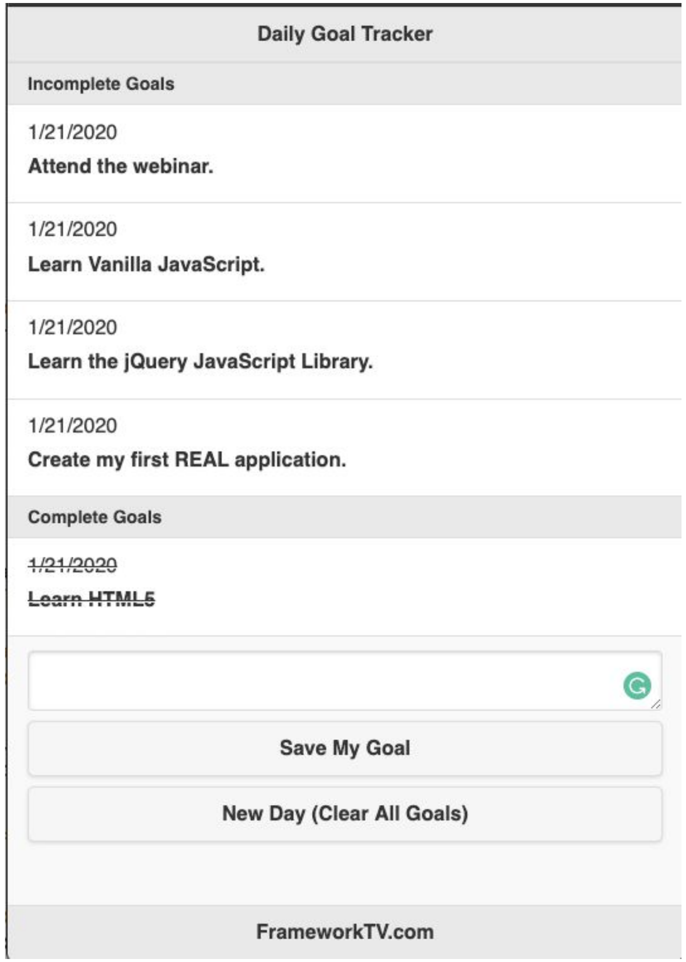
BUILD A

DAILY GOAL TRACKER APP

WITH MARK


Daily Goal Tracker

Incomplete Goals

1/21/2020
Attend the webinar.

1/21/2020
Learn Vanilla JavaScript.

1/21/2020
Learn the jQuery JavaScript Library.

1/21/2020
Create my first REAL application.

Complete Goals

~~1/21/2020~~
~~Learn HTML5~~

Save My Goal

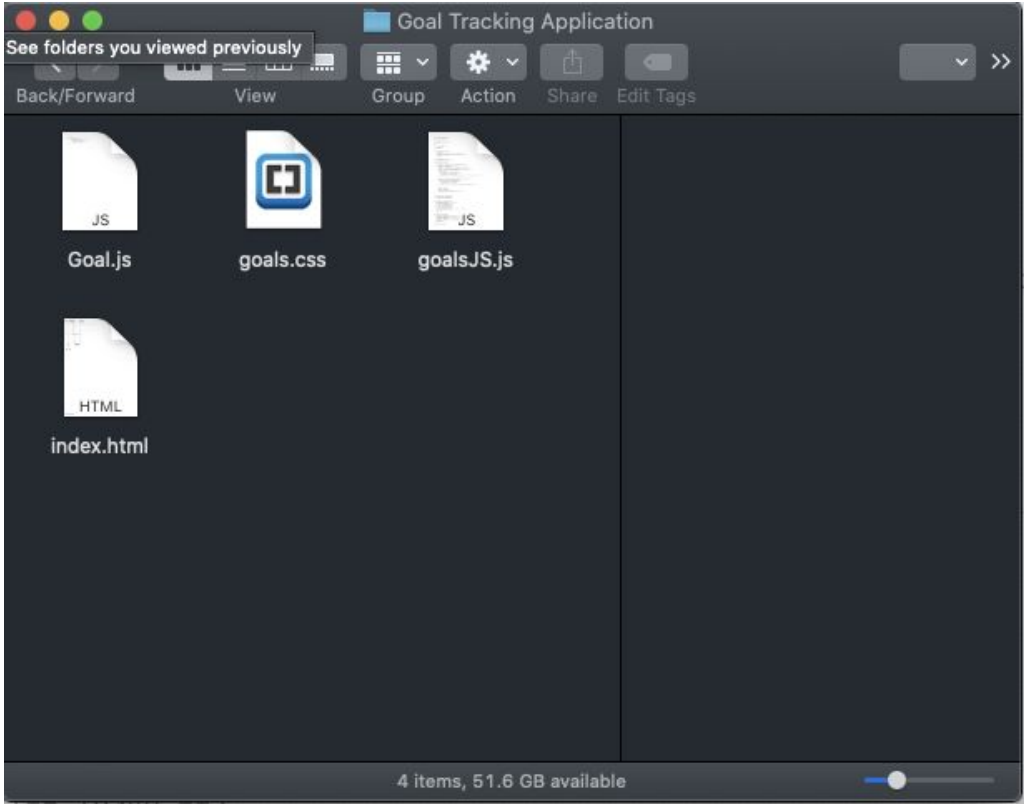
New Day (Clear All Goals)

