Building a CRUD App

CRUD App

- OBJECTIVES

1. To build a single page application that works on basic operations to read & manipulate data
2. Students will learn different functionalities of JavaScript such as querySelector, local storage etc.
3. To make students able to implement "*CRUD*" related operations as a front-end developer

CRUD App

- OUTCOMES
  - At the end of this module, students are expected to learn

✓ Adding & Removing data dynamically using JavaScript & HTML

✓ The basic functionality of a CRUD based application such as adding/removing/loading records dynamically

✓ Building Single Page Applications (SPA) & its importance in web development

CRUD APP

- Document/Video Links

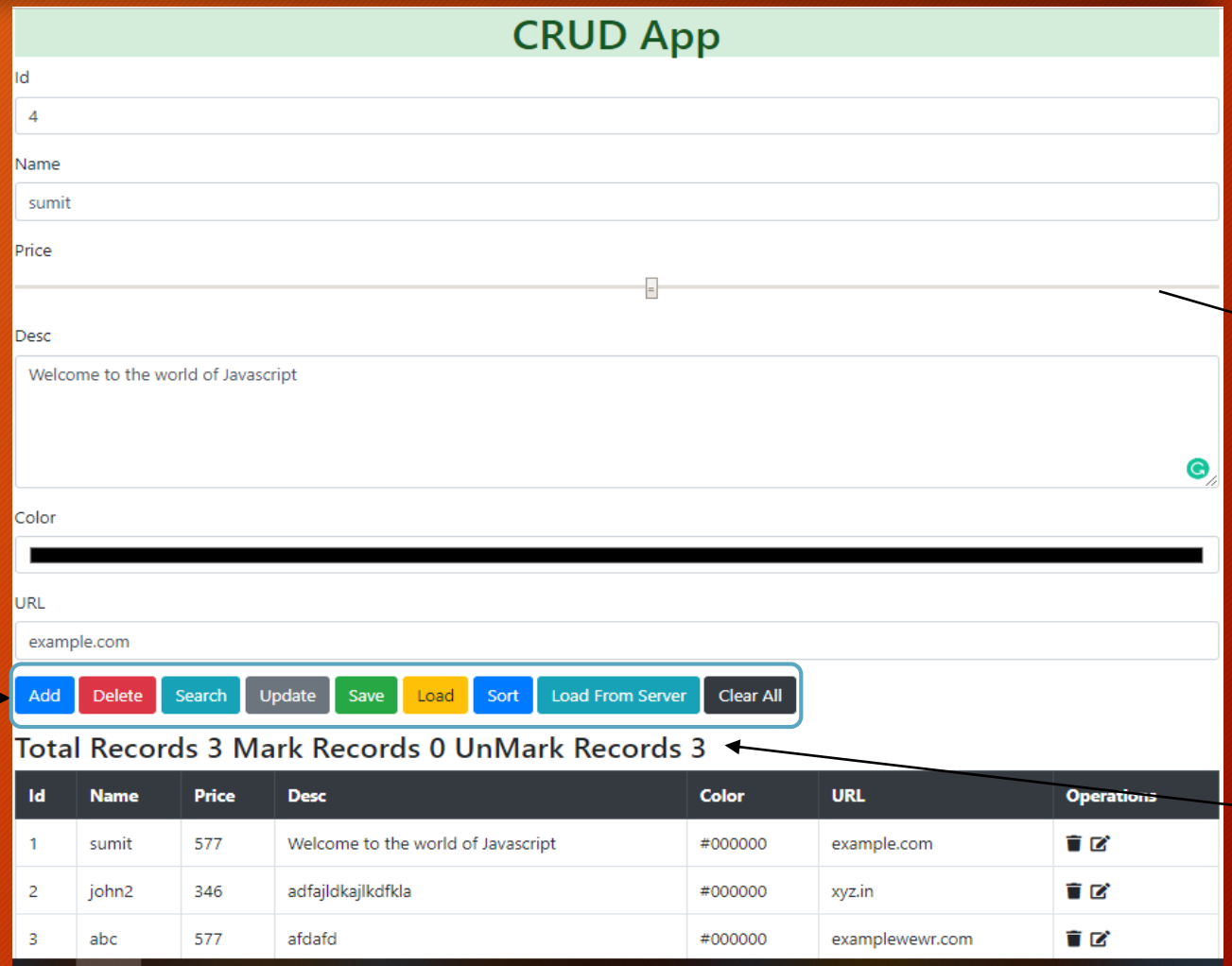
Topics	URL
How to Create a Simple CRUD Application using only JavaScript	https://www.encodedna.com/javascript/how-to-create-a-simple-crud-application-using-only-javascript.htm
JavaScript Array CRUD Example	https://www.codeofaninja.com/2012/11/javascript-array-crud-example.html
Pure JavaScript CRUD operations with HTML	https://www.youtube.com/watch?v=-rNQeJi3Wp4
JavaScript CRUD App – Part 1	https://www.youtube.com/watch?v=BDyw_3udjxU
CRUD in local storage in JavaScript	https://www.youtube.com/watch?v=qFHZsgW7nYI

Introduction

- “CRUD” is an acronym that stands for “Create, Read, Update and Delete”
- It is a single page application that works on *four* basic functions to manipulate data
- The data will be stored in a JavaScript object (JSON)
  - The data will be extracted from that object and displayed using dynamically created HTML table
  - Each row will have a few more dynamically created HTML elements like *edit & trash* with an event attached at the run-time

The CRUD App

- Final App Layout



Auto-Generated (RO)

