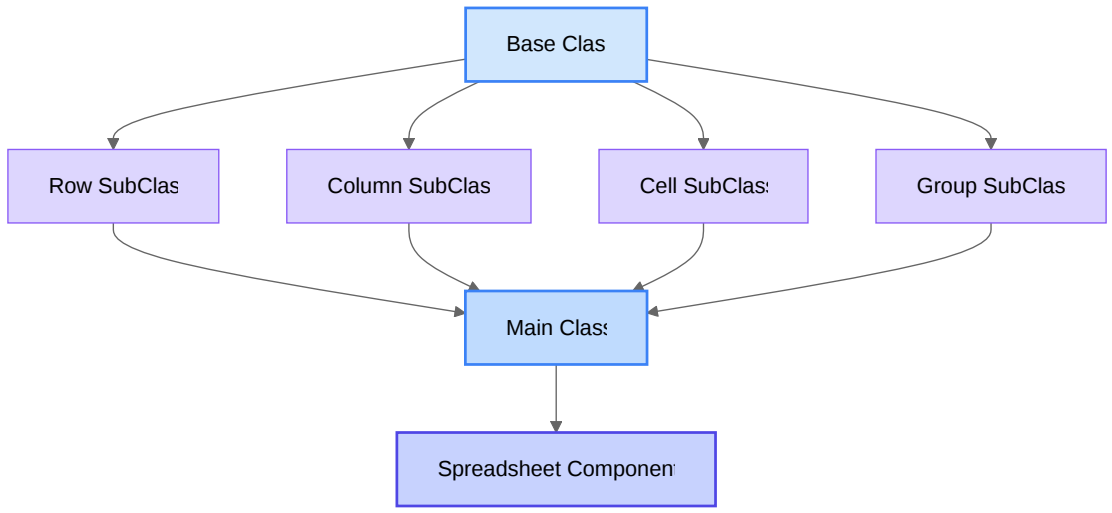
Angular Spreadsheet Component

Architecture Overview

Class Hierarchy



Base Class

Contains cross-function utilities shared across all SubClasses to prevent TypeScript errors

SubClasses

Specialized Row, Column, Cell, and Group classes that all extend the Base Class

Main Class

Extends all SubClasses (Row, Column, Cell, Group) to compose a comprehensive spreadsheet functionality

Spreadsheet Component Implementation

Angular Integration

Spreadsheet Component handles Angular-specific logic including:

- **Input/Output** - Data binding and events
- **Event Binding** - User interactions
- **View/Content Child** - Template references
- **Change Detection** - Performance optimization

```
spreadsheet/
├── components/
│   ├── sub-classes/
│   │   ├── base.ts       // Declare abstract cross-function
│   │   ├── cell.ts       // Cell management
│   │   ├── column.ts     // Column operations
│   │   ├── group.ts      // Grouping functionality
│   │   ├── main.ts       // Combines all SubClasses
│   │   └── row.ts        // Row handling
│   ├── sub-components/
│   │   ├── cells/          // Different cell type implementation
│   │   └── virtual-scroll/ // Performance optimized scrolling
│   └── sub-templates/    // Angular template definitions
├── helpers/              // Utility functions
│   ├── group.ts    // Group processing
│   ├── sort.ts     // Sorting functionality
│   ├── search.ts   // Search operations
│   ├── paste.ts    // Clipboard handling
│   ├── keyboard.ts // Keyboard event handling
│   └── event-emit-controller.ts // Event batching optimization
├── spreadsheet.pug
└── spreadsheet.scss
```

Helper Functions

Specialized utilities for data manipulation, sorting, grouping, searching, and clipboard operations

Event Optimization

Throttled event emission to batch updates and minimize Angular change detection cycles

Sub-Classes Architecture

Using TS-Mixer for Composition

The TS-Mixer Pattern

The Angular Spreadsheet uses a composition pattern with **TS-Mixer** to organize code effectively.

```
import { Mixin } from 'ts-mixer';

class Foo {
    protected makeFoo() {
        return 'foo';
    }
}

class Bar {
    protected makeBar() {
        return 'bar';
    }
}

class FooBar extends Mixin(Foo, Bar) {
    public makeFooBar() {
        return this.makeFoo() + this.makeBar();
    }
}
```

Class Structure Benefits

- **Type Safety** - Base Class prevents TypeScript errors when SubClasses access each other's methods
- **Code Organization** - Logically separates functionality into specialized SubClasses
- **Maintainability** - Makes the codebase easier to understand and modify
- **Composition Pattern** - Main Class extends all SubClasses to create a unified API

Class Structure Details

Base Class

Provides cross-function utilities that can be accessed by all SubClasses. This prevents TypeScript errors when methods need to call functions defined in other SubClasses.

SubClasses

Each SubClass (Row, Column, Cell, Group) extends the Base Class and focuses on a specific aspect of spreadsheet functionality. This modular approach improves code organization and maintainability.

