Full-stack Application Development

Redux – predictable state container for JavaScript apps

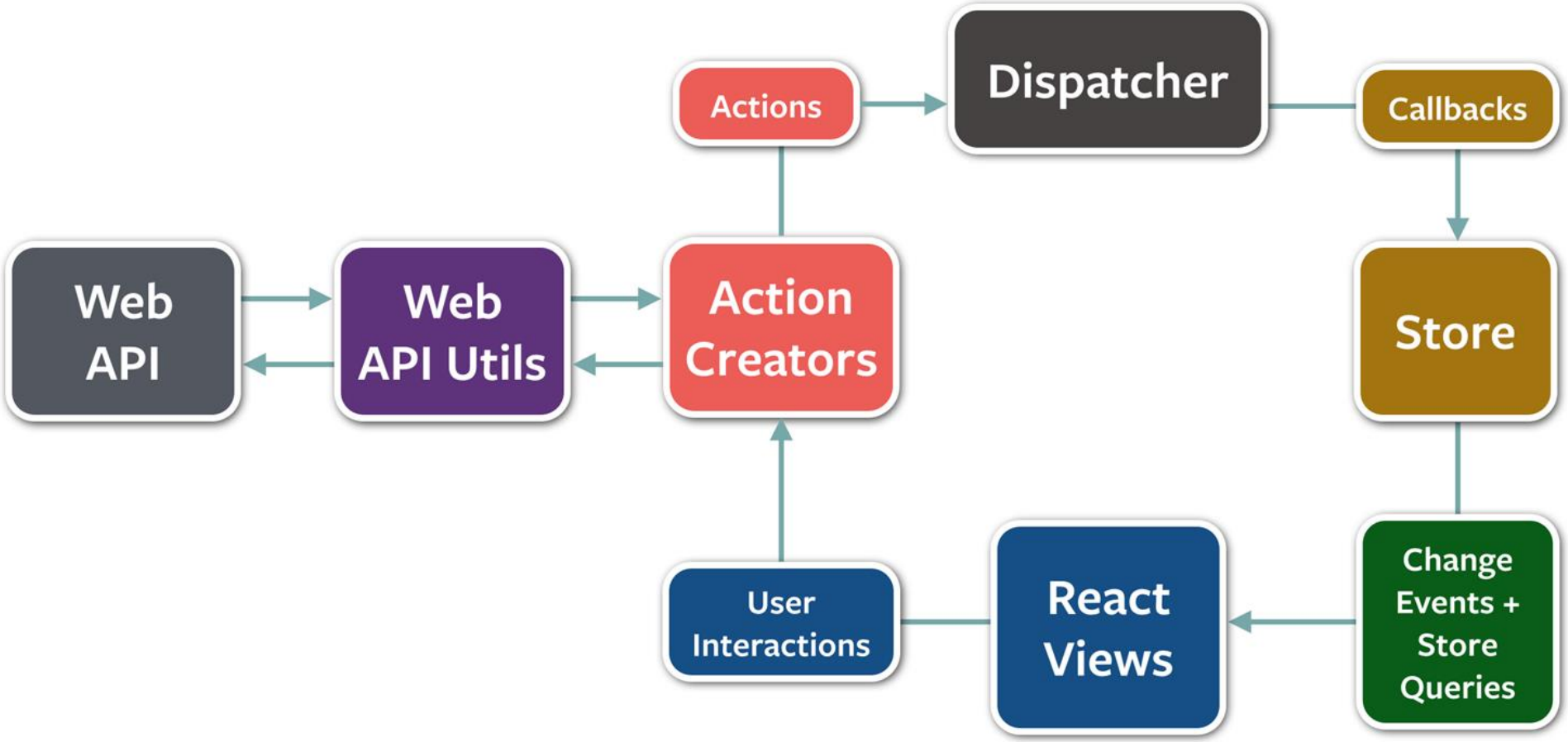
Where to Find The Code and Materials?

