**Financial Tracker**

A Financial Tracker is a great tool to help you track your income and expenses. In this article I’ll show you how to create a simple yet effective Financial Tracker using **HTML, CSS, and JavaScript**.

**Main Components of the Financial Tracker**

The Financial Tracker I will be creating now consists of several key components:

* **User interface**: We will use HTML and CSS to design and style the user interface for our Financial Tracker.
* **Input forms**: We will use HTML forms to allow users to input their income and expenses.
* **Local storage**: We will use a browser’s local storage to store the user’s Financial data.
* **Calculation logic**: We will use JavaScript to calculate the user’s total income, expenses, and Financial balance.

By adding these essential components, I will be able to create a functional Financial Tracker that allows users to easily input, store, and view their budget data. Now, I will go through the process of implementing each of these features in our Financial Tracker application.

**Steps to creating the Financial Tracker**

I will create a simple budget tracker in five easy steps. But you can also add additional features to your Financial Tracker as desired.

* Setting up the project
* Designing the user interface
* Implementing the local storage
* Adding the calculation logic
* Testing and debugging

**1. Setting up the project**

In the first step, I will set up the necessary files and folders for our project. Create a new folder of any name for your Financial Tracker and open it in a text editor. Inside this folder, create an index.html file for the HTML structure, a style.css file for the CSS styles, and a scripts.js file for the JavaScript code.

In the index.html file, set up the basic HTML structure by including the necessary HTML tags, such as the <html>, <head>, and <body> tags. Your index.html file should look something like this:

<!DOCTYPE html>

**<**html**>**

**<**head**>**

**<**title**>**Budget Tracker | CodingNepal**</**title**>**

**<**link rel="stylesheet" href="style.css"**>**

**</**head**>**

**<**body**>**

<!-- HTML content goes here -->

**<**script src="script.js"**></**script**>**

**</**body**>**

**</**html**>**

This initial setup provides the structure for your budget tracker project and enables you to link your CSS and JavaScript files to your HTML file. You can now add HTML content such as headers, buttons, and text.

**2. Designing the user interface**

In the second step, we will design and style the Financial Tracker using HTML and CSS. To begin, we will add the basic HTML elements for the interface in “index.html”, including a title and a form for inputting income and expenses to track and manage our budget. The following code can be used to add these elements:

**<**div class="headerBar"**>**

**<**header**>**

**<**h1 class="title"**>**My Budget Tracker**</**h1**>**

**<**h2 class="topbar"**>**Your Current Balance**</**h2**>**

**<**p**>**

**<**span class="currency"**>**$**</**span**>**

**<**span class="balance"**></**span**>**

**</**p**>**

**<**header**>**

**</**div**>**

**<**div class="content"**>**

**<**h3 class="secondTitle"**>**Add a new transaction: **</**h3**>**

**<**div class="form"**>**

**<**form id="expForm"**>**

**<**div class="formLine left"**>**

**<**span for="type"**>**Type:**</**span**>**

**<**select id="type"**>**

**<**option value="chooseOne"**>**Choose one...**</**option**>**

**<**option value="income"**>**Income**</**option**>**

**<**option value="expense"**>**Expense**</**option**>**

**</**select**>**

**</**div**>**

**<**div class="formLine right"**>**

**<**span for="name"**>**Name:**</**span**>**

**<**input type="text" id="name"**>**

**</**div**>**

**<**div class="formLine right"**>**

**<**span for="amount"**>**Amount:**</**span**>**

**<**input type="text" id="amount"**>**

**</**div**>**

**<**button type="submit" class="buttonSave"**>**Add to transactions**</**button**>**

**</**form**>**

**</**div**>**

**</**div**>**

To display the transactions that have been input from the form, we will include a table in our HTML. This can be done with the following code:

**<**div class="content"**>**

**<**table class="table"**>**

**<**thead**>**

**<**tr**>**

**<**th**>**Type**</**th**>**

**<**th**>**Name**</**th**>**

**<**th**>**Amount**</**th**>**

**<**th**>**Options**</**th**>**

**</**tr**>**

**</**thead**>**

**<**tbody id="transactionTable"**></**tbody**>**

**</**table**>**

**</**div**>**

To style the Financial Tracker, we will open the “style.css” file and add some basic styles to define the layout and appearance of the user interface. CSS allows us to customize the font, color, and size of text, as well as the background color and layout of the page.

The following is an example of some basic CSS styles that can be used for the Financial Tracker:

\* **{**

*margin*: 0;

*font-family*: Arial, Helvetica, sans-serif;

**}**

body **{**

*min-height*: 1000px;

*display*: flex;

*flex-direction*: column;

*background-color*: rgb**(**106, 166, 245**)**;

*color*: black;

**}**

.headerBar **{**

*background-color*: blue;

*color*: bisque;

*text-align*: center;

*padding*: 20px;

**}**

.title **{**

*margin-bottom*: 20px;

*color*: white;

**}**

.topbar **{**

*margin-bottom*: 10px;

**}**

.currency **{**

*font-size*: 30px;

*font-weight*: 300;

**}**

.balance **{**

*font-size*: 30px;

*font-weight*: 300;

**}**

.content **{**

*width*: 580px;

*margin*: 0 auto;

*padding*: 3%;

*padding-left*: 6%;

**}**

.secondTitle **{**

*background-color*: blue;

*color*: white;

*text-align*: center;

*margin-top*: 100px;

*padding*: 20px;

*font-size*: 25px;

**}**

.form **{**

*padding*: 5px;

*padding-top*: 20px;

*padding-left*: 10%;

*justify-content*: center;

*background-color*: bisque;

**}**

.formLine **{**

*display*: inline-flex;

*padding*: 5px 0px;

**}**

.left **{**

*float*: left;

**}**

.right **{**

*float*: right;

*margin-right*: 100px;

**}**

input,

select **{**

*width*: 130px;

*margin-left*: 10px;

**}**

/\* table style \*/

table **{**

*width*: 100%;

**}**

thead **{**

*background-color*: blue;

*color*: white;

*line-height*: 30px;

**}**

tbody **{**

*background-color*: bisque;

*line-height*: 30px;

*text-align*: center;

**}**

/\* Button \*/

button **{**

*width*: 200px;

*color*: #fff;

*padding*: 10px;

*text-align*: center;

*font-size*: 1.1em;

*line-height*: 20px;

*background-color*: blue;

*border-radius*: 5px;

*margin*: 14px 25%;

*cursor*: pointer;

**}**

button:hover **{**

*box-shadow*: 0 0 0 2px grey;

*transition*: 0.5s;

**}**

a **{**

*text-decoration*: underline;

*cursor*: pointer;

**}**

**3. Implementing the local storage**

In the third step, I will use local storage, a feature of the browser that allows us to store data locally in the user’s browser. The setItem and getItem methods can be used to store and retrieve data from local storage in JavaScript.

The following is an example of how these methods can be used to store and retrieve a transaction object using local storage. Remember to paste the JavaScript codes in the script.js file.

document.getElementById**(**'expForm'**)**.addEventListener**(**'submit', addTransaction**)**;

// initial array of transactions, reading from localStorage

const transactions = JSON.parse**(**localStorage.getItem**(**'transactions'**))** || **[]**;

**4. Adding the calculation logic**

In the fourth step, we will implement the calculation logic for our Financial Tracker. To begin, we will create functions to add transactions from the input form and display them.

The following is an example of how this can be done using a transaction object that contains all of the relevant fields from the form:

function addTransaction**(**e**)** **{**

e.preventDefault**()**;

// get type, name, and amount

let type = document.getElementById**(**'type'**)**.value;

let name = document.getElementById**(**'name'**)**.value;

let amount = document.getElementById**(**'amount'**)**.value;

if **(**type != 'chooseOne'

&& name.length > 0

&& amount > 0**)** **{**

const transaction = **{**

type,

name,

amount,

id: transactions.length > 0 ? transactions**[**transactions.length - 1**]**.id + 1 : 1,

**}**

transactions.push**(**transaction**)**;

// localStorage

localStorage.setItem**(**'transactions', JSON.stringify**(**transactions**))**;

**}**

document.getElementById**(**'expForm'**)**.reset**()**;

showTransactions**()**;

updateBalance**()**;

**}**

const showTransactions = **()** => **{**

const transactionTable = document.getElementById**(**'transactionTable'**)**;

transactionTable.innerHTML = '';

for **(**let i = 0; i < transactions.length; i++**)** **{**

transactionTable.innerHTML += `

<tr>

<td>${transactions[i].type}</td>

<td>${transactions[i].name}</td>

<td>$${transactions[i].amount}</td>

<td><a class="deleteButton" onclick="deleteTransaction(${transactions[i].id})">

Delete</td>

</tr>

`;

**}**

**}**

After adding the ability to input and display transactions, we will create functions to delete transactions from the table. The following is an example of how this can be achieved:

const deleteTransaction = **(**id**)** => **{**

for **(**let i = 0; i < transactions.length; i++**)** **{**

if **(**transactions**[**i**]**.id == id**)** **{**

transactions.splice**(**i, 1**)**;

**}**

**}**

// localStorage

localStorage.setItem**(**'transactions', JSON.stringify**(**transactions**))**;

showTransactions**()**;

updateBalance**()**;

**}**

It’s important to constantly update the table and local storage at the end of each function in order to ensure that the correct set of transactions is displayed. To update the balance tracker in the header, we can create a updateBalance function that recalculates the balance after every add or delete function call.

