An evaluation of HTML5 components for web-based manipulation of tabular data

Calle Alexandersson

Calle Alexandersson
Spring 2015
Bachelor's thesis, 15 hp
Supervisor: Pedher Johansson
Extern Supervisor: Daniel Hellström
Examiner: Andrew Wallace
Bachelor's program in Computing Science, 180 hp

Abstract

HTML5 is a promising technology that is on its way to becoming a standard for the web. Companies that have built their web application components using plugins now have to move to a entirely new JavaScript environment. One such component is data grids or tables and will be the focus of this report.

In this report I present a proposal for evaluation criteria for tabular components in JavaScript frameworks. Using these criteria, grid components in some of the market leading frameworks are evaluated. Further I will for one of these frameworks present a test implementation and performance test focusing on load time with and without UI Virtualization.

Contents

1 Introduction

For several years now, plugins have been used to handle non-native functionality in the web. But now with the new version of HTML, namely HTML5, plugins are becoming redundant when its key functionality can be done without them. This along with the many drawbacks of plugins have started a trend among companies to switch to plugin free solutions[11]. This trend is pressed on even more by Googles choice of completely disabling plugins such as Java and Silverlight in Google Chrome from September 2015[13]. Companies now has to replace complex parts of their web pages from scratch in to pure HTML/Javascript. Which can be very expensive, especially if web development is not one of the company's areas of work. To take advantage of this, there are several companies that have built frameworks that is aimed at replacing the properties that previously were done via plugins. These companies and frameworks are targeting companies that want to have a plugin free application but not put to much resources into building it.

This report will present a evaluation of some of the market leading frameworks for JavaScript grid components. These will be evaluated by a set of criteria/functionalities derived from looking at current plugins based grid components. All frameworks presented are enterprise sized, with good documentation and customer support.

These frameworks have been evaluated against the requirements presented in Chapter 3. There one of the frameworks have been ranked as the most promising one. It has been implemented as a proof of concept and tested. The test focus on load time for some defined data sets of tabular data with and without UI Virtualization (explained in Section 6.1).

This evaluation has been done in collaboration with Trimma. Trimma is a company developing a web-based business intelligence package named INSIKT[20]. They want to replace their current Silverlight grid with a HTML5 based grid. The requirements in Chapter 3 have been set in collaboration with them and they have been involved in determining the most promising framework for implementation in Chapter 5.

1.1 Goals and Methods

The goals of the report is to evaluate HTML5 frameworks by identifying plugin functionality, these are used as requirements for determining how each framework handles each of these, implementing a proof of concept on one framework, and to perform a performance test on the the same framework as the implementation. This to reach a conclusion, either not to migrate from Silverlight or to recommend a framework to migrate to.

All alternatives, plugins, frameworks and requirements of this report are restricted to grid/table/tabular data only. Other features of plugins and JavaScript frameworks are not to be take into account in this paper. Plugin functionality only refers to the functionality associated with the area of previously stated use.

The reason Silverlight is used as a base for this evaluation and performance test is that Silverlight have been one of the biggest plugin used for this kind of functionality[26]. This as well as being the plugin Trimma uses in their current application (see Appendix Figure 1).

Evaluation

The evaluation can be divided into two parts. The first is to identify functionality for existing plugins (Chapter 3). The second part is to determine how each framework handles each functionality referred to as requirements (Chapter 4).

To identify requirements, the current functionality, look and use of the Silverlight grid component is used as reference. This work is done in collaboration with Trimma. Some requirements are added from a performance perspective. General wishes from Trimma are used to define the technical requirements. The motivation for using a old system as requirements for a new is that Trimmas customers is used to their current functionality. The reason for the desired switch to HTML5 is not primarily to gain new functionality, but to get away from the disadvantages of plugins (Section 2.5) and to offer usage on mobile devices.

The requirements, good documentation and customer support are the basis for which frameworks have been chosen. Therefore big enterprise solutions are chosen in the evaluation since they are more likely to have these two requirements. Which companies that passes this description is assessed by reading their respective homepages.

The evaluation is done by reading documentation, forums and contacting technical support from the framework companies. Each requirement for each framework have been given a status of, *Supported*, *Partial Support*, *Achievable with modification*, or *Not Supported or unknown*. This is summarised in Figure 2 (Chapter 4). This as well as a short summary of their licensing model and support options since those requirements benefit from a more specific answer then *Supported* or *Not Supported*.

Implementation

The most promising framework is chosen together with Trimma, by looking at the summary of the evaluation. A number of requirements that I have deemed interesting to implement is chosen. These are implemented to form a proof of concept (Chapter 5).

Performance testing

A performance test is made on the framework chosen for the implementation (Chapter 6). The performance test measures load time with and without virtual scrolling, (explained in Section 6.1). The grid is built entirely by JavaScript, so the test measures the time by calculating the difference between two timestamps. The first when the JavaScript is initiated, and the second one when it is done loading. This is done 10 times with five different data sets. The test result is then loosely compared to a study with similar data in a Silverlight grid. The size of the data sets is taken from this study for comparison purposes. This purpose of the test it to get a understanding of the performance between HTML5 Grid and a Silverlight Grid.

2 What is HTML?

HTML5 refers not only to the HTML code, but also the associated CSS and JavaScript. No big professional sites consists entirely of HTML code since HTML on its own can not be used for animation or interactivity. In other words the word HTML5 refers to the whole package of HTML5, CSS and JavaScript.

2.1 HTML

HTML stands for HyperText Markup Language and is the World Wide Web's core markup language. In the early days of the web, HTML was primarily designed as a language for semantically describing scientific documents. Its general design, however, has enabled it to be adapted, over the subsequent years, to describe a number of other types of documents and even applications [4].

The last major revision of HTML prior to HTML5, namely HTML4, came out in 1997. Back then most web pages had much simpler layouts and consisted mainly of static images, GIF's and text. Since then, annual Internet traffic has increased by many thousand-folds[18] and the functionality that we demand of the web has completely changed. But after HTML4 was released the development of the web drew to a halt. For over a decade there did not really happen anything new. So when the need for interactive elements, video or audio streaming came there was not any native support for it. To solve this a bunch of plugins where created by different companies to handle such content[19].

2.2 JavaScript

JavaScript is a programming language developed by Netscape to enable Web authors to write interactive sites. Although it shares many of the features and structures of the full Java language, it was developed independently and is a completely different language. JavaScript can interact with the Document Object Model (DOM) of the HTML code, enabling Web authors to spice up their sites with dynamic content[24][1]. Many of the features regarding this report will concern JavaScript

2.3 CSS

CSS stands for Cascading Style Sheets and it defines how HTML elements are to be displayed. CSS is designed primarily to enable the separation of document content from document presentation, including elements such as the layout, colors, and fonts. This separation

of formatting and content reduces complexity and repetition. Instead of having to declare the style of each page it is possible to declare the style across a whole website by a single CSS document. It can also be used to display the web page differently depending on the screen size or device on which it is being viewed[23].

2.4 HTML5

The final version of HTML5 was released October 28, 2014[3]. HTML5 brings a lot of new functionality to the web that was previously handled by complex JavaScript and Plugins. Also it is backwards compatible with older versions of HTML[6]. Some of the new functionality of HTML5 are as follows[2].

- **New semantic elements and attributes** will give pages additional structure and meaning.
- **New APIs** to assist in building web applications. APIs like drag-and-drop, web storage, document editing, offline application and protocols a just a few of the new APIs that will be accessible through JavaScript.
- **Added support for audio and video.**

This along with syntax error handling, specifications for XHTML 1.0, DOM level2, and web forms 2.0 strives to reduce the reliability on third-party plugins and bring all this functionality directly to the browser. All this have been added to address the deficiencies of HTML4 and mature the language into a more capable mobile and web authoring tool[3].

2.5 Plugins

Back when the plugins where created they where the best way to handle non native functionality in HTML. With the plugins it is possible to do a verity of things that just was not possible with HTML alone. Some examples of the plugins where as follows.[7]

- **Adobe's Flash player** - Video, animations, and more.
- **Microsoft Silverlight** - Streaming media, animations, interactive applications.
- **Unity plugin** - Provides 3D graphics.
- **Windows Media Player, QuickTime, or RealPlayer** - All video players.

