

# Documentación Completa del JavaScript - Lista de Tareas

## Estructura General

La aplicación utiliza **programación orientada a objetos** con una clase principal `TodoList` que maneja toda la funcionalidad de la lista de tareas, incluyendo persistencia local, filtros, y manipulación del DOM.

---

## Clase TodoList

### Constructor

```
javascript

class TodoList {
  constructor() {
    this.tasks = [];
    this.currentFilter = 'all';
    this.taskIdCounter = 1;

    this.initializeElements();
    this.loadFromStorage();
    this.addEventListeners();
    this.updateDisplay();
  }
}
```

### Explicación línea por línea:

- `this.tasks = [];` - Array que almacena todas las tareas de la aplicación
  - `this.currentFilter = 'all';` - String que mantiene el filtro actual ('all', 'pending', 'completed')
  - `this.taskIdCounter = 1;` - Contador para generar IDs únicos para cada tarea
  - `this.initializeElements();` - Inicializa todas las referencias a elementos del DOM
  - `this.loadFromStorage();` - Carga datos guardados desde localStorage
  - `this.addEventListeners();` - Configura todos los event listeners
  - `this.updateDisplay();` - Actualiza la visualización inicial
- 

### Método initializeElements()

```

javascript

initializeElements() {
    this.taskInput = document.getElementById('taskInput');
    this.addBtn = document.getElementById('addBtn');
    this.taskList = document.getElementById('taskList');
    this.emptyState = document.getElementById('emptyState');
    this.totalTasksEl = document.getElementById('totalTasks');
    this.pendingTasksEl = document.getElementById('pendingTasks');
    this.completedTasksEl = document.getElementById('completedTasks');
    this.filterBtns = document.querySelectorAll('.filter-btn');
    this.clearCompletedBtn = document.getElementById('clearCompleted');
}

}

```

### Explicación línea por línea:

- `this.taskInput = document.getElementById('taskInput');` - Obtiene referencia al campo de entrada de texto
  - `this.addBtn = document.getElementById('addBtn');` - Obtiene referencia al botón de agregar tarea
  - `this.taskList = document.getElementById('taskList');` - Obtiene referencia al contenedor de la lista de tareas
  - `this.emptyState = document.getElementById('emptyState');` - Obtiene referencia al estado vacío
  - `this.totalTasksEl = document.getElementById('totalTasks');` - Obtiene referencia al contador de tareas totales
  - `this.pendingTasksEl = document.getElementById('pendingTasks');` - Obtiene referencia al contador de tareas pendientes
  - `this.completedTasksEl = document.getElementById('completedTasks');` - Obtiene referencia al contador de tareas completadas
  - `this.filterBtns = document.querySelectorAll('.filter-btn');` - Obtiene todos los botones de filtro
  - `this.clearCompletedBtn = document.getElementById('clearCompleted');` - Obtiene referencia al botón de limpiar completadas
- 

### Método `loadFromStorage()`

```

javascript

loadFromStorage() {
    const savedTasks = localStorage.getItem('todoTasks');
    const savedCounter = localStorage.getItem('taskIdCounter');

    if (savedTasks) {
        this.tasks = JSON.parse(savedTasks);
    }

    if (savedCounter) {
        this.taskIdCounter = parseInt(savedCounter);
    }
}

```

### Explicación línea por línea:

- `const savedTasks = localStorage.getItem('todoTasks');` - Obtiene las tareas guardadas del localStorage
  - `const savedCounter = localStorage.getItem('taskIdCounter');` - Obtiene el contador guardado del localStorage
  - `if (savedTasks) {}` - Verifica si existen tareas guardadas
  - `this.tasks = JSON.parse(savedTasks);` - Convierte el string JSON a array de objetos y lo asigna
  - `if (savedCounter) {}` - Verifica si existe un contador guardado
  - `this.taskIdCounter = parseInt(savedCounter);` - Convierte el string a número y lo asigna
- 

### Método saveToStorage()

```

javascript

saveToStorage() {
    localStorage.setItem('todoTasks', JSON.stringify(this.tasks));
    localStorage.setItem('taskIdCounter', this.taskIdCounter.toString());
}

```

### Explicación línea por línea:

- `localStorage.setItem('todoTasks', JSON.stringify(this.tasks));` - Convierte el array de tareas a JSON y lo guarda en localStorage

- `localStorage.setItem('taskIdCounter', this.taskIdCounter.toString());` - Convierte el contador a string y lo guarda en localStorage
- 

## Método addEventListeners()

javascript

```
addEventListeners() {
    this.addButton.addEventListener('click', () => this.addTask());
    ...
    this.taskInput.addEventListener('keypress', (e) => {
        if (e.key === 'Enter') {
            this.addTask();
        }
    });
    ...
    this.filterBtns.forEach(btn => {
        btn.addEventListener('click', (e) => {
            this.setFilter(e.target.dataset.filter);
        });
    });
    ...
    this.clearCompletedBtn.addEventListener('click', () => {
        this.clearCompleted();
    });
}
```

### Explicación línea por línea:

- `this.addButton.addEventListener('click', () => this.addTask());` - Añade evento de click al botón agregar
- `this.taskInput.addEventListener('keypress', (e) => {})` - Añade evento de tecla presionada al input
- `if (e.key === 'Enter') {}` - Verifica si la tecla presionada es Enter
- `this.addTask();` - Llama al método addTask si se presiona Enter
- `this.filterBtns.forEach(btn => {})` - Itera sobre todos los botones de filtro
- `btn.addEventListener('click', (e) => {})` - Añade evento de click a cada botón de filtro
- `this.setFilter(e.target.dataset.filter);` - Llama a setFilter con el valor del atributo data-filter