This function should be called at the end of the add and delete functions to ensure that the balance is always up to date.

const updateBalance = **()** => **{**

let balance = 0;

transactions.forEach**((**transaction**)** => **{**

if **(**transaction.type === "income"**)** **{**

balance += Number**(**transaction.amount**)**;

**}** else if **(**transaction.type === "expense"**)** **{**

balance -= transaction.amount;

**}**

**})**;

document.querySelector**(**".balance"**)**.textContent = balance;

**}**

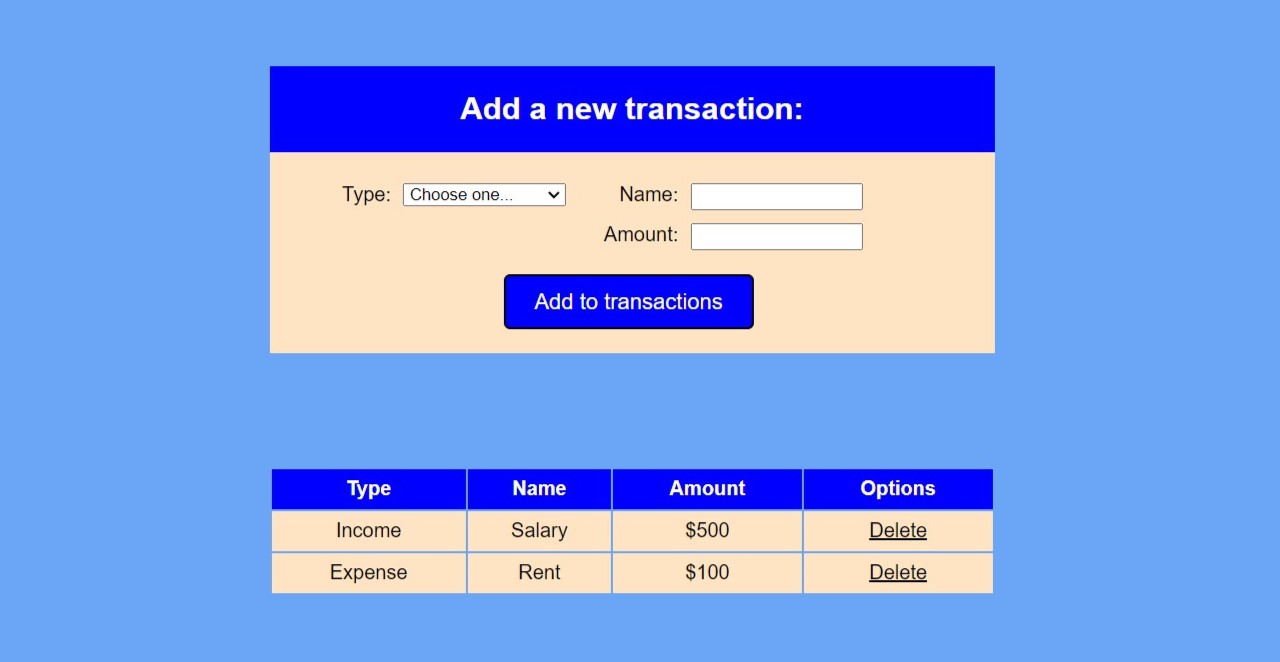
By implementing these functions, you’re now able to modify the budget object and save the changes to the local storage.

**5. Testing and debugging**

In the final step, we will test and debug our Financial Tracker to ensure it is functioning correctly. To do this, we will run the Financial Tracker in a web browser and interact with it to verify that it behaves as expected.

This includes adding and deleting income and expenses and checking that the budget data is calculated and displayed correctly, as well as ensuring the financial data is stored and retrieved properly from the browser’s local storage.

If any issues arise during testing, you can use the browser console to identify and fix the problem. To open the browser console press the F12 shortcut key or right-click on the page and select the Inspect option.

[](https://www.codingnepalweb.com/wp-content/uploads/2023/01/Budget-Tracker-Testing-and-debugging-1.jpg)

**Conclusion and Final Words**

By following the I have created a functional Financial Tracker using HTML, CSS, and JavaScript that allows users to easily track their income and expenses. There are many additional features and functionality that can be added to this Financial Tracker to make it more powerful and useful for users. Thank You !