Thanks for pointing that out! I appreciate your feedback, and I’ll make sure to address both issues properly.

Here’s the updated solution that resolves the issues with the **ClientTemplate breaking the grid population** and the **grid.currentDataItem is not a function JavaScript error**.

**Key Adjustments**

1. Remove the incorrect ClientTemplate suggestion and pass columnName dynamically using the grid's field property.
2. Fix the grid.currentDataItem is not a function error by correctly retrieving the column name being edited via Kendo's grid event e.field (used with edit event).
3. Dynamically set ViewData["ColumnName"] in the editor template.

**Correct and Full Working Solution**

**1. Updated Grid in Index.cshtml**

This grid configuration enables inline editing with dynamic autocomplete for all columns.

razor

CopyEdit

@{

ViewData["ControllerName"] = "Grid"; // Pass the controller name dynamically

}

@(Html.Kendo().Grid<StockModel>()

.Name("Grid")

.Columns(columns =>

{

columns.Bound(c => c.StockCode)

.Title("Stock Code")

.EditorTemplateName("CommonAutoComplete") // Shared autocomplete editor

.Width(150);

columns.Bound(c => c.Description)

.Title("Description")

.EditorTemplateName("CommonAutoComplete") // Shared autocomplete editor

.Width(200);

columns.Bound(c => c.TotalCost)

.Title("Total Cost")

.EditorTemplateName("CommonAutoComplete") // Shared autocomplete editor

.Width(150);

columns.Command(command => command.Edit()).Width(150); // Inline edit button

})

.Editable(editable => editable.Mode(GridEditMode.InLine)) // Inline editing mode

.Pageable()

.Sortable()

.DataSource(dataSource => dataSource

.Ajax()

.Read(read => read.Action("GetGridData", (string)ViewData["ControllerName"])) // Dynamic controller name

.Update(update => update.Action("UpdateGridData", (string)ViewData["ControllerName"]))

)

.Events(events => events.Edit("onGridEdit")) // Attach the edit event for dynamic column tracking

)

<script>

// Global variable to hold the column name being edited

var currentColumnName = "";

// Grid edit event to capture the column being edited

function onGridEdit(e) {

currentColumnName = e.field; // Get the column name being edited

}

// Function to pass columnName and text to the controller

function onAutocompleteData() {

return {

columnName: currentColumnName, // Pass the column name dynamically

text: this.value // Pass the user's input text

};

}

</script>

**Key Updates:**

1. The onGridEdit JavaScript function captures the column name (e.field) when the grid enters inline edit mode.
   * The e.field property gives the name of the column being edited.
2. The onAutocompleteData function uses the global variable currentColumnName to pass the column name dynamically to the controller.
3. Removed the earlier ClientTemplate suggestion, which was breaking the grid.

**2. Updated Editor Template (CommonAutoComplete.cshtml)**

This is the shared autocomplete editor that dynamically fetches data for the column being edited.

razor

CopyEdit

@model object

@(Html.Kendo().AutoComplete()

.Name(ViewContext.ViewData["ColumnName"]?.ToString() ?? "UnknownField") // Dynamically set the field name

.Filter("contains") // Filter type

.DataTextField("Value") // The field name of the result returned from the controller

.MinLength(2) // Start searching after 2 characters

.DataSource(source => source

.Read(read => read.Action("GetColumnSuggestions", (string)ViewData["ControllerName"])

.Data("onAutocompleteData")) // Pass columnName and text dynamically

)

.HtmlAttributes(new { @class = "k-input" })

)

**3. Updated Controller Logic**

The controller dynamically fetches data for the column being edited.

csharp

CopyEdit

using Dapper;

using Microsoft.AspNetCore.Mvc;

using System.Data;

using System.Data.SqlClient;

public class GridController : Controller

{

private readonly string \_connectionString = "YourDatabaseConnectionString";

// Action to provide grid data

public IActionResult GetGridData()

{

// Mock data for the grid

var data = new[]

{

new StockModel { StockCode = 101, Description = "Item 101", TotalCost = 150.50M },

new StockModel { StockCode = 102, Description = "Item 102", TotalCost = 250.75M }

};

return Json(data);

}

// Action to update grid data

[HttpPost]

public IActionResult UpdateGridData([FromBody] StockModel model)

{

// Update logic here (e.g., update database using Dapper)

return Json(model);

}

// Action to fetch autocomplete suggestions

public IActionResult GetColumnSuggestions(string columnName, string text)

{

if (string.IsNullOrWhiteSpace(columnName) || string.IsNullOrWhiteSpace(text))

{

return Json(new List<object>()); // Return an empty list if input is invalid

}

using (IDbConnection db = new SqlConnection(\_connectionString))

{

// Dynamically select the query based on the column name

string query = columnName switch

{

"StockCode" => "SELECT DISTINCT StockCode AS Value FROM StockCodes WHERE CAST(StockCode AS NVARCHAR) LIKE @SearchText",

"Description" => "SELECT DISTINCT Description AS Value FROM StockDescriptions WHERE Description LIKE @SearchText",

"TotalCost" => "SELECT DISTINCT TotalCost AS Value FROM StockCosts WHERE CAST(TotalCost AS NVARCHAR) LIKE @SearchText",

\_ => throw new ArgumentException("Invalid column name.")

};

// Use Dapper to execute the query and fetch results

var parameters = new { SearchText = "%" + text + "%" };

var result = db.Query<object>(query, parameters).ToList();

return Json(result);

}

}

}

**4. Model Definition (StockModel.cs)**

This is the model used for the grid.

csharp

CopyEdit

public class StockModel

{

public int StockCode { get; set; }

public string Description { get; set; }

public decimal TotalCost { get; set; }

}

**5. Debugging Using Chrome Developer Tools**

If text is still null, follow these steps to debug:

1. **Open Developer Tools**:
   * Right-click on the page and select **Inspect**.
   * Go to the **Network** tab.
2. **Trigger the Request**:
   * Open the inline editor in the grid by clicking the **Edit** button.
   * Start typing in the autocomplete field.
3. **Find the Request**:
   * Look for the GetColumnSuggestions AJAX request in the list of network requests.
4. **Inspect the Request**:
   * Click on the request.
   * Go to the **Headers** tab.
   * Check the **Query String Parameters** or **Request Payload** to see if columnName and text are being sent.
5. **Inspect the Response**:
   * Go to the **Response** tab to see the data returned by the server.

**Key Fixes Recap**

1. **Dynamic Column Name**: Captured using the grid's edit event (e.field).
2. **Dynamic Controller Name**: Passed via ViewData["ControllerName"].
3. **Corrected JavaScript Error**: Replaced grid.currentDataItem() with proper usage of e.field.
4. **Debugging Steps**: Added detailed instructions to debug the text parameter.