An Introduction

@manoj_nama

Multiple Reducers

- We can have multiple reducers which can act on the state specified to that reducer
- All the reducers can **act** on any *Action*, but they have **access** to manipulate only their state.
- We can add multiple reducers by using `combineReducers` utility from **redux**
- `import { combineReducers } from 'redux'`
- We specify the state tree we want with each of the reducer

Multiple Reducers

```
const reducers = combineReducers({
    users: userReducer,
    products: productsReducer
});
// each reducer will have a state tree as its initial state
// userReducer is not aware about existence of products
// reducer and vice-versa

// create the store with combined reducers
const store = createStore(reducers);
```

Middlewares

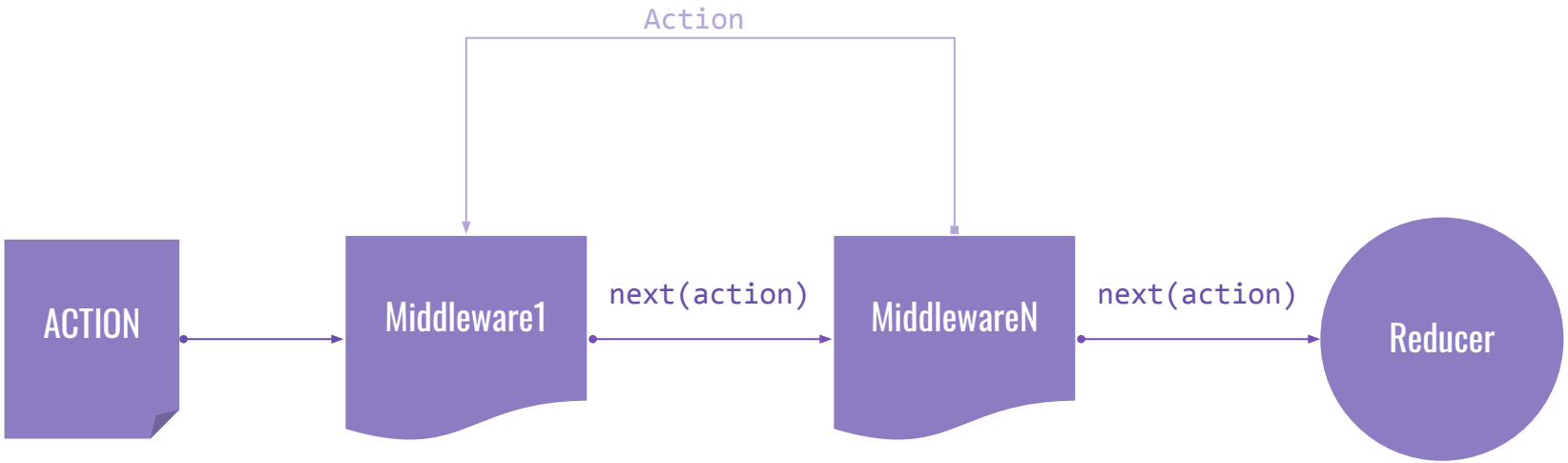
- Are simple functions intercepting the actions **before** reaching to the reducers
- They can **manipulate** actions and can **block** any actions to be forwarded to the reducer
- Middleware have the power to do async actions, i.e API calls etc. Since they have control over actions to be forwarded
- These are a chain of **thunks**, i.e functions returning functions
- `import { applyMiddleware } from 'redux'`

Middlewares

```
const applyMiddleware = (store) => (next) => (action) {
    // manipulate action here
    ...
    // forward action to next middleware(s) or to a reducer
    next(action)
};

To use middlewares, use applyMiddleware utility from redux
const middlewares = applyMiddleware(...list of middlewares)
const store = createStore(reducers, middlewares);
```

How redux-thunk works?



Async Actions

- Actions are synchronous, i.e they happen instantly
- For things like API calls or network requests, these need to be used in a different way
- We use `redux-thunk` to implement *async* actions
- These are multiple actions fired over a period of a Network request
- I.e Request fired, Request succeeded, Request failed
- `redux-thunk` provides a middleware for this called `thunkMiddleware`

Async Actions

```
const asyncAction = () => {
    return (dispatch) => { // this is store's dispatch method
        dispatch(apiCallStarted()); // call started
        fetch('http://rest.learncode.academy/api/ttn/users')
            .then(response => response.json())
            .then(data => {
                dispatch(apiCallSuccess(data));  // success
            })
            .catch(err => {
                dispatch(apiCallFailed(err));    // failure
            });
    }
};
```