FrameworkTV.com

Framework Project Guide

Welcome. This guide is for the Goal Tracking App project. The application allows the user to save goals daily and mark goals completed as the day proceeds.

This web-based application runs on any contemporary web browser and uses the jQuery Mobile library so that it is easily adaptable to a mobile application environment.



The Code

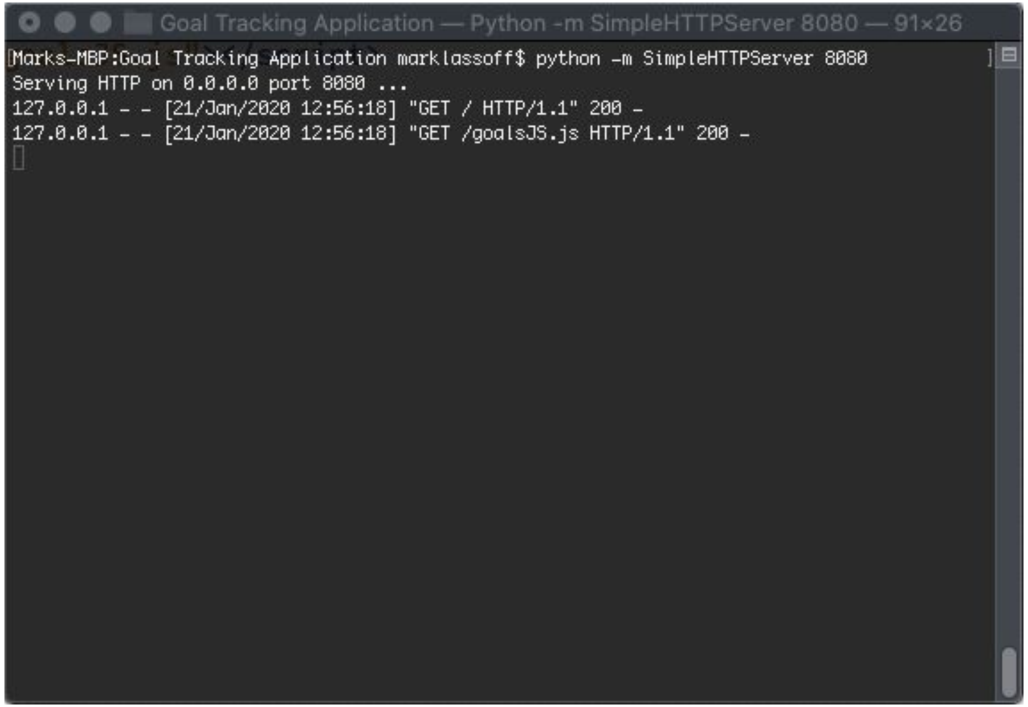
The code for this application is packaged with this guide. There are four files that come with the application code.

Goal.js: This file contains the goal object that is used each time a new goal is established.

goals.css: Most of the styling for this app is done using jQuery Mobile, however, this file contains a custom style that was used.

goalsJS.js: The JavaScript core of the application.

Index.html: The application scaffolding and links to libraries.

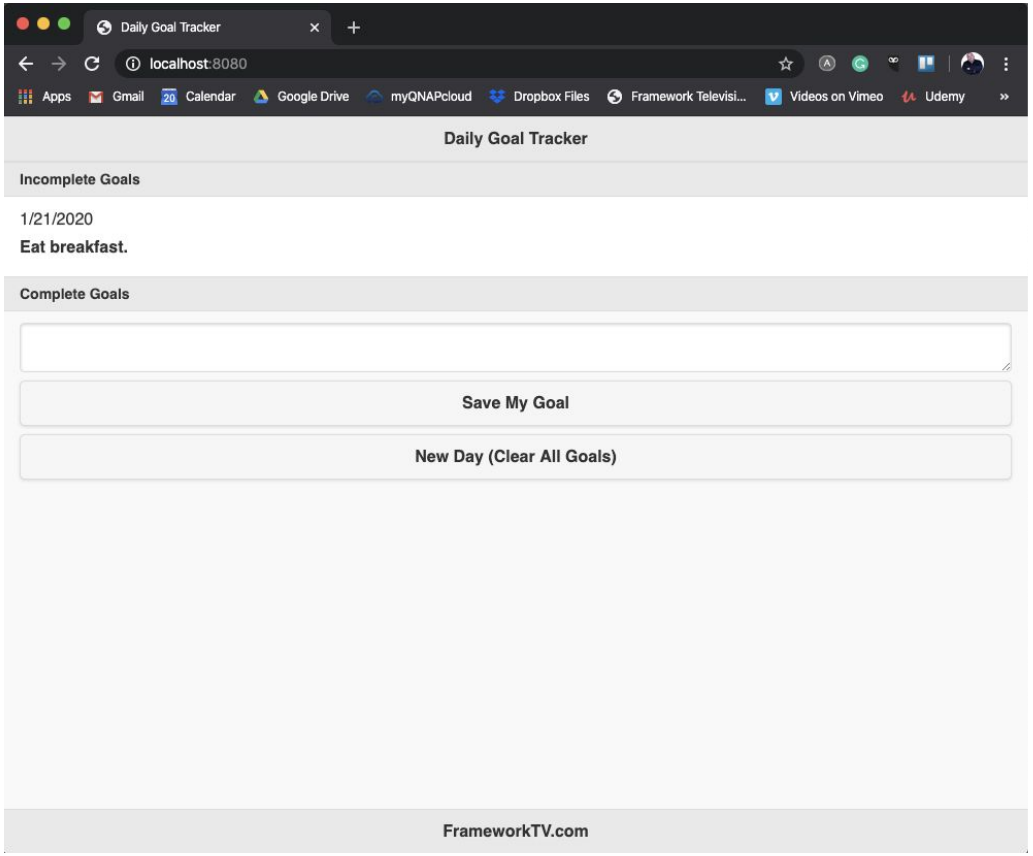

Goal Tracking Application — Python -m SimpleHTTPServer 8080 — 91×26

```
Marks-MBP:Goal Tracking Application marklassoff$ python -m SimpleHTTPServer 8080
Serving HTTP on 0.0.0.0 port 8080 ...
127.0.0.1 - - [21/Jan/2020 12:56:18] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [21/Jan/2020 12:56:18] "GET /goalsJS.js HTTP/1.1" 200 -
```

Running the Application
It is advisable to run this application in a web server. The easiest way to do this is on your command line. The Mac has a built in HTTP Server available.

1) Navigate to the folder that contains your code using the command line. (Remember **cd** changes the director. **cd myApp** will open a directory in your current directory called **myApp)**.

2) Start the server with the following command:
```
python -m SimpleHTTPServer
8080
```



Daily Goal Tracker

Incomplete Goals

1/21/2020
Eat breakfast.

Complete Goals

FrameworkTV.com

Running the Application

3) Open your web browser and type the following address in the address bar:

```
localhost:8080
```

Your application should load into the browser at this point.

Once you've got the application running, test it by creating and completing a few goals. To complete a goal, click on it and respond to the prompt.

The process for windows machines is similar by you may have to install Python and SimpleHTTPServer first.

```
<!DOCTYPE html>
<html>
   <head>
      <title>Daily Goal Tracker App</title>
   </head>
   <body>
   </body>
</html>
```

Building the App: Index.html
Like all web-based apps (or mobile apps based on the web languages HTML, CSS and JavaScript) we start with a basic document structure.

This structure underlies every web site and manny mobile apps that you use daily.

```
<head>
   <title>Goal Tracking App</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <script src="Goal.js"></script>
   <link rel="stylesheet" href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css" />
   <script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>
   <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js"></script>
   <link href="goals.css" type="text/css" rel="stylesheet" />
</head>
```

Next we'll add the necessary content to the document head element. The **meta** tag correctly sizes the viewport for the screen the app is being displayed on. It also ensures that magnification is at 100% with "initial-scale=1".

The balance of the code links to JavaScript code or style sheets. You'll notice that after the link to the Goal.js document there are three lines linking to external document. These three lines of code are loading the jQuery Mobile libraries. These were taken directly from the jQuery Mobile documentation.

The final **link** tag in the head of the document loads the little bit of custom CSS that the application uses to anchor the footer displayed in the interface.

```
<div data-role="page">
   <div data-role="header">
      <h1>Page Title</h1>
   </div>
   <!-- /header -->
   <div role="main" class="ui-content">
      <p>Page content goes here.</p>
   </div>
   <!-- /content -->
   <div data-role="footer">
      <h4>Page Footer</h4>
   </div>
   <!-- /footer -->
</div>
<!-- /page -->
```

A jQuery Mobile Page template actually has three distinct areas. The header section the main section and the footer section.

```
<!-- Start of first page -->
<div data-role="page" id="enterGoals">
<div data-role="header">
   <h1>Daily Goal Tracker</h1>
</div>
<!-- /header -->
```

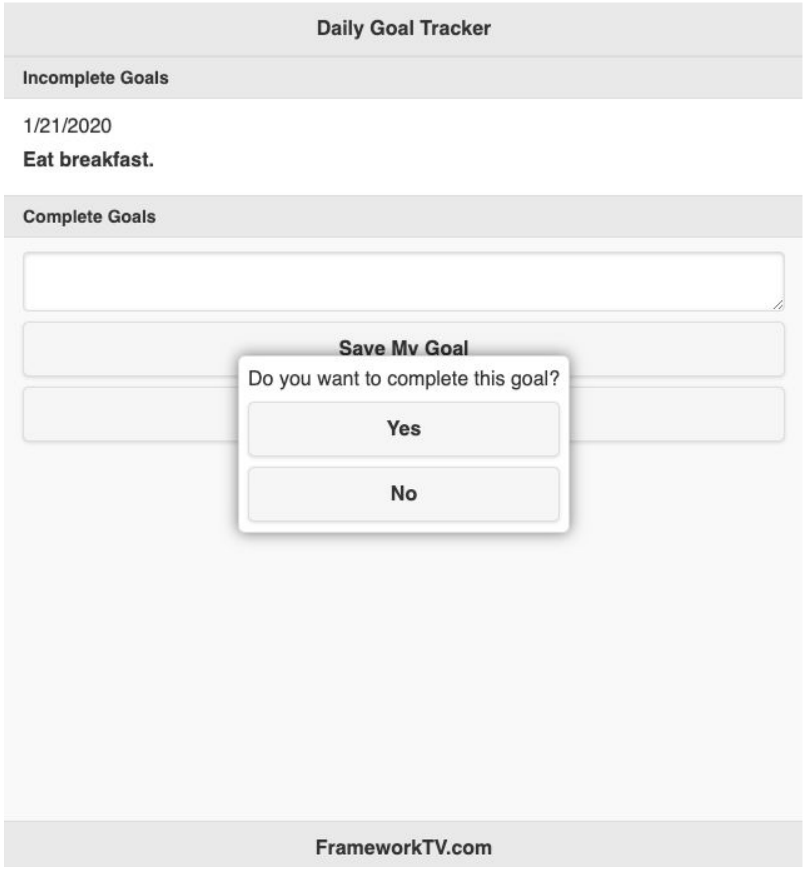
In the header for our document we only placed the name of the application in heading one tags.

```
<div role="main" class="ui-content">
   <div id="goalsList">
   </div>
   <form>
      <label for="goal">Enter Your Goal</label>
      <textarea id="goal" data-mini="true" cols="40"
rows="6"></textarea>
   </form>
   <button id="btnSave">Save My Goal</button>
   <button id="btnNewDay">New Day (Clear All
Goals)</button>
   <div data-role="popup" id="popupBasic">
      <p>Do you want to complete this goal?
      <p>
         <button id="btnComplete"
onclick="completeGoal()">Yes</button>
         <button id="btnNo"
onclick="closeDialog()">No</button>
   </div>
</div>
<!-- /content -->
```

Most of what is statically displayed on screen is in the main section. First you have a **div** id'ed as "goalsList". This is where the goals appear in a ListView component from jQuery.