https://github.com/iproduct/react-native-training

Agenda

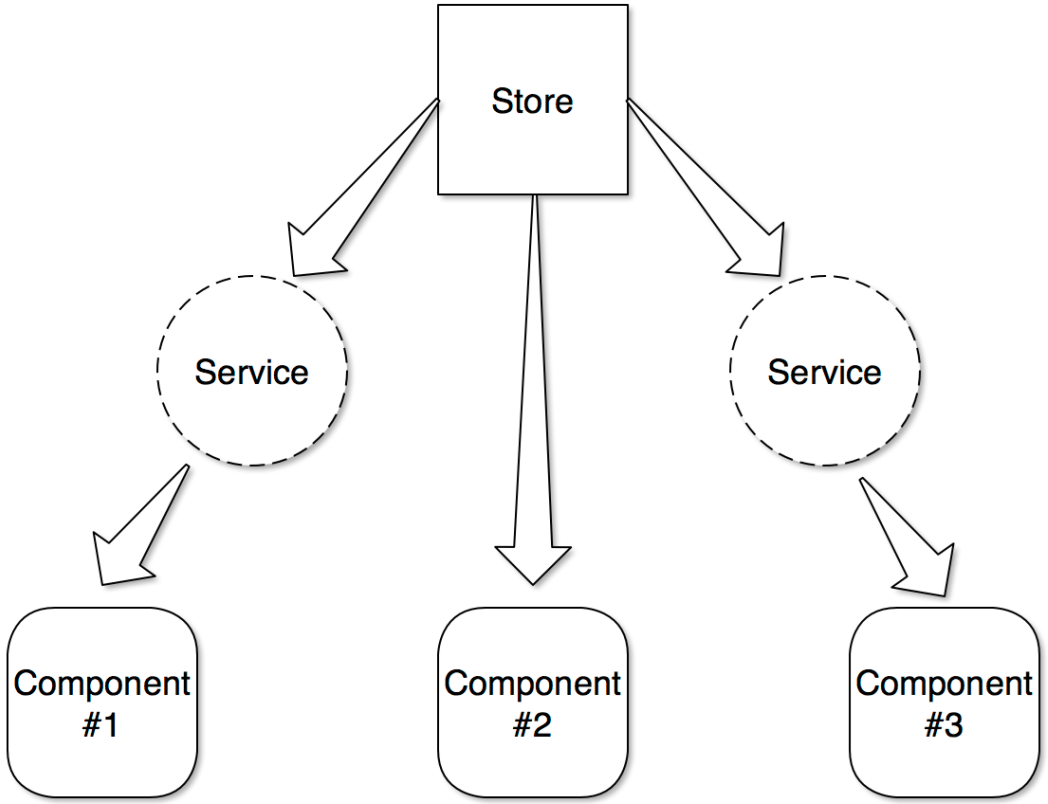
1. Flux and Redux design patterns
2. Main components: store, actions, action creators, reducers, selectors
3. Redux Recommended (Basic) Project Structure
4. Bootstrapping the Redux App
5. Redux Action Creators
6. Redux Reducers
7. Redux Root Reducer
8. Redux Containers and *connect* decorator
9. Using Redux Router Redux and Redux Thunk
10. Advanced Redux using Middleware Libraries

Flux Design Pattern



Source: Flux in GitHub, https://github.com/facebook/flux, License: BSD 3-clause "New" License

Redux Design Pattern



Linear flow: Dispatch → Reducers → New State → Store

Source: @ngrx/store in GitHub, https://gist.github.com/btroncone/a6e4347326749f938510

Redux

- Streamlined state management for **React.js** applications, inspired by **Redux**
- **State** is a single immutable data structure
- **Actions** describe state changes
- Pure functions called **reducers** take the previous state and the next action to compute the new state
- State is kept within single **Store**, and accessed through sub-state **selectors**, or as Observable of state changes
- Components are by default performance optimized using the shouldComponentUpdate() → performant change detection

Three Principles of Redux

- Single source of truth - the global state of your application is stored in an object tree within a single store.
- State is read-only - the only way to change the state is to emit an action, an object describing what happened.
- Changes are made with pure functions - to specify how the state tree is transformed by actions, you write pure reducers.

