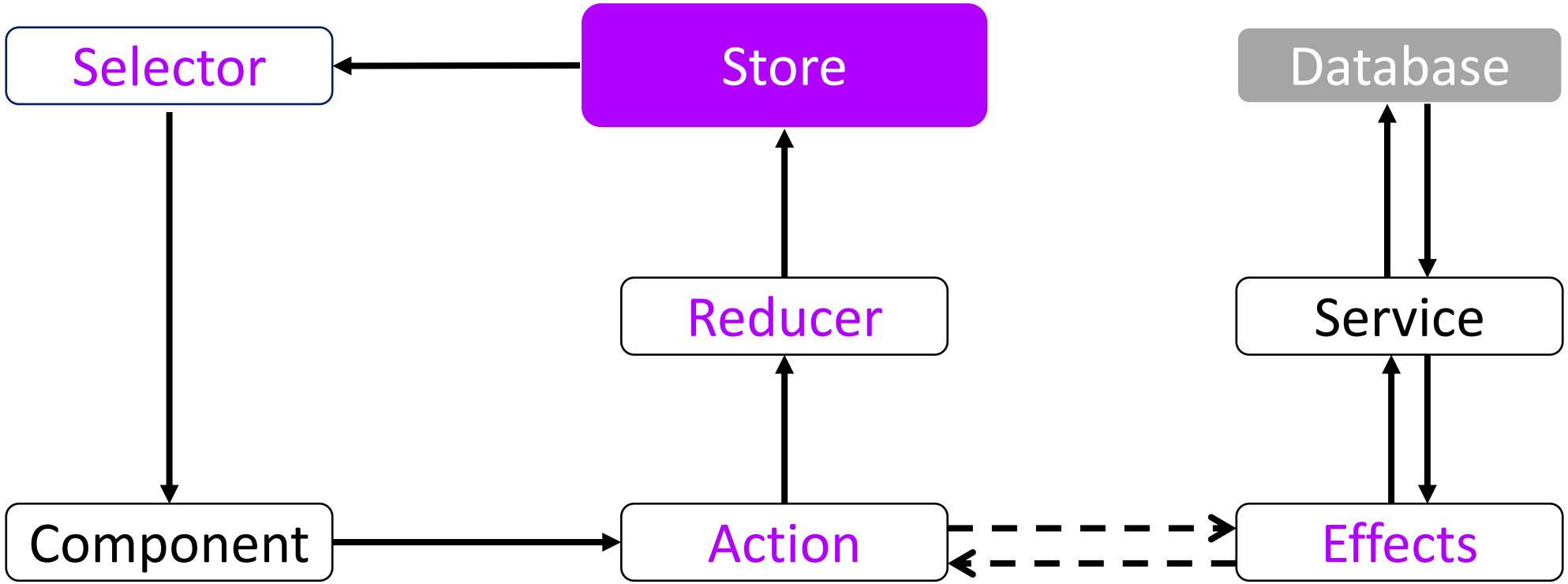

ng add @ngrx/store
ng add @ngrx/store-devtools
ng add @ngrx/effects
ng add @ngrx/schematics (możliwość generowania plików za pomocą **ng generate**)

NgRx

- **Store** – zarządzanie stanem aplikacji Angular oparte na RxJs (inspirowane na wzorcu Redux). **Store** jest jeden dla całej aplikacji, można go określić jako duży obiekt, który przechowuje cały stan aplikacji.

```
export interface AppState {
  3 usages
  todos: fromTodos.TodoState
}
```

```
export class TodoListComponent{
  no usages
  constructor(private store: Store<fromApp.AppState>) {}
```

```
this.store.select( key: 'todos').subscribe( observerOrNext: {
  next: todosArr => {
    this.todos = [...todosArr.todos];
  }
})
```

