-quartz.css",
],

...
```


Do it now! Add the style entries shown above to **angular.json**. Note that additional stylesheets providing different themes are also available within the **node_modules** folder and you can experiment with these to change the appearance of your grid.

Once the style are added we can apply them to the grid by applying the `ag-theme-quartz` class to the grid component. We also apply a height of 500px to the element, leaving the width to occupy as much space as is required.

File: src/app/app.component.html

```
<ag-grid-angular
  class="ag-theme-quartz"
  style="height: 500px"
  [rowData]="data"
  [columnDefs]="headings" />
```

Do it now! Apply the **class** and **style** attributes demonstrated in the code box above to **app.component.html**.

Finally, we need to modify **main.js**, which is the entry point of the Angular application, to add provision for the **HttpClient**, reconfiguring the file to take account of the existing code.

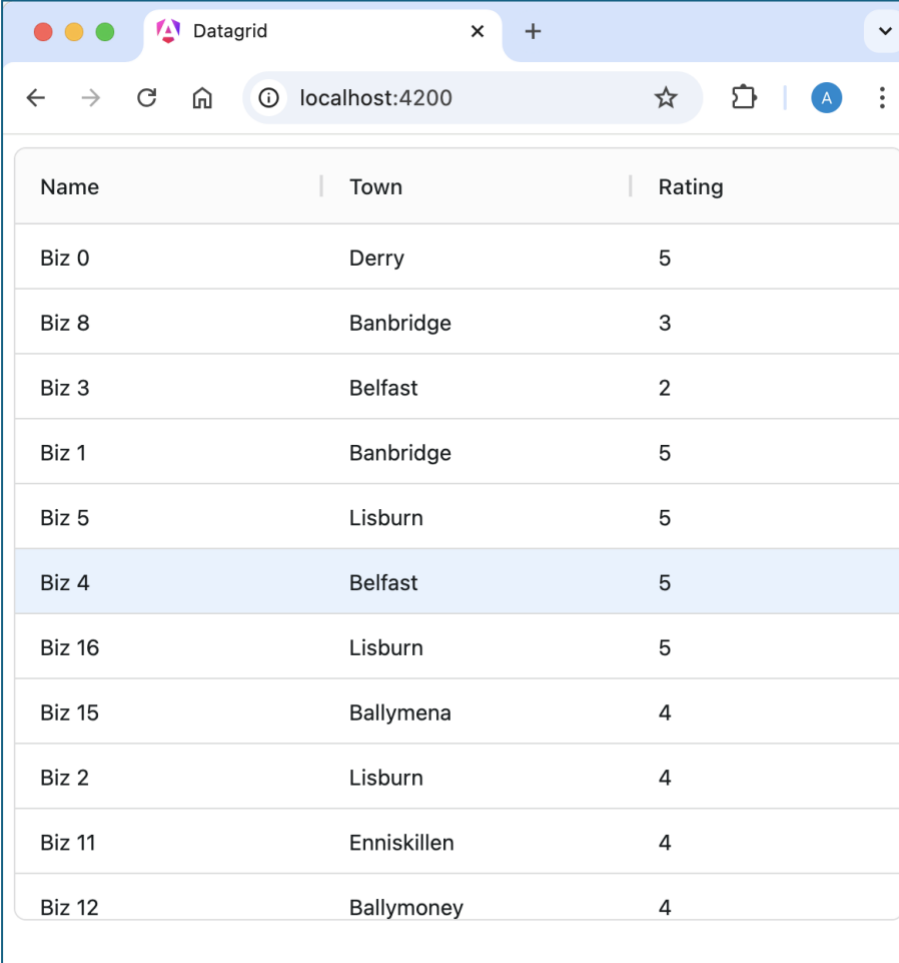
File: src/main.js