Input form-fields

Range Field

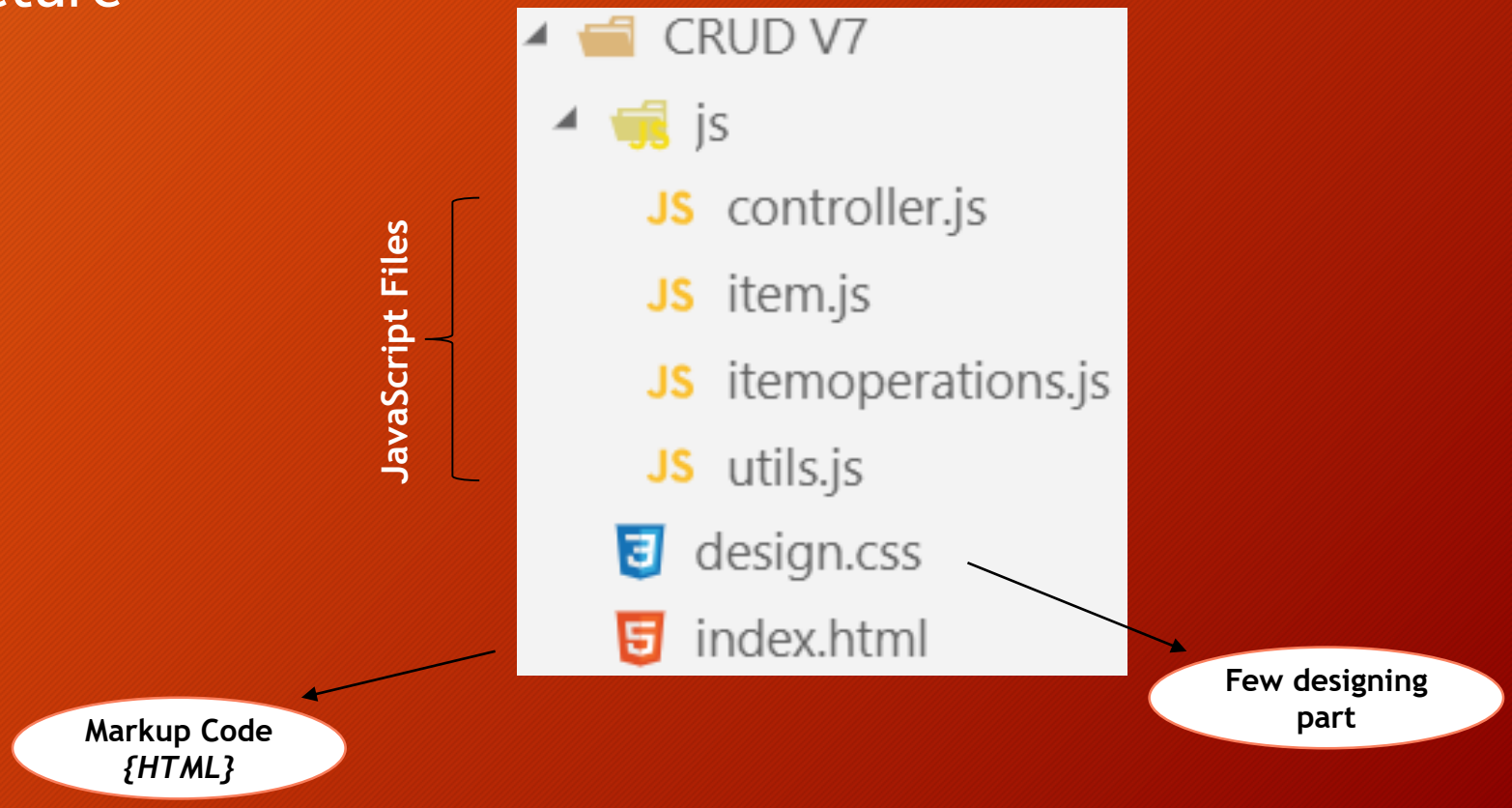
Diff. Operations

Items Description

Counters

Directory Structure

- Given is the Directory Structure for our CRUD APP
  - JS Folder
    - Controller.js
    - Item.js
    - itemOperations.js
    - Utils.js
  - Index.html
  - Design.css



HTML Code (index.html)

- Linking all the JS files inside *<head> using script tag*
- Linking Boot Strap and font awesome CSS

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css" integrity="sha384-fnmOCqbTlWIlj8LyTjo7mOUStjsKC4p
    <script src="js/item.js"></script>
    <script src="js/itemoperations.js"></script>
    <script src="js/controller.js"></script>
</head>
```

The Markup Code

- Make a *div* class “container”
  - Add different “*form-group*” classes for the following fields
    - ID
    - Name
    - Price
    - Description
    - Color
    - URL

```
<body>
    <div class='container'>
        <h1 class='alert-success text-center'>CRUD App</h1>
        <div class='form-group'>
            <label for="">Id</label>
            <input id='id' class='form-control' placeholder='Type Id Here' type="text">
        </div>
        <div class='form-group'>
            <label for="">Name</label>
            <input id='name' class='form-control' placeholder='Type Name Here' type="text">
        </div>
        <div class='form-group'>
            <label for="">Price</label>
            <input id='price' class='form-control' type="range" min="100" max="1000">
        </div>
        <div class='form-group'>
            <label for="">Desc</label>
            <textarea id='desc' placeholder="Type Desc Here" class='form-control' cols="30" rows="5"></textarea>
        </div>
        <div class='form-group'>
            <label for="">Color</label>
            <input id='color' type='color' class='form-control'>
        </div>
        <div class='form-group'>
            <label for="">URL</label>
            <input id='url' placeholder="http://sample.com" type='url' class='form-control'>
        </div>
```

The Markup Code

- Continued…
- Adding buttons to perform necessary operations
  - Make a "*form-group*" to add buttons: Add, Delete, Search, Update etc.

