To do List project technical documentation

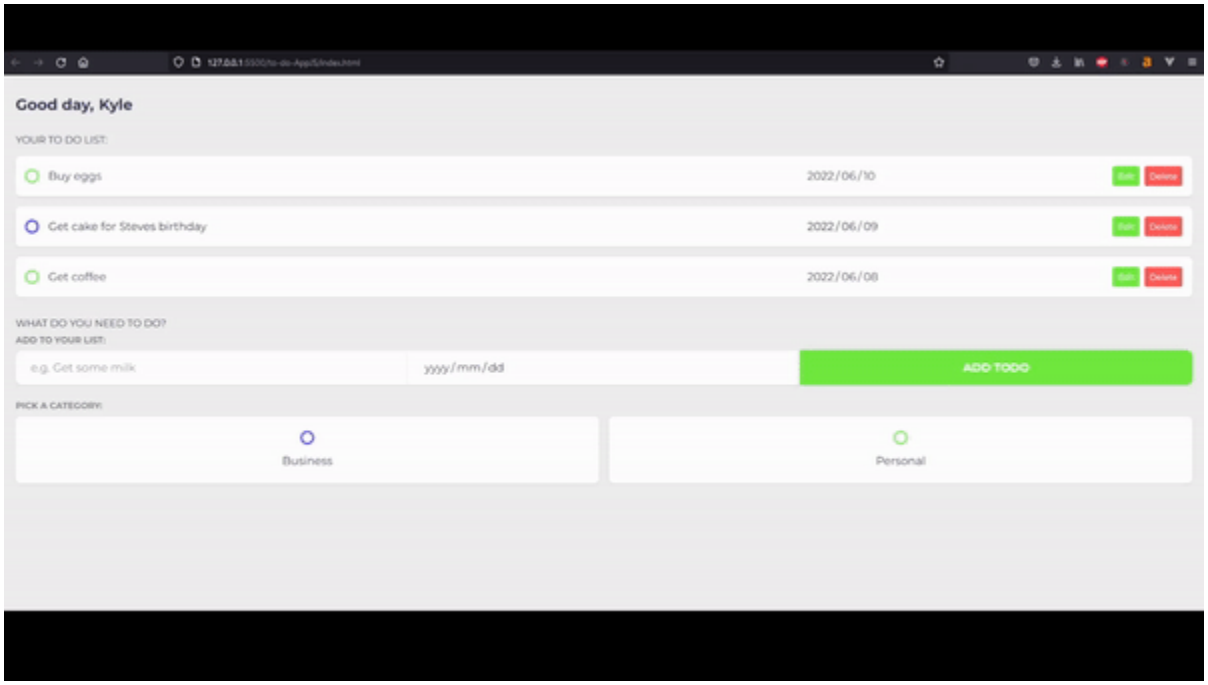
Overview

The project is a simple yet effective todo list CRUD application utilizing local storage along with JavaScript created elements. The user is asked ‘What do you need to do’ prompting the user to enter or ‘add’ to their list anything that they require to do as a reminder along with a date if they so choose.

This is a basic example of the app running during runtime

The user may then strike out(line through) any element they choose in the todo list to show that the task is complete or edit and delete which task is no longer required.

Basic CRUD application



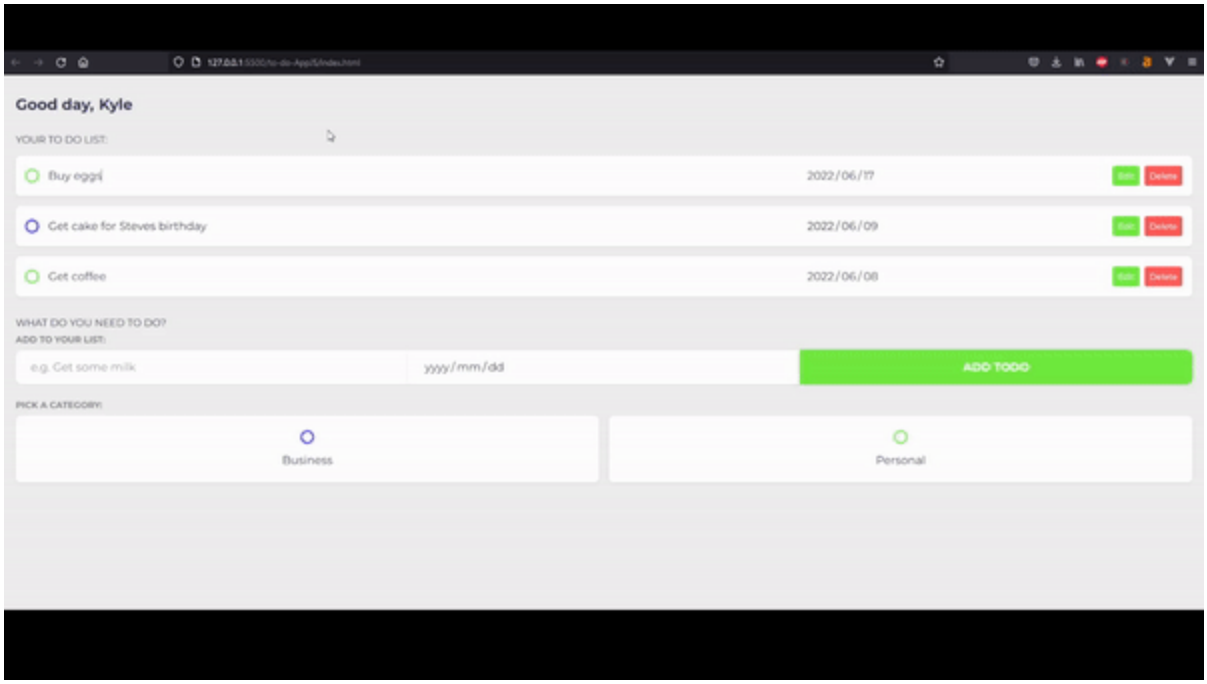
Strike out example usage

Striking out a todo is simply done by selecting the applicable radio button to show that the task is completed.

```
244  .todo-item.done .todo-content input {
245      text-decoration: line-through;
246      color: var(--grey);
```

