Simple Angular Todo App — NgRx + RxJS

This is a minimal, well-documented example of a Todo app built with Angular and NgRx (store) to help you practice state management and RxJS patterns.

What you'll get in this document

- Quick setup steps
- File structure
- All the source files you need (TypeScript, HTML, CSS)
- How to run the app

Quick setup

1. Create a new Angular project (requires Angular CLI):

```
ng new todo-ngrx --routing=false --style=css
cd todo-ngrx
```

1. Install NgRx store:

```
npm install @ngrx/store
```

1. Replace/add the files below in your `src/app` folder. Keep backups if you already have code.

2. Run the app:

```
ng serve
# open http://localhost:4200
```

File structure (src/app)

```
app/
├─ models/todo.model.ts
├─ store/todo.actions.ts
├─ store/todo.reducer.ts
├─ store/todo.selectors.ts
├─ app.module.ts
├─ app.component.ts
├─ app.component.html
└─ app.component.css
```

models/todo.model.ts

```
export interface Todo {
  id: string;
  title: string;
  completed: boolean;
}
```

store/todo.actions.ts

```
import { createAction, props } from '@ngrx/store';
import { Todo } from '../models/todo.model';

export const addTodo = createAction('[Todo] Add', props<{ title: string }
>());
export const toggleTodo = createAction('[Todo] Toggle', props<{ id: string }
>());
export const removeTodo = createAction('[Todo] Remove', props<{ id: string }
>());
export const clearCompleted = createAction('[Todo] Clear Completed');
```

store/todo.reducer.ts

```
import { createReducer, on } from '@ngrx/store';
import { Todo } from '../models/todo.model';
import * as TodoActions from './todo.actions';

export interface TodoState {
  todos: Todo[];
}

export const initialState: TodoState = {
  todos: [],
};

function uid() {
  // simple id generator
  return Math.random().toString(36).substr(2, 9);
}

export const todoReducer = createReducer(
  initialState,
  on(TodoActions.addTodo, (state, { title }) => ({
```

```
    ...state,
    todos: [...state.todos, { id: uid(), title, completed: false }],
  })),
  on(TodoActions.toggleTodo, (state, { id }) => ({
    ...state,
    todos: state.todos.map(t => (t.id === id ? { ...t, completed: !
t.completed } : t)),
  })),
  on(TodoActions.removeTodo, (state, { id }) => ({
    ...state,
    todos: state.todos.filter(t => t.id !== id),
  })),
  on(TodoActions.clearCompleted, state => ({
    ...state,
    todos: state.todos.filter(t => !t.completed),
  }))
);
```

---

store/todo.selectors.ts

```
import { createFeatureSelector, createSelector } from '@ngrx/store';
import { TodoState } from './todo.reducer';

export const selectTodoState = createFeatureSelector<TodoState>('todo');

export const selectTodos = createSelector(
  selectTodoState,
  state => state.todos
);

export const selectCompletedCount = createSelector(
  selectTodos,
  todos => todos.filter(t => t.completed).length
);

export const selectRemainingCount = createSelector(
  selectTodos,
  todos => todos.filter(t => !t.completed).length
);
```

---

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { StoreModule } from '@ngrx/store';
```

```
import { AppComponent } from './app.component';
import { todoReducer } from './store/todo.reducer';
import { ReactiveFormsModule } from '@angular/forms';

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    ReactiveFormsModule,
    StoreModule.forRoot({ todo: todoReducer }),
  ],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

---

app.component.ts

```
import { Component } from '@angular/core';
import { FormControl } from '@angular/forms';
import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';
import { Todo } from './models/todo.model';
import * as TodoActions from './store/todo.actions';
import * as TodoSelectors from './store/todo.selectors';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
})
export class AppComponent {
  todos$: Observable<Todo[]>;
  remainingCount$: Observable<number>;

  titleInput = new FormControl('');

  constructor(private store: Store) {
    this.todos$ = this.store.select(TodoSelectors.selectTodos);
    this.remainingCount$ =
this.store.select(TodoSelectors.selectRemainingCount);
  }

  add() {
    const title = (this.titleInput.value || '').trim();
    if (!title) return;
    this.store.dispatch(TodoActions.addTodo({ title }));
```

```
    this.titleInput.reset();
  }

  toggle(id: string) {
    this.store.dispatch(TodoActions.toggleTodo({ id }));
  }

  remove(id: string) {
    this.store.dispatch(TodoActions.removeTodo({ id }));
  }

  clearCompleted() {
    this.store.dispatch(TodoActions.clearCompleted());
  }
}
```

---

app.component.html

```
<div class="container">
  <h1>NgRx Todo (simple)</h1>

  <div class="new-todo">
    <input [formControl]="titleInput" (keydown.enter)="add()"
placeholder="What needs to be done?" />
    <button (click)="add()">Add</button>
  </div>

  <div class="stats">
    <span *ngIf="remainingCount$ | async as rem">{{ rem }} remaining</span>
    <button (click)="clearCompleted()">Clear completed</button>
  </div>

  <ul class="todo-list">
    <li *ngFor="let t of (todos$ | async)">
      <label>
        <input type="checkbox" [checked]="t.completed"
(change)="toggle(t.id)" />
        <span [class.completed]="t.completed">{{ t.title }}</span>
      </label>
      <button class="remove" (click)="remove(t.id)">X</button>
    </li>
  </ul>
</div>
```

`app.component.css`

```css
.container {
  max-width: 600px;
  margin: 40px auto;
  font-family: Arial, sans-serif;
}

.new-todo {
  display: flex;
  gap: 8px;
  margin-bottom: 16px;
}

.new-todo input {
  flex: 1;
  padding: 8px;
  font-size: 16px;
}

.todo-list {
  list-style: none;
  padding: 0;
}

.todo-list li {
  display: flex;
  align-items: center;
  justify-content: space-between;
  padding: 8px 0;
  border-bottom: 1px solid #eee;
}

.completed {
  text-decoration: line-through;
  color: #888;
}

.remove {
  background: transparent;
  border: none;
  cursor: pointer;
  font-size: 18px;
}
```

Notes & next steps (ideas for practice)

- Add persistence using `localStorage` (use an NgRx meta-reducer or subscribe to the store in a service).
- Add `@ngrx/effects` to simulate async saves (practice RxJS operators like `switchMap`, `of`, `delay`).
- Add filtering (all/active/completed) using router or local component state.
- Add unit tests for reducer and selectors.

If you'd like, I can: - Produce the exact file contents as downloadable files - Add persistence with localStorage - Extend the example to include NgRx Effects and a mock API

Tell me which of those you'd like next and I'll extend the example.