```
<div class='form-group'>
    <button id='add' class='btn btn-primary'>Add</button>
    <button class='btn btn-danger'>Delete</button>
    <button class='btn btn-info'>Search</button>
    <button class='btn btn-secondary'>Update</button>
    <button class='btn btn-success'>Save</button>
    <button class='btn btn-warning'>Load</button>
    <button class='btn btn-primary'>Sort</button>
    <button class='btn btn-info'>Load From Server</button>
    <button class='btn btn-dark'>Clear All</button>
</div>
```

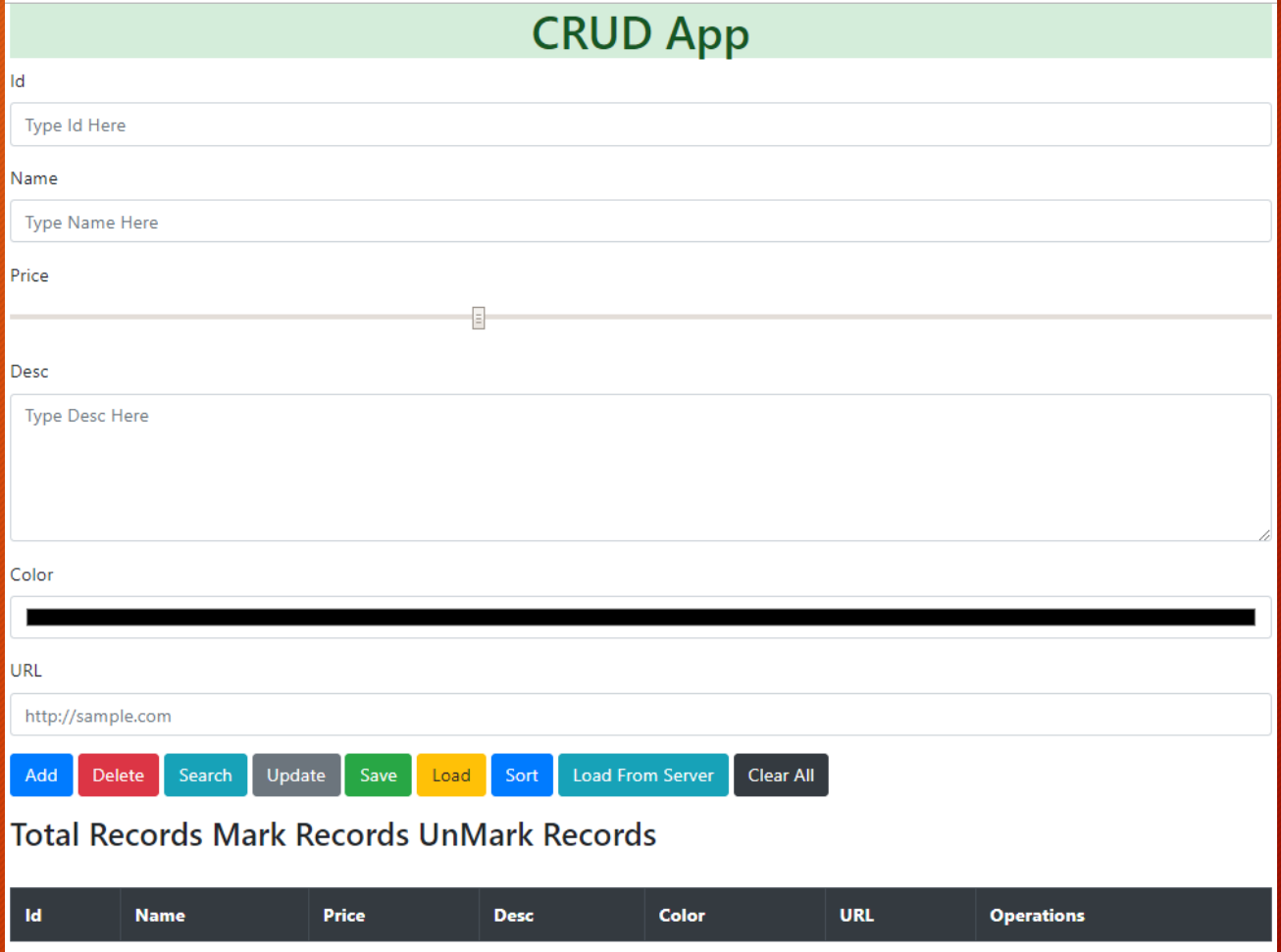
The Markup Code

- Adding the table fields
  - Add another table class with the following fields
    - ID | Name | Price | Desc| Color…
  - Make the table body element to group the dynamically generated elements
  - As we click on the “Add” button, a row will be dynamically generated inside <*tbody*> tag

```
        </div>
        <h3>Total Records Mark Records UnMark Records</h3>
        <table class='table table-bordered'>
            <thead class='thead-dark'>
                <tr>
                    <th>Id</th>
                    <th>Name</th>
                    <th>Price</th>
                    <th>Desc</th>
                    <th>Color</th>
                    <th>URL</th>
                    <th>Operations</th>
                </tr>
            </thead>
            <tbody id='items'>