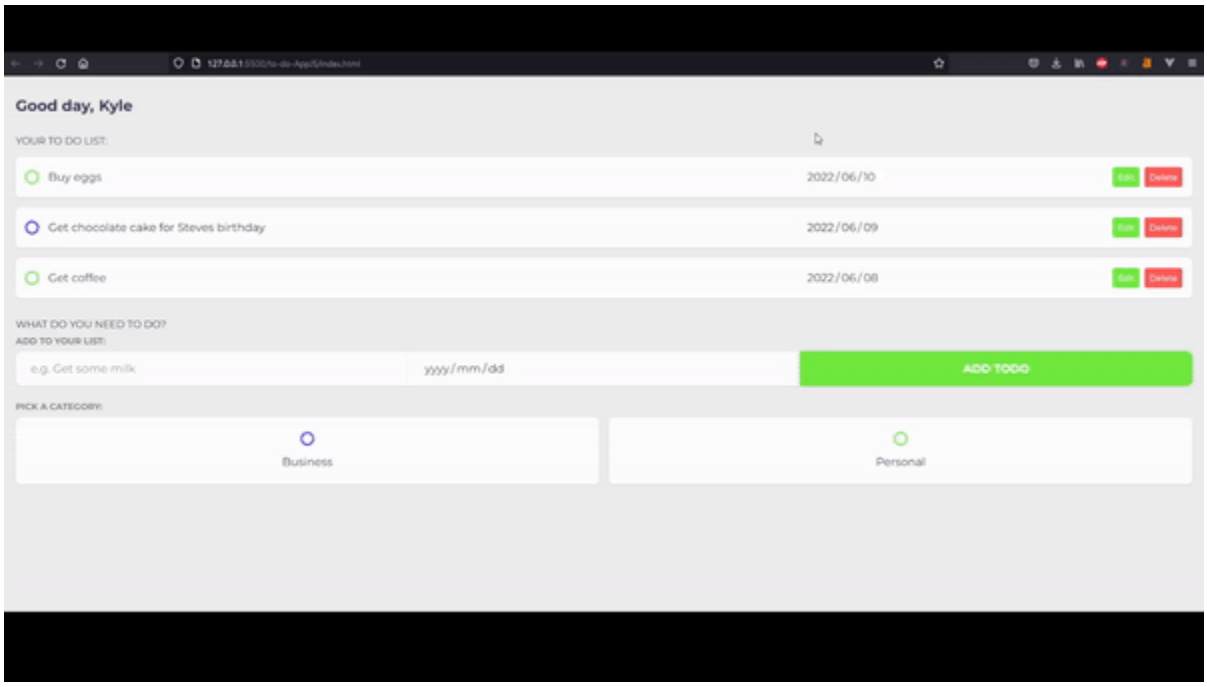
Strike out CSS styling

The CSS styling is only applied to the created element after an event listener on change is added and an if statement is ran to check if the target has been selected for each CRUD application.



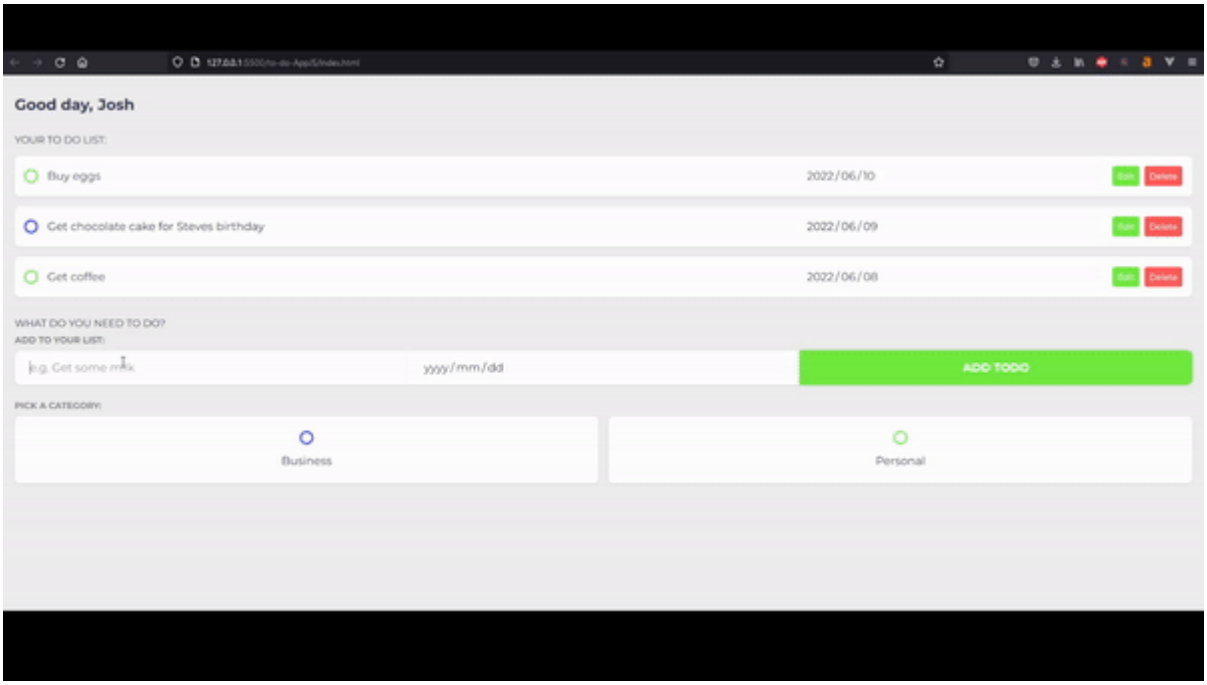
Edit example usage

Through the use of the button edit, a cursor will be focused on the selected input element and allow the user to edit the current selection.