Redux Recommended (Basic) Project Structure

- **actions** – action creator factory functions (design pattern Command)
- **assets** – static assets (css images, fonts, etc.) folder
- **components** – simple (dumb) react components – pure render
- **container** – Redux Store aware (smart) component wrappers
- **reducers** – the only way to advance state:

function(OldStoreState, Action) => NewStoreState // = Rx **scan()**

- **index.js** – bootstraps app providing access to Store for all containers (smart components) using React context

Installing Redux + Redux Thunk + Redux Devtools

- Installing Redux and React-Redux integration:
  npm install redux react-redux
- Installing React-Redux typings (Redux comes with own .d.ts file):
  npm install --save-dev @types/react-redux
- Installing Redux Thunk library for handling async operations:
  npm install redux-thunk
- Installing Redux Thunk typings:
  npm install --save-dev @types/redux-thunk
- Installing Redux Devtools: **npm install redux-devtools-extension**
- Installing Redux Thunk typings:
  npm install --save-dev @types/redux-devtools-extension

Bootstrapping the Redux App – index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import { createStore } from 'redux';
import rootReducer from './reducers';
import { FilteredTodoApp } from './containers/filtered-todo-app';
const store = createStore(
   rootReducer, window.__REDUX_DEVTOOLS_EXTENSION && window.__REDUX_DEVTOOLS_EXTENSION__() );
const render = (Component) => {
 ReactDOM.render(
   <Provider store={store}>
       <FilteredTodoApp />
   </Provider>,
    document.getElementById('root')
 );
};
```

← Redux store provider

← Top level container component

Redux Action Creators – /actions/index.js

```
let nextTodoId = 0;
export const addTodo = (text) => ({
  type: 'ADD_TODO',
  id: nextTodoId++,
  text
});
export const setVisibilityFilter = (filter) => ({
  type: 'SET_VISIBILITY_FILTER',
  filter
});
export const changeStatus = (id, status) => ({
  type: 'CHANGE_STATUS',
  id,
  status
});
...
```

Redux Reducers – /reducers/todo.js

```
const todoReducer = (state = {}, action) => {
  switch (action.type) {
  case 'ADD_TODO':
    return {
      id: action.id,
      text: action.text,
      status: 'active'
    };
  case 'CHANGE_STATUS':
    if (state.id !== action.id) {
      return state;
    }
    return Object.assign({}, state, { status: action.status });
  default:
    return state;
  }
};
```

Redux Reducers – /reducers/todos.js

```
const todosReducer = (state = [], action) => {
  switch (action.type) {
  case 'ADD_TODO':
    return [
      ...state,
      todoReducer(undefined, action)
    ];
  case 'CHANGE_STATUS':
    return state.map( todo => todoReducer(todo, action) );
  case 'DELETE_TODOS':
    return state.filter(todo =>
        todo.status !== action.status
      );
  default:
    return state;
  }
};
```

Redux Root Reducer – /reducers/index.js

```
import { combineReducers } from 'redux';
import todos from './todos';
import visibilityFilter from './visibilityFilter';

const rootReducer = combineReducers({
  todos,
  visibilityFilter
});

export default rootReducer;
```

Redux Containers – /conatiners/visible-todo-list.js

```
const getVisibleTodos = (todos, filter) => todos.filter(
  todo => filter === 'all' ? true: todo.status === filter);
const mapStateToProps = (state) => ({
  todos: getVisibleTodos(state.todos, state.visibilityFilter)
});
const mapDispatchToProps = (dispatch) => ({
  onCompleted: (id) => {
    dispatch(changeStatus(id, 'completed'));
  },
  onCanceled: (id) => {
    dispatch(changeStatus(id, 'canceled'));
  }
});
const VisibleTodoList = connect(mapStateToProps, mapDispatchToProps) (TodoList);
export default VisibleTodoList;
```

Redux App using ES7 @connect Decorator

```
const getVisibleTodos = (todos, filter) => todos.filter(
  todo => filter === 'all' ? true: todo.status === filter);
const mapStateToProps = (state) => ({
  todos: getVisibleTodos(state.todos, state.visibilityFilter)
});
const mapDispatchToProps = (dispatch) => ({
  onCompleted: (id) => {
    dispatch(changeStatus(id, 'completed'));
  },
  onCanceled: (id) => {
    dispatch(changeStatus(id, 'canceled'));
  }
});
@connect(mapStateToProps, mapDispatchToProps)
export default class TodoList extends React.Component {
  constructor(props) { ...
```

Using Redux Router Redux and Redux Thunk (1)

- Idea: **history** + **store (redux)** → **react-router-redux** → **enhanced history** → **react-router**