The new HTML5 is now supporting functionality previously achieved by plugins. But why go through the trouble of moving from well know plugins to a new universal standard. The first reason is as just mentioned, HTML5 is now a global standard. Instead of having performance demanding third-party programs working in the browser, there will only be native HTML5 with JavaScript running. Another reason is that companies like Google is actively working to remove old plugins like Java and Silverlight[13]. This by removing

NPAPI from Google Chrome by September 2015. NPAPI is the API that plugins from the 1990s use to extend the browsers functionality. Google is phasing out NPAPI because it is a big cause of lags, crashes, security incidents, and code complexity. Below follows a further explanation of the biggest problems with plugins[7].

- **Security:** Browser plugins have proven to be more insecure than clean browsers, where Flash and Java plugins have been some of the biggest attack vectors on the web. This is aggravated by the fact that everyone have the same Flash or Java plugin, no matter what browser or operating system. This means that a virus that attacks a plugin works across every browser and operating system.
- **No Sandboxing:** Security problems are made worse because traditional browser plugins written using NPAPI (Netscape Plugin Application Programming Interface) or ActiveX are not sandboxed. This means that plugins have complete access to the entire user account and its operating system permissions. A hole in the plugin gives access to the entire operating system. Meanwhile, browsers render web pages in a sandbox, which is harder to escape.
- **Cross-Platform Problems:** Plugins are created by a single vendor, which means there's only a single implementation and it only runs on the vendor's supported platforms. This means that for example Silverlight do not work on mobile platforms and Linux because Microsoft do not provide it. Another example is Flash that do not work on iOS for the same reason. In both cases, Linux developers or Apple developers can't write their own support for Silverlight or Flash. It's not an open standard like web standards are, where you can have multiple implementations implemented by different people.
- **Stability:** Plug-ins have also been a leading cause of crashes, especially when their crashes brought down entire web browsers. Thankfully, due to Chrome's sandboxing (with PPAPI[5]) and Firefox's plug-in isolation, crashing plug-ins only crash themselves nowadays. There's no way for browser developers to fix these crashes; they have to rely on the plug-in's developers to fix them.

3 Requirement Specification

Functionalities that have something in common have been put as a sub-functionality to a more common head functionality. This to make it easier to understand the each functionality. For example Requirement 1 is Performance, and 1.1 is UI Virtualization, a sub-functionality to performance. Just looking at 1.1 it might be hard to understand what it is about, but because it is under Requirement 1, you at least know it has to do with performance.

The sought component have been weighted against certain functionality/requirements stated below. Each functionality requirement will be denoted by a number that can be referenced later in the report.

3.1 Functional requirements

1. Performance

    1. 1 It should not draw all the lines that are not visible and are outside the scrolling interface (UI Virtualization).
    1. 2 Scrollbar should show that the entire grid is plotted even if new rows are visualized only when they come into view.
    1. 3 Page navigation (pagination) shall not be required to handle large data sets.
    1. 4 For large amounts of data (number of rows), the retrieval of data should happen only if necessary, for example when the user scrolls down in the grid (the data virtualization).

2. Interaction

    2. 1 Selection of rows, columns and cells shall be possible, there different selections can be made with the CTRL and SHIFT. The selection should resemble excels way to make selects [12].
    2. 2 Subtractive markings should also be possible through the CTRL key and click on the already selected cell.
    2. 3 It should be possible to lock so that the marking over certain rows, columns and cells are not possible, e.g. not possible to do a marking over cell A and cell B

3. Cell content

    3. 1 Type of cell should not be limited to the column but may vary in different cells in the same column.

3. 2 A input cell should have the possibility to be disabled, with tooltip explaining why.

3. 3 Ideally, the cell must be very flexible and can be adapted to contain virtually anything.

3. 4 The cell can be adapted to react to focus and left- and right-click.

3. 5 The cell should be updated with content without the need for the entire table to be reloaded.

3. 6 Header cells should be also be able to include icons, e.g. for sorting and filtering.

3. 7 The input cells, i.e. cells with text box, should be limited in what it contains / permits, for example, just numbers.

4. The table component must have flexible style so they can implement there own conditional formatting. The styling should be able to change without reloading the page(change colors etc).

5. Column width

5. 1 The column width shall be set automatically based on content or changed manually by the user via drag and drop.

5. 2 When a table is initialized some columns width should be automatic set, while others are fixed (depending on the formatting rules saved in the report).

5. 3 Cell contents/text in the header cells shall wrap automatically if the column can not contain the content. This applies primarily to column headings

5. 4 Cell contents/text in all cells should be completed with "..." if a column can not contain the content. A tooltip should then show the full content.

6. Appearance

6. 1 Design of the table should be free and current design should be possible to achieve, possibly with some minor exceptions. (See Appendix Figure 1)

7. Contextual menus - It should be possible to integrate contextual menus on the right-click in both cells and outside the table.

8. Indicator

8. 1 A cell can have an indicator that shows that there is a description. such as Excel and their current solution. The description shall be shown as a tooltip (Figure 1)

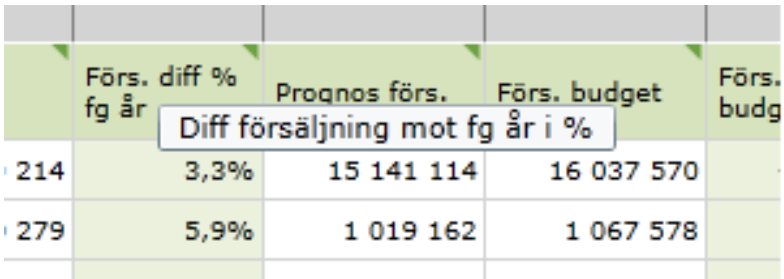


Figure 1: Indicator description in Trimmas current component

9. Input
    - 9. 1 The ability to use tabbing to move forward and and SHIFT to go backwards between cells.
    - 9. 2 The ability to be able to control the tab order based on custom logic.
    - 9. 3 The ability to put the focus in a specific row/cell based on the surrounding code.
10. Color animation at input
    - 10. 1 Cell color turns to red when the value becomes dirty.
    - 10. 2 Cell color turns to green and back to the original color when the value is updated as a result of a write in another cell.
11. Columns
    - 11. 1 Change the order of columns (and preferably rows) with drag and drop.
    - 11. 2 It should be possible to get a preview on where the column will be released and it shall be possible to decide where it should be placed (based on custom logic).
    - 11. 3 The ability to have multiple column headers.
12. Frozen headers - Any number of rows and columns from the top and from the left.
13. Spanning cells - The cell should be able to span multiple rows and / or columns.
14. It shall be adapted for mobile devices and for touch.

3.2 Technical requirements

This functionalities are not code specific, but are requirements surrounding the framework like documentation, integration and support.

15. Development environment / method
    - 15. 1 Active support for JavaScript frameworks such as Knockout and Angular is preferred.
    - 15. 2 Licensing model that allows every developer to work with the code.
    - 15. 3 Well-documented API with examples.
    - 15. 4 Source code is included.
    - 15. 5 Support from the supplier.
16. Integration
    - 16. 1 Should be possible to integrate in a surrounding HTML part where no classic ASP.NET postbacks are done but only JavaScript logic and REST calls to the server.

16. 2 It must also be possible to create multiple tables next to each other dynamically without them causing interference with each other.

16. 3 There is a clear advantage if the component also works in classic ASP.NET environment with postbacks and also supports simple classic ASP.NET approach where properties are added in the code-behind and no Javascript is used at all.

4 Comparison of Frameworks

The frameworks presented in this report have been chosen by the following criteria. Big enterprise solutions with good documentation and customer support. Frameworks not made by big companies have been left out of the comparison because they tend to not have these criteria. Not all frameworks that matches this description has been chosen due to time constraints.

4.1 Frameworks

This section presents the companies behind each framework, as well as their support and licensing options.

Syncfusion - ejGrid

Founded in 2001 and with their headquarters in Research Triangle Park, North Carolina, Syncfusion, Inc. provides a broad range of enterprise-class software components and tools for the Microsoft .NET platform. Since 2013 they also provides JavaScript/HTML5 components for Webb-applications. Syncfusion has more than 10,000 customers, including large financial institutions, Fortune 100 companies, and global IT consultancies[15].

They have an active community forum, online documentation and direct-trac support with 24 hours guaranteed response on business days[17].

Their licensing model have no royalties, run-time, or server deployment fees. A license for their HTML5/JavaScript costs $599 and is per developer based. Every license include a one-year subscription with access to their direct-trac support and new releases. Renewals of subscription cost 60 % of full license list price if more than two weeks have passed after your subscription expires. If the subscriptions has lapsed over 30 days, the reinstatement/renewal will be 75% of the license list price. The prise for renewal within two weeks of expression date is not available on their home page[16].

jQWidgets - jqxGrid

Founded in 2011 jQWidgets specializes in the development of platform independent and cross-browser compatible presentation layer components for building modern web-based applications for PC, Touch and Mobile. They have more than 300,000 software developers from 120 countries using their products[9].

They have a big community forum with daily activity, online documentation and examples. They have standard support, with 20 tickets a year and response within 48 hours. Premium support with 40 tickets a year and response within 24 hours. And platinum support with

120 tickets a year and response within 24 hours. All response times are Mon-Fri except on major holidays[10].

The license is perpetual and allows royalty free distribution with your websites and applications. Each license includes 1-year subscription for platinum, premium or standard technical support plan. They have three different license types, Website License + Standard Support for one developer and one website with a price of $199. Developer License + Premium Support + Source Code for one developer and unlimited websites with a price of $399. And Enterprise License + Platinum Support + Source Code for unlimited number of developers within one organization and unlimited number of websites with a price of $899. All licenses include one year subscription with free updates to newer versions. Renewals of subscription cost 50% of the price of a new license[10].

Webix - DataTable

Webix was released in July, 2013 and is a JavaScript and HTML5 framework for developing cross-platform data-rich mobile and desktop web applications with highly responsive user interfaces[22].

Webix have a community forum and a searchable documentation with some examples. They have three kinds of support options, standard premium and premium plus. Standard have 6 support tickets a year with unlimited 1st month and guaranteed response within 48 hours. Premium have 50 support tickets a year with unlimited 1st month and guaranteed response within 24 hours. Premium plus have unlimited tickets with 10 live chat tickets a year and a guaranteed response within 24 hours. They do not use a external support staff where the support is implemented only by Webix development team[21].

They have three different license types, developer, team and enterprise pack. The developer pack is for one developer with the standard support for a price of $469. The team pack is for five developers with the premium support for a price of $1299. The enterprise pack is for 20 developers with premium plus support for a price of $3999. All licenses include one year subscription for the support and free updates. Renewals of subscription cost between $300 and $1700 depending on support option[21].

4.2 Comparison

Here is a summary of all requirements for each framework (Figure 2). Green means that the framework have support for a requirement. Yellow means partial support, if the framework supports maybe half of the requirement. Red means that it is not supported. Red do not mean that it is impossible to achieve a requirement, just that it is not possible without customisation.

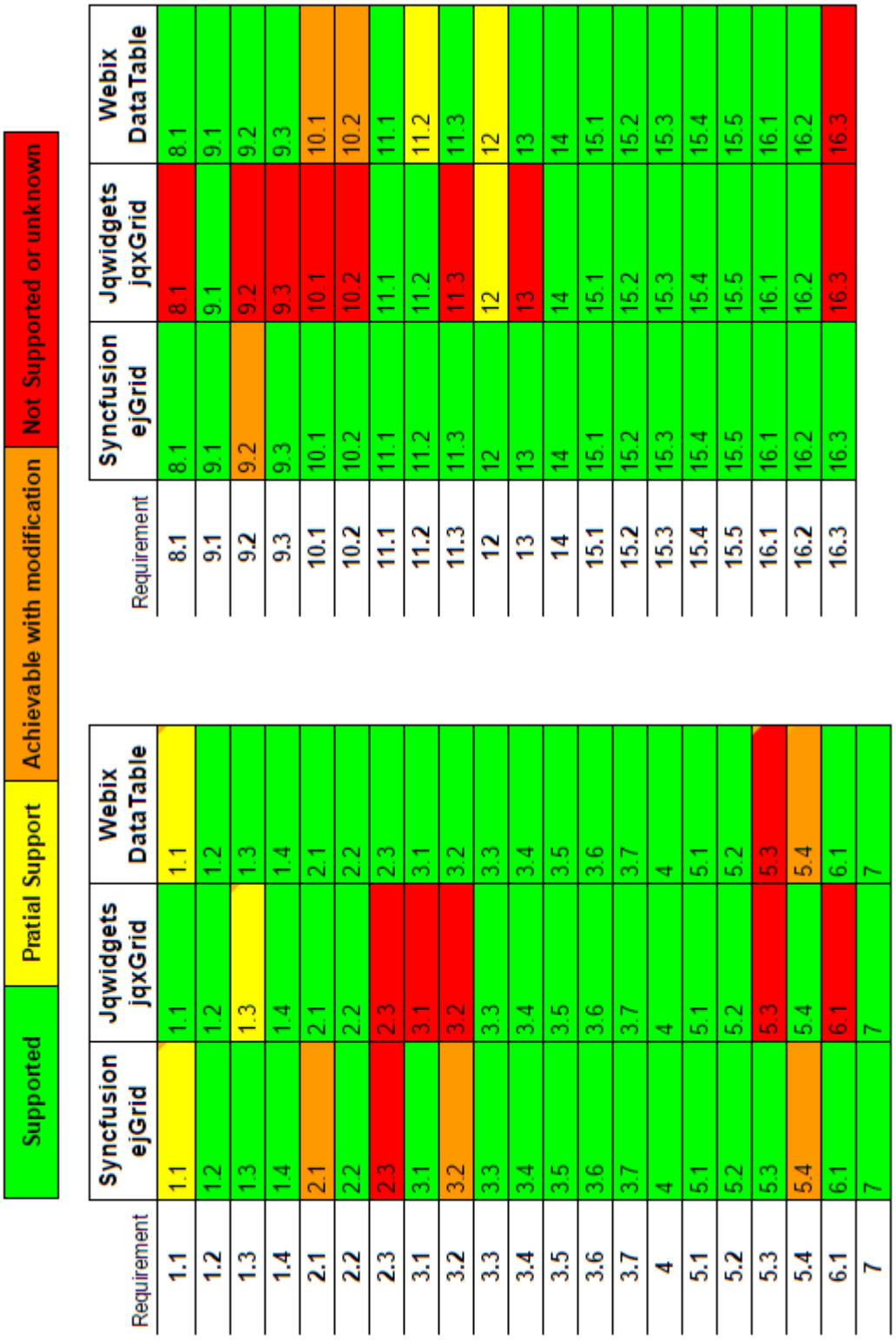


Figure 2: Summary of each requirement for each framework. Grayscale friendlier version in Appendix Figure 2.

5 Implementation

In this chapter a test implementation will be made from the framework chosen as the best from Chapter 4. A number of requirements from Chapter 3 will be chosen and implemented as a proof of concept.

In collaboration with Trimma, **Syncfusion's ejGrid** has been chosen for the implementation because it seems the most promising.

This implementation have been built during a 30-day free trial of Syncfusion's Essential Studio for JavaScript. All imported files in the code examples has been downloaded as a part of the trial. Essential Studio 2015 Volume 1 Service Pack v13.1.0.26 have been used during this implementation[14]. For use of this code or recreation of this example, visit Syncfusion's home page for information regarding licensing[15].

The goal of the implementation is to build a component that has some of the functionalities from all requirements, as a proof of concept. The requirements chosen to be implemented are as follows.

- 2.1 - Selection of rows, columns and cells shall be possible, there different selections can be made with the CTRL and SHIFT.
- 2.2 - Subtractive markings should also be possible through the CTRL key and click on the already selected cell.
- 3.1 - Type of cell should not be limited to the column but may vary in different cells in the same column.
- 11.3 - The ability to have multiple column headers.
- 12 - Frozen headers, Any number of rows and columns from the top and from the left.
- 13 - Spanning cells, The cell should be able to span multiple rows and / or columns.

5.1 Introduction

Figure 3 shows an example of a basic grid with code in listing 5.1. This will be used as a base for the implementation. For full code with surrounding HTML and imports, see A.1.

```
1  $("#Grid").ejGrid({
2    // the datasource "window.gridData" is referred from jsondata.min.js
3    dataSource: window.gridData,
4
5    allowScrolling: true,
6    scrollSettings: { width: 550, height: 300},
7
8    // sets the columns, the field: "tag" corresponds to a tag in window.gridData
         for automatic data binding.
9    columns: [
10         { field: "OrderID", headerText: "", textAlign: ej.TextAlign.Right, width:
               100 },
11         { field: "EmployeeID", headerText: "Employee ID", textAlign: ej.TextAlign.
               Right, width: 100 },
12         { field: "Freight", headerText: "Freight", textAlign: ej.TextAlign.Right,
               width: 100, format: "{0:C}" },
13         { field: "ShipCity", headerText: "Ship City", width: 100 },
14         { field: "ShipCountry", headerText: "Ship Country", width: 100 }
15     ]
16 });
```

Listing 5.1: Basic example of ejGrid.

