

# Redux State Management

Introduction to Redux. Using NgRX.



**SoftUni Team**  
Technical Trainers



**SoftUni**



**Software University**

<https://softuni.bg>

## 1. Introduction to NgRx

- NgRx Packages

## 2. NgRX Store

- Actions
- Reducers



sli.do

**#js-web**



# **State Management**

## **Introduction to NgRx**

# Introduction to NgRx

- NgRx is a framework for building reactive applications in Angular
- NgRx provides libraries for:
  - Managing **global state**
  - **Isolation** of side effects
  - Entity **collection management**
  - **Integration** with the Angular Router
  - **Developer tooling**

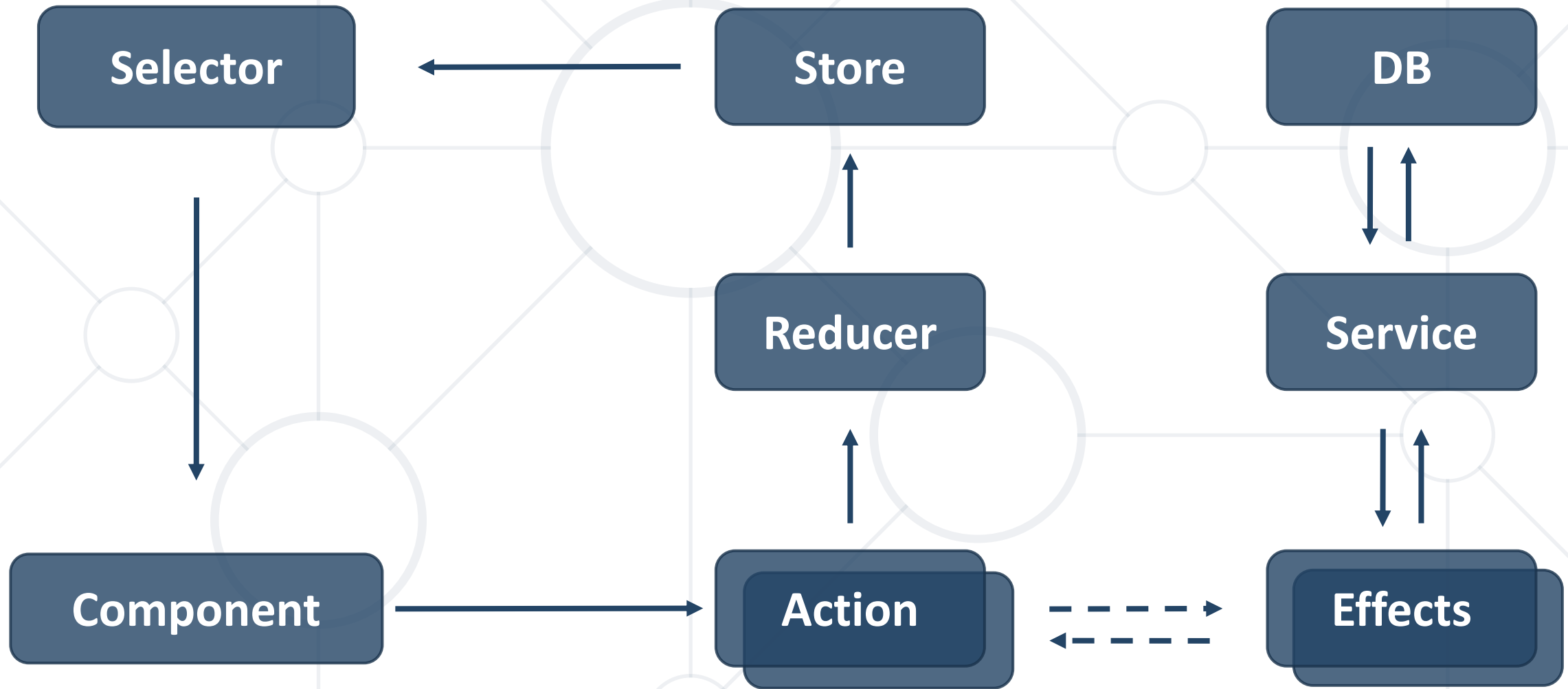


# NgRx Packages

- NgRx packages categories:
  - **State:**
    - **Store** - RxJS powered global state management
    - **Effects** - side effect model
    - **Router store** - Bindings to connect the Angular Router
    - **Entity** - Entity State adapter for managing record collections
    - **ComponentStore**
  - **Data:** Extension for simplified entity data management
  - **View:** Extension for fully reactive Angular applications
  - **Developer tooling:** Store Devtools, Schematics



