Enhanced Task Management Application with Angular Signals and Resource Handling

Introduction

In this example, we will build a more complex task management application using Angular **Signals**, with the addition of **resource handling features**. We'll cover the following concepts: - **signal**: For managing state (e.g., task list). - **computed**: For derived state (e.g., count of pending tasks). - **effect**: For reacting to state changes (e.g., persisting tasks to local storage). - **resource**: For managing external resources (e.g., fetching tasks from an API or database).

Code

task.service.ts

```
import { Injectable, signal, computed, effect, resource } 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()));
  });

  // Resource: Fetch tasks from an external API (simulated)
  private taskResource = resource(() => this.fetchTasks());

  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();
  }

  // Fetch tasks from a simulated external API
  private async fetchTasks(): Promise<Task[]> {
    // Simulated delay
    await new Promise((resolve) => setTimeout(resolve, 1000));
    return [
      { id: 1, title: 'Task 1', completed: false },
      { id: 2, title: 'Task 2', completed: true },
      { id: 3, title: 'Task 3', completed: false },
    ];
  }
}
```

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 = '';

  // Loading tasks from the resource
  tasks$ = this.taskService.taskResource.read();

  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 tasks$ | async">
      <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>
```

cart.component.ts

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-cart',
  template: `
    <h2>Shopping Cart</h2>
    <div *ngFor="let item of cartItems">
      {{ item.name }} - {{ item.price | currency }}
    </div>
  `,
  styleUrls: ['./cart.component.css']
})
export class CartComponent {
  @Input() cartItems: any[] = [];
}
```

task.component.css

```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 list of tasks.
- The state is mutated using `tasks.mutate()`, ensuring the updates propagate reactively.

computed

- The `pendingTasks` computed property derives the count of pending tasks based on the current state of `tasks`.

effect

- The `persistTasks` effect persists the tasks to local storage whenever the state changes.

resource

- The `taskResource` utilizes the resource function to fetch tasks from an external resource. In this example, we simulate fetching tasks from an API with a delay using `fetchTasks()`.
- The component reads the task resource using `tasks$ = this.taskService.taskResource.read()` and displays the data synchronously.