- `this.clearCompletedBtn.addEventListener('click', () => {})` - Añade evento de click al botón limpiar completadas
  - `this.clearCompleted();` - Llama al método clearCompleted
- 

## Método addTask()

```
javascript

addTask() {
    const taskText = this.taskInput.value.trim();

    if (taskText === '') {
        this.showInputError();
        return;
    }

    const newTask = {
        id: this.taskIdCounter++,
        text: taskText,
        completed: false,
        createdAt: new Date().toISOString()
    };

    this.tasks.unshift(newTask);
    this.taskInput.value = '';
    this.saveToStorage();
    this.updateDisplay();
    this.showSuccessAnimation();
}
```

### Explicación línea por línea:

- `const taskText = this.taskInput.value.trim();` - Obtiene el texto del input y elimina espacios en blanco
- `if (taskText === '') {}` - Verifica si el texto está vacío
- `this.showInputError();` - Muestra error si el input está vacío
- `return;` - Sale de la función si hay error
- `const newTask = {}` - Crea un nuevo objeto tarea con:
- `id: this.taskIdCounter++,` - ID único (incrementa el contador)

- `text: taskText`, - Texto de la tarea
  - `completed: false`, - Estado inicial no completada
  - `createdAt: new Date().toISOString()` - Fecha de creación en formato ISO
  - `this.tasks.unshift(newTask);` - Añade la nueva tarea al inicio del array
  - `this.taskInput.value = '';` - Limpia el campo de entrada
  - `this.saveToStorage();` - Guarda en localStorage
  - `this.updateDisplay();` - Actualiza la visualización
  - `this.showSuccessAnimation();` - Muestra animación de éxito
- 

## Método showInputError()

javascript

```
showInputError() {
    this.taskInput.style.borderColor = '#e53e3e';
    this.taskInput.placeholder = 'Por favor, escribe una tarea válida';

    ...
    setTimeout(() => {
        this.taskInput.style.borderColor = '#e2e8f0';
        this.taskInput.placeholder = 'Escribe una nueva tarea...';
    }, 2000);
}
```

## Explicación línea por línea:

- `this.taskInput.style.borderColor = '#e53e3e';` - Cambia el borde a color rojo para indicar error
  - `this.taskInput.placeholder = 'Por favor, escribe una tarea válida';` - Cambia el placeholder a mensaje de error
  - `setTimeout(() => {}` - Inicia un temporizador de 2 segundos
  - `this.taskInput.style.borderColor = '#e2e8f0';` - Restaura el color original del borde
  - `this.taskInput.placeholder = 'Escribe una nueva tarea...';` - Restaura el placeholder original
  - `}, 2000);` - Ejecuta después de 2000 milisegundos
- 

## Método showSuccessAnimation()

```
javascript

showSuccessAnimation() {
    this.addButton.style.transform = 'scale(0.95)';
    setTimeout(() => {
        this.addButton.style.transform = 'scale(1)';
    }, 150);
}
```

### Explicación línea por línea:

- `this.addButton.style.transform = 'scale(0.95)';` - Reduce el tamaño del botón al 95%
  - `setTimeout(() => {` - Inicia temporizador de 150ms
  - `this.addButton.style.transform = 'scale(1)';` - Restaura el tamaño original
  - `}, 150);` - Ejecuta después de 150 milisegundos
- 

### Método toggleTask()

```
javascript

toggleTask(taskId) {
    const task = this.tasks.find(t => t.id === taskId);
    if (task) {
        task.completed = !task.completed;
        this.saveToStorage();
        this.updateDisplay();
    }
}
```

### Explicación línea por línea:

- `const task = this.tasks.find(t => t.id === taskId);` - Busca la tarea por ID
  - `if (task) {` - Verifica si la tarea existe
  - `task.completed = !task.completed;` - Invierte el estado completado/no completado
  - `this.saveToStorage();` - Guarda los cambios en localStorage
  - `this.updateDisplay();` - Actualiza la visualización
- 

### Método deleteTask()

```

javascript

deleteTask(taskId) {
    if (confirm('¿Estás seguro de que quieres eliminar esta tarea?')) {
        this.tasks = this.tasks.filter(t => t.id !== taskId);
        this.saveToStorage();
        this.updateDisplay();
    }
}

```

### Explicación Línea por línea:

- `if (confirm('¿Estás seguro de que quieres eliminar esta tarea?')) {` - Muestra diálogo de confirmación
  - `this.tasks = this.tasks.filter(t => t.id !== taskId);` - Filtra el array excluyendo la tarea con el ID especificado
  - `this.saveToStorage();` - Guarda los cambios en localStorage
  - `this.updateDisplay();` - Actualiza la visualización
- 

### Método setFilter()

```

javascript

setFilter(filter) {
    this.currentFilter = filter;

    this.filterBtns.forEach(btn => {
        btn.classList.remove('active');
        if (btn.dataset.filter === filter) {
            btn.classList.add('active');
        }
    });

    this.updateDisplay();
}

```

### Explicación Línea por línea:

- `this.currentFilter = filter;` - Establece el filtro actual
- `this.filterBtns.forEach(btn => {` - Itera sobre todos los botones de filtro

- `btn.classList.remove('active');` - Remueve la clase 'active' de todos los botones
  - `if (btn.dataset.filter === filter) {` - Verifica si el botón corresponde al filtro seleccionado
  - `btn.classList.add('active');` - Añade la clase 'active' al botón correspondiente
  - `this.updateDisplay();` - Actualiza la visualización con el nuevo filtro
- 

## Método getFilteredTasks()

javascript

```
getFilteredTasks() {
  switch (this.currentFilter) {
    ...
    case 'pending':
      return this.tasks.filter(task => !task.completed);
```