# State Management Lifecycle






**NgRx**




# NgRX Store

- 
- RxJS powered global **state management tool** for Angular
  - Helps writing **performant, consistent** applications on top of Angular
  - **Installation**

```
npm install @ngrx/store --save
```

# Actions

- An action in NgRX/store
  - Is a **type** in the form of a **string**
  - Contains a **payload** of data
- Create an **actions.ts**



```
import { createAction } from '@ngrx/store';

// An action to increment
export const increment = createAction(
  '[Counter Component] Increment'
)
```

- An action to **decrement**

```
export const decrement = createAction(  
  '[Counter Component] Decrement'  
);
```

- An action to **reset**

```
export const reset = createAction(  
  '[Counter Component] Reset'  
);
```

# Reducers

- A **reducer**
  - is responsible for **handling transitions** from one state to another
  - is a **pure function**
  - handles each state transition **synchronously**
- A **reducer function**
  - handles transitions by determining which action to handle based on the **action's type**



# Define a Reducer Function

```
import { createReducer, on } from '@ngrx/store';  
import { increment, decrement, reset } from './counter.actions';  
  
export const initialState = 0;  
  
const _counterReducer = createReducer(  
  initialState,  
  on(increment, (state) => state + 1),  
  on(decrement, (state) => state - 1),  
  on(reset, (state) => 0)  
);  
  
export function counterReducer(state, action) {  
  return _counterReducer(state, action);  
}
```

# Add the StoreModule.forRoot

- Import **StoreModule** and the **reducer**

```
import { StoreModule } from '@ngrx/store';
import { counterReducer } from '../counter.reducer';

@NgModule({
  // Other code removed for brevity
  imports: [
    BrowserModule,
    StoreModule.forRoot({
      count: counterReducer
    })
  ],
  // Other code removed for brevity
})
```

# Create a Counter Component

- Import **Observable**, **Store** and the **actions**

```
// Other code removed for brevity
export class MyCounterComponent {
  count$: Observable<number>
  constructor(private store: Store<{ count: number }>) {
    // TODO: This stream will connect to the current store `count` state
    this.count$ = store.select('count');
  }
  increment() {
    this.store.dispatch(increment());
  }
  decrement() {
    // TODO: Dispatch a decrement action
  }
  reset() {
    // TODO: Dispatch a reset action
  }
}
```

# Create a Counter Component (2)

- Generate HTML

```
<button (click)="increment()">Increment</button>  
<div>Current Count: {{ count$ | async }}</div>  
<button (click)="decrement()">Decrement</button>  
<button (click)="reset()">Reset Counter</button>
```