```
import { bootstrapApplication } from '@angular/platform-browser';
import { appConfig } from './app/app.config';
import { AppComponent } from './app/app.component';
import { provideHttpClient } from '@angular/common/http';

bootstrapApplication(AppComponent, {
  providers: [
    provideHttpClient(),
    appConfig.providers
  ]
}).catch((err) => console.error(err));
```

Do it now! Update *main.js* with the highlighted code above.

Now we can view the data grid in action, so we run the application using ng serve and see the grid component as shown in Figure 18.23, below. If your application is already running, you should stop it first (by CTRL-C) and then re-start it to make sure that the stylesheets are loaded.



Name	Town	Rating
Biz 0	Derry	5
Biz 8	Banbridge	3
Biz 3	Belfast	2
Biz 1	Banbridge	5
Biz 5	Lisburn	5
Biz 4	Belfast	5
Biz 16	Lisburn	5
Biz 15	Ballymena	4
Biz 2	Lisburn	4
Biz 11	Enniskillen	4
Biz 12	Ballymoney	4

Figure 18.2 Basic AG Grid

Even in its most basic form, the grid contains some useful functionality. As we specified a fixed width, it provides vertical scrolling within the grid so that all rows can be seen. In addition, each of the row heading is clickable, with a click toggling between sorting that field in ascending order, in descending order, and with no sort applied.

Do it now! Run the application with `ng serve` and visit <http://localhost:4200> in a Web Browser. Verify that you see an interactive data grid as illustrated in Figure 18.2, above. Experiment with clicking on the column headings to demonstrate dynamic sorting on any selected field and see how the grid scrolls vertically to display the entire collection of businesses.

18.3 Configuring Additional Functionality

The **AG Grid** component already provides a useful means of browsing a large collection of data, but there are many additional elements of functionality and presentation that we can specify.

18.3.1 Filtering

Filtering a table of data on a selected column is when we allow the user to provide a value to be used as the filter and only rows where the column matches that value (either partially or fully) are displayed. In AG Grid, we can specify a partial match filter for any column by adding the `filter: true` attribute to the definition of the heading. This is demonstrated in the code box below where we add a filter to the town column.

File: src/app/app.component.ts

```
...

export class AppComponent {

  headings: ColDef[] = [
    { field: "name" },
    { field: "town", filter: true },
    { field: "rating" }
  ];

  data: any = [];

