Creating a CRUD form with HTML5 Local Storage and JSON

See in this article how to create a simple CRUD program, inputing data in the HTML5 Local Storage and JSON.

For those that don't know yet the Local Storage, this is a HTML resource that works, let we say, like a cookie that doesn't have expiration time. Or else, it is a local to store data that are not lost at the end of the section, thus, we can close and open the browser several times and the information will remain there.

In this article, we'll see how to use the Local Storage in practical terms, creating a simple cadastration program of clients, with CRUD basic operations. We are not going to describe the design that should be used, but the functionaties. In the source-code available in the top of this page, interface is enhenced.

Listing 1: HTML structure

```
<html>
<head>
	<meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
	<script type="text/javascript" src="http://code.jQuery.com/jQuery-
1.7.2.min.js"></script>
	<script type="text/javascript" src="functions.js"></script>
</head>
<body>
	<form id="frmCadastre">
		<ul>
			<li>
				<label for="txtID">ID:</label>
				<input type="text" id="txtID"/>
			</li>
			<li>
				<label for="txtName">Name:</label>
				<input type="text" id="txtName"/>
			</li>
			<li>
				<label for="txtPhone">Phone:</label>
				<input type="text" id="txtPhone"/>
			</li>
			<li>
				<label for="txtEmail">Email:</label>
				<input type="text" id="txtEmail"/>
			</li>
			<li>
				<input type="submit" value="Save" id="btnSave"/>
			</li>
		</ul>
	</form>

	<table id="tblList">

	</table>
</body>
</html>
```

It is important to observe that we make reference to two JavaScript files. The first one is JQuery and the second one is the file containing the functions that will treat the cadastration itself.

Throughout this article, we'll use some functions of JavaScript that deserve special attention, are they:

- localStorage.setItem(name, value): this function is used to store a value in the Local Storage. Each object stored is referenced by a key (name)
- localStorage.getItem(name): On the other hand, the getItem is used to recover a value stored from it's identifying key.
- JSON.stringfy(object): In order to store the data, we'll use the JSON format and this function transforms one object into string with a syntax adequate to the JSON.
- JSON.parse(object): Since the function parse transforms an item in JSON format into it's original format.

From here, shall we create then our file functions.js and into it we'll use also the syntax jQuery, thus, all content shall be in the body of the function that is executed on the loading of the page.

Listing 2: Global variables definition

```
$(function(){
        var operation = "A"; //"A"=Adding; "E"=Editing
        var selected_index = -1; //Index of the selected list item
        var tbClients = localStorage.getItem("tbClients");//Retrieve the stored
data
        tbClients = JSON.parse(tbClients); //Converts string to object
        if(tbClients == null) //If there is no data, initialize an empty array
                tbClients = [];
});
```

The variable "operation" will be used in order to define if the user is adding or editing a register. "selected_index" will serve to reference the register selected in a table. "tbClients" is ours "table of clients" and we start it with the value recovered from the Local Storage (saved with the key "tbClients"). In the case there's no content, then the variable will be initialized as an empty arrow.

By default, the variable "operation" will have the value "A", in other words, may the user type the data and click in the button to save the information, a new register will be added, except when one have clicked in the table in order to edit it.

Now, let’s go to the CRUD functions. We'll use 4 functions: Add, Edit, Delete e List. Let us see the code:

Listing 3: Add function

```
function Add(){
        var client = JSON.stringify({
                ID    : $("#txtID").val(),
                Name  : $("#txtName").val(),
                Phone : $("#txtPhone").val(),
                Email : $("#txtEmail").val()
        });
        tbClients.push(client);
        localStorage.setItem("tbClients", JSON.stringify(tbClients));
        alert("The data was saved.");
        return true;
}
```

Listing 4: Edit function

```
function Edit(){
        tbClients[selected_index] = JSON.stringify({
                        ID    : $("#txtID").val(),
                        Name  : $("#txtName").val(),
                        Phone : $("#txtPhone").val(),
                        Email : $("#txtEmail").val()
                });//Alter the selected item on the table
        localStorage.setItem("tbClients", JSON.stringify(tbClients));
        alert("The data was edited.")
        operation = "A"; //Return to default value
        return true;
}
```