            </tbody>
        </table>
    </div>
</body>


</html>
```

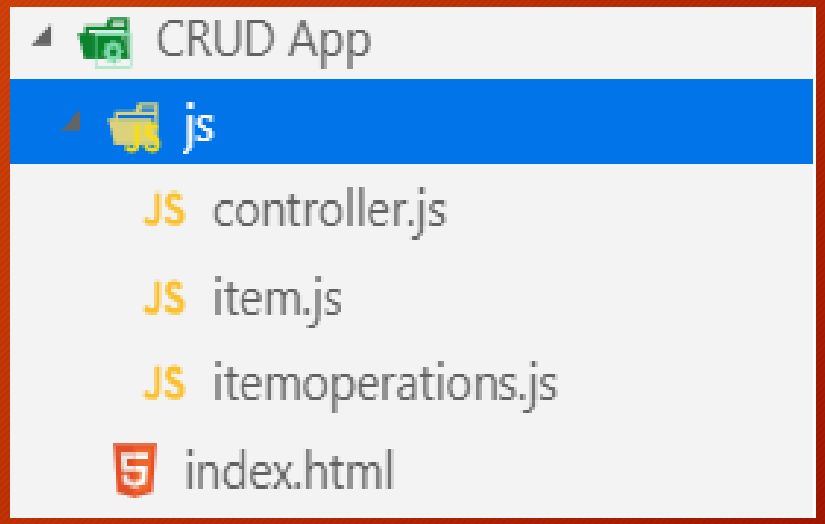
Output Form (open with Live Server)

- HTML Output



Creating JavaScript Files

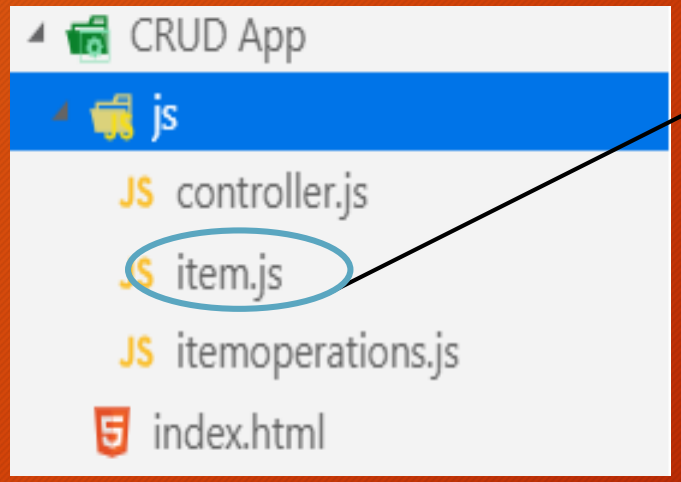
- Add the following three files under *js* directory
  - *Item.js*
  - *Controller.js*
  - *itemOperations.js*



The *item* Object

- Create an item class (inside "*item.js*") and make a constructor with the following arguments
- **Tip : Keep the class variables with the same name as the id created in HTML Form.**

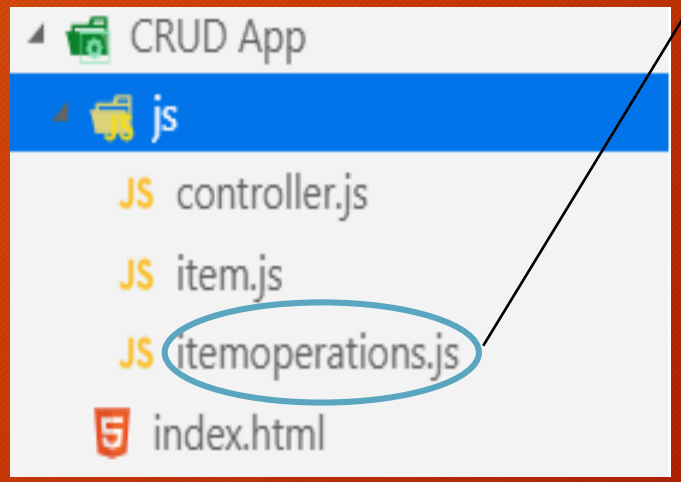
  - ID
  - Name
  - Price
  - Description
  - Color
  - URL




```
class Item{
    constructor(id , name, price, desc, color, url){
        this.id = id;
        this.name = name;
        this.price = price;
        this.desc = desc;
        this.color = color;
        this.url = url;
    }
}
```

Item Operations

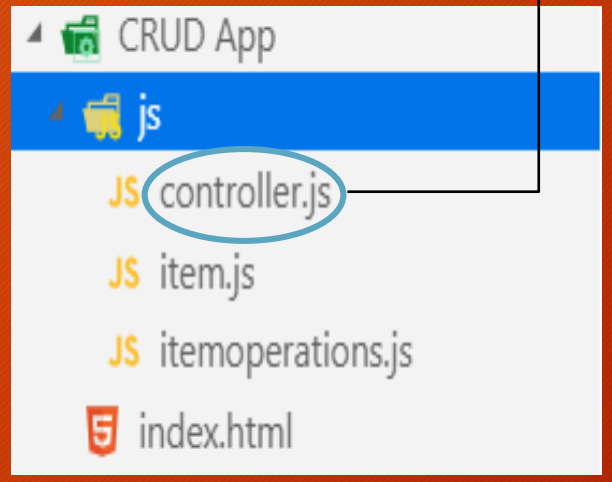
- Create an object itemOperations (inside *itemOperations.js)*
  - Initialize *items array*
  - Make *functions add, remove, sort, search, update…*
  - *Add()*
    - Fetch the items that are received from *itemObject*
    - Print these "*items*" on the screen in a grid layout using *push* function




```
const itemOperations = {
    items:[],
    add(itemObject){
        this.items.push(itemObject);
    },
    remove(){

    },
    search(){

    },
    sort(){

    },
    update(){

    }
}
```

The Controller

- Now go into "*controller.js*"
  - Here we will add all the logic code for different functions such as
    - *addRecord()*
    - *printRecord()*
    - *deleteRecord()*
    - *updateRecord()* ….