Next is the form where Goals are entered. The form is simple with only a **textarea** to enter the text of the goal and a button which daves the goal. The next button is the New Day button which clears the entire memory of the application and lets the user start over with new goals.

The final **div** id'ed as "popupBasic" is for a popup that is hidden when the application first runs but confirms that the user wants to complete a goal.



Daily Goal Tracker

Incomplete Goals

1/21/2020

Eat breakfast.

Complete Goals

Save My Goal

Do you want to complete this goal?

FrameworkTV.com

This popup is hidden when the application loads.

```
<div data-role="footer" class="footer">
   <h4>FrameworkTV.com</h4>
</div>
<!-- /footer -->
</div><!-- /page -->
<script src="goalsJS.js"></script>
</body>
</html>
```

The HTML file is completed by the footer section from the jQuery Mobile page template and a link to a final script which is the core JavaScript for the page.

By convention, this link is at the bottom of the page to ensure that all the HTML components are loaded before the JavaScript executes.

```
.footer{
     bottom: 0;
     position: fixed;
     width: 100%
}
```

Building the App: goals.css

The goals.css file is short because most of the styling is done using jQuery Mobile's default styles.

The style here is added because (for reasons no one understands) the footer in the jQuery page is not anchored to the bottom of the page and instead floats by default.

```
class Goal {
 constructor(date, goalText, status) {
  this.date = date;
  this.goalText = goalText;
  this.status = status;
 }
}
```

Building the App: Goal.js

This file is loaded in the **head** section of the index.html document. This is the template for a goal. Each goal object includes a date (automatically date-time stamped when the goal is created), the text of the goal and a status which will be “complete” or “incomplete.”

Each time a new goal is created in the application a new instance of the Goal class is instantiated with its own date, goalText and status. The constructor function in the Goal class sets up the goal instance with the instance data (date, goalText and status.)

```
let allGoals = new Array();
let selectedGoalIndex = 0;


$("document").ready(function(){
     init();
});

let init = function(){
    //if no goals
    getGoals();
}
```

Building the App: goalJS.js

This file contains the primary JavaScript that drives the application.

The initial part of the code sets up two variables that are used in multiple functions and need to be global in scope (available everywhere in the script).

They are:
allGoals: An array object that contains all of the Goal objects currently in memory and their attributes (Date, goalText, status).

selectedGoalIndex: Holds the index of whichever goal is currently being edited/completed.

```
let allGoals = new Array();
let selectedGoalIndex = 0;


$("document").ready(function(){
     init();
});

let init = function(){
    //if no goals
    getGoals();
}
```

The “document ready” function is written in jQuery shorthand.

`$(“document”)`
Selects the document from the DOM.

`.ready()`
Runs when the entire document has been loaded.

Essentially this code runs the **init()** function when all of the HTML has been loaded and drawn on the screen and all external libraries have been completely loaded.

The **init()** function simply runs another function called **getGoals()**.

```
let getGoals = function(){
    let savedGoals =
JSON.parse(localStorage.getItem("goals"));
    if(savedGoals != "" && savedGoals != null){
        allGoals = savedGoals;
        let output = "<ul data-role='listview'
id='goalListView'>";
        output += "<li
data-role='list-divider'>Incomplete Goals</li>";
        for(var x = 0; x < allGoals.length; x++){
        console.log(x);
        if(allGoals[x].status != "complete"){
            let goalText = parseGoal(allGoals[x]);
            let rawDate = allGoals[x].date;
            output = output +  "<li
onclick='editGoal(" + x + ")'>" + goalText + "</li>";
        }
        }
        output += "<li
data-role='list-divider'>Complete Goals</li>";
        for(var x= 0; x < allGoals.length; x++){
        if(allGoals[x].status == "complete"){
            let goalText = parseGoal(allGoals[x]);
            let rawDate = allGoals[x].date;
            output = output +  "<li><strike>" +
goalText + "</strike></li>";
        }
        }
```

```
        output += "</ul>"
        console.log(output);
        $("#goalsList").html(output);

$("#goalListView").listview().trigger("cre
ate");
    }
}
```

The **getGoals()** function is easily the most complex in the application. The function loads the goals from localStorage, loops through them and builds the output for the listView in the document where all the goals are displayed.