Listing 5: Delete function

```
function Delete(){
        tbClients.splice(selected_index, 1);
        localStorage.setItem("tbClients", JSON.stringify(tbClients));
        alert("Client deleted.");
}
```

Listing 6: List function

```
function List(){
        $("#tblList").html("");
        $("#tblList").html(
                "<thead>"+
                "       <tr>"+
                "       <th></th>"+
                "       <th>ID</th>"+
                "       <th>Name</th>"+
                "       <th>Phone</th>"+
                "       <th>Email</th>"+
                "       </tr>"+
                "</thead>"+
                "<tbody>"+
                "</tbody>"
                );
        for(var i in tbClients){
                var cli = JSON.parse(tbClients[i]);
                $("#tblList tbody").append("<tr>"+
                                                                        "
<td><img src='edit.png' alt='Edit"+i+"' class='btnEdit'/><img src='delete.png'
alt='Delete"+i+"' class='btnDelete'/></td>" +
                                                                        "
<td>"+cli.ID+"</td>" +
                                                                        "
<td>"+cli.Name+"</td>" +
                                                                        "
<td>"+cli.Phone+"</td>" +
                                                                        "
<td>"+cli.Email+"</td>" +

"</tr>");
        }
}
```

It is important to observe that in the first column of the table, it is added two images that will serve as buttons of Edit and Delete actions, which event onClick will be treated by jQuery means in the future. The images "edit.png" and "delete.png" are available in the source-code, but also may be any two images (here was used 16x16 dimension images as for the buttons to appear discreet on the table).

The List function should be called in the body of the main function, in order to the table be actualized whenever the site be reloaded.

Now, what's left is treat the events of the HTML's controls that will trigger the functions of CRUD. To store the data again in the register or in the on editing register, we'll use the event onSubmit of the form frmCadastre. See the following code:

Listing 7: Event onSubmit of the form

```
$("#frmCadastre").bind("submit",function(){
        if(operation == "A")
                return Add();
        else
                return Edit();
});
```

To the Edit and Delete buttons, on the other hand, were given classes "btnEdit"and "btnDelete", from which we'll treat the following onClick event:

Listing 8: onClick event of the edit buttons

```
$(".btnEdit").bind("click", function(){
        operation = "E";
        selected_index = parseInt($(this).attr("alt").replace("Edit", ""));
        var cli = JSON.parse(tbClients[selected_index]);
        $("#txtID").val(cli.ID);
        $("#txtName").val(cli.Name);
        $("#txtPhone").val(cli.Phone);
        $("#txtEmail").val(cli.Email);
        $("#txtID").attr("readonly","readonly");
        $("#txtName").focus();
});
```

Listing 9: onClick event of the Delete buttons

```
$(".btnDelete").bind("click", function(){
        selected_index = parseInt($(this).attr("alt").replace("Delete", ""));
        Delete();
        List();
});
```

The routines used are pretty simple, Here we didn't concern ourselves about validations and error treatment, for instance. Therefore, in the source-code of the article the code is a little bit more complex, with some more functionalities.

