* Redux is library which help us to manage state of our application
* Large SPA with complex data flows
* It is not for simple app and simple data flow as it add complexity to the application
* Multiple view working with same type of data and do not have parent child relationship
* Usually we will have multiple copies of data which is independent of each other.
* Example facebook
* Common solution could be Events: data could be updated in unpredictable way. if there is a way we have to jump all over the place to get flow of data.
* Same as Redux Facebook has flux architecture.
* Benefits:

Predictable application state

Decoupled Architecture.

Testibility

Great Tooling

Undo/Redo

* When To Use:

1-Independent copies of same data ta multiple places

2-Multiple view that need to work with same data and be in sync.

3-Data can be updated by multiple users

4-Data can be updated by multiple actors

* Building Blocks of Redux:

Store

Actions

Reducers

**Store:** Store is a simple javascript object that contains store of the application, it is like local client side database.

**Component Component Component**

Store

**Actions:** Plain simple object that represent that something has happened. Semantically they are more like events.

They are CQRS architectural style(Query and Command pattern) Or Commands And Events

{type: ‘MARK\_AS\_READ’}

{type:’post\_message’, body:’…..’}

**Reducer:** The functions that specifies how the state changes in response to an action. Semantically they are like event Handler or Action Handler.

**Note:** A Reducer does not modify the state, it returns new state. Then store update the state.

A reducer is a pure function

What is pure function:

1. Same input🡪 Same output
2. No side effects

Impure function:

function increment(input)

{

Input.counter++;

}

function increment(input)

{

Service.addMessage(input);

}

function increment(input)

{

Input.count += Math.random();

}

Changing above all the functions to pure function:

function increment(input)

{

Return {counter: input.count+1}

}

In pure function we should be mutate of modify any of the argumants and it should return same result always if same input.

**Reducer function:**

function(state, actions){

switch(action.type){

case ‘INCREMENT’:

return{count:state.count +1};

}}

**Benefits:**

1. Easy testability
2. Easy to undo/redo
3. Time travel debugging

Redux implementation for angular:

1. Ngrx/store: github.com/ngrx/store
2. Ng2-redux: github.com/angular-redux/ng2-redux

<https://angular-2-training-book.rangle.io/handout/state-management/ngrx/adding_ngrx_to_your_project.html>