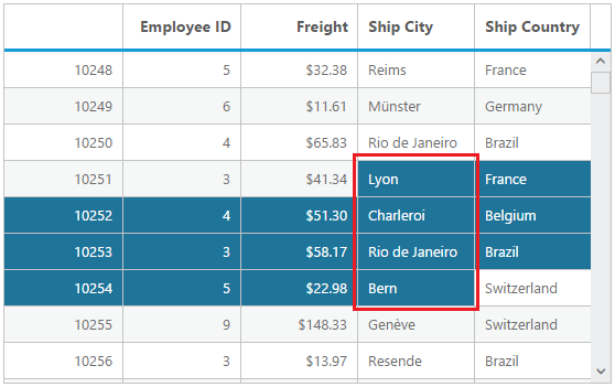
	Employee ID	Freight	Ship City	Ship Country
10248	5	$32.38	Reims	France
10249	6	$11.61	Münster	Germany
10250	4	$65.83	Rio de Janeiro	Brazil
10251	3	$41.34	Lyon	France
10252	4	$51.30	Charleroi	Belgium
10253	3	$58.17	Rio de Janeiro	Brazil
10254	5	$22.98	Bern	Switzerland
10255	9	$148.33	Genève	Switzerland
10256	3	$13.97	Resende	Brazil

Figure 3: Picture of a basic grid.

Requirement 2.1 and 2.2

The default cell multi-select functionality of the grid works as follows.

Look at Figure 4 where a selection have been made by clicking on *Ship City Lyon*, holding SHIFT, and then clicking on *Ship City Bern*. As shown in the figure, cells on both sides are also selected like the selection functionality of a regular HTML page. The excel like selection desired in this report would only select the cells circled by red in the figure.



	Employee ID	Freight	Ship City	Ship Country
10248	5	$32.38	Reims	France
10249	6	$11.61	Münster	Germany
10250	4	$65.83	Rio de Janeiro	Brazil
10251	3	$41.34	Lyon	France
10252	4	$51.30	Charleroi	Belgium
10253	3	$58.17	Rio de Janeiro	Brazil
10254	5	$22.98	Bern	Switzerland
10255	9	$148.33	Genève	Switzerland
10256	3	$13.97	Resende	Brazil

Figure 4: Default multi-select functionality.

To achieve this, a custom selection function have been implemented in listing 5.2 with full example in appendix A.2. The rows marked in yellow are default functionality to enable multi-cell selection. The `cellSelecting function (args)` is the custom overriding function that replaces the default. Basically it looks at the last and currently clicked cells if the SHIFT key is pressed. Then it loops through and makes a custom select cells call to the grid with only the cells wanted.

With this solution, a click on *Employee ID 6*, followed by holding SHIFT and clicking on *Ship City Bern* will result in the desired selection show in Figure 5.

The subtractive markings by holding CTRL is handled natively when enableToggle is set to false.

	Employee ID	Freight	Ship City	Ship Country
10248	5	$32.38	Reims	France
10249	6	$11.61	Münster	Germany
10250	4	$65.83	Rio de Janeiro	Brazil
10251	3	$41.34	Lyon	France
10252	4	$51.30	Charleroi	Belgium
10253	3	$58.17	Rio de Janeiro	Brazil
10254	5	$22.98	Bern	Switzerland
10255	9	$148.33	Genève	Switzerland
10256	3	$13.97	Resende	Brazil

Figure 5: Custom multi-select functionality

```
$("#Grid").ejGrid({

     ...

    allowSelection: true,
    selectionType: "multiple",
    selectionSettings: { selectionMode: ["cell"], enableToggle:false },

    cellSelecting: function (args) {

      if (args.isShiftPressed) {
            this.multiSelectShiftRequest = false;
            if (!args.previousRowCell.length)
                args.previousRowCell = $(this.getRowByIndex(args.
                    previousRowCellIndex[0][0]).find(".e-rowcell:eq(" + args.
                    previousRowCellIndex[0][1][0] + ")"))
            var startIndexFrozenCount = 0, endIndexFrozenCount = 0;
            if (args.currentCell.parents("table").parent("div").hasClass("e-
                movablecontentdiv"))
                endIndexFrozenCount = this.model.scrollSettings.frozenColumns;
            if (args.previousRowCell.prev().parents("table").parent("div").
                hasClass("e-movablecontentdiv"))
                startIndexFrozenCount = this.model.scrollSettings.frozenColumns;
            var startIndex = args.previousRowCell[args.previousRowCell.length -
                1].cellIndex - 1 + startIndexFrozenCount, endIndex = args.
                currentCell[0].cellIndex + endIndexFrozenCount;
            var startRow = args.currentCell.parent()[0].rowIndex, endRow = args.
                previousRowCell.parent()[0].rowIndex;
            var temp;
            if (startIndex > endIndex) {
                temp = startIndex; startIndex = endIndex; endIndex = temp;
            }
            if (startRow > endRow) {
                temp = startRow; startRow = endRow; endRow = temp;
            }
            var cells = [];
            for (var i = startRow; i <= endRow ; i++) {
                var currentRow = [i, []];
                for (var j = startIndex; j <= endIndex; j++) {
                    currentRow[1].push(j);
                }
                cells.push(currentRow);
            }
            flag = false;
            this.selectCells(cells);
            flag = true;
            this.multiSelectShiftRequest = true;
            args.cancel = true;
        }
    },
});
```

Listing 5.2: ejGrid with advanced selection from requirement 2.1 and 2.2.

Figure 6 shows the example grid with different cell types in the same column, multiple column headers, frozen rows and columns and spanning cells. The full code for this example can be seen in Appendix A.2.

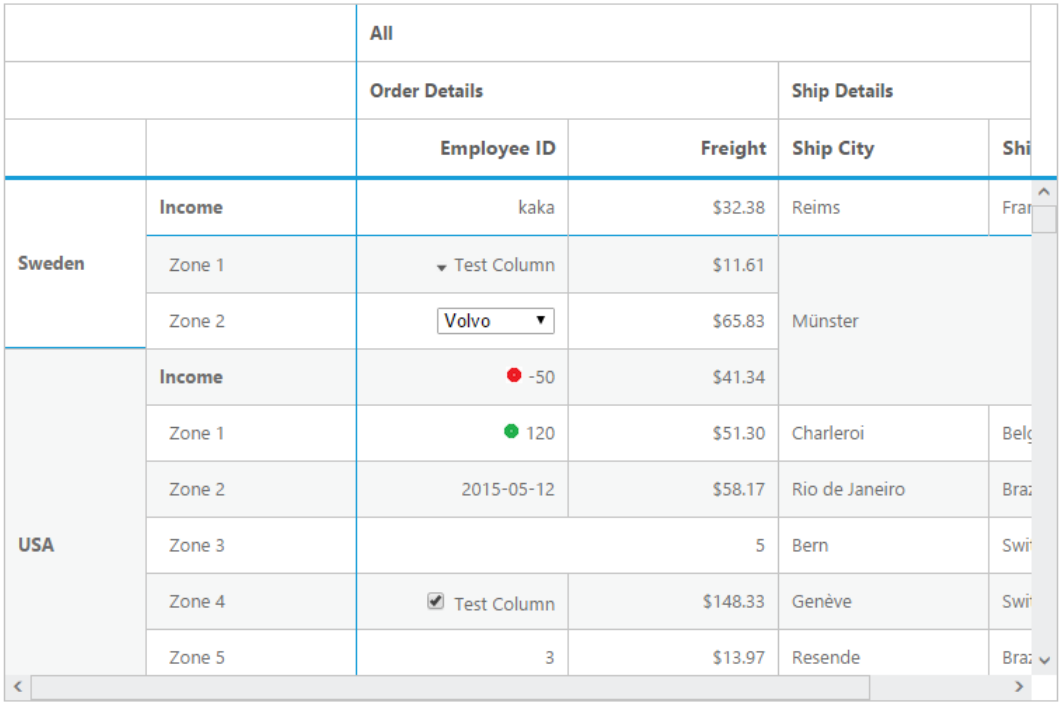


Figure 6: Example grid with different cell types in same column, multiple column headers, frozen rows and columns and spanning cells.

Requirement 3.1

(Here no actual code have been written.) The column is only a normal text cell with custom HTML code. This way anything can be placed in a cell, regardless of column type. This can be seen in Figure 6 where the cells in column *Employee ID* have changed to different texts, images, numbers, dates, check boxes etc. The check box cell is for example filled with the following string `"<input type='checkbox' checked=checked/> Test Column"`, which is normal HTML code. This can be seen i Figure 6 in column Employee ID.

Requirement 11.3

As seen in Figure 7 two extra header rows have been added with three headers, All, Order Details and Ship Details. This is native functionality where a `stackedHeaderColumns` is added to the `stackedHeaderRows` list. A name for the header and with columns the header shall span is declared inside the `stackedHeaderColumns`.

The code in listing 5.3 is from A.2 but made wider so the full header is visible in Figure 7.

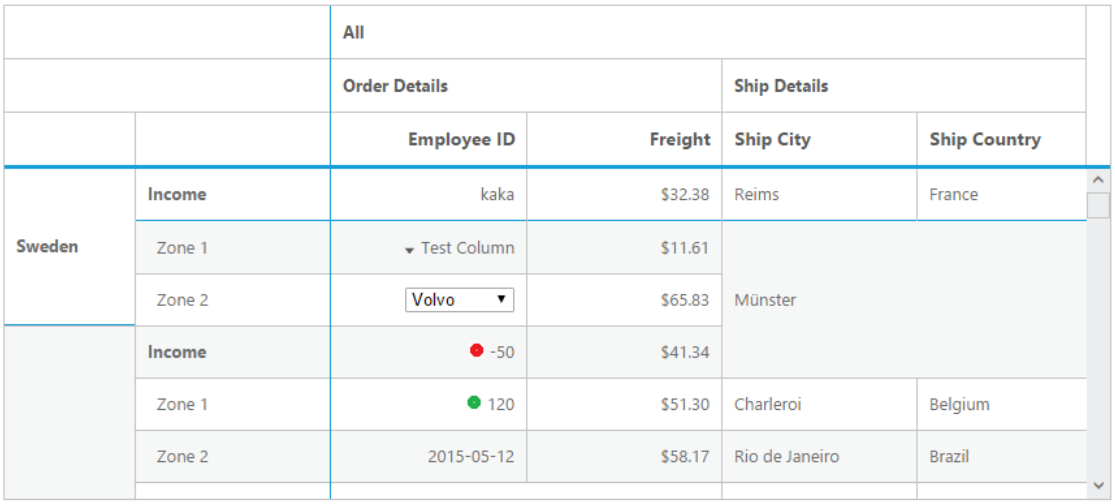


Figure 7: Example with extra headers rows and headers.

```
    ...

scrollSettings: {   width: 850   , height: 250, frozenRows:1 },

    ...

showStackedHeader:true,
stackedHeaderRows:[
  {stackedHeaderColumns:[
    {headerText:"All",column:"EmployeeID,Freight,ShipCity,ShipCountry"}
  ]},
  {stackedHeaderColumns:[
    {headerText:"Order Details",column:"EmployeeID,Freight"},
    {headerText:"Ship Details",column:"ShipCity,ShipCountry"}
  ]}
],

    ...
```

Listing 5.3: ejGrid with multiple header from requirement 11.3.

Requirement 12

As seen in Figure 6 the two first columns and the first row are frozen. That means that even if the user scrolls down or to the right, those cells will always be visible. This is done by changing the `scrollSettings:` to manage frozen rows. By adding the attribute `isFrozen:true` to a column for freezing it. This can be sen in listing 5.4 where it is highlighted. Full code example in appendix A.2.

```
 ...

scrollSettings: { width: 750, height: 250,  frozenRows:1   },

columns: [
  { field: "OrderID", headerText: " ",  isFrozen:true,     width: 100 },
  { field: "CustomerID", headerText: " ",  isFrozen:true,    width: 150 },
  { field: "EmployeeID", headerText: "Employee ID", textAlign: ej.TextAlign.Right,
      width: 150 },
  { field: "Freight", headerText: "Freight", textAlign: ej.TextAlign.Right, width:
      150, format: "{0:C}" },
  { field: "ShipCity", headerText: "Ship City", width: 150 },
  { field: "ShipCountry", headerText: "Ship Country", width: 150 }
]
```

Listing 5.4: ejGrid with frozen rows and columns from requirement 12.

Requirement 13

As seen in Figure 7 where the first column have some spanning cells. The first three rows have been merged by the highlighted code in listing 5.5. This is native functionality where a `mergeCellInfo:function(args)` is set, that will be called for every cell in the grid when it is loaded. When a specific cell is found by looking at `args.column` and `args.data`, a merge can be done to the right (`args.colMerge(2)`), down (args.rowMerge(3)) or both, where 2 and 3 are the numbers of columns and rows to be merged. Full code example in appendix A.2.

```
    ...
// Cell merging
allowCellMerging: true,
mergeCellInfo: function (args){

  if (args.column.field == "EmployeeID" && args.data.OrderID == 10254) {
     args.colMerge(2);
  }
  if (args.column.field == "ShipCity" && args.data.OrderID == 10249) {
    args.rowMerge(3);
    args.colMerge(2);
  }
  if (args.column.field == "OrderID" && args.data.OrderID == "<b>Sweden</b>") {
    args.rowMerge(3);
  }
  if (args.column.field == "OrderID" && args.data.OrderID == "<b>USA</b>") {
    args.rowMerge(7);
  }
},

    ...
```

Listing 5.5: ejGrid with spanning cells from requirement 13.

6 Testing

This chapter presents a performance test with with and without virtual scrolling. The test will focus on load time for different sizes of data sets. The test will be done on a computer with a local web-page and local data. To measure the time the following JavaScript code from listing 6.1 have been used. The function set as `ejGtid load: "function"` is called when DOM is ready and the grid initialization starts. The function set as `ejGtid create: "function"` is called when the grid has been created and is shown.

```
<label id="time"></label>
<script type="text/javascript">
    $(function () {
      $("#Grid").ejGrid({
            ...
        load: "onLoad",
        create: "onComplete",
            ...
      });
    });

  var startTime, endTime;

  function onLoad(args) {
    startTime = new Date();
  }
  function onComplete(args) {
    endTime = new Date();
    $("#time").text("Load time: "+Math.abs(endTime- startTime)+ " ms");
  }
</script>
```

Listing 6.1: JavaScript code used for time measurement.

The sizes of the data set have been taken from a technical report containing performance testing between HTML and Silverlight grids[8]. This in order to loosely compare the results of both reports. The different sizes of data set will be referred as tiny, small, medium, large, and huge reports, as in[8]. The sizes of the reports can be seen in Figure 8. The sizes of the reports come from real reports from Trimma[20], therefore the irregular size increase of rows and columns. All cells in the test reports contains a randomized float value ranging from -180,0 to 180,0.

Report size	Rows	Columns
Tiny	25	7
Small	100	7
Medium	652	7
Large	1985	9
Huge	54787	2

Figure 8: The sizes of the reports used for testing.

6.1 Virtual Scrolling

The test will be performed both with and without virtual scrolling. Virtual scrolling is a sort of UI Virtualization which loads the grid with data while scrolling (requirement 1.1). When the grid is initialized, only the rows seen in the grid window is loaded. The rest of the grid is empty with rows being added when the user scrolls down. By using `virtualScrollMode: ej.Grid.VirtualScrollMode.Normal` as seen in listing 6.2 the scroll-bar show that the entire report is plotted even if new rows are visualized only when they come into view (requirement 1.2).

The advantages of using virtual scrolling are that huge reports can be loaded with basically no penalty to the performance, at least in load and render time. The scrolling performance can be reduced due to the fact that the grid needs to load in rows when they come in to view. Another disadvantage is that some grid functionality become disabled when using virtual scrolling. Functionality like editing, frozen rows and columns, and cell merging. For very small reports the virtual scrolling have the same or even worse performance that normal.

```
1 $("#Grid").ejGrid({
2         ...
3
4   allowScrolling: true,
5   scrollSettings: { width: 1000, height: 500, allowVirtualScrolling: true,
        virtualScrollMode: ej.Grid.VirtualScrollMode.Normal },
6
7         ...
8
9 });
```

Listing 6.2: Code example of virtual scrolling.

6.2 Results

Each test have been done 10 times to decrease the chance of variations. All times presented are therefore an average. The full test data can be found in appendix fugue 3. Large and huge reports diagram are presented in their own diagram because the tiny and small stack became to small to see.

Figure 9 shows the load time for tiny, small and medium reports. As expected the normal mode fast exceeds the virtual in load time as the report grows bigger.

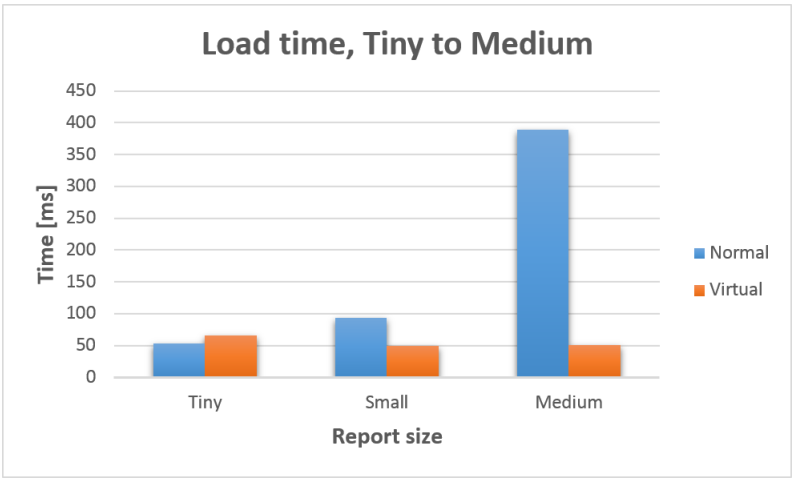


Figure 9: Load time for Tiny, Small and Medium report size with and without virtual scrolling measured in ms.

Figure 10 shows the load time for large and huge reports.

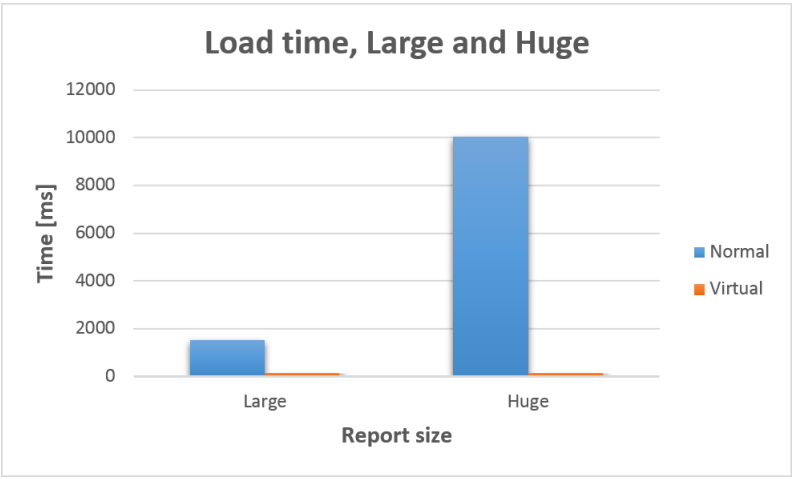


Figure 10: Load time for Large and Huge report size with and without virtual scrolling measured in ms.

Figure 11 shows the load times per cell for all reports with virtual scrolling disabled. The reason this i per cell is that the regular data was not that exiting. The overhead[25] seems to fade out between small and medium.

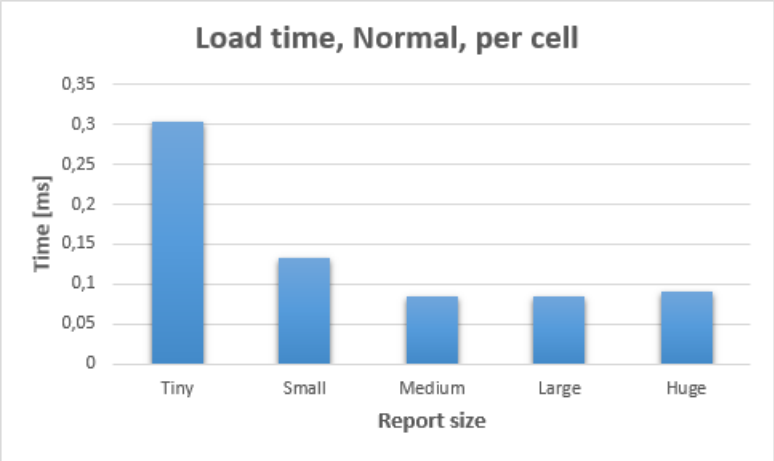


Figure 11: Load time per cell for all report sizes without virtual scrolling measured in ms.

Figure 12 shows the load times for all reports with virtual scrolling enabled. As expected virtual scrolling performs worse for small reports and seems to work best for the really huge reports. For small, medium and large the difference is negligible.

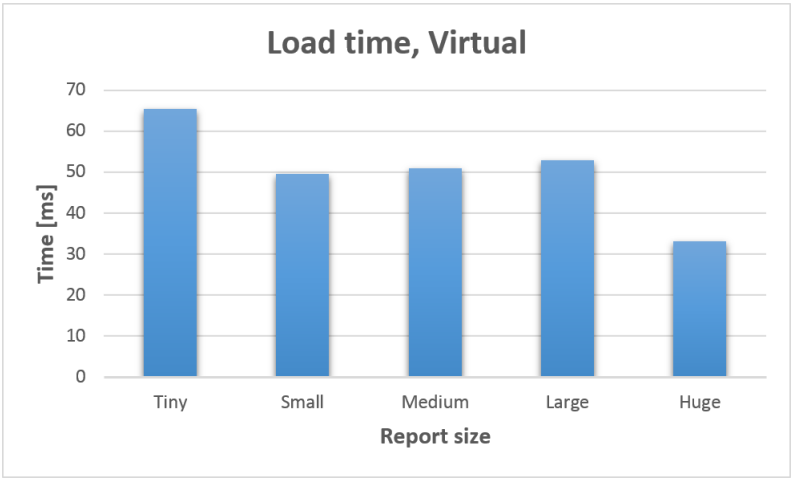


Figure 12: Load time for all report sizes with virtual scrolling measured in ms.

7 Discussion and Conclusion

HTML5 is without doubt a replacement for plugins. But the most advanced features might not be as well supported or even possible. JavaScript and for example Silverlight are two different languages so some things just can not work the same way. But with some compromises i believe a switch from plugins to HTML5 is possible.

Note that some of this comments and conclusion do not regard generalized JavaScript but only the framework chosen as test implementation.

7.1 Evaluation

Looking at Figure 2 (in Chapter 4), most of the requirements are supported. The orange ones should be fine if implementing some custom JavaScript is not a problem. It is the yellow and red ones that are hard or impossible to solve.

The yellow ones might be hardest to fix. For example requirement 1.1 on the grid implemented had partial support for UI Virtualization. I am talking about the virtual scrolling that disabled some other features. This is not a question about adding custom code but rather changing source code of the framework, which can be extremely hard.

7.2 Implementation

The test implementation went good overall. All requirements chosen for implementation was successfully implemented with both code and picture examples. However there was some problems. On paper, the grid support almost every functionality. But when some of them are put together, there might be some complications. For example, Frozen rows and Spanning cells did not work great together. Also Multiple selection and frozen columns had problems.

7.3 Performance

Comparing the results of the performance tests to the result of the Silverlight study[8], page 60, table 8, Rendering Time. The JavaScript virtual scrolling load time is much better, however, the JavaScript Gird is much simpler with less functionality enabled then the test in [8]. Also these test are very different except in report size, (number of rows and columns), but one can at least hint that they can have a similar performance.

The performance of the virtual scrolling is very good with extremely low load time. How-

ever, the possibility to load huge report almost instantly comes with a price. First of all you loose some functionality. secondly the scrolling do not flow very well and becomes very jerky. This behavior is noticeable in medium size reports and larger.

In normal mode everything less than huge reports flows well during scrolling but the load time for large reports are on the edge of being to much. The huge report is very slow and jerky during scrolling and the load time is way to high.

For huge reports neither virtual scrolling or normal mode works without any problems. It might be best to avoid reports of that size, at least for now.

Why virtual scrolling becomes faster the bigger the reports become is unknown (Figure 12). As well as how it is slowest on small report. This would be very interesting to investigate for a possible future work.

Looking at Figure 11, the load time have been divided by the total numbers of cells for each report. There is always some overhead, but somewhere between small and medium reports, the load time per cell becomes almost linear. The reason for the per cell calculation is that full reports in normal mode preformed as expected. More data means longer load time, and since the size increment between reports are not linear, no real conclusions could be made.

7.4 Limitations

The performance test is way to basic to be able to do more than just hint about its performance compared to Silverlight. The study it was compared against had a much more advanced implementation, and with additional functionality that most certainly affected its performance. Even that they had the same numbers of cells with similar content, the data structures was designed completely different.

There exist many more big frameworks than the three presented in this report. One of them might have been the most promising after the evaluation if it had been a part of it. The reason this three got selected for this study was primarily that they had the fastest and best support.

7.5 Conclusion

After performing this evaluation the following recommendations are made.

- A migration from a Silverlight component to a HTML5 based is definitely possible, with some compromise.
- Syncfusions ejGrid is the recommended framework after the evaluation, and since the implementation and performance tests where good overall, they support this claim.

Acknowledgements

I would like to thank Trimma for the opportunity to do this project and work with them. A special thanks to my external supervisor Daniel Hellström and his team for helping me during this project. I also want to thank my internal supervisor at Umeå University, Pedher Johansson.

Bibliography

[1] Stephen Chapman. What is javascript? [Online] (visited 2015-05-06) http://javascript.about.com/od/reference/p/javascript.htm.

[2] World Wide Web Consortium. THE TECHNOLOGY. [Online] (visited 2015-04-08) http://www.w3.org/html/logo/#the-technology.

[3] World Wide Web Consortium. A vocabulary and associated apis for html and xhtml. [Online] (visited 2015-04-08) http://www.w3.org/TR/html5/.

[4] World Wide Web Consortium. W3c editor's draft 23 march 2015. [Online] (visited 2015-04-07) http://www.w3.org/html/wg/drafts/html/master/introduction.html.

[5] Google. Pepper plugin api (ppapi). [Online] (visited 2015-04-16) http://www.chromium.org/nativeclient/getting-started/getting-started-background-and-basics#TOC-Pepper-Plugin-API-PPAPI.

[6] Ian Hixson. W3c workshop on web applications and compound documents (day 1) jun 1, 2004. [Online] (visited 2015-04-08) http://www.w3.org/2004/04/webapps-cdf-ws/minutes-20040601.html#topic18.1.

[7] Chris Hoffman. Why browser plug-ins are going away and what's replacing them. [Online] (visited 2015-04-07) http://www.howtogeek.com/179213/why-browser-plug-ins-are-going-away-and-whats-replacing-them/.

[8] Adam Holmström. Performance and usability improvements for massive data grids using silverlight. Technical report, Umeå universitet, Teknisk-naturvetenskapliga fakulteten, Institutionen för datavetenskap, 2011. [Online] (visited 2015-05-15) http://www8.cs.umu.se/education/examina/Rapporter/AdamHolmstrom_final.pdf.

[9] jQWidgets. jqxgrid. [Online] (visited 2015-04-02) http://www.jqwidgets.com/.

[10] jQWidgets. Licensing and support. [Online] (visited 2015-05-04) http://www.jqwidgets.com/license/.

[11] JP Mangalindan. Why companies are flocking to HTML5. [Online] (visited 2015-04-15) http://fortune.com/2011/08/30/why-companies-are-flocking-to-html5/.

[12] Microsoft. Select cells, ranges, rows, or columns on a worksheet. [Online] (visited 2015-05-19) https://support.office.com/en-us/article/Select-cells-ranges-rows-or-columns-on-a-worksheet-23f64223-2b6b-453a-8688-248355f10fa9.

[13] Bright Peter. Chrome starts pushing Java off the Web by disabling plugins. [Online] (visited 2015-04-16) http://arstechnica.com/information-technology/2015/04/chrome-starts-pushing-java-off-the-web-by-disabling-plugins/.

[14] Syncfusion. Downloads and service packs. [Online] (visited 2015-05-13) http://www.syncfusion.com/downloads/servicepacks.

[15] Syncfusion. Essential studio javascript grid. [Online] (visited 2015-04-02) http://www.syncfusion.com/.

[16] Syncfusion. Subscription service. [Online] (visited 2015-04-30) https://www.syncfusion.com/sales/faq.

[17] Syncfusion. Support and product maintenance sla. [Online] (visited 2015-04-30) https://www.syncfusion.com/Content/downloads/Syncfusion_Software_Support_SLA-13.1.pdf.

[18] Cisco Systems. Global Internet traffic by year, table. [Online] (visited 2015-04-07) http://en.wikipedia.org/wiki/Internet_traffic#Global_Internet_traffic.

[19] Techquickie. Html5 as fast as possible. [Online] (visited 2015-04-07) https://www.youtube.com/watch?v=IsXEVQRaTX8.

[20] Trimma. Trimma. [Online] (visited 2015-04-01) http://www.trimma.se/.

[21] Webix. Licensing and support. [Online] (visited 2015-05-04) http://webix.com/licenses/.

[22] Webix. Webix datatable. [Online] (visited 2015-04-27) http://webix.com/.

[23] Wikipedia. Cascading style sheets. [Online] (visited 2015-05-06) http://en.wikipedia.org/wiki/Cascading_Style_Sheets.

[24] Wikipedia. Javascript. [Online] (visited 2015-05-06) http://en.wikipedia.org/wiki/JavaScript.

[25] Wikipedia. Overhead (computing). [Online] (visited 2015-06-02) http://en.wikipedia.org/wiki/Overhead_%28computing%29.

[26] Wikipedia. Rich internet application market share. [Online] (visited 2015-06-02) http://en.wikipedia.org/wiki/Microsoft_Silverlight#Adoption.

A First Appendix

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <meta charset="utf-8" />
  <script src="scripts/jquery-1.10.2.min.js"></script>
  <script src="scripts/jquery.easing.1.3.min.js"></script>
  <script src="scripts/jquery.globalize.min.js"></script>
  <script src="scripts/jsrender.min.js"></script>
  <script src="scripts/jsondata.js"></script>
  <script src="scripts/ej.web.all.min.js"></script>
  <script src="Scripts/bootstrap.min.js"></script>
  <link href="themes/bootstrap.css" rel="stylesheet" />
  <link href="themes/responsive-css/ejgrid.responsive.css" rel="stylesheet"/>
  <link href="themes/default-theme/ej.web.all.min.css" rel="stylesheet"/>
</head>
<body>
  <div id="Grid"></div>
  <script type="text/javascript">
    $(function () {
      $("#Grid").ejGrid({
        // the datasource "window.gridData" is referred from jsondata.min.js
        dataSource: window.gridData,

        allowScrolling: true,
        scrollSettings: { width: 550, height: 300},

        // sets the columns, the field: "tag" corresponds to a tag in window.
            gridData for automatic data binding.
        columns: [
          { field: "OrderID", headerText: "", textAlign: ej.TextAlign.Right, width
              : 100 },
          { field: "EmployeeID", headerText: "Employee ID", textAlign: ej.
              TextAlign.Right, width: 100 },
          { field: "Freight", headerText: "Freight", textAlign: ej.TextAlign.Right
              , width: 100, format: "{0:C}" },
            { field: "ShipCity", headerText: "Ship City", width: 100 },
          { field: "ShipCountry", headerText: "Ship Country", width: 100}
        ]
      });
    });
  </script>
</body>
</html>
```

Listing A.1: Full basic example of ejGrid with surrounding HTML code and imports.

```
$(function () {
  $("#Grid").ejGrid({
    // the datasource "window.gridData" is referred from jsondata.min.js
    dataSource: window.gridData,

    // Scrolling frozen rows
    allowScrolling: true,
    scrollSettings: { width: 750, height: 400, frozenRows:1},

    // Cell merging
    allowCellMerging: true,
    mergeCellInfo: function (args){

      if (args.column.field == "EmployeeID" && args.data.OrderID == 10254) {
         args.colMerge(2);
      }
      if (args.column.field == "ShipCity" && args.data.OrderID == 10249) {
        args.rowMerge(3);
        args.colMerge(2);
      }
      if (args.column.field == "OrderID" && args.data.OrderID == "<b>Sweden</b>")
          {
        args.rowMerge(3);
      }
      if (args.column.field == "OrderID" && args.data.OrderID == "<b>USA</b>") {
        args.rowMerge(7);
      }
    },

    // Multiple headers
    showStackedHeader:true,
    stackedHeaderRows:[
      {stackedHeaderColumns:[
        {headerText:"All",column:"EmployeeID,Freight,ShipCity,ShipCountry"}
      ]},
      {stackedHeaderColumns:[
        {headerText:"Order Details",column:"EmployeeID,Freight"},
        {headerText:"Ship Details",column:"ShipCity,ShipCountry"}
      ]}
    ],

    enableRowHover:false,
    allowSelection: true,
    selectionType: "multiple",
    selectionSettings: { selectionMode: ["cell"], enableToggle:false },

    cellSelecting: function (args) {

      if (args.isShiftPressed) {
        this.multiSelectShiftRequest = false;
        if (!args.previousRowCell.length)
          args.previousRowCell = $(this.getRowByIndex(args.previousRowCellIndex
              [0][0]).find(".e-rowcell:eq(" + args.previousRowCellIndex[0][1][0] +
               ")"))
        var startIndexFrozenCount = 0, endIndexFrozenCount = 0;
        if (args.currentCell.parents("table").parent("div").hasClass("e-
            movablecontentdiv"))
          endIndexFrozenCount = this.model.scrollSettings.frozenColumns;
        if (args.previousRowCell.prev().parents("table").parent("div").hasClass("e
            -movablecontentdiv"))
          startIndexFrozenCount = this.model.scrollSettings.frozenColumns;
        var startIndex = args.previousRowCell[args.previousRowCell.length - 1].
            cellIndex - 1 + startIndexFrozenCount, endIndex = args.currentCell[0].
            cellIndex + endIndexFrozenCount;
        var startRow = args.currentCell.parent()[0].rowIndex, endRow = args.
            previousRowCell.parent()[0].rowIndex;
        var temp;
        if (startIndex > endIndex) {
```

```
61            temp = startIndex; startIndex = endIndex; endIndex = temp;
62          }
63          if (startRow > endRow) {
64            temp = startRow; startRow = endRow; endRow = temp;
65          }
66          var cells = [];
67          for (var i = startRow; i <= endRow ; i++) {
68            var currentRow = [i, []];
69            for (var j = startIndex; j <= endIndex; j++) {
70              currentRow[1].push(j);
71            }
72            cells.push(currentRow);
73          }
74          flag = false;
75          this.selectCells(cells);
76          flag = true;
77          this.multiSelectShiftRequest = true;
78          args.cancel = true;
79        }
80      },
81      columns: [
82        { field: "OrderID", headerText: " ",isFrozen:true,  width: 100 },
83        { field: "CustomerID", headerText: " ",isFrozen:true, width: 150 },
84        { field: "EmployeeID", headerText: "Employee ID", textAlign: ej.TextAlign.
              Right, width: 150 },
85        { field: "Freight", headerText: "Freight", textAlign: ej.TextAlign.Right,
              width: 150, format: "{0:C}" },
86        { field: "ShipCity", headerText: "Ship City", width: 150 },
87        { field: "ShipCountry", headerText: "Ship Country", width: 150 }
88      ]
89    });
90  });
```

Listing A.2: Full code example of ejGrid with advanced custom selection different cell types in same column multiple column headers frozen rows and columns and spanning cel

Figure 1: A Screenshot of Trimmas application named INSIKT(insight) with their current Silverlight grid component marked in red.

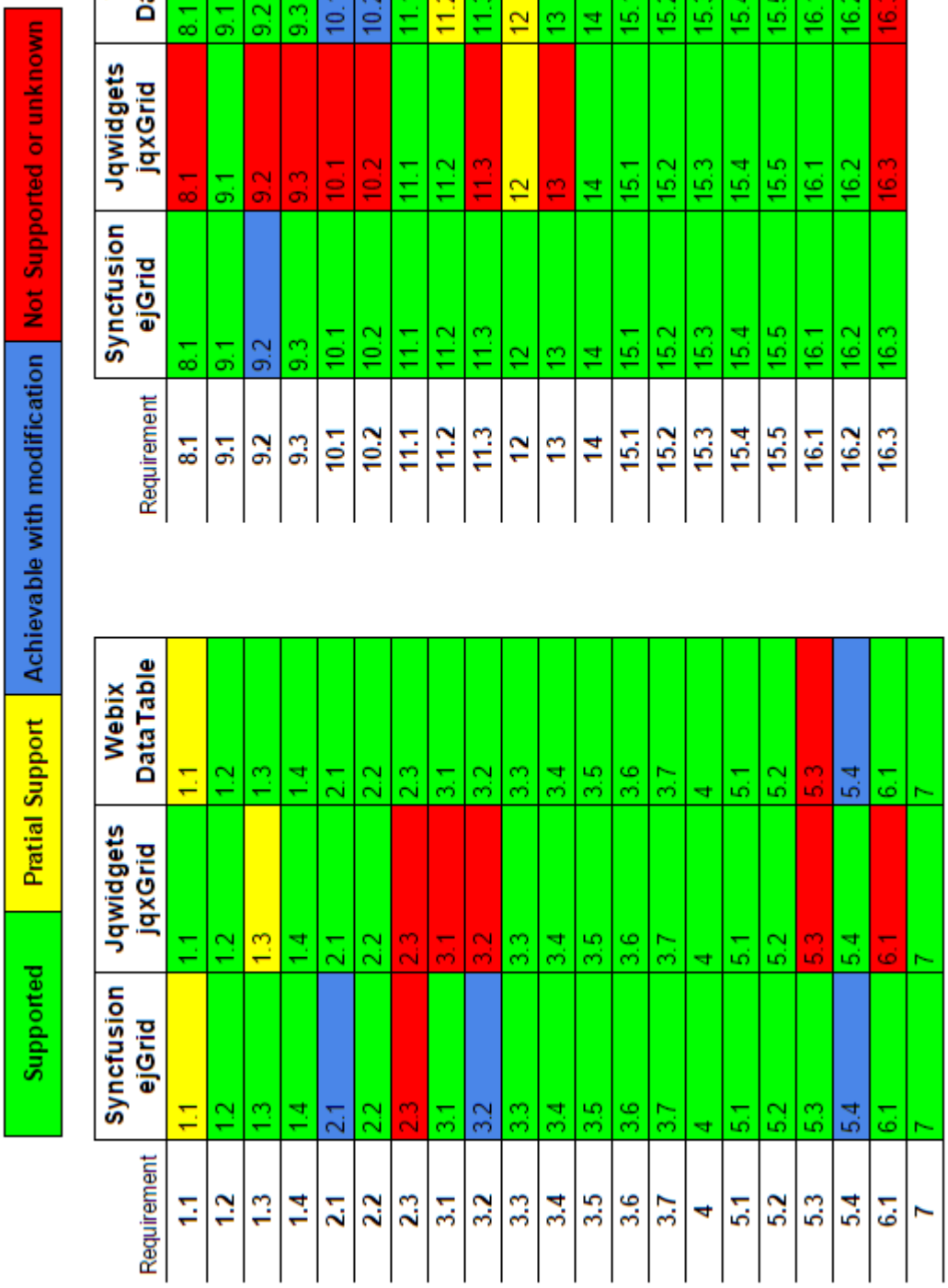


Figure 2: Summary of each requirement for each framework. Suited for grayscale.

	Tiny		Small		Medium		Large		Huge		
Test number	Normal	Virtual	Normal	Virtual	Normal	Virtual	Normal	Virtual	Normal	Virtual	Time in MS
1	61	67	99	57	394	47	1600	52	9882	34	
2	40	68	96	49	393	46	1537	49	9933	33	
3	41	62	99	48	393	49	1505	53	10130	34	
4	42	68	83	46	408	72	1502	56	9978	32	
5	63	73	86	43	383	60	1534	50	9912	33	
6	60	64	81	44	390	48	1498	57	10197	32	
7	44	64	82	50	384	47	1555	56	10117	33	
8	61	61	86	61	380	46	1449	52	10006	33	
9	62	63	110	47	384	50	1442	55	10334	32	
10	57	64	107	50	385	45	1469	50	9993	35	
Average Load Time [ms]	**53,1**	**65,4**	**92,9**	**49,5**	**389,4**	**51**	**1509,1**	**53**	**10048,2**	**33,1**	
Average Load Time [s]	**0,05**	**0,07**	**0,09**	**0,05**	**0,39**	**0,05**	**1,51**	**0,05**	**10,05**	**0,03**	

Figure 3: The performance test data