# Ricardos Adventure With Web Components!!!!

In the beginning.....

alrighty alrighty if you would like to follow a long I will be using the following tutorial

https://dev.to/thepassle/web-components-from-zero-to-hero-4n4m, and I will be attempting to change it over into TypeScript using the following resource

https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html, If anythign else I will add it and away we go

I began at 4:44pm

```
class TodoApp extends HTMLElement {

    constructor() {

    super();

    this._shadowRoot = this.attachShadow({'mode' : 'open'})

    this._shadowRoot.appendChild(template.content.cloneNode(true));

    this.$todoList = this._shadowRoot.querySelector('ul');
```

Our first journey with typescript so right now its crying about all the this.<stuff> statements because these types do not exist.

Fear not somehow we will fix this issue (flash backs of O.O programming)

ok so this is my first thought I gota declare these things somewhere but that $$$$$$ sign in $todoList kinda scares me but alas we must go forward so heres

### Solution 1.0

```
class TodoApp extends HTMLElement {
    shadowRoot: any
    $todoList: any
    constructor() {
        super();
        this.shadowRoot = this.attachShadow({'mode' : 'open'})
        this.shadowRoot.appendChild(template.content.cloneNode(true));
        this.$todoList = this.shadowRoot.querySelector('ul');
    }
}
```

so typescript has this cool father/mother of all data types called any dun ask how I knew i read about it somewhere randomly and thought oh this is cool and sure enough it has come to save me. so now i got my declared types and errors have vanished but are we really safe I duno

ONWARDS!!!!!!! (Untill that previous bug decides to appear :))

### lets dissect

```
const template = document.createElement('template');
template.innerHTML = `
<style>
    :host{
    display: block;
    font-family: sans-serif;
    text-align: Center;
    }

    button {
    border: none;
    cursor: pointer;
    }

    ul {
    list-style: none;
    padding: 0;
    }
</style>
<h1>To do</h1>
<input type = "text" placeholder = "add a new to do"></input>
<button>✅</button>
<ul id="todos"></ul>
`;
```

so whats happening here is its creating a template and then we are placing some inner html in it see the cool thing that templates allow us to clone our component and its super fast in comparison to the other methods or at least this is what the interwebs have told me and surely they wouldn't lie about anything :) from what I remember it has something to do with it getting saved somewhere so it just gets called back up again instead of having to do the whole read load shenanigans I could be completely wrong though I read about this awhile ago.

```
  this.shadowRoot = this.attachShadow({'mode' : 'open'})
  this.shadowRoot.appendChild(template.content.cloneNode(true));
```

alrighty alrighty so this is actually really cool piece of code the first line is attaching the shadowroot obvs and setting the mode to open which I have no idea what that does. what would happen if we set it to close ???? now the cool part is the second bit this is what i was talking about before about the clones this is the line that clones the template we set previously to the shadowROOOOOOT :) !another question: what if we defined the template outside of this file could we pull it up? what if we had multiple templates in one file?? Oh man so much stuff could happen !!!

THIS IS COOLL!!!

Note how in our to-do-app component, we've used a :host pseudo class, this is how we can add styling to the component from the inside and I've been wondering what the :host thing was doing I had guessed but now I know!!!

Big NOTE: An important thing to note is that the display is always set to display: inline;, which means you can't set a width or height on your element. So make sure to set a :host display style (e.g. block, inline-block, flex) unless you prefer the default of inline.

I FOUND THE CLOSED AND OPEN SHADOW DOM !!!!!