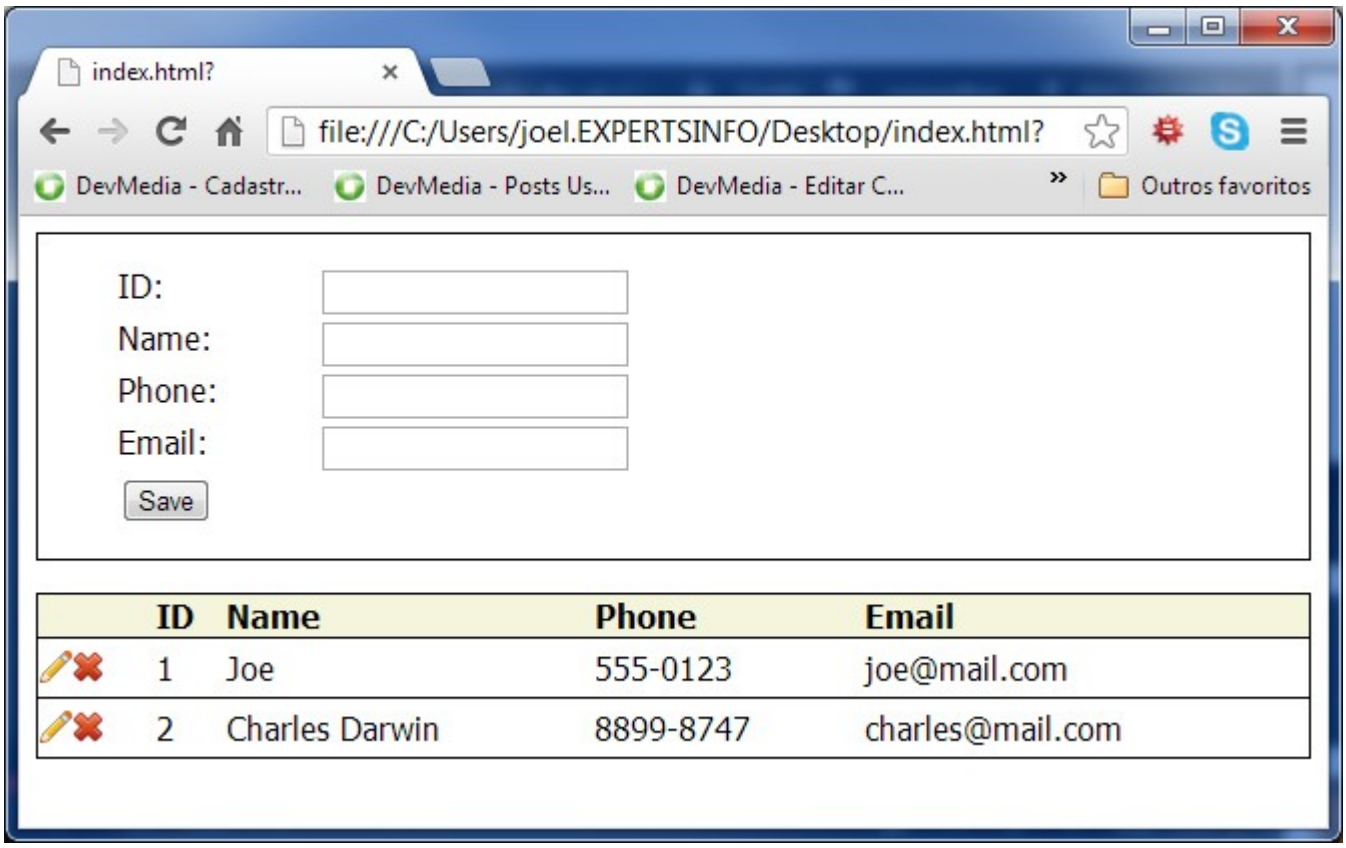

Figura 1: The code running on the browser

So, as for today that's all folks. I put myself at disposal to answer any questions about the article, you just need to comment right below.

A big hug and shall we meet in the next article.

