Angular Signals Example: Task Management Application

Introduction

This example demonstrates how to use Angular Signals in a realistic, task management application. It covers the usage of `signal`, `computed`, and `effect` functionalities within Angular. The task manager allows users to manage tasks, with automatic persistence to local storage and reactive updates.

Code

task.service.ts

```
import { Injectable, signal, computed, effect } 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[]>([]);

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

  // 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));
  }

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

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);
  }
}
```

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>

  <ul class="task-list">
    <li *ngFor="let task of taskService.getTasks()">
      <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-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

signal

The `tasks` signal in `TaskService` holds the array of tasks. By calling `tasks.mutate`, we can directly modify the state in an immutable way, ensuring updates propagate.

computed

The `pendingTasks` computed property derives the count of pending tasks based on the current `tasks` state. This ensures reactive updates whenever the `tasks` signal changes.

effect

The `persistTasks` effect saves the tasks to local storage whenever `tasks` is updated. Effects are used for side effects, like logging, persisting data, or updating the DOM.

Workflow

- **Add Task**: The `addTask` method updates the `tasks` signal. The UI and local storage automatically update.
- **Toggle Completion**: The `toggleTaskCompletion` method flips a task's completed status, updating the signal and derived state.
- **Remove Task**: The `removeTask` method filters the tasks list, automatically triggering UI and local storage updates.

This example demonstrates how Angular Signals simplify state management and reactivity while maintaining a declarative programming style.