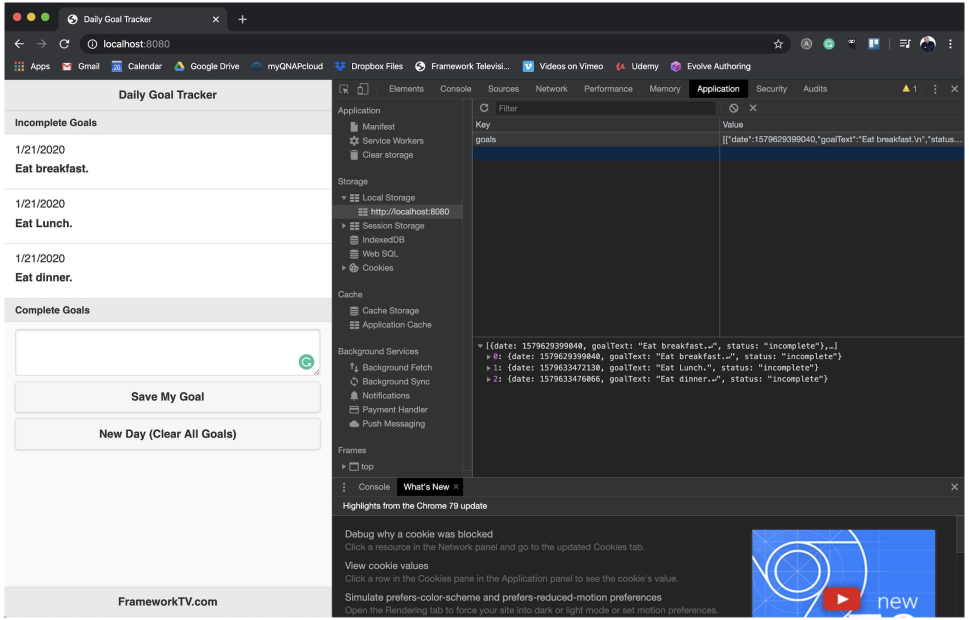
It then refreshes the listView so that the jQuery styles are applied.

Let’s break this down.

```
let savedGoals =
JSON.parse(localStorage.getItem("goals"));
```

The actual goal data is stored in something called localStorage which is a text repository in your browser. There you can store data and retrieve it. (For security data can only be retrieved by the domain that stored it.)

This code is getting all of the data stored in localStorage and assigned it to a variable called **savedGoals.**



Key	Value
goals	[{"date":1579629399040,"goalText":"Eat breakfast.\n","status...

```
▼ [{date: 1579629399040, goalText: "Eat breakfast.↵", status: "incomplete"},…]
  ▶ 0: {date: 1579629399040, goalText: "Eat breakfast.↵", status: "incomplete"}
  ▶ 1: {date: 1579633472130, goalText: "Eat Lunch.", status: "incomplete"}
  ▶ 2: {date: 1579633476066, goalText: "Eat dinner.↵", status: "incomplete"}
```

Using the “Developer Tools” Application tab in Chrome you can actually see the individual Goal objects stored in localStorage.

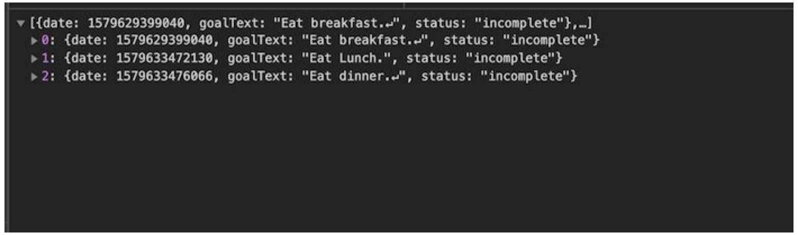
```
 if(savedGoals != "" && savedGoals != null){
        allGoals = savedGoals;
        let output = "<ul data-role='listview' 
id='goalListView'>";
        output += "<li 
data-role='list-divider'>Incomplete Goals</li>";
        for(var x = 0; x < allGoals.length; x++){
        console.log(x);
        if(allGoals[x].status != "complete"){
            let goalText = parseGoal(allGoals[x]);
            let rawDate = allGoals[x].date;
            output = output +  "<li 
onclick='editGoal(" + x + ")'>" + goalText + "</li>";
        }
        }
```

The **if** statement checks to make sure that what was returned from localStorage was not empty. The information is then copied into the global **allGoals** variable. The **output** variable is then introduced to begin building the HTML that will output the goals. Note that the jQuery Mobile library styles the unordered list produced as a listView.

The divider for the Incomplete goals is added to the output.

Next comes the loop where all the goals that are now stored in **allGoals** are iterated through.

```
for(var x = 0; x < allGoals.length; x++){
        console.log(x);
        if(allGoals[x].status != "complete"){
            let goalText = parseGoal(allGoals[x]);
            let rawDate = allGoals[x].date;
            output = output +  "<li
onclick='editGoal(" + x + ")'>" + goalText + "</li>";
        }
        }
```



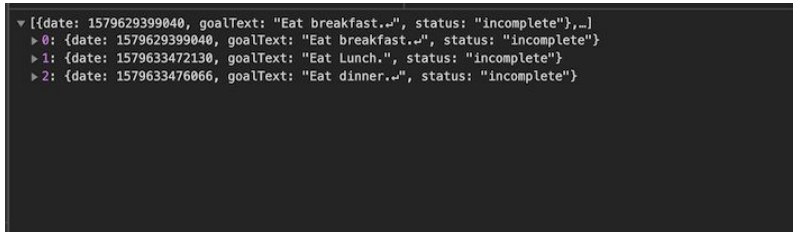
```
▼ [{date: 1579629399040, goalText: "Eat breakfast.↵", status: "incomplete"},…]
  ▶ 0: {date: 1579629399040, goalText: "Eat breakfast.↵", status: "incomplete"}
  ▶ 1: {date: 1579633472130, goalText: "Eat Lunch.", status: "incomplete"}
  ▶ 2: {date: 1579633476066, goalText: "Eat dinner.↵", status: "incomplete"}
```

This loop is critical as it will iterate through all the Goal objects returned from localStorage now stored in the **allGoals** array.

In this pass only the goals that have the status of Complete are added to the **output** variable creating the HTML that will be displayed.

Note the use of the **parseGoal()** function which formats the actual goal by parsing the date and goalText.

```
output += "<li data-role='list-divider'>Complete
Goals</li>";
        for(var x= 0; x < allGoals.length; x++){
        if(allGoals[x].status == "complete"){
            let goalText = parseGoal(allGoals[x]);
            let rawDate = allGoals[x].date;
            output = output +  "<li><strike>" +
goalText + "</strike></li>";
        }
}
```



```
▼ [{date: 1579629399040, goalText: "Eat breakfast.↵", status: "incomplete"},…]
  ▶ 0: {date: 1579629399040, goalText: "Eat breakfast.↵", status: "incomplete"}
  ▶ 1: {date: 1579633472130, goalText: "Eat Lunch.", status: "incomplete"}
  ▶ 2: {date: 1579633476066, goalText: "Eat dinner.↵", status: "incomplete"}
```

The list divider for the Complete Goals section is added to the **output** variable.

Then, Goal objects stored in the **allGoals** array are iterated through a second time and this time the goals with the "complete" status are added to output after being formatted by parseGoal(). The **<strike>** tag is used so these goals appear to be crossed out when displayed.

```
 output += "</ul>"
        console.log(output);
        $("#goalsList").html(output);

$("#goalListView").listview().trigger("create");
    }
}
```

Finally, the **output** variable is completed and the unordered list built within it is closed.

Using a jQuery shortcut, the list is placed in the HTML logical division id'ed as **goalsList**. The entire ListView object is then refreshed so jQuery can apply the appropriate formatting.

```
let parseGoal = function(goalText) {
    console.log(goalText);
    let goalDate = new Date(goalText.date);
    let month = goalDate.getMonth();
    let day = goalDate.getDate();
    let year = goalDate.getFullYear();
    let text = goalText.goalText;
    let out = (month+1) + "/" + day + "/" + year +
"<br/>";
    out += "<h2>" + text + "</h2>";
    return out;
}
```

The **parseGoal()** function is a utility function that pulls the date and text out of the Goal object passed too it. It uses the JavaScript date utility to get the month, date and year of the goal stored. (It is stored in millisecond format and therefore not easily parsed by humans).