Open:

- Open shadow DOM allows us to create a sub DOM tree next to the light DOM to provide encapsulation for our components. Our shadow DOM can still be pierced by javascript like so: document.querySelector('our-element').shadowRoot

Closed:

- Closed shadow roots are not very applicable, as it prevents any external javascript from piercing the shadowroot. Closed shadow DOM makes your component less flexible for yourself and your end users and should generally be avoided.

ahh my old friend the ARROW FUNCTION !!!!!

```
  this._todos.forEach((todo, index) => {
              let $todoItem = document.createElement('div');
              $todoItem.innerHTML = todo.text;
              this.$todoList.appendChild($todoItem);
          });
```

Alrighty TypeScirpt is dead heres where I left it

```
import * as $ from 'jquery'
const template = document.createElement('template');
template.innerHTML = `
<style>
    :host{
    display: block;
    font-family: sans-serif;
    text-align: Center;
    }

    button {
    border: none;
    cursor: pointer;
    }

    ul {
    list-style: none;
    padding: 0;
    }
</style>
<h1>To do</h1>
<input type = "text" placeholder = "add a new to do"></input>
<button>✅</button>
<ul id="todos"></ul>
`;

class TodoApp extends HTMLElement {
    shadowRoot: ShadowRoot
    todosVar: any
    todo: Element
    todoList: Element
    constructor(){
        super();
        this.shadowRoot = this.attachShadow({'mode' : 'open'});
        this.shadowRoot.appendChild(template.content.cloneNode(true));
        this.todoList = this.shadowRoot.querySelector('ul');
    }

    _renderTodoList(){
        this.todoList.innerHTML = ' ';
        this.todosVar.forEach((todo, index) => {
            let $todoItem = document.createElement('div');
            $todoItem.innerHTML = todo.text;
            this.todoList.appendChild($todoItem);
        })
    }

    set todos(value) {
        this.todos = value;
        this._renderTodoList();
    }

    get todos() {
        return this.todos;
    }
}

window.customElements.define('to-do-app', TodoApp);
//let todo = <Element>document.querySelector('#to-do-app');
document.querySelector('to-do-app').todos = [
    {text: "Make a to-do list", checked: false},
    {text: "Finish blog post", checked: false} ];
```

Heres what I'm continuing with

```
const template = document.createElement('template');
template.innerHTML = `
<style>
    :host{
    display: block;
    font-family: sans-serif;
    text-align: Center;
    }

    button {
    border: none;
    cursor: pointer;
    }

    ul {
    list-style: none;
    padding: 0;
    }
</style>
<h1>To do</h1>
<input type = "text" placeholder = "add a new to do"></input>
<button>checkMark</button>
<ul id="todos"></ul>
`;
```

```
class TodoApp2 extends HTMLElement {
    constructor() {
        super();
        this._shadowRoot = this.attachShadow({'mode' : 'open'})

         this._shadowRoot.appendChild(template.content
         .cloneNode(true));
        this.$todoList = this._shadowRoot.querySelector('ul');
    }
    _renderTodoList() {
        this.$todoList.innerHTML = '';

        this._todos.forEach((todo, index) => {
            let $todoItem = document.createElement('div');
            $todoItem.innerHTML = todo.text;
            this.$todoList.appendChild($todoItem);
        });
    }

    set todos(value) {
        this._todos = value;
        this._renderTodoList();
    }

    get todos() {
        return this._todos;
    }
}

window.customElements.define('to-do-app', TodoApp2);
document.querySelector('to-do-app').todos = [
    {text: "Make a to-do list", checked: false},
    {text: "Finish blog post", checked: false}
];
```

oh and heres the html file

```
<html>
<script
 src="./RTutorial.js"></script>
</script>
<head>
  <title></title>
```

```
</head>
<body>
 <h1>bob</h1>
<to-do-app> </to-do-app>
</body>
</html>
```

### Big stufff

Attributes can only take strings

Properties can take rich data types like arrays

Alrighty so theres a big problem for some reason I lost the todos bit of code

### Path to salvation

1. so this happened after i did the add item part of the tutorial and I could not for the life of me get that damn list to reappear so I began by reverting all the code back to its orignal form and still had no luck
2. make a new file and then just straight up copy and past the code from the tutorial again fail
3. use the code I posted here which i verified had been working again fail
4. cry
5. move around document.query and get errors ultimately failing
6. cry some more
7. inspect the page to see what da heck is going on, ok looks like todos is not loading but it should be getting info from the document.query so wtf is going on
8. print statement the render function to see if its getting called and discover that theres an error in the console log. uncaught TypeError: Cannot set property 'todos' of null. hmm wtf you talking about console this wasn't an issue before. now why is todos null? well after some quick googling I discovered that the issue is with the page not loading all the way so I'm guessing when the document.query() gets called on to-do-app it cant find it because its not on the page yet so it kinda fails silently and then obviously it cant find todos so heres the fix
9. you use the defer keyword and it will wait to load the script

