
epam
REACT.JS + REDUX
MANAGING YOUR APP STATE
SEPTEMBER, 2016

AGENDA

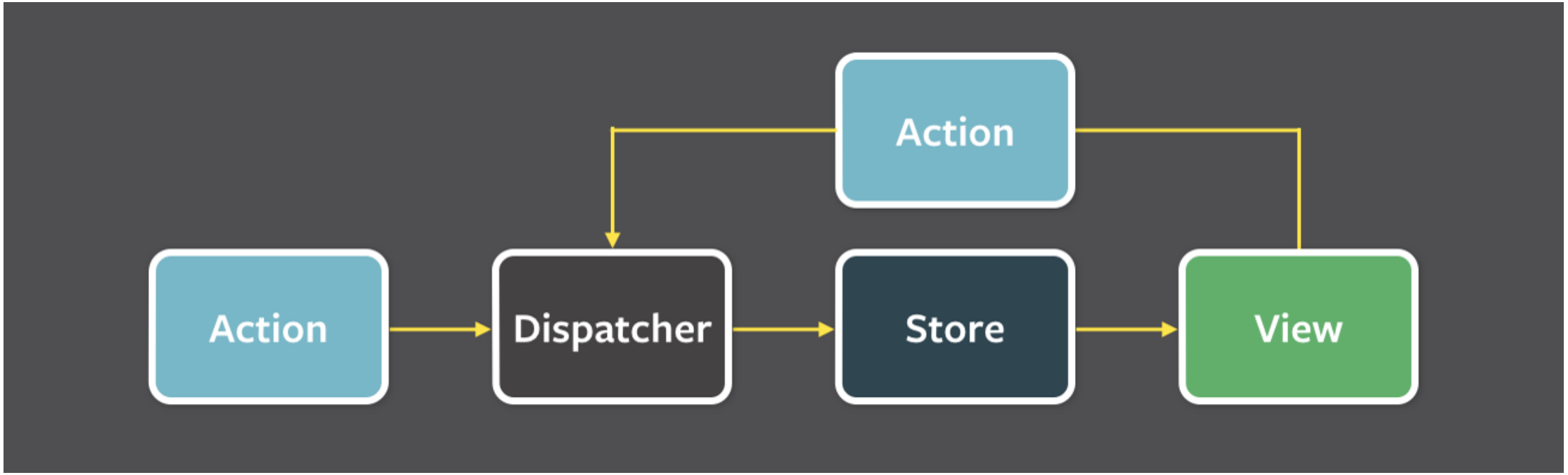
1. Managing application state
2. Understanding Redux
3. Integrating with React
4. Final Project

MANAGING APPLICATION STATE

STATE IN REACT COMPONENTS

- Hard to maintain
- Hard to test
- Hard to share data with other components
- Breaks the separation of concerns
  - Component is dealing with interface AND handling data

FLUX PATTERN



- Unidirectional data flow
- Stores handle the state management
- Application state becomes more predictable

IMPROVING FURTHER: REDUX

IMPROVING FURTHER: REDUX

- The whole state in a single object tree (single store)
  - Removes the need for a dispatcher

IMPROVING FURTHER: REDUX

- The whole state in a single object tree (single store)
  - Removes the need for a dispatcher
- Actions as the single way to change store data

IMPROVING FURTHER: REDUX

- The whole state in a single object tree (single store)
  - Removes the need for a dispatcher
- Actions as the single way to change store data
- Changes are described using pure reducers

UNDERSTANDING REDUX

REDUCERS

- Signature: `(state, action) => state`
- Pure functions

```
function counter(state = 0, action) {
  switch (action.type) {
    case 'INCREMENT':
      return state + 1;
    case 'DECREMENT':
      return state - 1;
    default:
      return state;
  }
}
```

STORE

- The single source of truth for your app state

```
import createStore from 'redux';
const store = createStore(myReducer);

const currentState = store.getState();

// Send actions with dispatch method
store.dispatch({ type: 'MY_ACTION' });

// Listen to changes with subscribe
store.subscribe(() => {
  // do something (render?)
});
```

UPDATING ARRAYS WITHOUT MUTATIONS

- **ES5:** `array.concat` method
- **ES6:** spread operator

```
// Adding:
list = list.concat([ newItem ]);

list = [...list, newItem];

// Removing:
list = list.slice(0, index)
  .concat(list.slice(index + 1));

list = [
  ...list.slice(0, index),
  ...list.slice(index + 1)
];
```

```
// Updating:
list = list.slice(0, index)
  .concat(list[index] + 1)
  .concat(list.slice(index + 1));

list = [
  ...list.slice(0, index),
  list[index] + 1,
  ...list.slice(index + 1)
];
```

UPDATING OBJECTS WITHOUT MUTATIONS

- **ES5:** No simple solution. Use a lib or include polyfill for ES6
- **ES6:** `Object.assign` method
- **Stage 2:** spread operator

```
// ES6
obj = Object.assign({}, obj, {
  prop: 'new-value'
});

// ESNext
obj = {
  ...obj,
  prop: 'new-value'
};
```

UPGRADE ACTIONS WITH MIDDLEWARES

Middlewares provides a third-party extension point between dispatching an action, and the moment it reaches the reducer.