```
import { compose, createStore, combineReducers, applyMiddleware } from 'redux';
import { Provider } from 'react-redux';
import createHistory from 'history/createBrowserHistory';
import { ConnectedRouter, routerReducer, routerMiddleware, push } from
  'react-router-redux';          // React Router to Redux integration
import thunk from 'redux-thunk'; // Allows using thunks = async actions
import reducers from './reducers'; // your reducers here
const history = createHistory();   // use browser History API
// middleware for intercepting & dispatching navigation & async actions
const middleware = [routerMiddleware(history), thunk];
// Enable Redux Devtools
const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
```

Using Redux Router Redux and Redux Thunk (2)

```
const store = createStore(
  combineReducers({ ...reducers,
    router: routerReducer // Add the reducer to store on `router` key
  }),
  /* preloadedState, */
  composeEnhancers( applyMiddleware(...middleware) ));
store.dispatch(push('/repos/react/redux'));//dispatch navigation action
ReactDOM.render(
  <Provider store={store}>//ConnectedRouter use store from the Provider
    <ConnectedRouter history={history}>
        <App />
    </ConnectedRouter>
  </Provider>,
  document.getElementById('root')
)
```

Redux Thunk Async Actions - /actions/counter.js

```
export function increment(x) {
  return { type: INCREMENT, amount: x }
}

export function incrementAsync(x) {
  return dispatch => //Can invoke sync or async actions with `dispatch`
    setTimeout(() => { dispatch(increment(x));  }, 2000);
}

export function incrementIfOdd(x) {
  return (dispatch, getState) => {
    const { counter } = getState();
    if (counter.number % 2 === 0) return;
    dispatch(increment(x));  //Can invoke actions conditionally
  };
}
```

Advanced Redux using Middleware Libraries

- **Normalizr** – normalizing and denormalizing data in state, helps to transform nested JSON response structures into a relational DB-like plain entities, referenced by Id in the Redux store.
- **redux-thunk** – in addition to plain actions, **action creators** can now return **Thunks** – callback functions of **dispatch** and **getState** arguments, allowing to handle async operations like data fetch from REST endpoint and Promise-like composition.
- **redux-promise/ redux-promise-middleware** – thunk alternatives
- **redux-observable** – really powerfull reactive transforms of async action events as RxJS Observables, called **Epics**:

(action$:Observable<Action>, store:Store) => Observable<Action>

- **redux-saga** - a library that aims to make application side effects (i.e. asynchronous things like data fetching and impure things like accessing the browser cache) easier to manage, more efficient to execute, easy to test, and better at handling failures. Uses generator functions instead of promises.

Installing Redux Toolkit

- Installing Redux Toolkit with ECMAScript using official template:

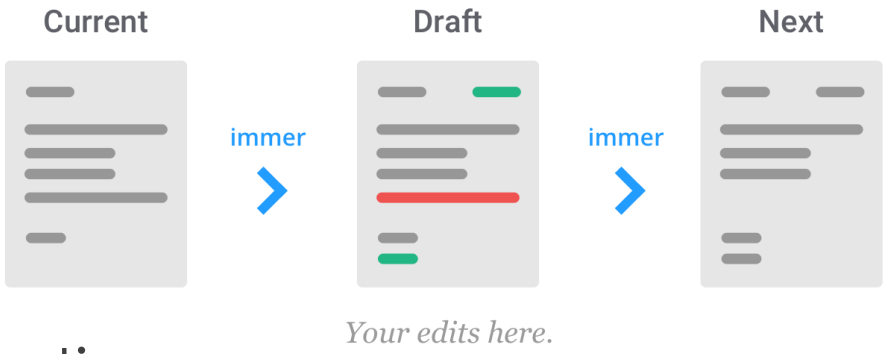
npx create-react-app my-app --template redux

- Installing Redux Toolkit in exisiting TypeScript project:

NPM: **npm install @reduxjs/toolkit**

Yarn: **yarn add @reduxjs/toolkit**

- Under the hood Redux Toolkit installs:
- **Redux** & **Redux-Thunk**
- **Immer** – easy to implement immutability
- **Reselect** – combining selectors with memoization



Redux Toolkit Helpers

- Store Setup
  - configureStore - abstraction over the standard Redux createStore function that adds good defaults to the store setup for better development experience.
  - getDefaultMiddleware - returns array containing the default list of middleware.
  - Immutability Middleware - any detected mutations will be thrown as errors.
  - Serializability Middleware - any detected non-serializable values will be logged.
- Reducers and Actions
  - createReducer - creating reducers, as lookup tables action_type -> handler
  - createAction – a helper function for defining a Redux action type and creator.
  - createSlice - initial state + object of reducer functions + "slice name" -> action creators + action types + reducers
  - createAsyncThunk - accepts action type + callback function returning promise
  - createEntityAdapter – generates a set of prebuilt reducers and selectors for performing CRUD operations on a normalized state structure
- Other – createSelector - utility from the Reselect library, re-exported

Redux Saga

- Asynchronous – using ES6 generators makes asynchronous flows easy to read, write, and test and allows to create complex side effects without getting distracted by the details.
- Composition-focused - Sagas enable different approaches to cope with parallel execution, task concurrency, racing, cancellation, and to have detailed control over the asynchronous flow of your code.
- Sagas are easy to test – allowing to assert results of each step in a generator or for the whole saga, and making testing of side effects quick, concise, and painless, as it should be.

Installing Redux Saga with TypeScript

- Installing Redux Saga:
  yarn add redux-saga

- Installing Redux Saga typings:
  yarn add –D @types/redux-saga

Redux Saga Example – I

