# INTERNSHIP PROJECT

KANAGALA SAI VASANTH KUMAR

TO DO A LIST USING HTML

## ABSTRACT

To-do lists are essential tools for organizing tasks and improving productivity . In HTML, CSS, and JavaScript, developers create sleek and user-friendly to-do list applications where users can effortlessly add, edit, and remove tasks . These lists typically display tasks in a format that allows users to prioritize them based on importance or urgency.

HTML provides the structure for to-do lists, defining elements such as lists, input fields, and buttons . CSS is then used to style these elements, ensuring an attractive and intuitive interface . JavaScript adds functionality to the list, enabling dynamic features like adding new tasks, marking tasks as complete, and deleting tasks .

Users interact with the to-do list by entering tasks into input fields and pressing buttons to perform actions such as adding, editing, or deleting tasks .As tasks are added or modified, the list updates dynamically to reflect changes, providing users with real-time feedback on their actions.

Overall, to-do lists created with HTML offer a simple yet powerful solution for task management, allowing users to stay organized and focused on their priorities.By combining HTML, CSS, and JavaScript, developers can create customizable and responsive to-do list applications to meet the diverse needs of users.

To-do lists implemented in HTML provide users with a straightforward interface for managing tasks efficiently. With HTML as the backbone, developers utilize CSS to enhance the visual appeal of the list, ensuring an intuitive and aesthetically pleasing design. JavaScript plays a crucial role in adding interactivity to the list, enabling features like drag-and-drop functionality for task reordering.

Users can customize their to-do lists according to their preferences, such as organizing tasks by priority or due date. Additionally, developers can incorporate features like notifications or reminders to help users stay on track with their tasks. These customizable options enhance the user experience, making the to-do list application adaptable to various workflows and preferences.

Furthermore, to-do lists created with HTML offer accessibility features, ensuring that users with disabilities can also effectively utilize the application. Developers can implement keyboard shortcuts, screen reader compatibility, and other accessibility enhancements to make the to-do list inclusive and user-friendly for all individuals. Overall, HTML-based to-do lists provide a versatile solution for task management, catering to the diverse needs of users while prioritizing usability and accessibility.

# TABLE OF CONTENT

## OBJECTIVE

Creating a to-do list in HTML serves multiple objectives. Firstly, it aims to provide users with a straightforward and intuitive interface for managing their tasks efficiently. By incorporating HTML, CSS, and JavaScript, the objective is to create an interactive platform where users can easily add, edit, and delete tasks as needed . Additionally, the objective is to ensure the to-do list is visually appealing and user-friendly, encouraging users to engage with the application regularly.

Furthermore, the objective extends to enhancing the functionality of the to-do list by implementing features such as task prioritization, due date reminders, and task categorization. This aims to provide users with a comprehensive task management tool that can adapt to their individual preferences and workflow requirements. Moreover, the objective includes ensuring accessibility features are integrated into the to-do list, making it usable for individuals with disabilities. Overall, the objective of creating a to-do list in HTML is to facilitate effective task organization and productivity enhancement for users across various contexts and needs.

Expanding on the objectives for creating a to-do list in HTML, additional goals include enhancing user engagement through gamification elements such as progress tracking, rewards, or achievement badge. Moreover, the objective extends to integrating collaborative features, allowing multiple users to access and update the same to-do list simultaneously. This fosters teamwork and coordination, particularly in group projects or shared tasks.

Furthermore, the objective involves implementing responsive design principles to ensure the to-do list is accessible across various devices and screen sizes.Additionally, optimizing the performance of the application by minimizing loading times and reducing server requests contributes to providing a seamless user experience . Moreover, the objective includes integrating data analytics functionalities to gather insights into user behavior and preferences, enabling continuous improvement and optimization of the to-do list application.

# INTRODUCTION

In today's fast-paced world, staying organized and managing our daily tasks efficiently has become more crucial than ever. With numerous responsibilities demanding our attention, a well-structured to-do list has emerged as an indispensable tool for keeping track of our activities, boosting productivity, and maintaining a sense of accomplishment. However, in a digital landscape that extends across various devices and screen sizes, it's imperative that our to-do list isn't confined to a single platform. This is where the concept of a responsive to-do list comes into play.