   <script src="test2.js" defer></script>

my current html document

```
<html>
<script src="test2.js" defer></script>
<head>
  <title></title>
</head>
```

```
<body>
<h1>bob</h1>
<to-do-app></to-do-app>
</body>
</html>
```

alrighty I'll finish this up after the gym

### querySelector

we need to investigate this query selector to my understanding it looks like its able to access the dom and then grabs a specified Element and if you look in the previous code

```
document.querySelector('to-do-app').todos =
```

you can see that it first grabs our web component and then it accesses the property todos now the big question is by simply adding like a button with an ID make it a property or is it that and adding the set and get functions.

### adding functionality to the button and the text box

first looking at input

```
<input type = "text" placeholder = "add a new to do"></input>
```

we have this piece of code in our shadow Dom you can see we have an input element and we want to use it so I believe we do this using jquery

such as with the following code

```
this.$input = this._shadowRoot.querySelector('input');
```

so now we got the input but how do we use it?

first lets look at button

```
<button>checkMark</button>
```

we have the above code in the shadowDom so we should know now by following the logic from before we can access this with a jquery call

```
this.$submitButton = this._shadowRoot.querySelector('button');
```

and now we can add functionality to this button by adding an event listener for when a user clicks it with the following code

```
this.$submitButton.addEventListener('click',
  this._addTodo.bind(this));
```

but what is this._addTodo.bind(this) well this bind function lets you continue to access the variables from where this is called well that seems to be the gist of it, heres a nice link on it

https://javascript.info/bind

### Connecting WebComponents

update on what my files look like

.html

```
<html>
<script src="RTutorial.js" defer></script>
<head>
  <title></title>
</head>
<body>
<h1>bob</h1>
<to-do-app todos="[{...}, {...}]"></to-do-apptodos></to-do-app>
</body>
</html>
```

to-do-app.js

```
const template = document.createElement('template');
template.innerHTML = `
<style>
    :host{
    display: block;
    font-family: sans-serif;
    text-align: Center;
    }

    button {
    border: none;
    cursor: pointer;
    }

    ul {
    list-style: none;
    padding: 0;
    }
</style>
```

```
<h1>To do</h1>
<input type = "text" placeholder = "add a new to do"></input>
<button>checkMark</button>
<ul id="todos"></ul>
`;


class TodoApp extends HTMLElement {
    constructor() {
        super();
        this._shadowRoot = this.attachShadow({'mode' : 'open'});

         this._shadowRoot.appendChild(template.content
         .cloneNode(true));

        this.$todoList = this._shadowRoot.querySelector('ul');
        this.$input = this._shadowRoot.querySelector('input');

        this.$submitButton = this._shadowRoot.querySelector('button');
        this.$submitButton.addEventListener('click',
         this._addTodo.bind(this));
    }

    _addTodo(){
        if(this.$input.value.length > 0){
            this.todos.push({ text: this.$input.value, checked:
             false})
            this._renderTodoList();
            this.$input.value = '';
        }
    }
    _renderTodoList() {
        this.$todoList.innerHTML = '';

        this._todos.forEach((todo, index) => {
            let $todoItem = document.createElement('to-do-item');
            $todoItem.setAttribute('text', todo.text);
            this.$todoList.appendChild($todoItem);
        });
    }

    set todos(value) {
        this._todos = value;
        this._renderTodoList();
    }
    get todos() {
        return this._todos;
    }
}

window.customElements.define('to-do-app', TodoApp);
document.querySelector('to-do-app').todos = [
    {text: "Make a to-do list", checked: false},
    {text: "Finish blog post", checked: false},
    {text: "helllo old chap", checked: false}
];
```