  ...
}
```

The effect of adding the filter can be seen from Figure 18.3 below. Here, we provide the text “Ball” in the **town** filter, resulting in only towns that include the character sequence “ball” being displayed. All towns where the name does not include the sequence “ball” are hidden from view.

Note that the filter updates the display in real time as the user enters the filter value and that the column value does not have to begin with the filter sequence. Hence, entering “*r*” as the filter string displays Businesses in Derry, Banbridge, Coleraine, Lisburn and Newry, while extending the sequence to “*ri*” immediately limits the set of businesses displayed to those in Ban**bridge**.

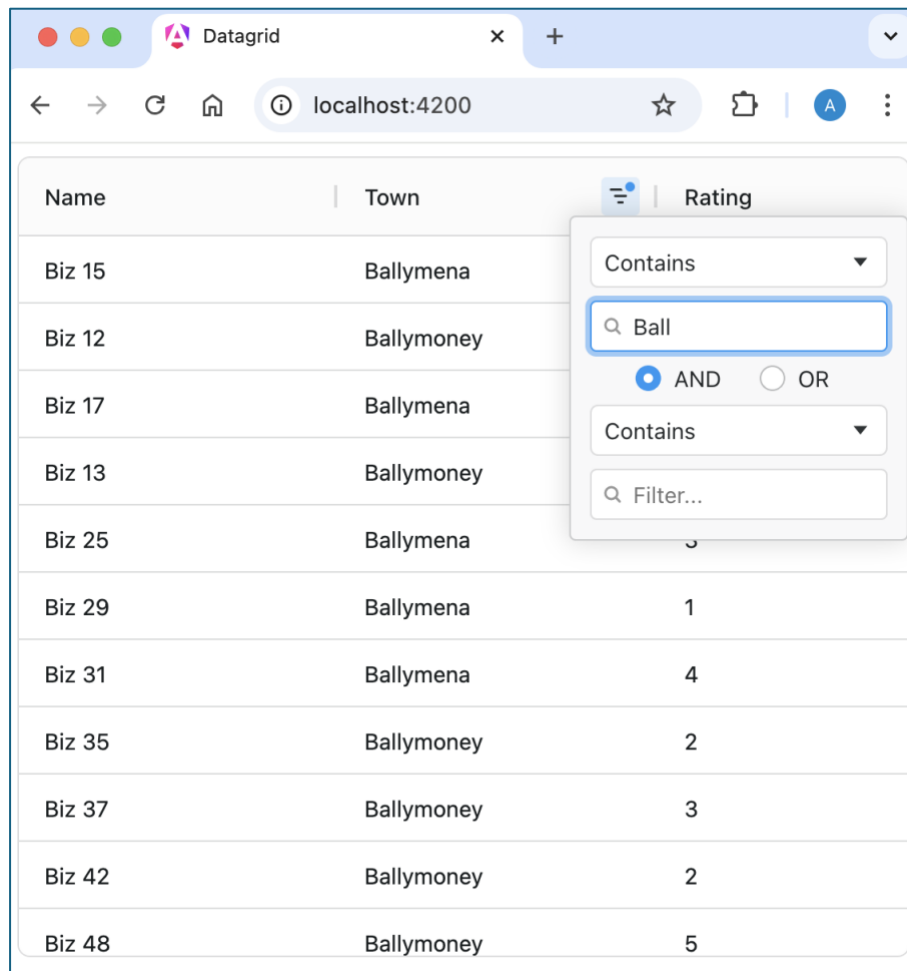


Figure 18.3 Adding a Filter

By default, the filter is hidden and appears when the icon to the right of the column heading is displayed. However, by adding `floatingFilter: true` to the heading definition, we can request that the filter is permanently displayed below the column heading.

File: src/app/app.component.ts

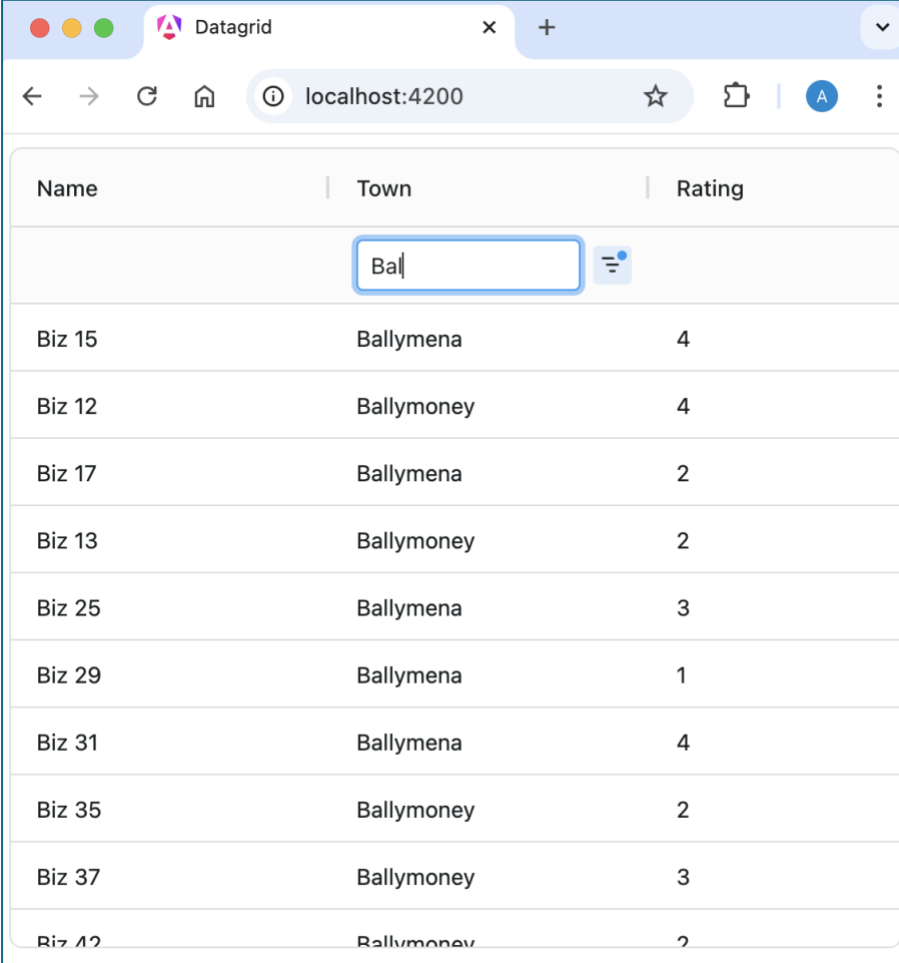
```
...