A responsive to-do list ensures that your task management companion is as adaptable as you are. Whether you're meticulously planning on your desktop, quickly checking off completed items on your tablet, or adding new tasks on the go from your smartphone, a responsive to-do list seamlessly tailors itself to your preferred device, enhancing your ability to manage tasks effectively regardless of your location or the tools at hand.

In this comprehensive tutorial, we'll embark on a journey to create our very own responsive to-do list application. We'll harness the combined power of HTML, CSS, and JavaScript to craft a dynamic and user-friendly interface that not only captures and displays your tasks but also adapts intelligently to the screen size you're using. By following along with each step of this guide, you'll gain invaluable insights into front-end web development, understand the nuances of responsive design, and acquire the skills needed to bring your creative ideas to life in the digital realm.

So, whether you're a budding web developer eager to expand your skill set, a task management enthusiast seeking a tailored solution, or simply someone who loves to learn by doing, this tutorial is your gateway to building a responsive to-do list that seamlessly integrates into your daily routine. Let's dive in and unlock the potential of creating a personalized, cross-device task management experience like never before.

Creating a to-do list in HTML involves a systematic methodology to ensure the efficient implementation of features and functionalities. This methodology outlines the step-by-step process for designing and developing a user-friendly and interactive to-do list application.

## METHODLOGY

### Step 1: Planning and Requirements Gathering

The first step in creating a to-do list in HTML is to gather requirements and plan the project. This involves identifying the target audience, understanding their needs, and defining the key features and functionalities of the to-do list application. Additionally, it is essential to outline the user interface design and determine the technologies and tools required for development.

### Step 2: Designing the User Interface

Once the requirements are gathered, the next step is to design the user interface of the to-do list application. This involves creating wireframes or mockups to visualize the layout, structure, and flow of the application. Design considerations include the placement of input fields, buttons, and lists, as well as the overall aesthetics and usability of the interface.

### Step 3: HTML Markup

With the user interface design finalized, the HTML markup for the to-do list application is created. This involves structuring the elements of the application using HTML tags such as **<div>**, **<ul>**, **<li>**, **<input>**, and **<button>**. Each task in the to-do list is represented as a list item (**<li>**), and input fields and buttons are added for adding, editing, and deleting tasks.

### Step 4: Styling with CSS

Once the HTML markup is in place, the next step is to style the to-do list application using CSS. This involves applying styles to the HTML elements to enhance the visual appeal and usability of the application. CSS properties such as **color**, **font-size**, **padding**, and **margin** are used to customize the appearance of text, buttons, and input fields.

### Step 5: Adding Interactivity with JavaScript

The final step in creating a to-do list in HTML is to add interactivity using JavaScript. This involves implementing functionalities such as adding new tasks, editing existing tasks, marking tasks as complete, and deleting tasks. Event listeners and DOM manipulation techniques are used to handle user interactions and update the UI dynamically.

## SOURCE CODE

**Step 1 (HTML Code):**
To get started, we will first need to create a basic HTML file. In this file, we will include the main structure for our to-do list.

After creating the files just paste the following codes into your file. Make sure to save your HTML document with a .html extension, so that it can be properly viewed in a web browser.

Let's break down the different sections and elements:

**1. <!DOCTYPE html>**: This declaration specifies that the document is an HTML5 document type.

**2. <html lang="en">**: The opening tag of the HTML document, with the "lang" attribute set
cating the language of the document's content.

**3. <head>**: The head section of the HTML document, which contains metadata and links to external
resources.

**4. <title>**To-Do List**</title>**: Sets the title of the web page displayed in the browser's title bar or tab.

**5. <meta charset="UTF-8" />**: Specifies the character encoding for the document (UTF-8, which supports a wide range of

**6. <meta name="viewport" content="width=device-width" />**: Configures the viewport width to match the device's width, providing better responsiveness
sizes.

**7. <link rel="stylesheet" href="styles.css" />**: Links to an external CSS file named "styles.css" to apply styling to the HTML eleme

**8. <body>**: The body section of the HTML document, which contains the visible content of the web
page.

**9. <section class="container">**: A section element with the class "container," used to group related content.

**10. <div class="heading">**: A division element with the class "heading" for the title and image at the top of the page.