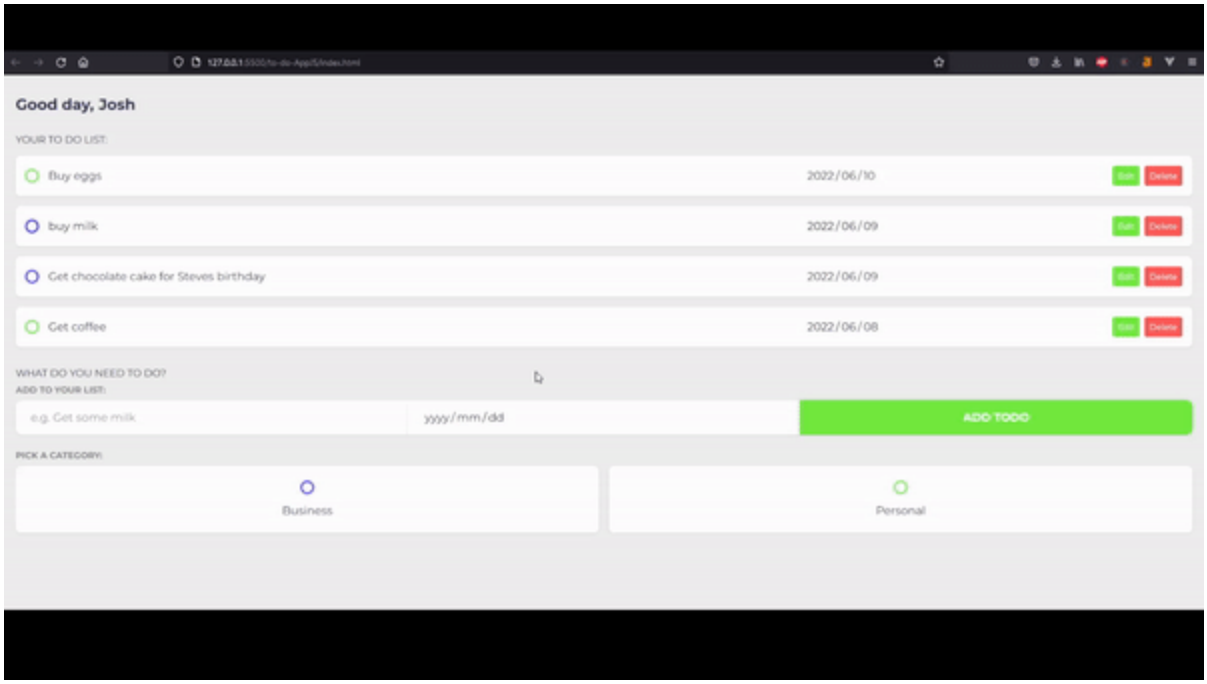
Name edit and refresh example

The following data is stored in local storage in an array of objects in the following order: {content, category, personal ,date}. Each object represents a single todo in the list. Edited information along with dates will be locally stored once edited.



Adding a todo to todo list application example

To do list objects can be added to the list using the ADD TODO button and will then be stored for later use.



Deleting an object entry example

Whenever a todo is deleted off the todo list it is also removed from the object in local storage.

Technical documentation

`window.addEventListener('load', () => {` `    todos = JSON.parse(localStorage.getItem('todos')) ||` `[];` `    const nameInput =` `tggasddocument.querySelector('#name');` `    const newTodoForm =` `document.querySelector('#new-todo-form');` `    const username = localStorage.getItem('username') ||` `'';`	When the application first loads into the browser the first event listener will retrieve the full list of items and name from local storage or present the user with empty strings or arrays if no local storage is present.
`    nameInput.value = username;` `nameInput.addEventListener('change', (e) => {` `localStorage.setItem('username', e.target.value);` `    })`	Sets the name to the name a user entered using an event listener for the user's name input.