Selectors

Selektory to pod spodem po prostu funkcje używane do pobierania części danych ze stanu **Store**. Nie ma konieczności ich używania (możemy też pobierać dane bez selektorów), ale mają kilka zalet. Upraszczają dostęp do stanu aplikacji, a także pozwalają na skupieniu się na konkretnych fragmentach stanu (zamiast pobierać całość i przekształcać dane). Dzięki **Selectors** mamy także możliwość stosowania memoizacji – ostatni zwrócony wynik jest zapamiętywany, dzięki czemu jeśli np. 3 komponenty będą chciały pobrać dane z tego samego selektora to będzie to dużo wydajniejsze (**MemoizedSelector**).

```
export const selectTodos = (state: AppState) => state.todos;

2 usages
export const selectActiveTodos = createSelector(
  selectTodos,
  projector: (state : TodoState ) => state.todos.filter(todo => !todo.isComplete)
)
```

```
this.store.select(selectActiveTodos).subscribe( observerOrNext: {
  next: val => {
    console.log(val)
  }
})
```

Actions

Actions – jedne z najważniejszych elementów NgRx. Akcje odnoszą się do unikalnych zdarzeń (event'ów), które się dzieją podczas działania całej aplikacji (np. dodawanie coś do tablicy). **Action** to obiekt, który zawiera informację o akcji, którą ma wykonać oraz może dodatkowo zawierać payload (np. zadanie do dodania do tablicy).

```
import {createAction, props} from "@ngrx/store";
import {Todo} from "../../shared/interfaces/todo.interface";

4 usages
export const addTodo = createAction(
  type: '[Todos] Add Todo',
  props<{todo: Todo }>()
);
```

```
this.store.dispatch(TodosActions.addTodo( props: {todo: {id: 1, name: todo, isComplete: false}}))
```

Reducers

Reducers to funkcje, które dają taki sam wynik, dla takich samych danych wejściowych. Działają w sposób synchroniczny. Każdy **reducer** odbiera akcję i potrafi określić jakiego typu jest dana akcja (np. pobranie tablicy zadań), do tego ma dostęp do aktualnego **state** (stanu). Dzięki temu decyduje, czy zwrócić nowy (zmodyfikowany) **state**, czy zwrócić oryginalny stan (niezmieniony). Reducer'y rejestruje się w **Store**.

```
const _todoReducer = createReducer(
  initialState, // początkowy stan - np. pusta tablica
  on(
    addTodo, // action - utworzone poprzez createAction()
    reducer: (state, action) => ({
      ...state,
      todos: state.todos.concat({ ...action.todo })
      // zapis w sposób immutable - najpierw kopiujemy (nie modyfikujemy
      // istniejącej tablicy), a później zmieniamy wartość!!!
    })
  )
);
2 usages
export function todosReducer(state: TodoState | undefined, action: Action) {
  return _todoReducer(state, action);
}
```

Effects

Effects izolują „efekty uboczne” od komponentów – zalicza się do tego między innymi pobieranie danych (zapytania HTTP), czy zadania które skutkują wieloma zdarzeniami. W standardowej aplikacji to komponent poprzez serwis komunikuje się z backend’em – w tym przypadku **effects** dostarczają sposób na interakcję z serwisami i izolowania tego od komponentów. **Effects** pozwalają na pracę w sposób asynchroniczny (ale także synchroniczny). Skutkują powstaniem nowej akcji (action).

Lazy loading, a NgRx

W głównym module aplikacji rejestrujemy to co należy do **Store** za pomocą dodania do tablicy **imports**:

```
StoreModule.forRoot( reducers: { todos:  todosReducer})
```

W przypadku, gdy posiadamy Lazy loaded modules i niektóre elementy są potrzebne tylko z poziomu tych modułów możemy wykorzystać w nich:

```
StoreModule.forFeature( slice: { todos:  todosReducer})
```

Ale należy z tym uważać! Jeśli dany lazy loaded module nie zostanie jeszcze załadowany – to ten element, który dodajemy do **Store** będzie **undefined**.

Wykorzystanie **forFeature** powoduje dodanie do globalnego **Store** nowych elementów, nie powoduje utworzenia nowego **Store**.