```
window.addEventListener("load",bindEvents);
function bindEvents(){
    document.querySelector('#add').addEventListener('click',addRecord);
}

function addRecord(){
    var item = new Item();
    for(let key in item){
        item[key] = document.querySelector('#'+key).value;
    }
    itemOperations.add(item);
    printRecord(item);
    console.log('Item Object is ',item);
}
function printRecord(item){
    var tbody = document.querySelector('#items');
    var tr = tbody.insertRow();
    var index = 0;
    for(let key in item){
        let cell = tr.insertCell(index);
        cell.innerText = item[key] ;
        index++;
    }
}
```

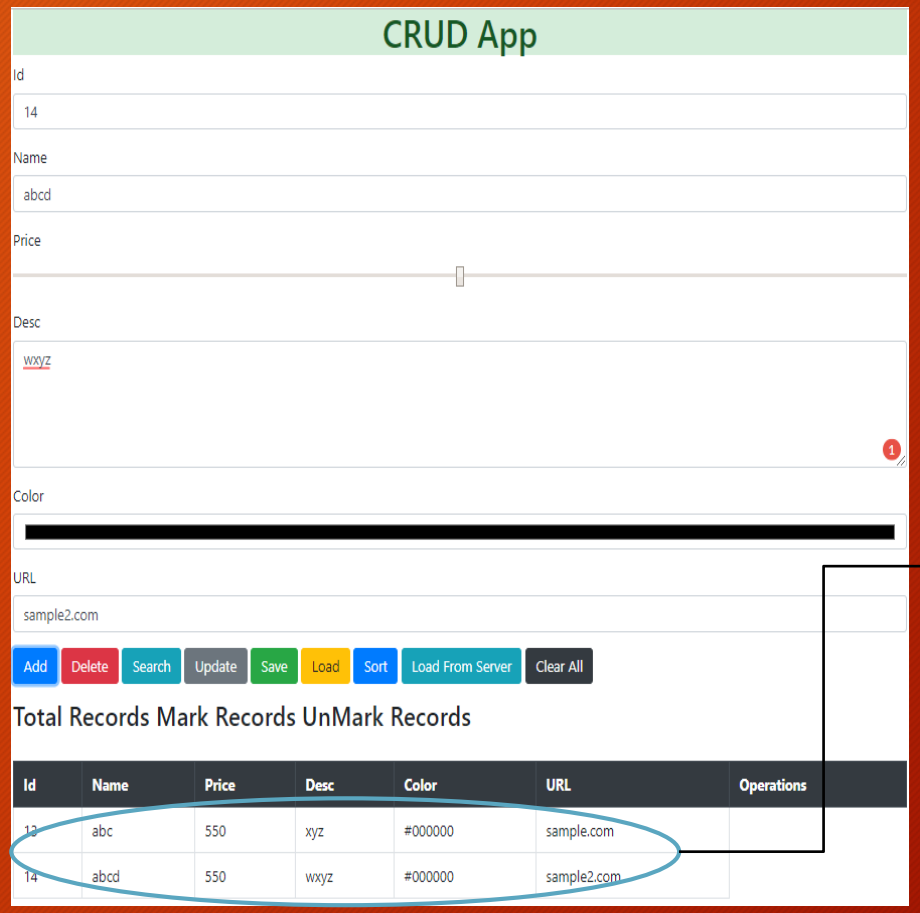
CRUD APP - Controller

- Create a file "*controller.js*"
  - Add an event listener
  - Use the *bindEvents()* function
- Function *addRecords()*
  - Pick data from *html*
  - Create an object "*item*"
  - Store the object into an array
  - Traverse array and print into a table

```
window.addEventListener("load",bindEvents);
function bindEvents(){
    document.querySelector('#add').addEventListener('click',addRecord);
}
let count = 0;
function addRecord(){
    var item = new Item();
    for(let key in item){
        item[key] = document.querySelector('#'+key).value;
    }
    count = count+1;
    itemOperations.add(item);
    printRecord(item);
    console.log('Item Object is ',item);
    console.log("Total Records",count);
}
```

CRUD APP - Controller

- Function *printRecord()*
  - Check the *'id'* attribute inside *<tbody>* tag (*index.html)*
  - use variable *tbody* to get the items (querySelector)
  - *Traverse* through the items and print items using insertCell() function
  - Items will be added dynamically at run-time



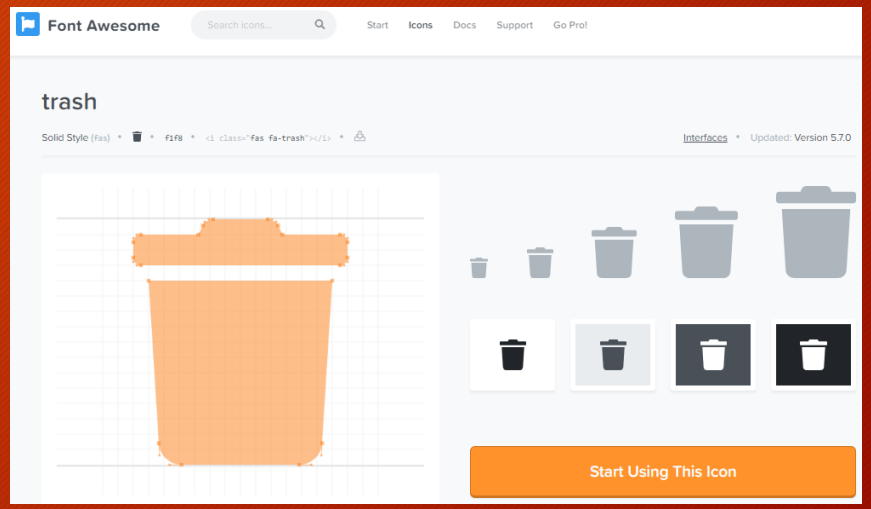

```
function printRecord(item){
    var tbody = document.querySelector('#items');
    var tr = tbody.insertRow();
    var index = 0;
    for(let key in item){
        let cell = tr.insertCell(index);
        cell.innerText = item[key] ;
        index++;
    }
}
```

CRUD APP - Controller

- Creating “*edit*” & “*trash*” icons

Id	Name	Price	Desc	Color	URL	Operations
12	afd	550	af	#000000	asdf	
13	adfasdf	550	afasdfasdfasfaddafdafda	#000000	asdf.org	

- Head to *fontawesome.com*
  - Search for *edit* & *trash* icons
  - Copy their class names



CRUD APP - Controller

- Function *createIcon()*
  - Use *iTag* variable to store the <i> element
  - Add event listener to this
  - Set attribute to item-id
  - Return it back to the parent function

Under *design.css*

```
i {
cursor:
pointer;
}
```

```
function printRecord(item){
    var tbody = document.querySelector('#items');
    var tr = tbody.insertRow();
    var index = 0;
    for(let key in item){
        if(key=='isMarked'){
            continue;
        }
        let cell = tr.insertCell(index);
        cell.innerText = item[key] ;
        index++;
    }
    var lastTD = tr.insertCell(index);
    lastTD.appendChild(createIcon('fas fa-trash mr-2',trash,item.id));
    lastTD.appendChild(createIcon('fas fa-edit',edit,item.id));
}
```

```
function createIcon(className,fn, id){
    // <i class="fas fa-trash"></i>
    // <i class="fas fa-edit"></i>
    var iTag = document.createElement("i");
    iTag.className = className;
    iTag.addEventListener('click',fn);
    iTag.setAttribute("data-itemid", id) ;

    return iTag;
}
```

CRUD APP - Controller

- Finding Total | Marked & Unmarked Records
  - Function *showTotal()*

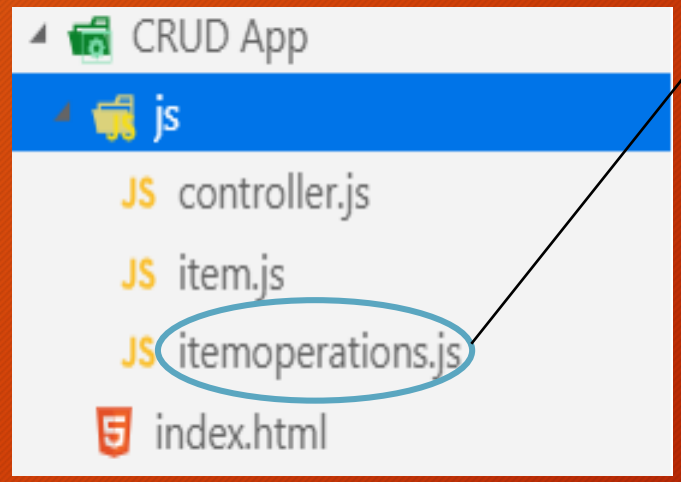
```
function showTotal(){
    document.querySelector('#total').innerText = itemOperations.items.length;
    document.querySelector('#mark').innerText = itemOperations.countTotalMark();
    document.querySelector('#unmark').innerText = itemOperations.items.length - itemOperations.countTotalMark();
}
```

  - Call this function from *init()*

```
function init(){
    showTotal();
    bindEvents();

}
```

Item Operations

- Go back to *"itemOperations.js"*
  - *Add the functionality for remove(), search(), markunmark()*




```
const itemOperations = {
    items:[],
    add(itemObject){
        this.items.push(itemObject);
    },
    remove(){

    },
    search(){

    },
    sort(){

    },
    update(){

    }
}
```

Item Operations

- Adding some more functionality
  - *Remove()*
    - *As we click on the remove icon -> that particular row should be marked*
  - *Search()*
    - *It will search a particular item using its "id"*
  - *Markunmark()*
    - *To change the state of the selected row*

```
const itemOperations = {
    items:[],
    add(itemObject){
        this.items.push(itemObject);
    },
    remove(){
         this.items = this.items.filter(itemObject=>!itemObject.isMarked);
        return this.items;
        },
    search(id){
            return this.items.find(itemObject=>itemObject.id ==id);
    },
    markUnMark(id){
            this.search(id).toggle();
    },
    countTotalMark(){
           return this.items.filter(itemObject=>itemObject.isMarked).length;
    },
    sort(){

    },
    update(){

    }
}
```

CRUD APP - Controller

- Trash/Edit Icon
  - Making selection as we click on Trash icon

```
function trash(){
    let id = this.getAttribute('data-itemid');
    itemOperations.markUnMark(id);
    showTotal();
    let tr = this.parentNode.parentNode;
    /*if(tr.className){
    tr.className = '';
    }
    else{
        tr.className = 'alert-danger';
    }*/
    tr.classList.toggle('alert-danger');
    console.log("I am Trash ",this.getAttribute('data-itemid'))
}
```

Total Records 3 Mark Records 1 UnMark Records 2

Id	Name	Price	Desc	Color	URL	Operations
101	abcd	588	adfad	#000000	afsdaa	
102	abcd	588	adfad	#000000	afsdaa	
103	abcd	588	adfad	#000000	afsdaa	

CRUD APP - Controller

- Deleting Records
  - Function *deleteRecords()*
    - Delete items using *itemOperations.remove()*
    - Print the updated table
    - Call this method inside *bindEvents()*

```
function deleteRecords(){
    var items = itemOperations.remove();
    printTable(items);
}
```

```
function printTable(items){
   var tbody =  document.querySelector('#items');
   tbody.innerHTML ='';
    items.forEach(item=>printRecord(item));
    showTotal();
}
```

```
document.querySelector('#remove').addEventListener('click',deleteRecords);
```

CRUD APP - Controller

- Function *search()*
  - Make a div class

Inside ***design.css***

```
.hide {
display: none;
}
```

- create drop down + text box and it is hidden by default
- on *init()* function apply the hide class
- *display none* : For hiding the div class
- Select the drop down and enter the value in Textbox and focus out (blur event)
- Use *filter function* to search
  - *It return the sub-array & print by using any printable function*

```
function bindEvents(){
    document.querySelector('#searchTxt').addEventListener('change',search);
    document.querySelector('#search').addEventListener('click',showHideSearchBar);
    document.querySelector('#remove').addEventListener('click',deleteRecords);
    document.querySelector('#add').addEventListener('click',addRecord);
}
```