Html

```
<!-- This is the main navbar section -->
<nav>
  <div class="nav-wrapper z-depth-5 red darken-2">
    <!-- this handles the sliding bars -->
    <a href="#" data-activates="slide-out" class="button-collapse show-on-large"><i class="fa fa-
bars fa-2x"></i></a>
      <ul id="slide-out" class="side-nav">
        <li><a href="#"><i class="fa fa-television"> </i> Dashboard </a> </li>
        <li><a href="#" data-toggle="modal" data-target="#addNewProductModal"
class="addNewProduct" ><i class="fa fa-plus-circle"></i> Add New Product </a></li>
        <li><a href="#aboutUsModal" class="modal-trigger" data-target="#aboutUsModal"><i
class="fa fa-question-circle-o"></i> About Us </a></li>
      </ul>
    <!-- this handles the full-width navbar -->
    <a href="#" class="brand-logo center"><img class="imgLogo"
src="https://res.cloudinary.com/ethelbert007/image/upload/v1511873622/logo_geyrq5.jpg"
alt=""></a>
    <ul id="nav-mobile" class="right hide-on-med-and-down grey lighten-1">
      <li class=""><a href="#"><i class="fa fa-television"> </i> Dashboard </a> </li>
      <li><a href="#" data-toggle="modal" data-target="#addNewProductModal"
class="addNewProduct"><i class="fa fa-plus-circle"></i> Add New Product </a></li>
      <li><a href="#aboutUsModal" class="modal-trigger" data-target="#aboutUsModal"><i
class="fa fa-question-circle-o"></i> About Us </a></li>
    </ul>
  </div>
</nav>

<!-- This is the main body<div> container section -->
<div class="container-fluid">
  <!-- firstPart: handles the floating button -->
  <div class="fixed-action-btn vertical">
    <a class="btn-floating btn-large red darken-2"> <i class="fa fa-bars fa-4x"></i></a>
    <ul>
      <li><a class="btn-floating blue darken-2 modal-trigger" href="#aboutUsModal"><i class="fa
fa-question-circle-o"></i></a></li>
      <li><a class="btn-floating red darken-2"><i class="fa fa-television"></i></a></li>
      <li><a class="btn-floating green darken-2 addNewProduct" data-toggle="modal" data-
target="#addNewProductModal"><i class="fa fa-pencil"></i></a></li>
    </ul>
  </div>

  <!-- 2ndPart: handles the ProductDashboard display -->
  <div class="mx-auto">
    <div class="card border-primary">
      <div class="card-header" id="usernameSpan"><h3 id="h3header">Product
Dashboard</h3></div>
      <div class="card-body text-primary" id="productDashboard">
        <form>
          <div class="input-field">
```

```
          <input id="search" type="search" required class="autocomplete" placeholder="Search Product List">
        </div>
       </form>
       <h4 class="card-title" > Current Products</h4>
       <ul class="list-group" id="productDisplay">
        <!-- codes  here supplied via JS-->
       </ul>
      </div>
      <button class="btn btn-outline-success btn-sm m-3 addNewProduct" id="addNewProduct" data-toggle="modal" data-target="#addNewProductModal"
           style="width: 20%">Add New Product</button>
    </div>
  </div>
  <!-- 3rdPart: responds to the main modal call -->
  <div class="modal fade" id="addNewProductModal" tabindex="-1" role="dialog" aria-labelledby="addNewProductModalLabel" aria-hidden="true">
    <div class="modal-dialog">
     <div class="modal-content">
      <!-- header for the modal -->
      <div class="modal-header">
       <img class="imgLogo" src="https://res.cloudinary.com/ethelbert007/image/upload/v1511873622/logo_geyrq5.jpg"><h4 class="modal-title left" id="addNewProductModalLabel">New Product Form</h4>
      </div>
      <!-- body of the modal -->
      <div class="modal-body">
       <form class="form-horizontal" role="form" id="modalForm">
        <div class="form-group">
         <label for="productName">Product Name</label>
         <input id="productName" type="text" />
        </div>
        <div class="form-group">
         <label for="productDescription">Product description</label>
         <textarea id="productDescription" type="text"  style="height: 100px" ></textarea>
        </div>
        <div class="form-group mb-4">
         <label for="productCategory">Select product category</label>
         <select id="productCategory" class="custom-select">
          <option value="Business Products">Business Products</option>
          <option value="Cards">Cards</option>
          <option value="Calender">Calender</option>
          <option value="Fine Arts">Fine Arts</option>
          <option value="Photo Prints">Photo Prints</option>
          <option value="Poster">Poster</option>
          <option value="Photo Gifts">Photo Gifts</option>
          <option value="PhotoBook">PhotoBook</option>
          <option value="Sample Set">Sample Set</option>
          <option value="Wall Decors">Wall Decors</option>
         </select>
        </div>
        <div class="modal-footer align-content-center" id="modalFooter">
```

```
            <button type="button" class="btn btn-outline-danger btn-sm m-2" data-dismiss="modal" id="modalClose"><i class="fa fa-close"></i> Close
            </button>
            <button type="button" class="btn btn-outline-success btn-sm m-2" id="modalSubmit"><i class="fa fa-save"></i> Add Entry </button>
          </div>
        </form>
      </div>
    </div>
  </div>
</div>

<!-- 4thPart: responds to the About Us modal call -->
<div id="aboutUsModal" class="modal bottom-sheet">
  <div class="modal-content">
    <span class="h4"><img class="imgLogo" src="https://res.cloudinary.com/ethelbert007/image/upload/v1511873622/logo_geyrq5.jpg" alt="">About Us</span>
    <p class="h5">Saal Digital - Ihr Partner für den professionellen Auftritt Werbeprodukte in HighEnd Qualität</p>
  </div>
</div>

<script src="https://code.jquery.com/jquery-3.2.1.js" integrity="sha256-DZAnKJ/6XZ9si04Hgrsxu/8s717jcIzLy3oi35EouyE=" crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.3/umd/popper.min.js" integrity="sha384-vFJXuSJphROIrBnz7yo7oB41mKfc8JzQZiCq4NCceLEaO4IHwicKwpJf9c9IpFgh" crossorigin="anonymous"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/js/bootstrap.min.js" integrity="sha384-alpBpkh1PFOepccYVYDB4do5UnbKysX5WZXm3XxPqe5iKTfUKjNkCk9SaVuEZflJ" crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.100.2/js/materialize.min.js"></script>
<script>
$(document).ready(function () {
  $(".button-collapse").sideNav();
  $("#aboutUsModal").modal();
  $("input.autocomplete").autocomplete({
    data:{
      "Photo Plaques":null, "FineArt Prints":null, "Retro prints": null, "Photo Prints": null, "Mini Reporello": null,
      "Card":null, "Folded Card(Up Down)":null, "Folded Card(Left Right)":null, "Double Folded Card":null, "Memorial Card":null,
      "Wall Decors":null, "Alu-Dibond":null, "Gallery Print":null, "Floating Frame":null, "Envelopes":null
    },
    limit:7,
    minLength:1
  });
```

```
  });
</script>
```