to-do-item.js

```
const template = document.createElement('template');
template.innerHTML =`
<style>
    :host {
        display: block;
        font-family: sans-serif;
    }

    .completed {
        text-decoration: line-through;
    }

    button {
        border: none;
        cursor: pointer:
    }
</style>
<li class="item">
    <input type = "checkbox">
    <label></label>
    <button>X</button>
</li>
`;

class TodoItem extends HTMLElement {
    constructor() {
        super();
        this._shadowRoot = this.attachShadow({'mode' : 'open'});
```

```
     this._shadowRoot.appendChild(template.content
     .cloneNode(true));

    this.$item = this._shadowRoot.querySelector('.item');
    this.$removeButton = this.shadowRoot.querySelector('button');
    this.$text = this._shadowRoot.querySelector('label');
    this.$checkbox = this.shadowRoot.querySelector('input')

    this.$removeButton.addEventListener('click', (e) =>{
        this.dispatchEvent(new CustomEvent('onRemove', {detail:
         this.index}));
    });

    this.$checkbox.addEventListener('click', (e) => {
        this.dispatchEvent(new CustomEvent('onToggle', {detail:
         this.index}));
    });
}

connectedCallBack(){
    if(!this.hasAttribute('text')) {
        this.setAttribute('text', 'placeHolder');
    }

    this._renderTodoItem();
}

_renderTodoItem() {
    if(this.hasAttribute('checked')) {
        this.$item.classList.add('completed');
        this.$checkbox.setAttribute('checked', ' ');
    }
    else{
        this.$item.classList.remove('completed');
        this.$checkbox.removeAttribute('checked');
    }
    this.$text.innerHTML = this._text;
}

static get observedAttributes() {
    return ['text'];
}

attributeChangedCallback(name, oldValue, newValue) {
```

```
            this.text = newValue;
        }
    }

    window.customElements.define('to-do-item', TodoItem);
```

### Problem

1. cant get to-do-item to connect with to-do-app, my first guess is I'm making a refrence to to-do-item but I know my laws of programming and even in this fancy world of "JAVASCRIPT" it cant possibly make use of something it doesn't know exists so in my mind I have two options

```
      let $todoItem = document.createElement('to-do-item');
                  $todoItem.setAttribute('text', todo.text);
```

   1. Import
      - So the internet says apparently you cant import a javascript file into another javascript file but I coulda sworn I was doing this in react native I mean I personally like to keep my stuff separated out and each file have a specific purpose
      - let us say f the internet
        - Seems you are not allowed to load up local files unless your using http in order to prevent sending a custom script to a web server, apparently react has stuff to avoid this problem and this is why I was able to do it before
      - not over yet

### Solution

- alrighty alrighty first things first we have to start a web server so we can get access to using the modules type. The reason for this is apparently there is some kind of security risk when loading scripts locally so they do this so you cant load a script into some random website.
- you also need to add an export declaration to your todoItem class and then you just need to add an import statement to your todoApp.
  - you can also add this export declaration to functions and then import them and use em its really cool.

### Problem

- So the checklist items are there but their text value is not appearing

### Path to Solution

- we know attributes are strings and what we want is a string so we're solid
- we need to use getAttribute()

- Tried using getAttribute() and looked at the code on the tutorial and discovered a typo but none of these actually fixed anything
- So we need to start walking through the code so we can figure out where the problem is occuring. The error seems that none of the connectedCallback methods are working along with the change stuff this is probabaly because my code is different then the tutorials and my components are loading with these attributes rather then loading and then gaining the attributes, im not really sure but its someting along these lines

### Solution

- Since all the functions that call render are not getting called you need to add render somewhere it wasn't before

```
    attributeChangedCallback(name, oldValue, newValue) {
            this._text = newValue;
            this._renderTodoItem();
        }
```

### What Got Done

- learned about attributes and properties
- made two web components
- connected two components together
- learned how to import javascript into another javascript
- learned some java script