**11. <img class="heading__img" src="...">**: An image element with the class "heading__img" that displays an image of a laptop.

**12. <h1 class="heading__title">**To-Do List**</h1>**: A heading level 1 element with the class "heading__title," displaying the te - List."

**13. <form class="form">**: A form element with the class "form" for user input.

**14. <label class="form__label" for="todo">**...: A label for the input field, indicating what the user should input.

**15. <input class="form__input" type="text" id="todo" name="to-do" size="30" required>**: An input field for the user to enter their to-do item. It has a class "form__ir
"todo," a name "to-do," a size of 30 characters, and is required.

**16. <button class="button"><span>**Submit**</span></button>**: A button element with the class "button," containing the text "Sul

**17. <div>**: A division element containing an unordered list (ul) for displaying the to-do list items.

**18. <ul class="toDoList">**: An unordered list with the class "toDoList," where to-do list items will be added dynami

**19. <script src="script.js"></script>**: Links to an external JavaScript file named "script.js" for adding interactivity and functionali
page.

This is the basic structure of our to-do list using HTML, and now we can move on to styling it using CSS.

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>To-Do List</title>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width" />
    <link rel="stylesheet" href="styles.css" />
  </head>
  <body>
    <section class="container">
      <div class="heading">
        <img class="heading__img" src="https://s3-us-west-
2.amazonaws.com/s.cdpn.io/756881/laptop.svg">
        <h1 class="heading__title">To-Do List</h1>
      </div>
      <form class="form">
        <div>
          <label class="form__label" for="todo">~ Today I need to
~</label>
          <input class="form__input"
               type="text"
               id="todo"
               name="to-do"
               size="30"
               required>
          <button class="button"><span>Submit</span></button>
        </div>
      </form>
      <div>
        <ul class="toDoList">
        </ul>
      </div>
    </section>
    <script src="script.js"></script>
  </body>
</html>
```

**Step 2 (CSS Code):**
Once the basic HTML structure of the to-do list is in place, the next step is to add styling to the to-do list using CSS.

Next, we will create our CSS file. In this file, we will use some basic CSS rules to style our to-do list.

**1. @import**: This line imports a Google Fonts stylesheet, making the "Gochi Hand" font
use on the page.

**2. body**: This block of code styles the entire body of the webpage. It sets the background
inimum height, padding, box-sizing, and various other properties to center the content
rizontally and vertically. It also sets the text color and font family to "Gochi Hand" for
e font style.

**3. @media only screen and (min-width: 500px)**: Inside this media query, the minimum height of the body is adjusted to take up the full vi
width is at least 500px.

**4. .container**: This class styles a container element. It sets the width, minimum and
dth, background color, border-radius, box shadow, and padding. It also adds a
t background pattern.

**5. .heading**: This class styles a container for the heading. It aligns the content both
and vertically and adds a margin at the bottom.

**6. .heading__title**: This class styles the heading title, giving it a rotated appearance, rounded
d color, and font size. Inside the media query, the font size is increased for larger screens.

**7. .form__label**: This class styles labels for form inputs, adding a margin at the bottom.

**8. .form__input**: This class styles form input fields. It sets the box-sizing, background color,
radius, and border style. The focus state adjusts the border color.

**9. @media only screen and (min-width: 500px)** :Inside this media query, the width of the input fields are adjusted for larger screens.

**10. .button**: This class styles a button element. It sets various properties for the button including padding, rotation, font family, border radius, box shadow,background

**11. .button span**: This styles the text inside the button, giving it a background color, -radius. padding, border,

**12. .button:active**, **.button:focus**: These styles define the appearance of the button when it is

**13. .toDoList**: This class styles a to-do list on the webpage, aligning the text to the left.

**14. .toDoList li**: This class styles individual list items in the to-do list, adding padding.

**15. .toDoList li:hover**: This style is applied when hovering over a list item, giving it a with a wavy effect and changing the color.

This will give our to-do list an upgraded presentation. Create a CSS file with the name styles.css and paste the given codes into your CSS file. Remember that you must create a file with the .css extension.

```
@import url("https://fonts.googleapis.com/css?family=Gochi+Hand");
body {
  background-color:
#a39bd2;   min-height:
70vh;   padding: 1rem;
box-sizing: border-box;
display: flex;   justify-
content: center;   align-
items: center;   color:
#494a4b;
  font-family: "Gochi Hand", cursive;
  text-align: center;   font-
size: 130%;
}

@media only screen and (min-width: 500px) {
body {
    min-height: 100vh;
  }
}
.container {   width:
100%;   height: auto;
min-height: 500px;
max-width: 500px;
min-width: 250px;
background: #f1f5f8;
  background-image: radial-gradient(#bfc0c1 7.2%, transparent
0);   background-size: 25px 25px;   border-radius: 20px;
  box-shadow: 4px 3px 7px 2px
#00000040;   padding: 1rem;   box-
sizing: border-box;
}

.heading {   display:
flex;   align-items:
center;   justify-
content: center;
margin-bottom: 1rem;
}

.heading__title {
transform: rotate(2deg);
padding: 0.2rem 1.2rem;
```

```
  border-radius: 20% 5% 20% 5%/5% 20% 25% 20%;
background-color: rgba(0, 255, 196, 0.7);   font-
size: 1.5rem;
}

@media only screen and (min-width: 500px) {
  .heading__title {     font-
size: 2rem;
  }
}
.heading__img {
width: 24%;
}

.form__label {
display: block;
margin-bottom: 0.5rem;
}

.form__input {
  box-sizing: border-box;   background-
color: transparent;   padding: 0.7rem;
  border-bottom-right-radius: 15px 3px;
border-bottom-left-radius: 3px 15px;
border: solid 3px transparent;   border-
bottom: dashed 3px #ea95e0;   font-family:
"Gochi Hand", cursive;   font-size: 1rem;
  color: rgba(63, 62, 65,
0.7);   width: 70%;   margin-
bottom: 20px;
}
.form__input:focus {
  outline: none;
  border: solid 3px #ea95e0;
}

@media only screen and (min-width: 500px) {
  .form__input {
width: 60%;
  }
}
.button {
padding: 0;
border: none;
  transform: rotate(4deg);
transform-origin: center;   font-
family: "Gochi Hand", cursive;
text-decoration: none;   padding-
bottom: 3px;   border-radius: 5px;
box-shadow: 0 2px 0 #494a4b;
  transition: all 0.3s cubic-bezier(0.175, 0.885, 0.32, 1.275);
  background-image:
url("data:image/gif;base64,R0lGODlhBAAEAIABAAAAAAAAACH/C1hNUCBEYXRhW
E1QPD94cGFja2V0IGJlZ2luPSLvu78iIGlkPSJXNU0wTXBDZWhpSHpyZVN6TlRjemtjO
WQiPz4gPHg6eG1wbWV0YSB4bWxuczp4PSJhZG9iZTpuczptZXRhLyIgeDp4bXB0az0iQ
WRvYmUgWE1QIENvcmUgNS4wLWMwNjEgNjQuMTQwOTQ5LCAyMDEwLzEyLzA3LTEwOjU3O
jAxICAgICAgICAiPiA8cmRmOlJERiB4bWxuczpyZGY9Imh0dHA6Ly93d3cudzMub3JnL
zE5OTkvMDIvMjItcmRmLXN5bnRheC1ucyMiPiA8cmRmOkRlc2NyaXB0aW9uIHJkZjphY
m91dD0iIiB4bWxuczp4bXA9Imh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8iIHhtb
G5zOnhtcE1NPSJodHRwOi8vbnMuYWRvYmUuY29tL3hhcC8xLjAvbW0vIiB4bWxuczpzd
FJlZj0iaHR0cDovL25zLmFkb2JlLmNvbS94YXAvMS4wL3NUeXBlL1Jlc291cmNlUmVmI
yIgeG1wOkNyZWF0b3JUb29sPSJBZG9iZSBQaG90b3Nob3AgQ1M1LjEgV2luZG93cyIge
G1wTU06SW5zdGFuY2VJRD0ieG1wLmlpZDo5NUY1OENCRDdDMDYxMUUyOTEzMEE1MEM5Q
zM0NDVBMyIgeG1wTU06RG9jdW1lbnRJRD0ieG1wLmRpZDo5NUY1OENCRTdDMDYxMUUyO
TEzMEE1MEM5QzM0NDVBMyI+IDx4bXBNTTpEZXJpdmVkRnJvbSBzdFJlZjppbnN0YW5jZ
UlEPSJ4bXAuaWlkOjk1RjU4Q0JCN0MwNjExRTI5MTMwQTUwQzlDMzQ0NUEzIiBzdFJlZ
jpkb2N1bWVudElEPSJ4bXAuZGlkOjk1RjU4Q0JDN0MwNjExRTI5MTMwQTUwQzlDMzQ0N
```

UEzIi8+IDwvcmRmOkRlc2NyaXB0aW9uPiA8L3JkZjpSREY+IDwveDp4bXBtZXRhPiA8P
3hwYWNrZXQgZW5kPSJyIj8+Af/+/fz7+vn49/b19PPy8fDv7u3s6+rp6Ofm5eTj4uHg3
97d3Nva2djX1tXU09LR0M/OzczLysnIx8bFxMPCwcC/vr28u7q5uLe2tbSzsrGwr66tr

Step 3 (JavaScript Code):
Finally, we need to create a function in JavaScript.

Let's break down the code step by step:

1. An Immediately Invoked Function Expression (IIFE) is used to encapsulate the entire code
This pattern is often used to create a private scop
ing the global scope.

2. Inside the IIFE, several state variables and UI variables are declared and initialized:

- toDoListArray: An array that will store the To-Do list items.
- form, input, an ul: These variables are used to reference specific elements
tructure. form refers to the HTML form element, input refers
ld within the form, and ul refers to an unordered list where the
To- ill be displayed.

3. Two event listeners are set up:

- The first listener is for the form's submit event. When the form is submitted (user presses Enter or clicks the "Submit" button), the function prevents the default form submission behavior (page reloads), generates a unique ID for the To-Do item, retrieves the input value, and then invokes two functions: addItemToDOM() and addItemToArray().
- The second listener is for the unordered list (ul). It listens for click events within the list. When an item within the list is clicked, it checks if the clicked element has a data-id attribute. If it does, it invokes two functions: removeItemFromDOM() and removeItemFromArray().

4. Four functions are defined within the IIFE:

- addItemToDOM(itemId, toDoItem): This function creates a new list item (<li>) element, assigns it a ttribute with the provided itemId, sets
tem to the unordered list (ul). he provided toDoItem, and appends the
list

- addItemToArray(itemId, toDoItem): This function adds a new object to the toDoListArray array, where each object represents a To-Do item with its associated itemId and toDoItem text.
- removeItemFromDOM(id): This function finds the list item element with the specified data-id attribute (id) and removes it from the unordered list (ul).
- removeItemFromArray(id): This function filters the toDoListArray to exclude the object with the matching itemId, effectively removing the corresponding To-Do item from the array.

5. The IIFE is immediately invoked by enclosing the entire code block in parentheses and

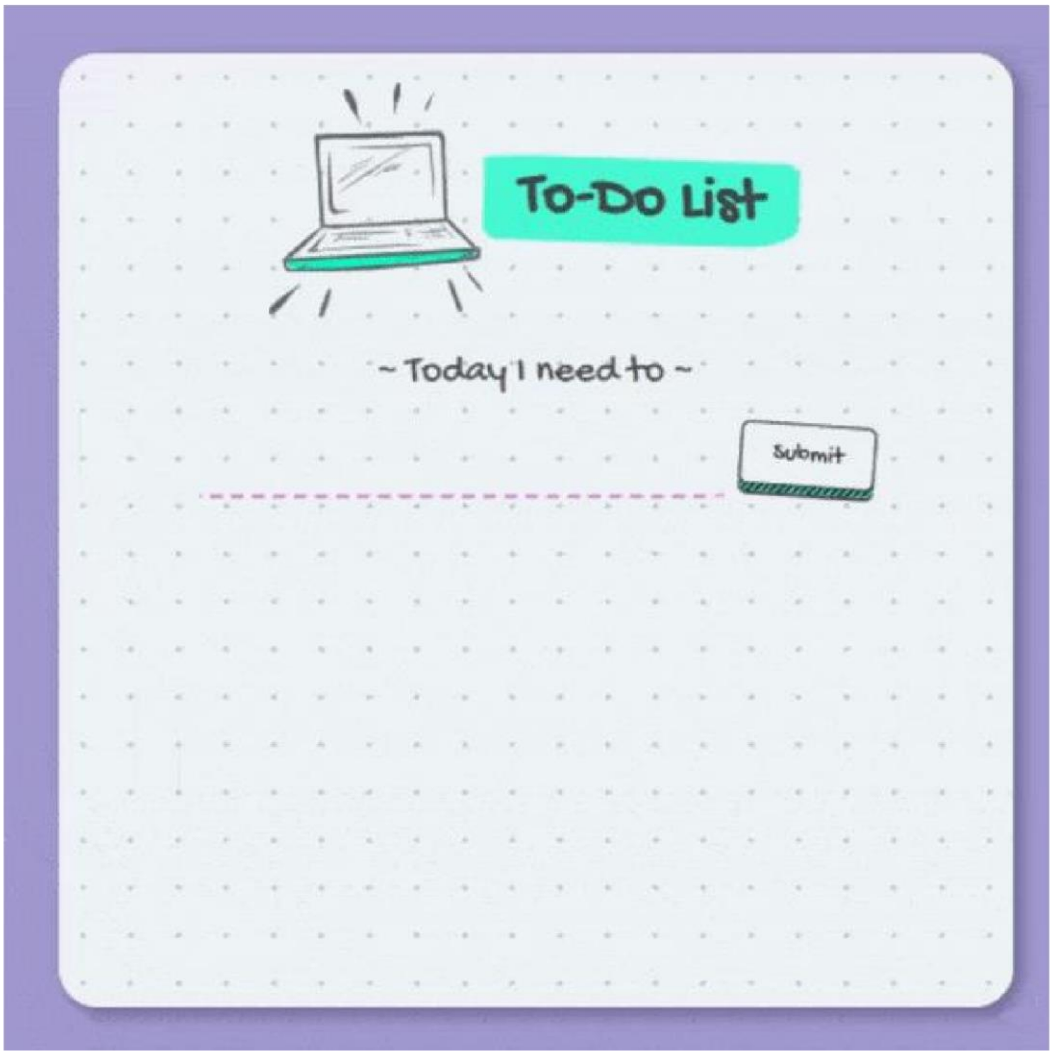
adding () at the end. This ensures that the code within the IIFE is executed as soon as the JavaScript file is loaded.

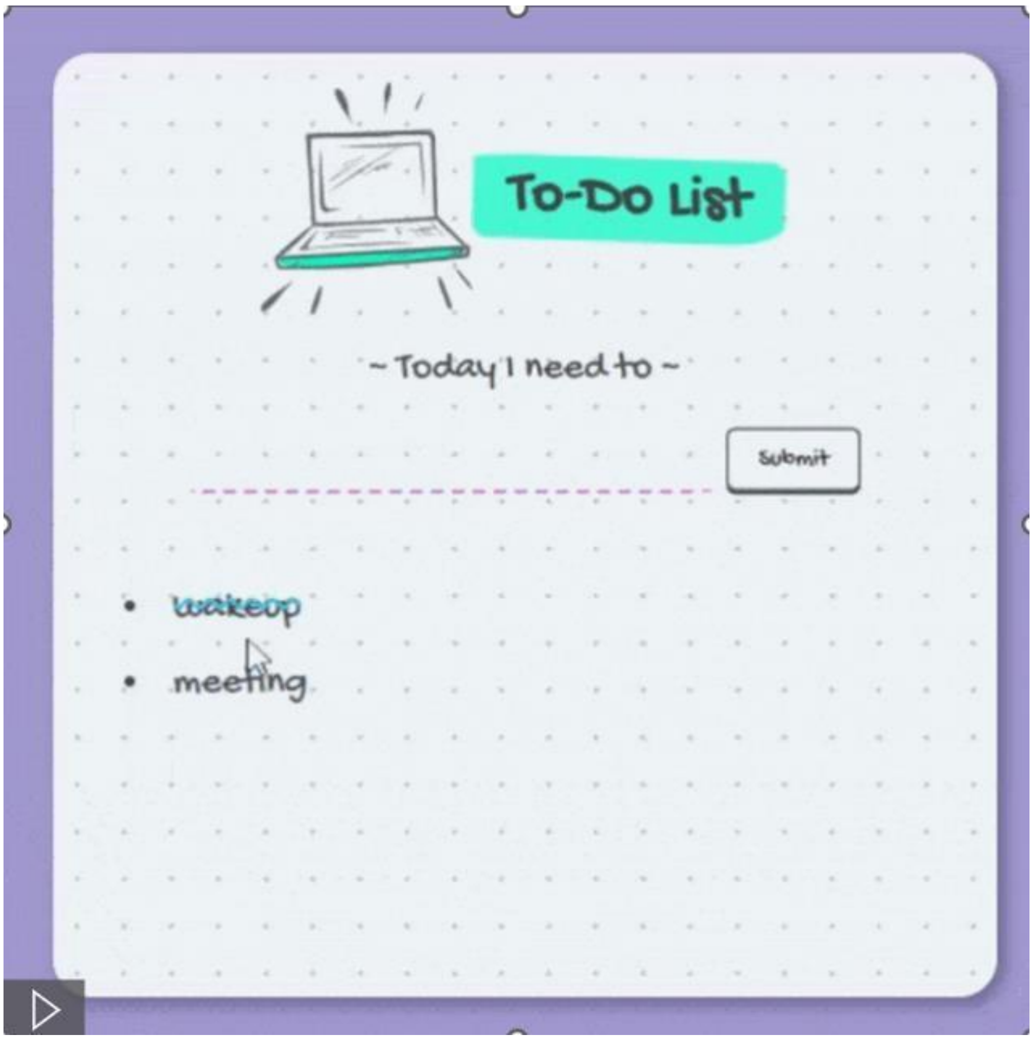
Create a JavaScript file with the name of script.js and paste the given codes into your JavaScript file and make sure it's linked properly to your HTML document, so that the scripts are executed on the page. Remember, you've to create a file with .js extension.