CSS

```
.imgLogo{
    height: 50px;
    width: 120px;
}
a:hover{
   text-decoration: none;
    color: navy;
    font-weight: bold;
}
#slide-out li a:hover{
    color: navy;
    font-weight: bold;
}
.divCenter{
    margin: auto;
    width: 50%;
}
.modal-body .form-horizontal{
    width: 100%;
}
```

JS

```
function randomNumberID(){
  return Math.floor(Math.random()*(1000002 - 1 + 1)) + 1;
}
$(document).ready( function () {
  getProductLists();
  document.getElementById('modalSubmit').addEventListener('click', modalSubmit);

  function modalSubmit (e) {
    let productTempId = randomNumberID();
    let productName = document.getElementById('productName').value;
    let productDescription = document.getElementById('productDescription').value;
    let productCategory = document.getElementById('productCategory').value;

    const productId = productTempId+productName+randomNumberID(); //Used to give each
product a unique id
    if(productName !== '' && productDescription !== ''){
      let newProduct = {
         id: productId,
        name: productName.toUpperCase(),
        category: productCategory,
        description: productDescription
       };

      //Add new product to localStorage. The localStorage key for all the product is productList'
```

```
      if(localStorage.getItem("productList") === null || localStorage.getItem("productList") === [] ||
localStorage.getItem("productList") === undefined ){
        let productList = [];
        productList.push(newProduct);
        localStorage.setItem("productList", JSON.stringify(productList));
      } else {
        let productList = JSON.parse(localStorage.getItem("productList"));
        productList.push(newProduct);
        localStorage.setItem("productList", JSON.stringify(productList));
      }
     } else{
       alert('All fields are required. Please check your entries again');
     }
    getProductLists();

    resetForm();
   e.preventDefault();
  }

}); //DocumentBody end tag

//get the data stored in the localStorage for display on load
function getProductLists() {
  if(localStorage.getItem("productList") === null){
    alert("Your dashboard is currently empty. Use the add button to add new products.");
    document.getElementById("search").disabled = true;
  } else {
    document.getElementById("search").disabled = false;
    let productList = JSON.parse(localStorage.getItem("productList"));
    let productDisplay = document.getElementById('productDisplay');
    //Display result
    productDisplay.innerHTML = '';
    for (let i = 0; i < productList.length; i++){
      let id = productList[i].id;
      let name = productList[i].name;
      let category = productList[i].category;
      let description = productList[i].description;

      productDisplay.innerHTML += '<li
class="list-group-item"><strong>'+name+'</strong><p>'+category+'</p><p>'+description+'</
p><p><a' +
          ' href="#" onclick="editProduct(\"+id+'\')" data-toggle="modal" data-
target="#addNewProductModal">' +
          '<i class="fa fa-edit green-text darken-2 "></i> Edit</a>    ' +
          '<a href="#" id="deleteId" onclick="deleteProduct(\"+id+'\')"><i class="fa fa-trash' +
          ' red-text' +
          ' darken-2"></i> ' +
          ' Delete</a>' +
          ' </p>' +
          '</li>';
      }
    }
```

```
  }


// deleting the main bookmark.
function deleteProduct(id) {
  let productList = JSON.parse(localStorage.getItem("productList"));
  for(let i = 0; i < productList.length; i++){
    if (productList[i].id === id) {
      productList.splice(i,1);
      //console.log(result);
    }
  }
  localStorage.setItem("productList", JSON.stringify(productList)); //reset the values in the local
storage
  getProductLists(); // to quickly display what is remaining from local storage.
}

// Editing a product
function editProduct(id) {
  "use strict";
  document.getElementById('modalSubmit').style.display = "none";
  document.getElementById("addNewProductModalLabel").textContent = "Edit Product";

  let tempId = id;
  let parentDiv = document.getElementById('modalFooter');
  let productList = JSON.parse(localStorage.getItem("productList"));


  if (parentDiv.contains(document.getElementById("editButton"))) {
    document.getElementById('editButton').disabled = false;
  } else {
    let editButton = document.createElement('button');
    editButton.id = "editButton";
    editButton.className = "fa fa-hdd-o btn btn-outline-primary btn-sm m-2";
    editButton.textContent = " Save data";
    parentDiv.appendChild(editButton);
  }
  for (let i = 0; i < productList.length; i++) {
    if (productList[i].id === id) {
      document.getElementById("productName").value = productList[i].name;
      document.getElementById("productDescription").value = productList[i].description;
      document.getElementById("productCategory").value = productList[i].category;
    }
  }

  document.getElementById("editButton").addEventListener("click", function () {
    addProduct();
    let productList = JSON.parse(localStorage.getItem("productList"));
    for(let i = 0; i < productList.length; i++){
      if(productList[i].id === tempId){
        productList.splice(i,1);
      }
```

```
    }
    localStorage.setItem("productList", JSON.stringify(productList));
    getProductLists();
    resetForm();
    document.getElementById("editButton").style.display = "none";

    $(".addNewProduct").on('click',productFormReset());

  });

}

function resetForm() {
  document.getElementById("productName").value = "";
  document.getElementById("productDescription").value = "";
  document.getElementById("productCategory").value = "";
}

function productFormReset() {
  document.getElementById('modalSubmit').style.display = "block";
  document.getElementById("addNewProductModalLabel").textContent = "New Product Form";
  document.getElementById('editButton').style.display = "none";
}

function addProduct() {
  let productTempId = randomNumberID();
  let productName = document.getElementById('productName').value;
  let productDescription = document.getElementById('productDescription').value;
  let productCategory = document.getElementById('productCategory').value;

  const productId = productTempId + productName + randomNumberID(); //Used to give each
product a unique id
  if (productName !== '' && productDescription !== '') {
    let newProduct = {
      id: productId,
      name: productName.toUpperCase(),
      category: productCategory,
      description: productDescription
    };
    if (localStorage.getItem("productList") === null || localStorage.getItem("productList") === [] ||
localStorage.getItem("productList") === undefined) {
      let productList = [];
      productList.push(newProduct);
      localStorage.setItem("productList", JSON.stringify(productList));
    } else {
      let productList = JSON.parse(localStorage.getItem("productList"));
      productList.push(newProduct);
      localStorage.setItem("productList", JSON.stringify(productList));
    }
  }
}
```