Once the date and text are extracted they are assigned to a variable **out** and then passed back to the caller within the loops in the **getGoals()** function.

```
let saveGoal = function(){
    console.log("saveGoal()");
    let goalText =
document.getElementById('goal').value;
    let theGoal = new Goal(Date.now(),goalText,
"incomplete" );
    allGoals.push(theGoal);
    localStorage.setItem("goals",
JSON.stringify(allGoals));
    getGoals();
    $("#goal").val("");
}
```

When the “Save Goal” button is pressed in the interface, this function executes.

This function extracts the text the user typed in the interface and stores it as **goalText**. That text is then used to instantiate a Goal object (from Goal.js) along with the current date.time and the status string “incomplete”.

That goal object is added to the **allGoals** global array. The global array is then stored in localStorage so that the new value is saved and can be later retrieved.

The goals display is updated as **getGoals()** is called and the textbox where a goal is added is emptied.

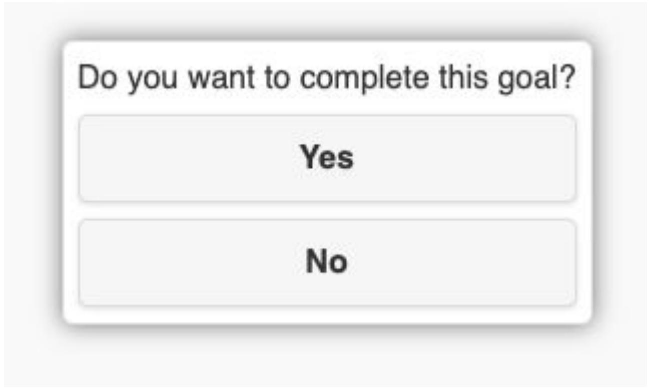
```
let newDay = function(){
    allGoals = [];
    localStorage.setItem("goals",
JSON.stringify(allGoals));
    $("#goalListView").html("<li></li>");

}
```

When the “New Dayl” button is pressed in the interface, this function executes.

This function erases all the goals in localStorage and memory and then resets the listView so that it reflects that no goals are currently stored.

Note that the **allGoals** global array is cleared by assigning the variable to an empty array.

```
let editGoal = function(goalIndex){
    $("#popupBasic").popup("open");
    selectedGoalIndex = goalIndex;
}
```



When an individual goal is pressed the **editGoal()** function runs. It opens the popup id'ed as "popupBasic" which displays the confirmation popup.

The confirmation popup was defined in index.html, but was hidden until this moment.

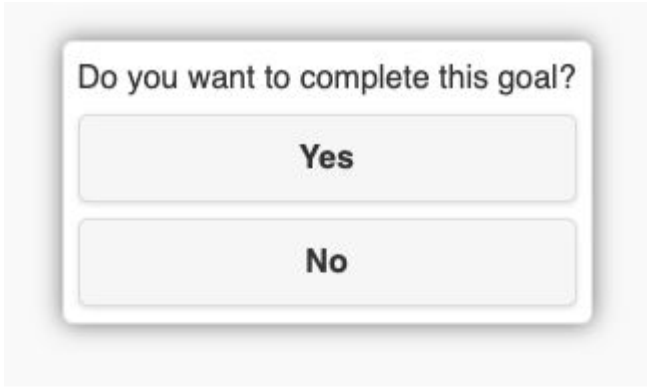
The global variable **selectedGoalIndex** is then updated with the index of the goal that is being completed.

The yes and no buttons each have their own call back functions.

```
let completeGoal = function(){
    closeDialog();
     allGoals[selectedGoalIndex].status = "complete";
     localStorage.setItem("goals",
JSON.stringify(allGoals));
     getGoals();
}

let closeDialog=function(){
    $("#popupBasic").popup("close");
}
```

If "Yes" is clicked, **completeGoal()** executes. It closes the dialog box, and then changes the status of the selected goal to "complete" in the **allGoals** global array. The updated array is then stored in localStorage and **getGoals()** is called to update the display.

If "No" is clicked, the dialog closes with no further action.


Do you want to complete this goal?
Yes
No