The most common are:

- redux-logger
- redux-thunk
- redux-promise

```
import { createStore, applyMiddleware } from 'redux';
import thunkMiddleware from 'redux-thunk';
import createLogger from 'redux-logger';

function configureStore(initialState) {
  const middlewares = [thunkMiddleware];

  if (process.env.NODE_ENV !== 'production') {
    middlewares.push( createLogger() );
  }

  return createStore(
    rootReducer,
    initialState,
    applyMiddleware(...middlewares)
  );
}
```

REDUX-THUNK EXAMPLE

- Gives the possibility of dispatching functions:

```
function getItems(dispatch) {
  dispatch({ type: 'GET_ITEMS_REQUEST' });

  fetch('http://api.example.com')
    .then(res => res.json())
    .then(data => {
      dispatch({
        type: 'GET_ITEMS_SUCCESS',
        data
      })
    });
}

store.dispatch(getItems);
```

LEARN MORE

- Official website: http://redux.js.org
- Dan Abramov's free video series:
  - Getting started with Redux
  - Building React Applications with Idiomatic Redux

INTEGRATING WITH REACT

DEALING WITH DATA

PRESENTATIONAL COMPONENTS

- Have no knowledge of outside world
- Relies only on props for data and events

CONTAINER COMPONENTS

- Aware of Redux/State tree/Actions
- Can be generated by `react-redux` library

COMPONENTS KINDS

```
// Presentational Component
const TodoList = ({ todos, onTodoClick }) => (
  <ul>
    {todos.map(todo => <Todo {...todo} />)}
  </ul>
);
```

```
// Container
import { connect } from 'react-redux';

const TodoApp = connect(
  mapStateToProps,
  mapDispatchToProps
)(TodoList);
```

CONNECT PARAMS: MAPSTATETOPROPS

```
const getVisibleTodos = (todos, filter) => {
  switch (filter) {
    case 'SHOW_ALL':
      return todos
    case 'SHOW_COMPLETED':
      return todos.filter(t => t.completed)
    case 'SHOW_ACTIVE':
      return todos.filter(t => !t.completed)
  }
}

const mapStateToProps = (state) => {
  return {
    todos: getVisibleTodos(state.todos, state.visibilityFilter)
  }
}
```

CONNECT PARAMS: MAPDISPATCHTOPROPS

```
const mapDispatchToProps = (dispatch) => {
  return {
    onTodoClick: (id) => {
      dispatch(toggleTodo(id))
    }
  }
}
```

... is equivalent to:

```
const mapDispatchToProps = {
  onTodoClick: toggleTodo
};
```

PROVIDER

Makes `store` available inside components context. It's required for `connect()` usage.

```
import { Provider } from 'react-redux';

ReactDOM.render(
  <Provider store={store}>
    <MyComponent />
  </Provider>,
  rootElement
);
```

FINAL PROJECT

MUSIC APP

Create a music app that allows for searching artists and checking all his albums.

You should consume Spotify's API:

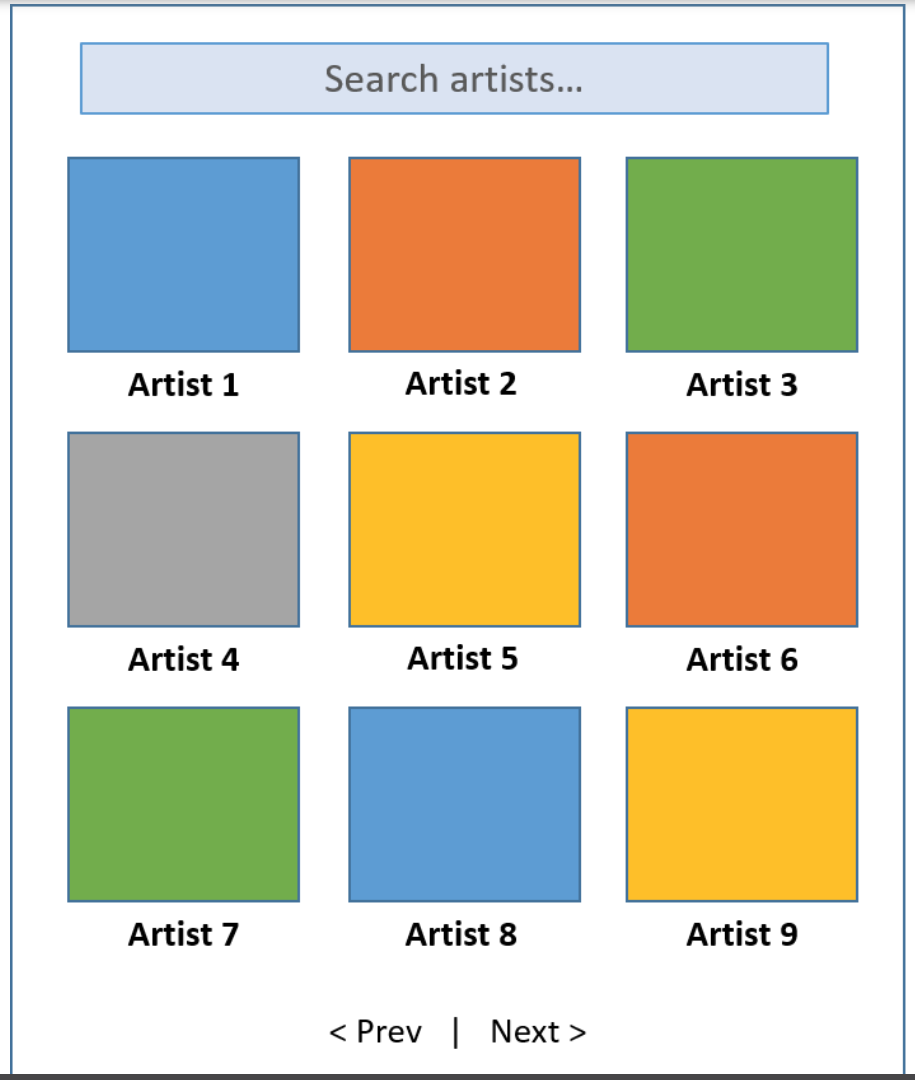
SEARCH ARTISTS

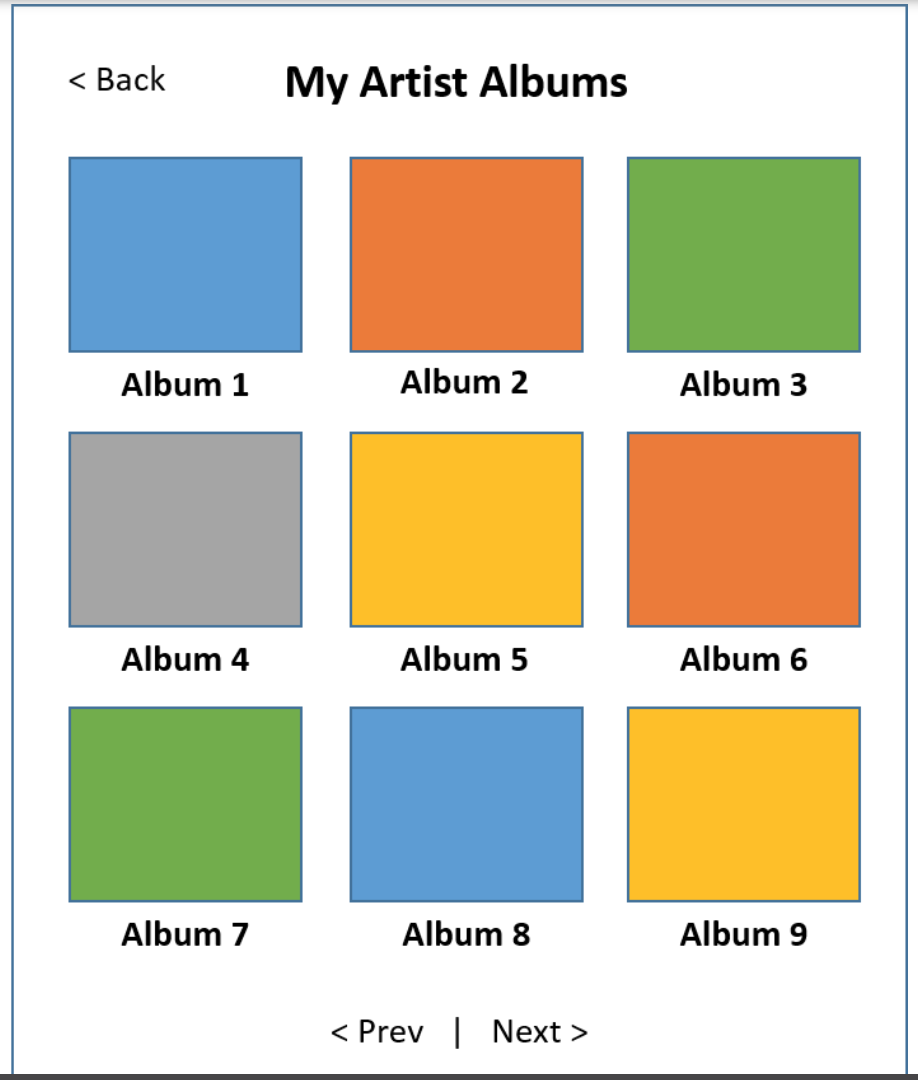
```
https://api.spotify.com/v1/search?type=artist&q=Metallica
```

LIST ALBUMS

```
https://api.spotify.com/v1/artists/{id}/albums
```

MUSIC APP WIREFRAME





MUSIC APP REQUIREMENTS

IMPORTANT PARTS

- Don't use React components' internal state, use Redux instead
- Display a loading indicator whenever you are waiting for an API response

Remember

- Containers doesn't handle UI
- Presentational components doesn't know the outside world

NICE TO HAVE

- Use react-router library
- Cache API results, so when you click on same artist again it doesn't trigger a new fetch

Q & A