Create a WYSIWYG Editor With the contentEditable Attribute

WYSIWYG editors are quite popular. You might also have used one at some point. There are a lot of libraries available to help you set up your own editor. Although they're quick to set up, there are also downsides of using these libraries. To begin with, they are bloated. Most of them have fancy features that you might not use. Moreover, customizing the appearance of those editors can be a headache.

In this tutorial, we will build our own lightweight WYSIWYG editor. By the end of this tutorial, you will have an editor with basic formatting capabilities that's styled according to your preferences.

Let's begin with the introduction of execCommand. We will be using this command to implement our editor extensively.

## Document.execCommand()

execCommand is a method of the document object. It allows us to manipulate the contents of an editable region. When used alongside contentEditable, it can help us create a rich-text editor. There are a lot of commands available like adding a link, making a selection **bold** or italic, and changing font size or color. This method follows the syntax:

|  |  |
| --- | --- |
| 1 | document.execCommand(CommandName, ShowDefaultUI, ValueArgument); |

CommandName is a string which specifies the name of the command to execute. ShowDefaultUIis a Boolean to indicate whether the supporting interface should be shown or not. This option is not fully implemented, and it is best to set it to false. ValueArgument is a string to provide information like image URL or foreColor. This argument is set to null when a command does not require a value to take effect.

We will need to use different versions of this method to implement various features. In the next few paragraphs, I will go over all of them one by one.

### Commands With No Value Argument

Commands like bold, justify, undo and redo don't need a ValueArgument. In such cases we use the following syntax:

|  |  |
| --- | --- |
| 1 | document.execCommand(commandName, false, null); |

CommandName is simply the name of command like justifyCenter, justifyRight, bold, etc.

### Commands With a Value Argument

Commands like insertImage, createLink and foreColor need a third argument to work properly. For these commands, you need the following syntax:

|  |  |
| --- | --- |
| 1 | document.execCommand(commandName, false, value); |

For insertImage, the value would be the URL of the image to be inserted. In the case of foreColor, it would be a color value like #FF9966 or a name like blue.

### Commands That Add Block-Style Tags

Adding HTML block-style tags requires you to use formatBlock as commandName and the tag name as valueArgument. The syntax would be similar to:

|  |  |
| --- | --- |
| 1 | document.execCommand('formatBlock', false, tagName); |

This method will add an HTML block-style tag around the line which contains the current selection. It also replaces any tag that already existed there. tagName can be any of the headline tags (h1-h6), p or blockquote.

I have discussed the most common commands here. You can visit [Mozilla](https://developer.mozilla.org/en-US/docs/Web/API/Document/execCommand) for a list of all available commands.

## Creating a Toolbar

With the basics out of our way, it is time to create the toolbar. I will be using [Font Awesome](http://fontawesome.io/) icons for the buttons. You might have noticed that leaving aside a few differences, all the execCommands have a similar structure. We can use this to our advantage by using the following markup for toolbar buttons:

|  |  |
| --- | --- |
| 1 | <a href="#" data-command='commandName'><i class='fa fa-icon'></i></a> |

This way whenever users click a button, we will be able to tell which version of execCommand to use based on the value of data-command attribute. Here are a few buttons for reference:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <a href="#" data-command='h2'>H2</a>  <a href="#" data-command='undo'><i class='fa fa-undo'></i></a>  <a href="#" data-command='createlink'><i class='fa fa-link'></i></a>  <a href="#" data-command='justifyLeft'><i class='fa fa-align-left'></i></a>  <a href="#" data-command='superscript'><i class='fa fa-superscript'></i></a> |

The data-command attribute value for the first button is h2. After checking for this value in JavaScript, we will use the formatBlock version of the execCommand method. Similarly, for the last button, superscriptsuggests that we need to use the no valueArgument version of execCommand.

Creating foreColor and backColor buttons is a different story. They pose two problems. Depending on how many colors we are providing to users to choose from, writing that much code can be tiresome and error prone. To tackle this issue we can use the following JavaScript code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | var colorPalette = ['000000', 'FF9966', '6699FF', '99FF66','CC0000', '00CC00', '0000CC', '333333', '0066FF', 'FFFFFF'];    var forePalette = $('.fore-palette');    for (var i = 0; i < colorPalette.length; i++) {    forePalette.append('<a href="#" data-command="forecolor" data-value="' + '#' + colorPalette[i] + '" style="background-color:' + '#' + colorPalette[i] + ';" class="palette-item"></a>');  } |

Notice that I am also setting a data-value attribute for each color. This will later be used as valueArgumentin the execCommand method.

The second issue is that we can't show that many colors all the time, because it would take a lot of space and result in a terrible user experience. Using a little CSS, we can make sure that the color palette appears only when a user hovers over respective buttons. The markup for these buttons needs to be changed to the following as well:

|  |  |
| --- | --- |
| 1  2  3  4 | <div class="fore-wrapper"><i class='fa fa-font'></i>    <div class="fore-palette">    </div>  </div> |

To display the palettes only on hover, we need the following CSS:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11 | .fore-palette,  .back-palette {    display: none;  }    .fore-wrapper:hover .fore-palette,  .back-wrapper:hover .back-palette {    display: block;    float: left;    position: absolute;  } |

There are a lot of other CSS rules in the CodePen demo to make the toolbar prettier, but this is all that's needed for the core functionality.

## Adding Functionality to the Editor

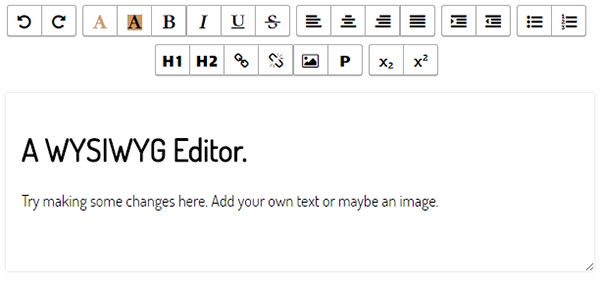
Now, it is time to make our editor functional. The code required to do so is surprisingly small.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20 | $('.toolbar a').click(function(e) {      var command = $(this).data('command');      if (command == 'h1' || command == 'h2' || command == 'p') {      document.execCommand('formatBlock', false, command);    }      if (command == 'forecolor' || command == 'backcolor') {      document.execCommand($(this).data('command'), false, $(this).data('value'));    }      if (command == 'createlink' || command == 'insertimage') {      url = prompt('Enter the link here: ','http:\/\/');      document.execCommand($(this).data('command'), false, url);    }      else document.execCommand($(this).data('command'), false, null);    }); |

We begin by attaching a click event to all toolbar buttons. Whenever a toolbar button is clicked, we store the value of the data-command attribute of the respective button in the variable, command. This is later used to call the appropriate version of the execCommand method. It helps in writing concise code and avoids repetition.

When setting foreColor and backColor, I am using the data-value attribute as the third argument. createLink and insertImage don't have a constant url value, so we use a prompt to get the values from the user. You may also like to perform additional checks to make sure the url is valid. If the command variable does not satisfy any of the if blocks, we run the first version of execCommand.

This is what [our WYSIWYG editor](http://codepen.io/Shokeen/pen/pgryyN) looks like.



You could also implement the auto-save functionality using localStorage that I discussed in my last tutorial.