- **Row SubClass:** Row-specific operations
- **Column SubClass:** Column handling
- **Cell SubClass:** Cell management
- **Group SubClass:** Grouping logic

Main Class

The Main Class extends all other SubClasses using the TS-Mixer pattern, combining all functionality into a single comprehensive class. This approach uses composition over inheritance to avoid the limitations of TypeScript's single inheritance model.

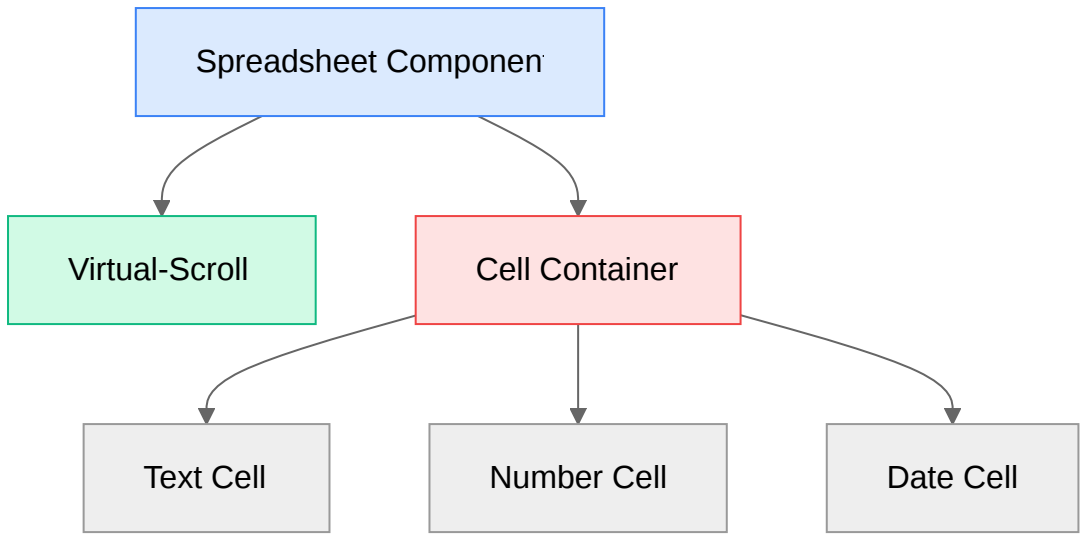
Spreadsheet Component

Handles Angular-specific logic like Input/Output bindings, Event Binding, View/Content Child, and other Angular features. It integrates with the Main Class to provide the complete spreadsheet functionality.

Sub-Components & Rendering

Efficient Field Rendering with Virtual Scroll

Sub-Components Architecture



Virtual-Scroll

Handles lazy-rendering of large datasets by only rendering visible rows and columns, drastically improving performance.

Cells Container

Manages different cell types and coordinates the rendering strategy for various field formats.

Dynamic Component Loading

Uses **viewContainerRef** to dynamically create embedded views based on field type (Text, Date, Number). Components are created on-demand to optimize memory usage.

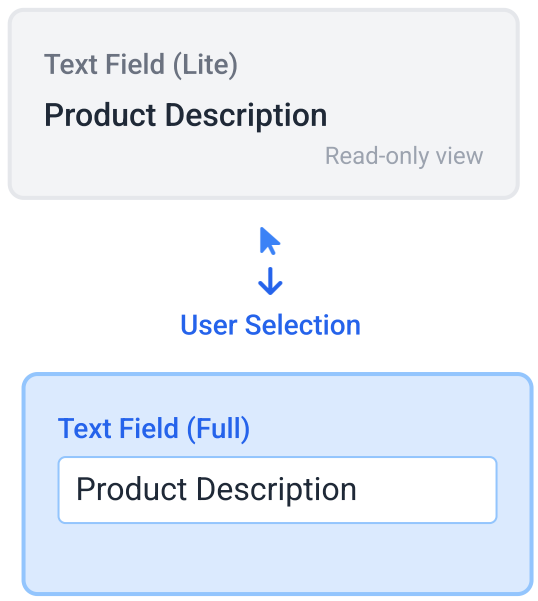
```
// Example of dynamic cell component creation
@Component({
  selector: 'app-cell-container',
  template: ``
})
export class CellContainerComponent {
  @ViewChild('cellContainer', { read: ViewContainerRef })
  cellContainer: ViewContainerRef;

  renderCell(field: Field, isEditing: boolean) {
    const componentType = isEditing
      ? this.getFullCellComponent(field.type)
      : this.getLiteCellComponent(field.type);

    this.cellContainer.clear();
    const ref = this.cellContainer.createComponent(componentType);
    ref.instance.data = field.data;
  }
}
```

Lite & Full Rendering Strategy

Optimized Rendering Approach

The spreadsheet uses a dual rendering strategy to optimize performance while maintaining interactivity:

- **Lite View:** Simple, lightweight rendering for browsing and scrolling
- **Full View:** Feature-rich, interactive component when cell is selected for editing



Lite Components

Minimal DOM elements, optimized for rendering thousands of cells with excellent scrolling performance.

Full Components

Rich interactive elements with validation, formatting options, and type-specific controls.