`` class List { constructor(content, category, done, date) { this.content = content; this.category = category; this.done = done; this.date = date; } } ```	A class list declaration with 4 parameters used for each todo list entry (object).
`` newTodoForm.addEventListener('submit', e => { e.preventDefault(); let date = document.getElementById('datepicker').value; const todo = new List(e.target.elements.content.value, e.target.elements.category.value,false, date); todos.push(todo); localStorage.setItem('todos', JSON.stringify(todos)); ```	An event listener for the submit button push (or enter key). The date is first read in after a preventDefault function then the elements are read into the class and pushed into an array declared in the first event listener on load then saved into local storage.
`` e.target.reset(); DisplayTodos() }) DisplayTodos() ```	A form reset is then done to display to the user the updated list.
`` function AlphabetSort(x, y){ return x.content.localeCompare(y.content) } ```	A sort function is done to each todo submission after the DisplayTodos function is complete.
`` function DisplayTodos () { todos.sort(AlphabetSort); const todoList = document.querySelector('#todo-list'); todoList.innerHTML = ""; console.log(todos); ```	The main function of the application (DisplayTodos). A sort is done first then the query selector finds all the elements under todo-list and the inner html is set to an empty string.

`` todos.forEach(todo => { const todoItem = document.createElement('div'); todoItem.classList.add('todo-item'); const date = document.createElement('input'); const label = document.createElement('label'); const input = document.createElement('input'); const span = document.createElement('span'); const content = document.createElement('div'); const actions = document.createElement('div'); const edit = document.createElement('button'); const deleteButton = document.createElement('button'); ```	A for each loop is performed to create elements for each todo list addition in the array of objects and elements are created to store the information provided by the user. (note the for each loop is used to the end of the script from this point onwards)
`` date.type = 'date'; input.type = 'checkbox'; input.checked = todo.done; span.classList.add('bubble'); if (todo.category == 'personal') { span.classList.add('personal'); } else { span.classList.add('business'); } ```	The created javaScript elements are given html types and a check is done to see which category the user selected then the appropriate CSS is applied directly to the created element using its class.
`` content.classList.add('todo-content'); actions.classList.add('actions'); edit.classList.add('edit'); deleteButton.classList.add('delete'); ```	Additional button creation to perform CRUD functionality.
`` content.innerHTML = `<input type="text" style="width: 1200px;"         value="${todo.content}" readonly> <input type="date" value="${todo.date}" required/> `; edit.innerHTML = 'Edit'; ```	The todo item is displayed on the todo list for the user.

`deleteButton.innerHTML = 'Delete';`	
`label.appendChild(input);` `label.appendChild(span);` `actions.appendChild(edit);` `actions.appendChild(deleteButton);` `todoItem.appendChild(label);` `todoItem.appendChild(content);` `todoItem.appendChild(actions);` `todoList.appendChild(todoItem);`	All the items are then appended to the list.
`if (todo.done) {` `todoItem.classList.add('done');` `}` `input.addEventListener('change', (e) => {` `todo.done = e.target.checked;` `localStorage.setItem('todos', JSON.stringify(todos));` `if (todo.done) {` `todoItem.classList.add('done');` `} else {` `todoItem.classList.remove('done');` `}` `DisplayTodos()` `})`	The HTML class done is then added to each item that is completed (user has struck out) and a check is performed if the user utilized the event listener change which checks the radio box if the task is complete. The HTML class done has CSS in place to strike out whenever the event listener happens then the front end is displayed to the user with the changes applied
`edit.addEventListener('click', (e) => {` `const input = content.querySelector('input');` `input.removeAttribute('readonly');` `input.focus();` `input.addEventListener('blur', (e) => {` `input.setAttribute('readonly', true);` `todo.content = e.target.value;`	The edit event listener listens for a click by the user then removes the attribute read only from the HTML input element and a focus is set for the user to edit his todo and another event listener is performed to check when the

`` localStorage.setItem('todos', JSON.stringify(todos)); DisplayTodos() }) }) deleteButton.addEventListener('click', (e) => { todos = todos.filter(t => t != todo); localStorage.setItem('todos', JSON.stringify(todos)); DisplayTodos() }) }) } ```	user clicks away to set the new edited todo in local storage and is displayed. The delete button event listener listens for a click then uses array filter method to remove the element from the todo list and the form is displayed.