```
function search(){
    var val = document.querySelector('#searchTxt').value;
    var key = document.querySelector('#searchby').value;
    var items = itemOperations.searchAll(key,val);
    printTable(items);
}
```

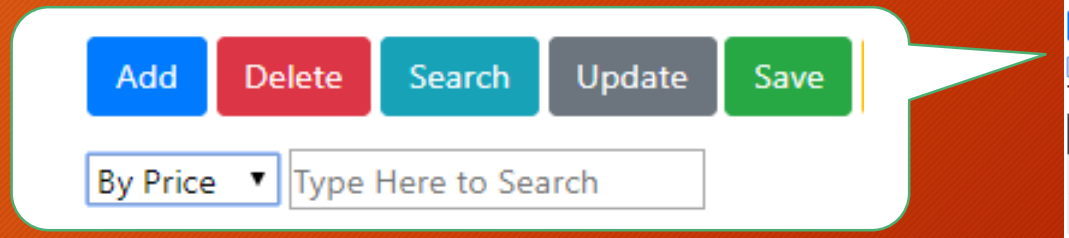
```
const showHideSearchBar = ()=> document.querySelector('#searchbar').classList.toggle('hide');

function init(){
    showTotal();
    bindEvents();
    showHideSearchBar();

}
```

Call inside *init()*

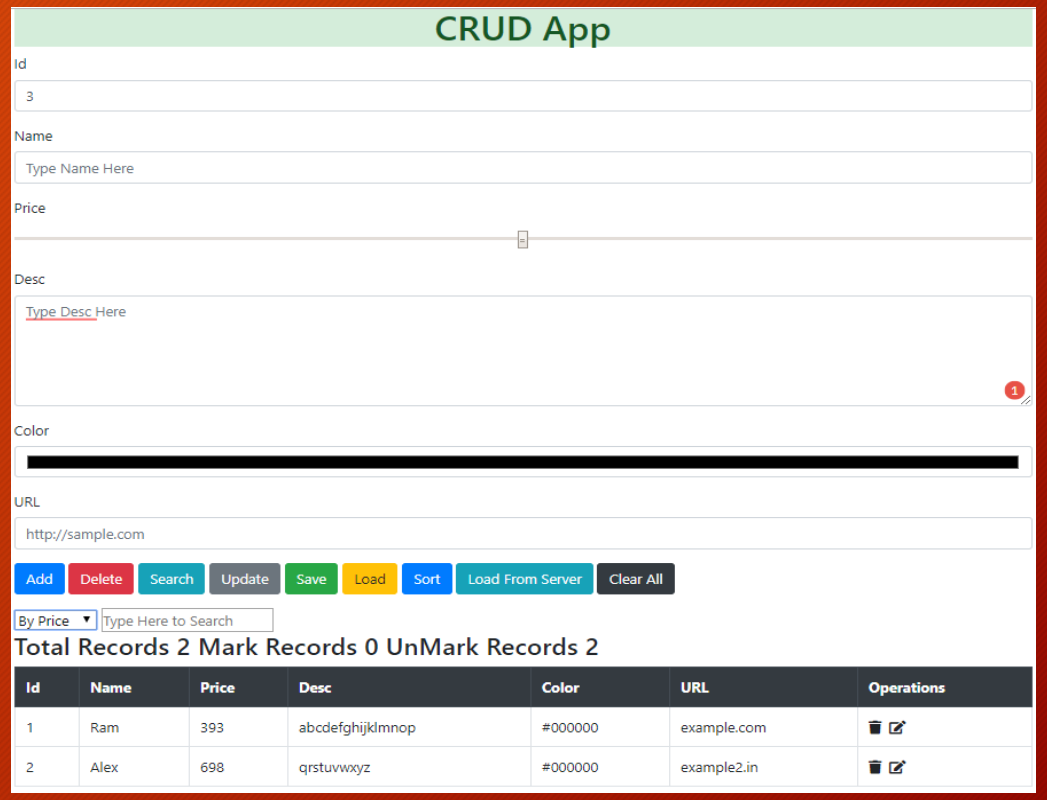
CRUD APP - Controller

- Search Bar HTML

```
<div id='searchbar'>
    <select id="searchby">
            <option value="price">By Price</option>
            <option value="name">By Name</option>
            <option value="id">By Id</option>
        </select>
    <input type="text" placeholder="Type Here to Search" id='searchTxt'>
</div>
```



- Output



CRUD APP - Controller

- Updating Records
  - Click on *edit* icon -> call edit() function
  - It will fetch the item-ID
  - Search by ID -> Get an object
  - Print the object inside textboxes using for-in loop
  - Edit the text inside the textboxes
  - Finally, click on update button, & it will replace the existing object with a new value

```
var itemObject;
function edit(){
    var id = this.getAttribute('data-itemid');
     itemObject = itemOperations.search(id);
     fillFields();
    console.log("i am Edit ",this.getAttribute('data-itemid'));
}

function fillFields(){
    for(let key in itemObject){
        if(key=='isMarked'){
            continue;
        }
        document.querySelector('#'+key).value = itemObject[key];
    }
}
```

CRUD APP - Controller

- *Function updateRecord()*

```
function bindEvents(){
    document.querySelector('#update').addEventListener('click',updateRecord);
    document.querySelector('#searchTxt').addEventListener('change',search);
    document.querySelector('#search').addEventListener('click',showHideSearchBar);
    document.querySelector('#remove').addEventListener('click',deleteRecords);
    document.querySelector('#add').addEventListener('click',addRecord);
}
```

```
function updateRecord(){
    for(let key in itemObject){
        if(key=='isMarked'){
            continue;
        }
        itemObject[key] = document.querySelector('#'+key).value;
    }
    printTable(itemOperations.items);
}
```

CRUD APP - Controller

- Sorting Operation
  - Here, we're sorting the table based on *price* value
  - Function *sortByPrice()*