### Final

.html

```
<html>
<script type="module" src="RTutorial.js" defer></script>
<head>
  <title></title>
</head>
<body>
<h1>bob</h1>
<to-do-app></to-do-app>
</body>
</html>
```

to-do-app.js

```
import  './to-do-item.js';

const template = document.createElement('template');
```

```
template.innerHTML = `
<style>
    :host{
    display: block;
    font-family: sans-serif;
    text-align: Center;
    }

    button {
    border: none;
    cursor: pointer;
    }

    ul {
    list-style: none;
    padding: 0;
    }
</style>
<h1>To do</h1>
<input type = "text" placeholder = "add a new to do"></input>
<button>checkMark</button>
<ul id="todos"></ul>
`;


class TodoApp extends HTMLElement {
    constructor() {
        super();
        this._shadowRoot = this.attachShadow({'mode' : 'open'});

         this._shadowRoot.appendChild(template.content
         .cloneNode(true));

        this.$todoList = this._shadowRoot.querySelector('ul');
        this.$input = this._shadowRoot.querySelector('input');

        this.$submitButton = this._shadowRoot.querySelector('button');
        this.$submitButton.addEventListener('click',
         this._addTodo.bind(this));
    }

    _addTodo(){
        if(this.$input.value.length > 0){
            this.todos.push({ text: this.$input.value, checked:
             false})
            this._renderTodoList();
            this.$input.value = '';
        }
    }
    _renderTodoList() {
        this.$todoList.innerHTML = '';

        this._todos.forEach((todo, index) => {
            let $todoItem = document.createElement('to-do-item');
            $todoItem.setAttribute('text', todo.text);
            this.$todoList.appendChild($todoItem);
        });
    }

    set todos(value) {
        this._todos = value;
        this._renderTodoList();
    }

    get todos() {
        return this._todos;
    }
}

window.customElements.define('to-do-app', TodoApp);
document.querySelector('to-do-app').todos = [
   {text: "Make a to-do list", checked: false},
   {text: "Finish blog post", checked: false},
   {text: "helllo old chap", checked: false},
   {text: "testing to see what", checked: false}
];
```

to-do-item.js

```
const template = document.createElement('template');
template.innerHTML =`
<style>
    :host {
        display: block;
        font-family: sans-serif;
    }

    .completed {
        text-decoration: line-through;
    }
    button {
        border: none;
        cursor: pointer:
    }
</style>
<li class="item">
    <input type = "checkbox">
    <label></label>
    <button>X</button>
</li>
`;

export default class TodoItem extends HTMLElement {
    constructor() {

        super();
        this._shadowRoot = this.attachShadow({'mode' : 'open'});

         this._shadowRoot.appendChild(template.content
         .cloneNode(true));

        this.$item = this._shadowRoot.querySelector('.item');
        this.$removeButton = this.shadowRoot.querySelector('button');
        this.$text = this._shadowRoot.querySelector('label');
        this.$checkbox = this.shadowRoot.querySelector('input')
        this.$value = this.shadowRoot.querySelector('text');

        this.$removeButton.addEventListener('click', (e) =>{
            this.dispatchEvent(new CustomEvent('onRemove', {detail:
             this.index}));
        });

        this.$checkbox.addEventListener('click', (e) => {
            this.dispatchEvent(new CustomEvent('onToggle', {detail:
             this.index}));
        });
    }

    connectedCallBack(){
        if(!this.hasAttribute('text')) {
            this.setAttribute('text', 'placeHolder');
        }
        else
        this._renderTodoItem();
    }

    _renderTodoItem() {
        if(this.hasAttribute('checked')) {
            this.$item.classList.add('completed');
            this.$checkbox.setAttribute('checked', '');
        }
        else{
            this.$item.classList.remove('completed');
            this.$checkbox.removeAttribute('checked');
        }
        this.$text.innerHTML = this._text;
    }

    static get observedAttributes() {
        return ['text'];
    }

    attributeChangedCallback(name, oldValue, newValue) {
        this._text = newValue;
        this._renderTodoItem();
    }
}

window.customElements.define('to-do-item', TodoItem);
```