export class AppComponent {

  headings: ColDef[] = [
    { field: "name" },
    { field: "town", filter: true, floatingFilter: true },
    { field: "rating" }
  ];

  data: any = [];
  ...
}
```

The floating filter applied to the town column is shown in Figure 18.4, below.



Name	Town	Rating
	Bal	
Biz 15	Ballymena	4
Biz 12	Ballymoney	4
Biz 17	Ballymena	2
Biz 13	Ballymoney	2
Biz 25	Ballymena	3
Biz 29	Ballymena	1
Biz 31	Ballymena	4
Biz 35	Ballymoney	2
Biz 37	Ballymoney	3
Biz 12	Ballymoney	2

Figure 18.4 Floating Filter

Do it now! Add a filter to the town field as described in this section. Try both the fixed and floating filter options and determine which you prefer to use going forward. Try adding additional filters for other fields and see how they can be combined.

18.3.2 Column Options

By default, the column names are derived automatically from the field name by capitalising it – hence our fields `name`, `town` and `rating` become the headings **Name**, **Town** and **Rating**. In addition, if the field name contains multiple words in camelCase, AG Grid will split and capitalise such that a field name of `myTopScore` will become the heading **My Top Score**.

However, sometimes we want to use a custom heading that is unrelated to the field name and AG Grid provides for this by the `headerName` attribute that can be specified as part of the header definition. In the code box that follows, we can see the `num_employees` field added to the grid, but with a heading of **Workforce**.

The code box below also demonstrates how to access data values that are embedded within the data structure. Our specification for a business includes the gross profit or loss for each of the past three years as shown below.

```
"profit" : [
  {
    "year" : "2022",
    "gross" : 50000
  },
  {
    "year" : "2023",
    "gross" : -10000,
  },
  {
    "year" : "2024",
    "gross" : 15000
  }
]
```

In order to access the individual values for gross profit, AG Grid provides the `valueGetter` attribute that allows us to access such fields using notation such as `profit[0].gross` for the 2022 value, `profit[1].gross` for the 2023 value, and so on.

File: src/app/app.component.ts

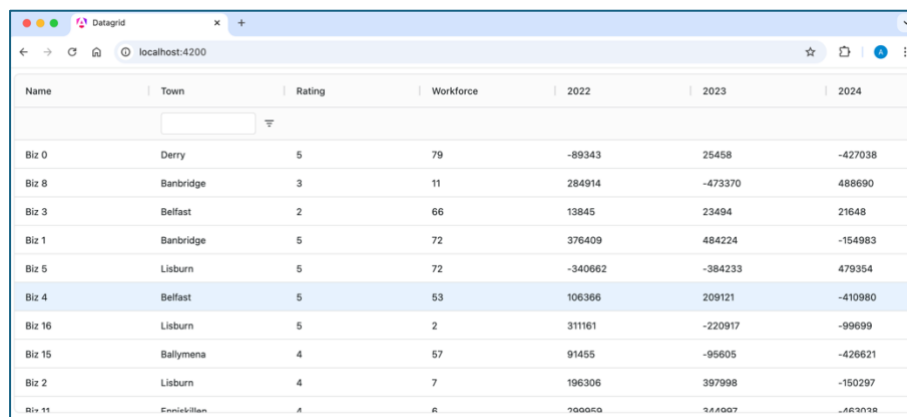
```
...

export class AppComponent {

  headings: ColDef[] = [
    { field: "name" },
    { field: "town", filter: true, floatingFilter: true },
    { field: "rating" },
    { field: "num_employees", headerName: "Workforce" },
    { valueGetter: "data.profit[0].gross", headerName: "2022" },
    { valueGetter: "data.profit[1].gross", headerName: "2023" },
    { valueGetter: "data.profit[2].gross", headerName: "2024" }
  ];