```
const sortByPrice=()=>printTable(itemOperations.sortByPrice());
```

```
function bindEvents(){
    document.querySelector('#sort').addEventListener('click',sortByPrice);
    document.querySelector('#update').addEventListener('click',updateRecord);
    document.querySelector('#searchTxt').addEventListener('change',search);
    document.querySelector('#search').addEventListener('click',showHideSearchBar);
    document.querySelector('#remove').addEventListener('click',deleteRecords);
    document.querySelector('#add').addEventListener('click',addRecord);
}
```

CRUD APP - Controller

- *Save* Operation
  - Check if your browser supports local storage
  - Convert all items into String (from JSON format)
  - Fill up the data & press “Save Button”
  - Restart your application
    - Go to inspect-> Application -> local Storage

```
function saveRecords(){
    if(localStorage){
        localStorage.myitems = JSON.stringify(itemOperations.items);
        alert("Data Store....")
    }
    else{
        alert("NO Local Storage Feature is Supported in Ur Browser...")
    }
}
```

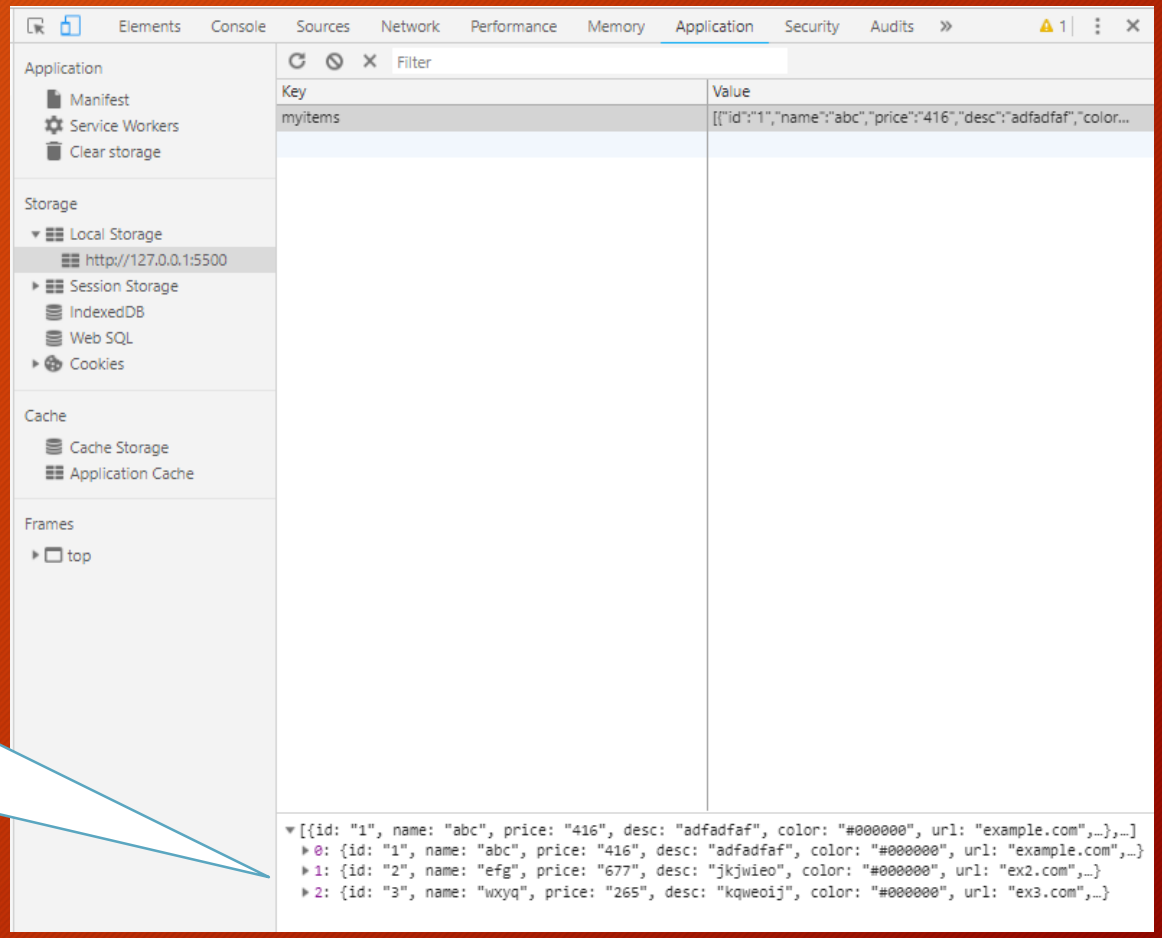
Add this function to *bindEvents()*

```
document.querySelector('#save').addEventListener('click',saveRecords);
```

CRUD App - Controller

- Checking the saved data
  - To remove local storage
    - *localStorage.clear* (on console)

```
[{id: "1", name: "abc", price: "316", desc: "adfadfa", color: "#000000", url: "example.com",…},…]
▶0: {id: "1", name: "abc", price: "316", desc: "adfadfa", color: "#000000", url: "example.com",…}
▶1: {id: "2", name: "def", price: "663", desc: "fadad", color: "#000000", url: "ex2.com", isMarked: false}
▶2: {id: "3", name: "wxyq", price: "223", desc: "kqwerad", color: "#000000", url: "ex3.com",…}
```



CRUD App - Controller

- Loading Records
  - If local storage is present
    - JSON.parse(localStorage.myitems)
    - pass it into itemOperations.items
    - Print items using *printable()*
  - Restart your application
    - Click on *Load* button

```
function loadRecords(){
    if(localStorage){
        if(localStorage.myitems){
            itemOperations.items = JSON.parse(localStorage.myitems);
            printTable(itemOperations.items);
        }
        else{
            alert("No Data to Load...");
        }
    }
    else{
        alert("NO Local Storage Feature is Supported in Ur Browser...")
    }
}
```

Add this function to *bindEvents()*

```
document.querySelector('#load').addEventListener('click',loadRecords);
```

Thank You