//holdval_ document.getElementById('editButton').style.display = "none"; //checks in case of disabled button.

Perform CRUD Operations with Local Storage Data

Environment

Product	Progress Kendo UI Grid
Operating System	All
Browser	All
Browser Version	All

Description

How can I perform CRUD operations with local storage data in the Kendo UI Grid?

Solution

The following example demonstrates how to perform CRUD operations with local storage data.

```
<button onClick="reset()" class="k-button">Reset test data</button>
<div id="grid"></div>
<script>

function setTestData(){
    var testData = [
      {ID: 1, Value: "TEST1"},
      {ID: 2, Value: "TEST2"},
      {ID: 3, Value: "TEST3"},
      {ID: 4, Value: "TEST4"},
      {ID: 5, Value: "TEST5"}
    ];
    localStorage["grid_data"] = JSON.stringify(testData);
}

function reset(){
    setTestData();
    $("#grid").data("kendoGrid").dataSource.read();
}

$(document).ready(function () {

    if(localStorage["grid_data"] == undefined){
        setTestData();
    }

    var dataSource = new kendo.data.DataSource({
        transport: {
          create: function(options){
            var localData = JSON.parse(localStorage["grid_data"]);
            options.data.ID = localData[localData.length-1].ID + 1;
            localData.push(options.data);
            localStorage["grid_data"] = JSON.stringify(localData);
```

```
            options.success(options.data);
          },
          read: function(options){
              var localData = JSON.parse(localStorage["grid_data"]);
              options.success(localData);
          },
          update: function(options){
            var localData = JSON.parse(localStorage["grid_data"]);

            for(var i=0; i<localData.length; i++){
              if(localData[i].ID == options.data.ID){
                localData[i].Value = options.data.Value;
              }
            }
            localStorage["grid_data"] = JSON.stringify(localData);
            options.success(options.data);
          },
          destroy: function(options){
            var localData = JSON.parse(localStorage["grid_data"]);
            for(var i=0; i<localData.length; i++){
                if(localData[i].ID === options.data.ID){
                    localData.splice(i,1);
                    break;
                }
            }
            localStorage["grid_data"] = JSON.stringify(localData);
            options.success(localData);
          },
        },
        schema: {
          model: {
            id: "ID",
            fields: {
              ID: { type: "number", editable: false },
              Value: { type: "string" }
            }}
        },
        pageSize: 20
    });

    var grid = $("#grid").kendoGrid({
        dataSource: dataSource,
        pageable: true,
        height: 500,
        toolbar: ["create", "save", "cancel"],
        columns: [
          { field: "ID", width: "100px" },
          { field: "Value", width: "100px"},
          { command: "destroy" }
        ],
        editable: "incell",
    }).data("kendoGrid");
});
</script>
```

https://docs.telerik.com/kendo-ui/knowledge-base/grid-localstorage-crud

Using the Local Storage

#javascript #webdev

Sarah Chima ・4 min read

This article discusses what the local storage is and JavaScript methods that we can use to manipulate it.

