Enhanced Angular Signals Example: Task Management Application with Linked Signals

Introduction

In this enhanced version of the Angular Signals example, we introduce the linkedsignal feature, which allows us to synchronize two signals. This is useful in cases where we need to keep different parts of the state in sync without manually updating them. For example, we'll link a signal that controls the task filtering state (e.g., "All", "Completed", "Pending") to the task list's display logic. The linked signal will automatically update based on the task filter, and vice versa.

Code

task.service.ts

```
import { Injectable, signal, computed, effect, linked } from '@angular/core';

export interface Task {
  id: number;
  title: string;
  completed: boolean;
}

@Injectable({
  providedIn: 'root'
})
export class TaskService {
  // State: Holds the list of tasks
  private tasks = signal<Task[]>([]);

  // Linked state: Filter for tasks (All, Completed, Pending)
  private filter = signal<'all' | 'completed' | 'pending'>('all');
  linkedFilter = linked(this.filter, (value) => value);

  // Derived state: Count of pending tasks
  pendingTasks = computed(() =>
    this.tasks().filter(task => !task.completed).length
  );

  // Derived state: Filtered tasks based on filter selection
  filteredTasks = computed(() => {
    const filterValue = this.linkedFilter();
    switch (filterValue) {
      case 'completed':
        return this.tasks().filter(task => task.completed);
      case 'pending':
        return this.tasks().filter(task => !task.completed);
      default:
        return this.tasks();
    }
  });

  // Effect: Persist tasks to local storage whenever the list changes
  private persistTasks = effect(() => {
    localStorage.setItem('tasks', JSON.stringify(this.tasks()));
  });

  constructor() {
    // Load tasks from local storage on initialization
    const storedTasks = localStorage.getItem('tasks');
    if (storedTasks) {
      this.tasks.set(JSON.parse(storedTasks));
    }
  }

  // Methods to update the state
  addTask(title: string): void {
    const newTask: Task = {
      id: Date.now(),
      title,
      completed: false
    };
    this.tasks.mutate(tasks => tasks.push(newTask));
  }

  toggleTaskCompletion(taskId: number): void {
    this.tasks.mutate(tasks => {
      const task = tasks.find(t => t.id === taskId);
      if (task) {
        task.completed = !task.completed;
      }
    });
  }

  removeTask(taskId: number): void {
    this.tasks.mutate(tasks => tasks.filter(t => t.id !== taskId));
  }

  setFilter(filter: 'all' | 'completed' | 'pending'): void {
    this.filter.set(filter);
  }

  getTasks(): Task[] {
    return this.tasks();
  }

  getFilter(): 'all' | 'completed' | 'pending' {
    return this.filter();
  }
}
```

task.component.ts

```
import { Component } from '@angular/core';
import { TaskService, Task } from './task.service';

@Component({
  selector: 'app-task',
  templateUrl: './task.component.html',
  styleUrls: ['./task.component.css']
})
export class TaskComponent {
  newTaskTitle = '';

  constructor(public taskService: TaskService) {}
  addTask(): void {
    if (this.newTaskTitle.trim()) {
      this.taskService.addTask(this.newTaskTitle.trim());
      this.newTaskTitle = '';
    }
  }

  toggleCompletion(task: Task): void {
    this.taskService.toggleTaskCompletion(task.id);
  }

  removeTask(task: Task): void {
    this.taskService.removeTask(task.id);
  }

  setFilter(filter: 'all' | 'completed' | 'pending'): void {
    this.taskService.setFilter(filter);
  }
}
```

task.component.html

```
<div class="task-app">
  <h1>Task Manager</h1>
  <form (submit)="addTask(); $event.preventDefault()">
    <input
      type="text"
      [(ngModel)]="newTaskTitle"
      placeholder="Enter new task"
    />

    <button type="submit">Add Task</button>
  </form>

  <div class="task-summary">
    <p>Total Tasks: {{ taskService.getTasks().length }}</p>
    <p>Pending Tasks: {{ taskService.pendingTasks() }}</p>
  </div>

  <div class="task-filter">
    <button (click)="setFilter('all')">All</button>
    <button (click)="setFilter('completed')">Completed</button>
    <button (click)="setFilter('pending')">Pending</button>
  </div>

  <ul class="task-list">
    <li *ngFor="let task of taskService.filteredTasks()">
      <input
        type="checkbox"
        [checked]="task.completed"
        (change)="toggleCompletion(task)"
      />
      <span [class.completed]="task.completed">{{ task.title }}</span>
      <button (click)="removeTask(task)">Remove</button>
    </li>
  </ul>
</div>
```

task.component.css

```
.task-app {
  max-width: 500px;
  margin: auto;
  font-family: Arial, sans-serif;
}

form {
  display: flex;
  gap: 8px;
  margin-bottom: 16px;
}

.task-filter {
  margin: 16px 0;
}

.task-list {
  list-style-type: none;
  padding: 0;
}

.task-list li {
  display: flex;
  align-items: center;
  gap: 8px;
  margin-bottom: 8px;
}

.completed {
  text-decoration: line-through;
  color: gray;
}
```

Explanation

linked-signal

The `filter` signal is linked to `linkedFilter`, which keeps it in sync with the rest of the application. By using `linked`, we ensure that any updates to the filter signal automatically update the linked signal.

computed

This `filteredTasks` computed property filters tasks based on the `linkedFilter` signal, which reacts to the filter's value ('all', 'completed', 'pending'). It ensures the UI reflects the correct set of tasks whenever the filter changes.

effect

The effect `persistTasks` saves the task list to local storage whenever the tasks signal is updated.

Workflow

- **Add Task**: Adds a new task to the list. The task is added to the `tasks` signal, and the UI is updated.
- **Toggle Completion**: Flips a task's completion status. The `tasks` signal and `filteredTasks` computed property are updated.

- **Remove Task**: Removes a task from the list. The task is removed from the `tasks` signal, and the UI is updated.
- **Set Filter**: Changes the task filter (All, Completed, Pending). The `filter` signal is updated, and the linked signal keeps the filter state in sync across the application