- Add your component to the AppModule

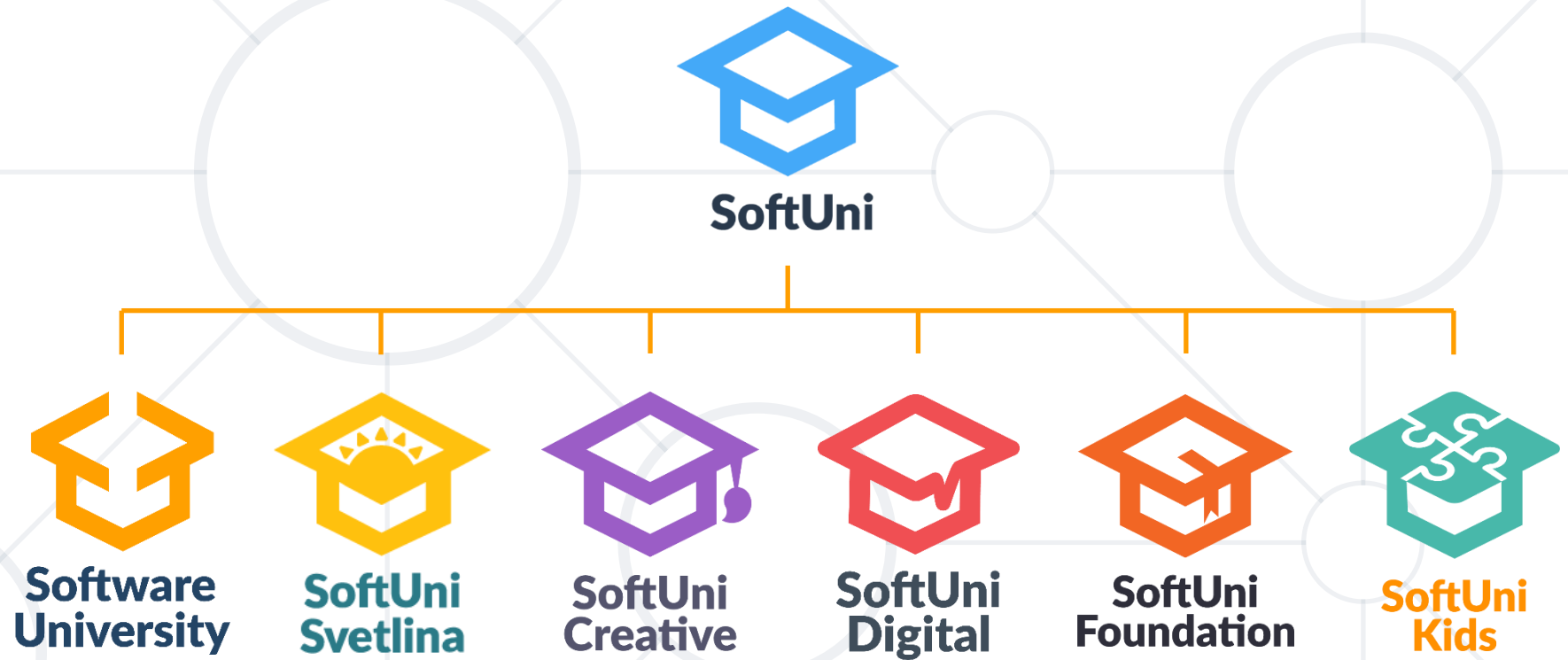
```
@NgModule({  
  declarations: [AppComponent, MyCounterComponent],  
  imports: [BrowserModule, StoreModule.forRoot({ count:  
    counterReducer })],
```



- **State Management Lifecycle**
- **NgRx** packages
- NgRX store is a **state management tool**
  - Store
  - Actions
  - Reducers



# Questions?



# SoftUni Diamond Partners

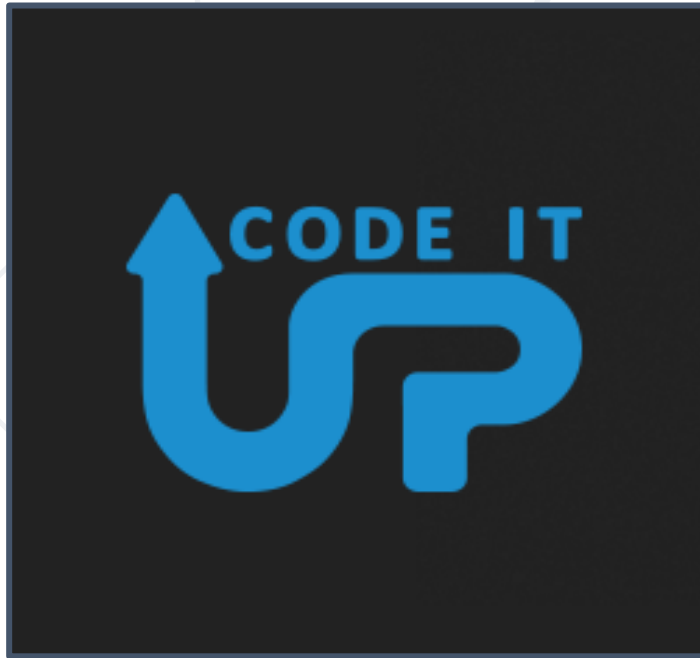


**SCHWARZ**



**SUPER  
HOSTING  
.BG**





**VIRTUAL RACING SCHOOL**



- Software University – High-Quality Education, Profession and Job for Software Developers

- [softuni.bg](http://softuni.bg), [softuni.org](http://softuni.org)

- Software University Foundation

- [softuni.foundation](http://softuni.foundation)

- Software University @ Facebook

- [facebook.com/SoftwareUniversity](https://facebook.com/SoftwareUniversity)

- Software University Forums

- [forum.softuni.bg](http://forum.softuni.bg)



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://softuni.org>
- © Software University – <https://softuni.bg>