  ...
}
```

Adding these fields to the headings definition results in the expanded grid as shown in Figure 18.5, below.



Name	Town	Rating	Workforce	2022	2023	2024
Biz 0	Derry	5	79	-89343	25458	-427038
Biz 8	Banbridge	3	11	284914	-473370	488690
Biz 3	Belfast	2	66	13845	23494	21648
Biz 1	Banbridge	5	72	376409	484224	-154983
Biz 5	Lisburn	5	72	-340662	-384233	479354
Biz 4	Belfast	5	53	106366	209121	-410980
Biz 16	Lisburn	5	2	311161	-220917	-99699
Biz 15	Ballymena	4	57	91455	-95605	-426621
Biz 2	Lisburn	4	7	196306	397998	-150297
Biz 11	Enniskillen	4	6	700000	344007	-463036

Figure 18.5 Custom Headers and Embedded Data Fields

Now that the grid has a larger number of fields, we can also demonstrate the horizontal scrolling provided by reducing the width of the browser window and observing that the grid only displays those columns for which there is space and makes those outside of that area available by scrolling. Figure 18.6 demonstrates a situation where there is enough horizontal space to show four columns with the others available by scrolling to the left or to the right.

Town	Rating	Workforce	2022
Derry	5	79	-89343
Banbridge	3	11	284914
Belfast	2	66	13845
Banbridge	5	72	376409
Lisburn	5	72	-340662
Belfast	5	53	106366
Lisburn	5	2	311161
Ballymena	4	57	91455
Lisburn	4	7	196306
Enniskillen	4	6	200050

Figure 18.6 Horizontal Scrolling

Do it now! Add the additional fields to the headings definition as shown in the code box above and see the horizontal scrolling in action.

18.3.3 Pagination

Pagination is a commonly found feature in data grid components and AG Grid provides a very flexible implementation based around three component attributes described as follows.

pagination	takes the value true if pagination is to be enabled, and false otherwise
paginationPageSize	the initial number of rows to display on a page. Should be one of the values specified in the paginationPageSizeSelector list
paginationPageSizeSelector	the list of possible page sizes offered to the user

The following code box shows values for these attributes initialized as properties of the App Component.

File: src/app/app.component.ts

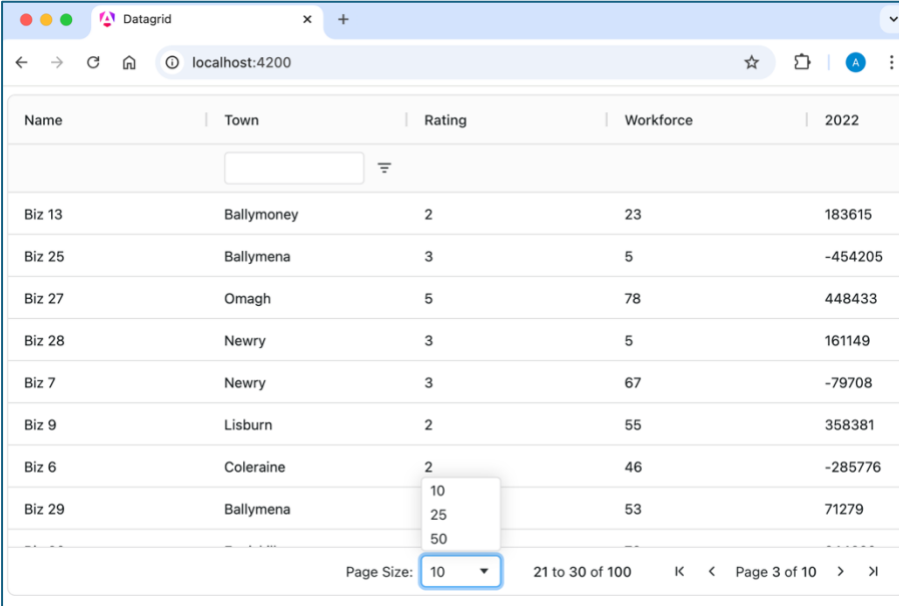
```
...  
  
data: any = [];  
  
pagination = true;  
paginationPageSize = 10;  
paginationPageSizeSelector = [10, 25, 50];  
  
...
```

Next, in the HTML template, we bind each of the ag-grid-angular attributes to the corresponding component property

File: src/app/app.component.html

