Create an Inline Text Editor With the contentEditable Attribute

Creating an inline editor requires effort. You begin by switching the element to be edited with an inputor textarea field. For a seamless user experience, you might also have to use some CSS in order to match the styling of swapped elements to the original one. Once the user is done editing, you will again have to switch the elements after copying all the content to the original ones.

The contentEditable attribute makes this task a lot easier. All you have to do is set this attribute to trueand standard HTML5 elements will become editable. In this tutorial, we will create an inline rich text editor based on this feature.

## The Basics

This attribute can take three valid values. These are true, false and inherit. The value true indicates that the element is editable. An empty string will also evaluate to true. false indicates that the element is not editable. The value inherit is the default value. inherit indicates that an element will be editable if its immediate parent is editable. This implies that if you make an element editable, all its children and not just immediate ones will become editable as well, unless you explicitly set their contentEditable attribute to false.

You can change these values dynamically with JavaScript. If the new value is none of the three valid ones then it throws a [SyntaxError](http://www.w3.org/TR/html5/infrastructure.html#syntaxerror) exception.

## Creating the Editor

To create the inline editor, you need to have the ability to change the value of the contentEditable attribute whenever a user decides to edit something.

While toggling the contentEditable attribute, it is necessary to know what value the attribute holds currently. To accomplish that, you can use the isContentEditable property. If isContentEditable returns true for an element then the element is currently editable, otherwise it is not. We will use this property shortly to determine the state of various elements in our document.

The first step in building the editor is the creation of a button to toggle editing and some editable elements.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <button id="editBtn" type="button">Edit Document</button>    <div id="editor">      <h1 id="title">A Nice Heading.</h1>      <p>Last Edited By - <span id="author">Monty Shokeen</span></p>      <p id="content">Some content that needs correction.</p>  </div> |

Each element that we intend to keep editable needs to have its own unique Id. This will be helpful when we have to save the changes or retrieve them later to replace the text inside each element.

The following JavaScript code handles all the editing and saving.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | var editBtn = document.getElementById('editBtn');  var editables = document.querySelectorAll('#title, #author, #content')    editBtn.addEventListener('click', function(e) {    if (!editables[0].isContentEditable) {      editables[0].contentEditable = 'true';      editables[1].contentEditable = 'true';      editables[2].contentEditable = 'true';      editBtn.innerHTML = 'Save Changes';      editBtn.style.backgroundColor = '#6F9';    } else {      // Disable Editing      editables[0].contentEditable = 'false';      editables[1].contentEditable = 'false';      editables[2].contentEditable = 'false';      // Change Button Text and Color      editBtn.innerHTML = 'Enable Editing';      editBtn.style.backgroundColor = '#F96';      // Save the data in localStorage      for (var i = 0; i < editables.length; i++) {        localStorage.setItem(editables[i].getAttribute('id'), editables[i].innerHTML);      }    }  }); |

We use querySelectorAll() to store all editable elements in a variable. This method returns a NodeListwhich contains all the elements in our document that are matched by specified selectors. This way it is easier to keep track of editable elements with one variable. For instance, the title of our document can be accessed by using editables[0], which is what we will do next.

Next, we add an event listener to our button's click event. Every time a user clicks on the Edit Document button, we check if the title is editable. If it is not editable, we set the contentEditable property on each of the editable elements to true. Moreover, the text 'Edit Document' changes to 'Save Changes'. After users have made some edits, they can click on the 'Save Changes' button and the changes made can be saved permanently.

If the title is editable, we set the contentEditable property on each of the editable elements to false. At this point, we can also save the content of our document on the server to retrieve later or synchronize the changes to a copy that exists somewhere else. In this tutorial I am going to save everything in localStorageinstead. When saving the value in localStorage, I am using the Id of each element to make sure that I don't overwrite anything.

[See my live CodePen demo](http://codepen.io/Shokeen/pen/XXggZr).

## Retrieving Saved Content

If you make changes to any of the elements in the previous demo and reload the page, you will notice that the changes you made are gone. This is because there is no code in place to retrieve the saved data. Once the content has been saved in localStorage, we need to retrieve it later when a user visits the webpage again.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14 | if (typeof(Storage) !== "undefined") {      if (localStorage.getItem('title') !== null) {      editables[0].innerHTML = localStorage.getItem('title');    }      if (localStorage.getItem('author') !== null) {      editables[1].innerHTML = localStorage.getItem('author');    }      if (localStorage.getItem('content') !== null) {      editables[2].innerHTML = localStorage.getItem('content');    }  } |

The code above checks if the title, author or content already exist in localStorage. If they do, we set the innerHTML of the respective elements to the retrieved values.

[See another CodePen demo](http://codepen.io/Shokeen/pen/OMggQQ).

## Making the Editor More User-Friendly

To further improve our inline editor, we need to make two changes. The first one is to provide a clear distinction between what is editable and what is not. This may be achieved by changing the appearance of editable elements with CSS. Changing the font and color of the concerned elements should do the trick. The [contenteditable="true"] selector will apply the following style to elements whenever the contenteditable attribute is set to true.

|  |  |
| --- | --- |
| 1  2  3  4 | [contenteditable="true"] {    font-family: "Rajdhani";    color: #C00;  } |

The second improvement would be the ability to auto-save data. You can do it in multiple ways, like auto-save every five seconds.

|  |  |
| --- | --- |
| 1  2  3  4  5 | setInterval(function() {    for (var i = 0; i < editables.length; i++) {      localStorage.setItem(editables[i].getAttribute('id'), editables[i].innerHTML);    }  }, 5000); |

You can also save the changes on every keydown event.

|  |  |
| --- | --- |
| 1  2  3  4  5 | document.addEventListener('keydown', function(e) {    for (var i = 0; i < editables.length; i++) {      localStorage.setItem(editables[i].getAttribute('id'), editables[i].innerHTML);    }  }); |

In this tutorial I am sticking with the former method. You are free to trigger auto-save based on any event that seems more appropriate in your projects.

[View the CodePen demo](http://codepen.io/Shokeen/pen/eJRRVw).

## Editing the Entire Page With Design Mode

contentEditable is useful when you have to edit a few elements on a webpage. When the content of all or almost all the elements on a webpage has to be changed, you can use the designMode property. This property is applicable to the whole document. To turn it on and off, you use document.designMode = 'on';and document.designMode = 'off'; respectively.

This will prove valuable in situations where you are the designer and someone else is the content creator. You provide them with a design and some dummy text. Later, they can replace it with real content. To see designMode in action, open up the console tab in your browser's developer tools. Type document.designMode = 'on'; into the console and press **Enter**. Everything on this page should be editable now.

## Final Thoughts

The contentEditable attribute is convenient in situations like quickly editing articles or enabling users to edit their comments with a single click. This feature was first implemented by IE 5.5. Later, it was standardized by WHATWG. The [browser support](http://caniuse.com/#feat=contenteditable) is also pretty good. All major browsers besides Opera Mini support this attribute.

JavaScript has become one of the de-facto languages of working on the web. It’s not without it’s learning curves, and there are plenty of frameworks and libraries to keep you busy, as well. If you’re looking for additional resources to study or to use in your work, check out what we have available in the [Envato marketplace](https://codecanyon.net/category/javascript).

This tutorial covered the basics of the contentEditable attribute and how it can be used to create a basic inline text editor. The next tutorial will teach you how to implement a toolbar and provide rich text editing capabilities.