I have always known about the local storage but I never got to use it on any project. So I decided to build a note app because I want to be able to use the local storage to store and manipulate data. I decided to share what I learnt while using it. First, let us understand what the local storage is.

What is Local Storage?

Local storage is a web storage object that is available in a user's browser. It allows JavaScript browsers store and access data right in the browser. Basic CRUD operations (create, read, update and delete) can be done on data in the local storage. Data stored in the local storage persists even when the browser window has been closed.

Another form of web storage is Session Storage. This is similar to local storage. The difference is that the data stored in the session storage gets cleared after the session ends, ie. the browser window is closed.

Local Storage Methods

Local Storage methods are the methods that help you manipulate the local storage. That is to save and access data stored in the local storage. These methods include:

1. setItem()
2. getItem()
3. removeItem()
4. clear()

Let us discuss each of them.

setItem()

We use this method to add new data items to the local storage object or update existing values if the data exists. It takes two arguments, the key of the item to create or update and the value to store. For example, if we want to store a name in the local storage, here is what we will do

```
localStorage.setItem('name', 'Sarah');
```

In the example above, `name` is the key and `Sarah` is the value.

This is a simple example. What if you want to store something a little more complex like an array or an object in the local storage? For example, store the notes of the note app in the local storage. It is important to note that local storage stores values as strings. We need to convert the arrays or objects to strings before passing it to the local storage.

We can use the `JSON.stringify()` method to convert an object or array to strings before passing it the `setItem()` method.

```
const notes = [
    {
        title: 'A note',
        text: 'First Note'
    },
    {
        title: 'Another note',
        text: 'Second Note'
    }
]

localStorage.setItem('notes', JSON.stringify(notes))
```

getItem()

This method is used to access data stored in the local storage. It accepts one argument: the key of the item you want to get the value of. It returns the value as a string.

Let us get the name we stored in the local storage.

```
const name = localStorage.getItem('name');
console.log(name) // 'Sarah'
```

What if we want to get the notes we stored in the local storage? we do the same thing, pass the key to the getItem method. However, to get our value as array, we need to parse it. Otherwise, it returns strings.

```
JSON.parse(localStorage.getItem('notes'))
```

removeItem()

The removeItem() method removes data from the local storage. It receives a key and removes the data item stored with that key from the local storage. If that key does not exist in the local storage, it does nothing.

```
localStorage.removeItem('name')

console.log(localStorage.getItem('name')) //null
```

clear()

The `clear()` method clears the entire local storage of all data stored in it. It does not receive any argument.

```
localStorage.clear()
```

Those are the methods available to store and retrieve data from the local storage. Next, let us see how we can listen to storage change events.

Event Listener for Storage Change

To listen to changes in the local storage, we add an event listener for storage.

```
// When local storage changes, execute the doSomething function
```

```
window.addEventListener('storage', doSomething())
```

The storage event fires when either the local storage or the session has been modified in the context of another document. This means that the storage event is not fired on the page that is making changes to the local storage. Rather it is fired in another tab or window if the same page is open there. The assumption is that your page already knows all changes that happen on it. That it will only need notification if the change happens on another page.

I encountered this challenge when building the note app. I was trying to update the part that displays the notes based on changes in the local storage. However, I noticed that when I add a new note, it does not update the notes. Rather it updates the same page opened in another tab. To solve this, I used a state object. After storing to the local storage, I stored or updated a new note in this state. The display of the notes depends on the changes to the state.

Important Things to Note about the Local Storage

One last thing before we go, there are important things about the local storage that we should know.

1. The local storage is limited to 5MB across all major browsers.
2. It can easily be accessed from the browser so it should not be used to store any sensitive data or user information.
3. Operations on the local storage are synchronous. Therefore, they are executed one after another.

Want to see the note app I built? Here's a link to the live app and a link to Github. Have any question on any part of this article or the app, feel free to ask.

You can follow me on Instagram where I post regularly on my tech journey. I also share short notes on things I have learnt.

https://dev.to/sarah_chima/using-the-local-storage-fn8