```
<ag-grid-angular  
  class="ag-theme-quartz"  
  style="height: 500px"  
  [pagination]="pagination"  
  [paginationPageSize]="paginationPageSize"  
  [paginationPageSizeSelector]="paginationPageSizeSelector"  
  [rowData]="data"  
  [columnDefs]="headings" />
```

Now, when we re-run the application, we see the pagination controls added below the grid, as shown in Figure 18.7, below.



Name	Town	Rating	Workforce	2022
Biz 13	Ballymoney	2	23	183615
Biz 25	Ballymena	3	5	-454205
Biz 27	Omagh	5	78	448433
Biz 28	Newry	3	5	161149
Biz 7	Newry	3	67	-79708
Biz 9	Lisburn	2	55	358381
Biz 6	Coleraine	2	46	-285776
Biz 29	Ballymena	5	53	71279

Page Size: 10 21 to 30 of 100 K < Page 3 of 10 > X

Figure 18.7 Adding Pagination

Do it now!

Add pagination to the data grid by making the modifications to **`app.component.ts`** and **`app.component.html`** shown in the code boxes above. Try different page size options and confirm their operation. Also combine pagination with filtering and verify that only the rows matching the filter are counted in the pagination calculation.

18.3.4 Styling

AG Grid comes with a number of themes that can be enabled by linking to their stylesheet files in **`angular.json`**, but each of these is fully customizable by overwriting their SS rules in a local component CSS file.

The code box below illustrates this by providing new values for the CSS rules for the **`ag-theme-quartz`** theme that we specified to be used in the **`<ag-grid-angular>`** tag in **`app.component.html`**.

File: src/app/app.component.css

```
.ag-theme-quartz {
  --ag-foreground-color: rgb(126, 46, 132);
  --ag-background-color: rgb(249, 245, 227);
  --ag-header-foreground-color: rgb(204, 245, 172);
  --ag-header-background-color: rgb(209, 64, 129);
  --ag-odd-row-background-color: rgb(0, 0, 0, 0.03);
  --ag-header-column-resize-handle-color: rgb(126, 46, 132);

  --ag-font-size: 17px;
  --ag-font-family: monospace;
}
```

Adding this code to app.component.css results in the grid style that we see in Figure 18.8, below.

Name	Town	Rating	Workforce ↓	2022
Ba				
Biz 91	Banbridge	2	94	-348696
Biz 71	Banbridge	4	90	249521
Biz 31	Ballymena	4	88	199037
Biz 63	Banbridge	4	88	305024
Biz 47	Banbridge	2	83	19469
Biz 93	Ballymena	4	82	-346934
Biz 86	Ballymoney	3	81	452006
Biz 19	Banbridae	1	76	391326

Page Size: 25 1 to 25 of 37 Page 1 of 2

Figure 18.8 Custom Style

Do it now!

Add code to **app.component.css** as shown above and modify the presentation style of your grid. Manipulate the colour values to create your own theme. You can return to the original theme at any time by removing the content from **app.component.css** and reverting to the default values.

18.3.5 Other Extensions

AG Grid contains many other features (even in the free tier) that can provide additional functionality and display options, including those identified below.

Cell components	Allow HTML elements such as buttons, checkboxes and images to be displayed in grid cells
Row and value selection	Add a column of checkboxes to the grid so that the user can indicate those in which they are interested. The selected rows can then be extracted using the AG Grid API for further processing.
Editing	AG Grid supports editing of the cell values, and these can be saved back to the component data structure. If required, changes can be POSTed back to the database via the Web Service
Export	Data can be exported to CSV format by using an AG Grid API call

Information and examples of these and many other facilities are available in the comprehensive online documentation at <https://www.ag-grid.com/angular-data-grid/getting-started/>.

18.4 Further Information

- <https://taglineinfotech.com/why-use-angular/>
Why use Angular in 2024?
- https://medium.com/@simon.sharp_25406/the-state-of-angular-by-the-google-angular-team-9ea8f6571c5f
The State of Angular by the Google Angular Team
- <https://angulardive.com/blog/angular-services-vs-components-understanding-when-to-use-each/>
Angular Components v Services
- <https://angular.dev/tools/libraries>
Overview of Angular Libraries
- <https://angular.dev/tools/libraries/creating-libraries>
Creating Libraries
- <https://www.ag-grid.com/>
AG Grid Home Page
- <https://www.ag-grid.com/angular-data-grid/compatibility/>
AG Grid and Angular Version Compatibility Table
- <https://www.youtube.com/@agGrid>
Ag Grid YouTube Channel
- <https://thinkster.io/tutorials/fundamentals-of-ag-grid-with-angular>
Fundamentals of AG Grid with Angular
- <https://www.linkedin.com/pulse/elevate-your-angular-applications-ag-grid-guide-examples-jaiswal/>
Elevate Your Angular Applications with AG Grid: A Comprehensive Guide with Examples