```
// IEFE
(() => {
  // state variables
let toDoListArray = [];
  // ui variables
  const form = document.querySelector(".form");
const input = form.querySelector(".form__input");
const ul = document.querySelector(".toDoList");

  // event listeners
  form.addEventListener('submit', e => {
// prevent default behaviour - Page reload
e.preventDefault();      // give item a unique
ID      let itemId = String(Date.now());
// get/assign input value      let toDoItem =
input.value;      //pass ID and item into
functions      addItemToDOM(itemId ,
toDoItem);      addItemToArray(itemId,
toDoItem);
```

```
    // clear the input box. (this is default behaviour but we got
rid of that)    value =
                ;

    input.
  });           tListener('click', e => {
ul.              e.target.getAttribute('data-id')
     let id =   return // user clicked in something else
   if (!id)      through to functions
    //pass id   emFromDOM(id);
                emFromArray(id);

  });
                   addItemToDOM(itemId,
  // function   oDoItem) {  an li
function         = document.createElement('li')
create      co  tribute("data-id", itemId);
li      li.     tem text to li
// add              = toDoItem
li.innerText     to the DOM
// add li       dChild(li);
ul.
  }
function        ddItemToArray(itemId, toDoItem) {
    // add de   tem to array as an object with an ID so we can find  nd
it              ter
toDoListArray       .push({ itemId, toDoItem});
console.        log(toDoListArray)
  }
function
// get
var li =        emoveItemFromDOM(id) {
                he list item by data ID
                 document.querySelector('[data-id="' + id + '"]');
    // remove list item
    ul.removeChild(li);
  }

  function removeItemFromArray(id) {
    // create a new toDoListArray with all li's that don't match the
ID
    toDoListArray = toDoListArray.filter(item => item.itemId !==
id);
    console.log(toDoListArray);
  }

})();
```

Final Output:





## Conclusion

To conclude, creating a to-do list in HTML is a straightforward process that involves utilizing HTML for the structure, CSS for styling, and JavaScript for functionality. Various tutorials and guides demonstrate how to build a basic to-do list app using these technology. Additionally, responsive designs can be implemented to ensure the app works seamlessly across devices.To-do lists are powerful tools for enhancing productivity, time management, and collaboration. In conclusion, creating a to-do list in HTML provides a customizable and efficient way to organize tasks and manage workflows for individuals.