```
import { call, put, takeEvery, takeLatest } from 'redux-saga/effects'
import Api from '...'

// worker Saga: will be fired on USER_FETCH_REQUESTED actions
function* fetchUser(action) {
    try {
        const user = yield call(Api.fetchUser, action.payload.userId);
        yield put({type: "USER_FETCH_SUCCEEDED", user: user});
    } catch (e) {
        yield put({type: "USER_FETCH_FAILED", message: e.message});
    }
}

/*
Starts fetchUser on each dispatched `USER_FETCH_REQUESTED` action.
Allows concurrent fetches of user.
*/
function* mySaga() {
    yield takeEvery("USER_FETCH_REQUESTED", fetchUser);
}
```

Redux Saga Example - II

```
/*
Alternatively you may use takeLatest.
Does not allow concurrent fetches of user. If "USER_FETCH_REQUESTED" gets
dispatched while a fetch is already pending, that pending fetch is cancelled
and only the latest one will be run.
*/
function* mySaga() {
yield takeLatest("USER_FETCH_REQUESTED", fetchUser);
}

export default mySaga;
```

Redux Saga Example - main

```
import { createStore, applyMiddleware } from 'redux'
import createSagaMiddleware from 'redux-saga'

import reducer from './reducers'
import mySaga from './sagas'

// create the saga middleware
const sagaMiddleware = createSagaMiddleware()
// mount it on the Store
const store = createStore(
      reducer,
      applyMiddleware(sagaMiddleware)
)

// then run the saga
sagaMiddleware.run(mySaga)

// render the application
```

Declarative Effects

- Sagas are implemented using Generator functions
- To express the Saga logic, we yield plain JavaScript Objects from the Generator. We call those Objects Effects. An Effect is an object that contains some information to be interpreted by the middleware. You can view Effects like instructions to the middleware to perform some operation (e.g., invoke some asynchronous function, dispatch an action to the store)
- To create Effects, you use the functions provided by the library in the **redux-saga/effects** package.
- Effects allows the Sagas to be easily tested, because they are declarative.
- Sagas can yield Effects in multiple forms. The easiest way is to yield a Promise.

Example - Saga watching a PRODUCTS_REQUESTED action:

```
import { takeEvery } from 'redux-saga/effects'
import Api from './path/to/api'

function* watchFetchProducts() {
      yield takeEvery('PRODUCTS_REQUESTED', fetchProducts)
}

function* fetchProducts() {
      const products = yield Api.fetch('/products')
      console.log(products)
}
```

=>

```
function* fetchProducts() {
      const products = yield call(Api.fetch, '/products')
      // ...
}
```

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

http://robolearn.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD