React Training

Instructor
Jukka Tupamäki, jukka@lemanse.fi

Redux

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

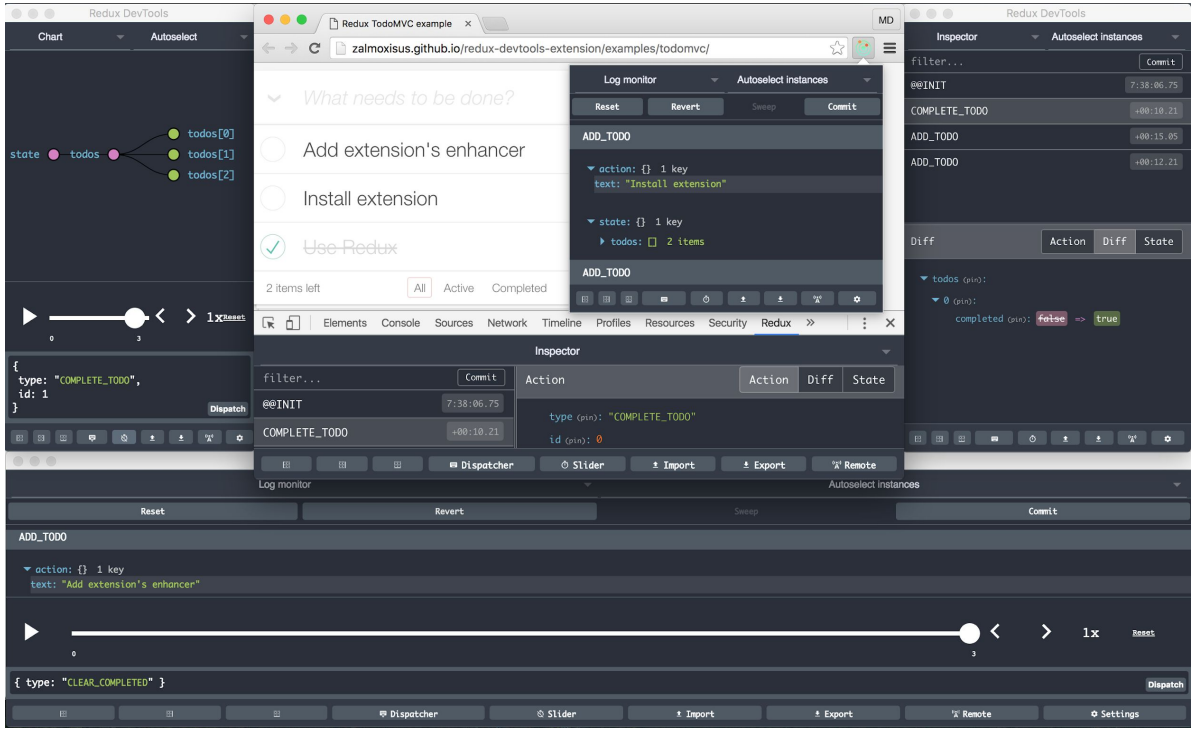
Redux (http://redux.js.org/)

- Redux implements an architecture similar to Flux
  - Redux is not just for React; it can be used in e.g. Ember, Angular, ...
  - Documentation: https://redux.js.org/
  - Video tutorials: https://app.egghead.io/playlists/fundamentals-of-redux-course-from-dan-abramov-bd5cc867

- Building blocks
  - **Action creator** functions (many)
  - **Reducer** functions (many)
  - **Store** (usually one)
  - **Middleware** functions (several; depending on what the app needs)

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

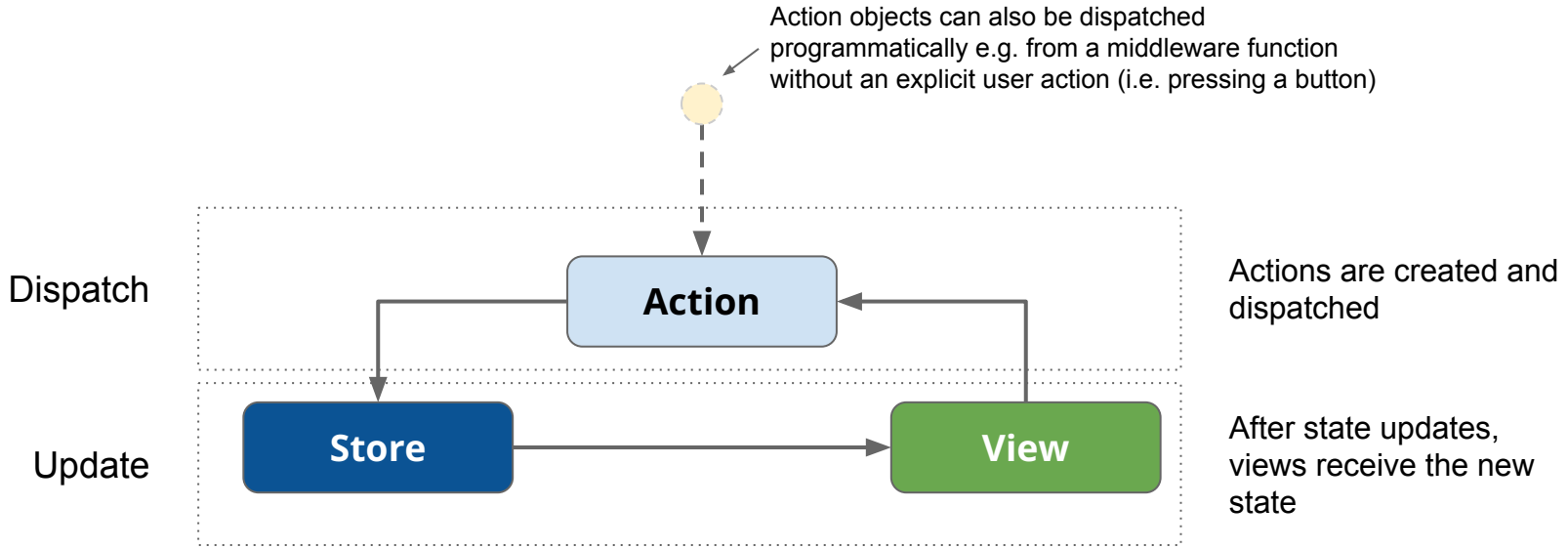
Redux DevTools (Chrome, Firefox)



https://github.com/zalmoxisus/redux-devtools-extension

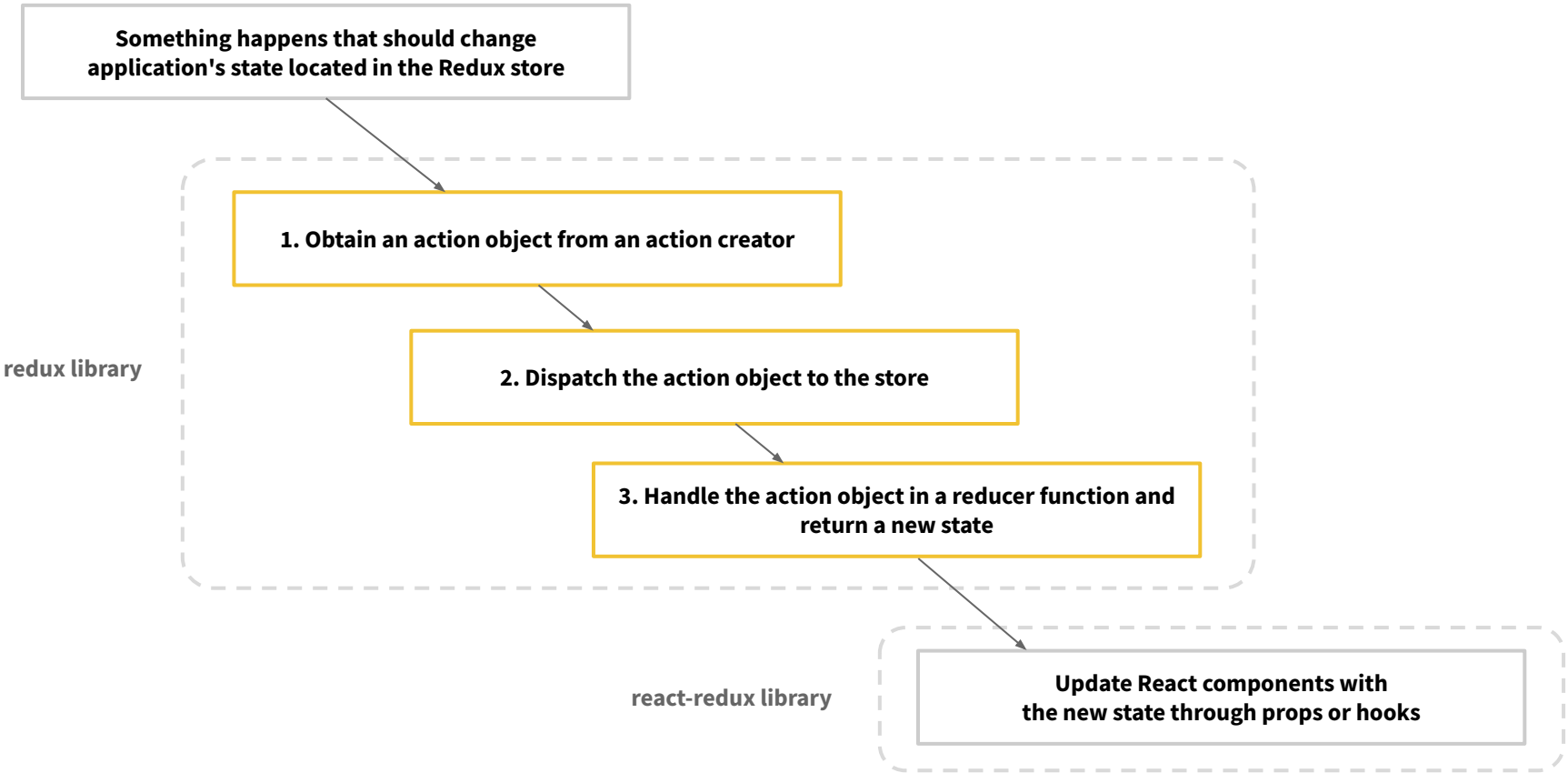
INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Redux architecture



INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

How Redux works



INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

How Redux works

- **Store's state can only be changed by dispatching an action object!**
  - Redux's three principles: https://redux.js.org/understanding/thinking-in-redux/three-principles
    - Single source of truth
    - State is read-only
    - Changes are made with pure functions

- Reducer functions decide which actions they handle
  - A reducer can change the state by returning a new state (immutability)
  - Each reducer function is responsible for a specific part of the state (state slice)
  - **Every action object goes through every reducer function!**

- *A store usually owns most of the application's state*
  - useState and this.state are still important utilities

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Slices

- "Slice" is a term used in Redux docs / Redux Toolkit that refers to a combination of
  - Action types
  - Action creators
  - A reducer function

- A slice implements a well defined feature
  - E.g. state handling & actions for interacting with a specific API route (e.g. /api/users)

- All the mentioned parts can be written in one file
  - In plain Redux, slices are just a convention; you can still write actions and reducers in separate files if you want to, or use any other convention
  - In Redux Toolkit, slices are a core concept and must be used

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Action types, action creators and action objects

```
const INCREMENT = 'INCREMENT';
```

Action types can be defined as ***unique*** strings. If you have many similar actions in different "slices", you can prefix the action types:

```
const INCREMENT = 'counter/INCREMENT';
```

```
function increment() {
  return {
    type: INCREMENT
  };
}
```

An **action creator** function returns **an action object.** Action creator functions are synchronous by default.

The **type** key is used in reducer functions to decide whether an action should be handled or not. Actions can contain any other data as well (payload) e.g. form data, an updated resource, etc..

An action object can contain any data in addition to the required **type** key (e.g. user input from a form, data from an API)

```
type IncrementAction = ReturnType<typeof increment>;
function isIncrementAction(action: AnyAction): action is IncrementAction {
  return action.type === INCREMENT;
}
```

Helpers for dealing with action object types in reducers

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Using an action creator and dispatching an action

```
const action = increment();
```

1. Calling an action creator creates an action object.

```
appDispatch(action);
```

2. By dispatching the action object, it goes through all middleware and reducer functions.

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Reducer functions

```
const initialState = { value: 0 };

function reducer(state = initialState, action: AnyAction) {
  if (isIncrementAction(action)) {
    return {
      ...state,
      value: state.value + 1
    };
  }
  return state;
}
```

Requirements

1) If the **state** parameter is **undefined**, the reducer function must return its initial state

2) If an action is handled, **a reducer function must return a new data structure**. *Changes are noticed only if the data structure's reference changes.*

3) If an action is not recognized, reducer must return the current state as-is.

- **A reducer** returns a new state based on the current state and the received action
  - A reducer function owns a part of the store's state and is responsible for managing it
  - Reducers use action object's **type** key to decide what to do
  - **Reducers must not mutate the state object**: always return a new data structure if the state changes (use either Object.assign or the spread operator)

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Store setup

Redux maintainers have decided to advertise Redux Toolkit aggressively by marking the createStore function as legacy. Legacy does not mean that createStore is deprecated / should not be used.

```
import { combineReducers, legacy_createStore as createStore, applyMiddleware, Middleware } from "redux";
import { composeWithDevTools } from "redux-devtools-extension";
import thunk from "redux-thunk";
import * as counterSlice from "./slices/counter";

const rootReducer = combineReducers({
  counter: counterSlice.reducer,
});

const middleware: Middleware[] = [thunk];

const store = createStore(
  rootReducer,
  composeWithDevTools(applyMiddleware(...middleware))
);

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;

export default store;
```

Reducers are bundled together with the combineReducers() helper. Each reducer has a unique key, e.g. "counter" for the counter reducer.

Middleware functions extend store's built-in features. Redux Thunk is a common middleware that allows async actions to be dispatched.

A store instance is created by calling createStore. At this point you can pass initial values to the store; e.g. restore previous state from local storage / database (rehydration, persistence).

Enable the Redux DevTools browser extension (recommended) by wrapping the middleware functions with composeWithDevTools

Infer types for root state and for the dispatch function to make the rest of the app aware of what kind of data the reducers return.

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Hooks & Types

- The **react-redux** library provides hooks for selecting data from the store (**useSelector**) and for dispatching actions (**useDispatch**)
- By default, the hooks do not know about our store's data types
- In order to make them type aware, we have to create app specific versions of the hooks

  Read more from the Redux docs: "Root state and dispatch types" and "Define typed hooks"

data/store.ts

```
export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;
```

data/hooks.ts

```
export const useAppDispatch: () => AppDispatch = useDispatch;
export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector;
```


INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Provider component (react-redux)

```
import { Provider } from 'react-redux';
import store from './data/store';

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
);
```

The Provider component is required to allow React components to access the Redux store and to dispatch actions

The Provider component becomes the new root component of the app. Redux and related hooks can be used only in Provider's child components.

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Using connect() to consume data in a class component

```
const ConnectedCounter = connect(
  function mapStateToProps(state, props) {
    return { value: state.counter.value };
  },
  function mapDispatchToProps(dispatch, props) {
    return {
      increment: () => dispatch({ type: 'INCREMENT' })
    };
  }
)(Counter);

export default ConnectedCounter;
```

mapStateToProps() selects values for the component from the state and from user-defined props. Values are passed to the component as props. *Returns an object.*

mapDispatchToProps() appends functions (action creators) to component's props. The functions are used by the component to dispatch actions.

connect() returns *a HOC function* that must be called with a component. The HOC returns the connected component that gets state updates via props, selected by the two functions: mapStateToProps and mapDispatchToProps

```
// Values returned by mapStateToProps and mapDispatchToProps can be found in Counter's props
this.props.value
this.props.increment()
```


INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Using connect() to consume data in a class component

```
import { increment } from './actions/counter';


export default connect(
  state => ({
    value: state.counter.value
  }),
  {
    increment
  }
)(Counter);

// The `value` variable and the `increment` function are passed as props to Counter component
this.props.value
this.props.increment();
```

The connect() call can be written in a bit shorter form by exporting the connected component without temporary variables. Written like this can look weird at first.

A *shorthand* notation for **mapDispatchToProps()** is to use an object that contains action creator functions.

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Hooks

Selecting data from the store – (compare to mapStateToProps())

```
const value = useSelector(state => state.counter.value);
```

Dispatch an action – (compare to mapDispatchToProps())

```
const appDispatch = useAppDispatch();
appDispatch(increment());
```

See https://react-redux.js.org/api/hooks

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Hooks

- You don't need to "connect" a component to the store; instead you pull the data that you need from the store using the useSelector() hook

- Hooks do not create an extra wrapper components to the component hierarchy like connect()

- Using Redux looks suddenly pretty easy...

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Counter component with hooks

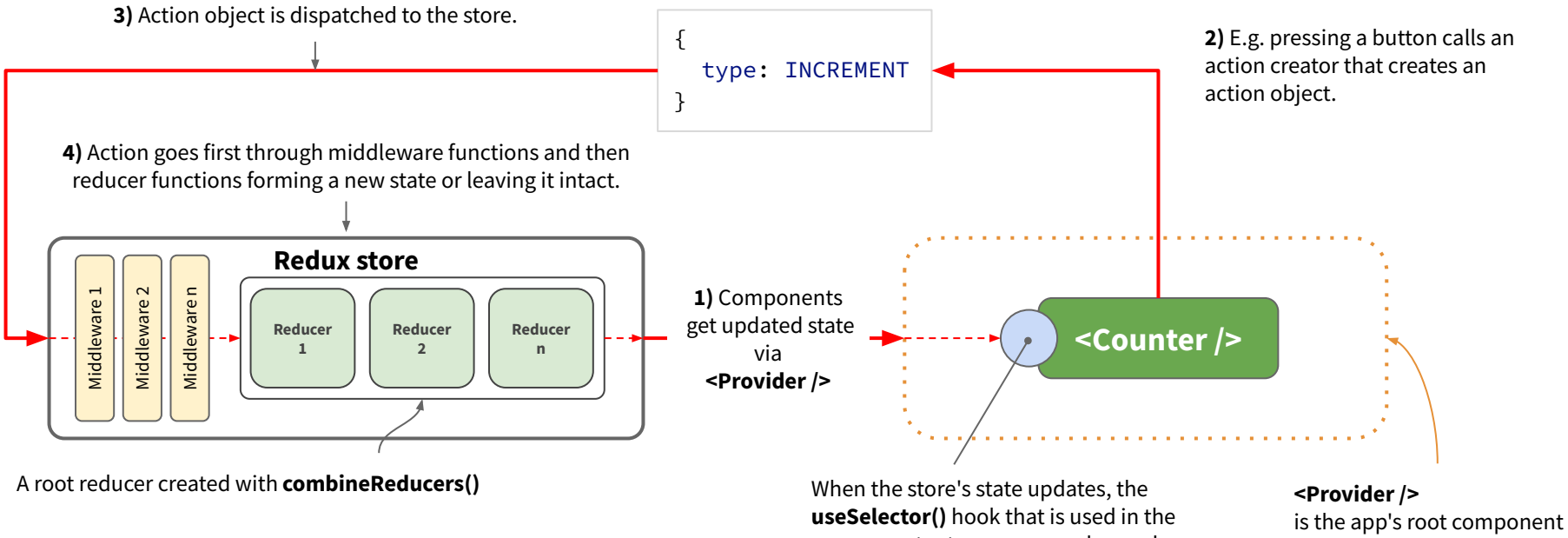
```
import { useSelector, useDispatch } from "react-redux";
import * as counterActions from "../data/actions/counter";

function Counter() {
  const value = useSelector((state) => state.counter.value);

  const dispatch = useDispatch();
  const decrement = () => dispatch(counterActions.decrement());
  const increment = () => dispatch(counterActions.increment());

  return (
    <div className={styles.root}>
      <div className={styles.button} onClick={decrement}>-</div>
      <div className={styles.value}>{value}</div>
      <div className={styles.button} onClick={increment}>+</div>
    </div>
  );
}
```

Select values from the store

Dispatch actions

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

React / Redux anatomy - If the component is connected to Redux using hooks

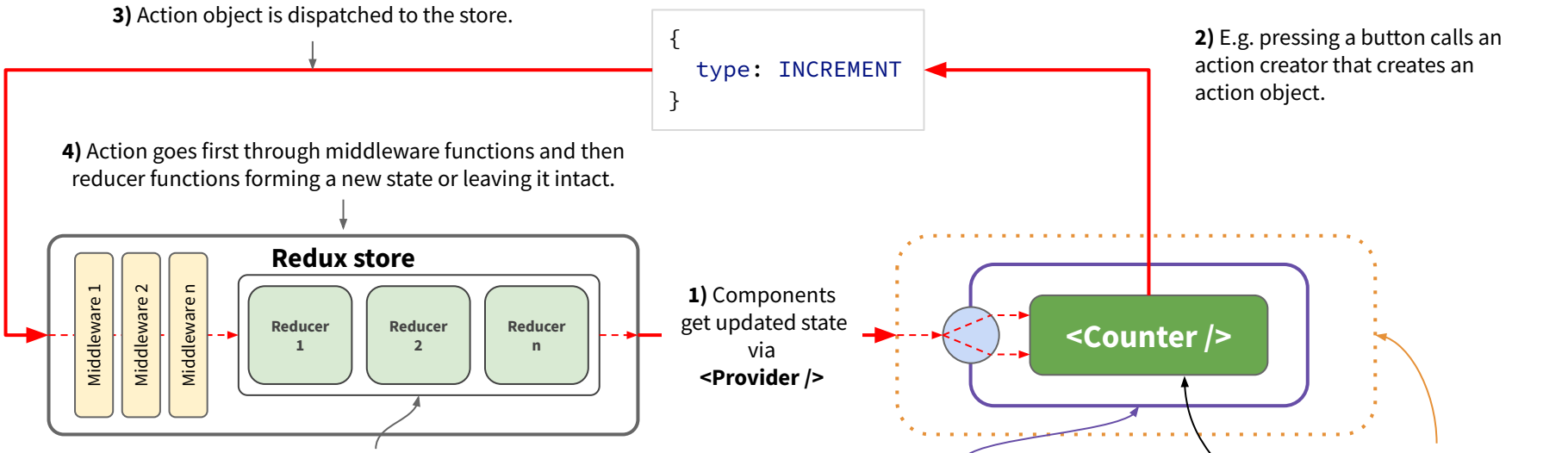



```
const rootReducer = combineReducers({
  counter: counterReducer,
  users: usersReducer
});
```

About creating a middleware function:
https://redux.js.org/api-reference/applymiddleware#example-custom-logger-middleware

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

React / Redux anatomy - If the component is connected to Redux using connect()




```
const rootReducer = combineReducers({
  counter: counterReducer,
  users: usersReducer
});
```

About creating a middleware function:
https://redux.js.org/api-reference/applymiddleware#example-custom-logger-middleware

```
export default connect(
  mapStateToProps,
  mapDispatchToProps
)(Counter);
```


INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Exercise

Getting started with Redux

1. Install a few modules:
   a. redux
   b. react-redux
   c. redux-thunk
   d. reselect
   e. redux-devtools-extension
2. Open **src/index.tsx**
3. Import the store from **data/store.ts**
4. Add the **Provider component** to the root of the app and pass the store as a prop

```
<Provider store={store}>
  <BrowserRouter>
    <App />
  </BrowserRouter>
</Provider>
```

5. Check that the app still works as usual.

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Exercise

Write a reducer function, add it to the store and select a value from the store

1. Open **src/data/slices/counter.ts**
2. In the file, write a reducer function using the following as the initial state:

```
const initialState = { value: 0 };
```

3. Open **data/store.ts**
   a. Add the counter reducer to the store's root reducer

4. Open **src/components/ReduxCounter.ts**
   a. Select the counter's value from the store and display it
   b. Remember to use the app-specific selector: **useAppSelector**

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Write action creators and related utilities

1. Open **src/data/slices/counter.ts**
2. Define action types:

```
const INCREMENT = 'counter/INCREMENT';
const DECREMENT = 'counter/DECREMENT';
```

3. Write action creator functions and export them:

```
export function increment(step: number) {}
export function decrement(step: number) {}
```

4. Add action object types for each action using return values

```
type DecrementAction = ReturnType<typeof decrement>;
```

5. Add type guards for each action

```
function isDecrement(action: AnyAction): action is DecrementAction {
  return action.type === DECREMENT;
}
```

Returns a type predicate

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Dispatch actions in the ReduxCounter component

1. Open **src/components/ReduxCounter.tsx**
2. Import the **increment** and **decrement** action creators from the counter slice
3. Define click handlers for the + and - buttons and assign them to the elements
    a. Remember to dispatch the actions using **useAppDispatch**

4. Open the Redux DevTools in the browser
5. Click the buttons and pay attention to the devtools' log

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Exercise

Handle the increment and decrement actions in the reducer

1. Open **src/data/slices/counter.ts**
2. Handle the actions in the reducer function
    a. Use the type guards for checking the action object type
    b. Remember to return a new state, do not mutate
3. Click ReduxCounter's buttons again. If the value changes, it works!

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Asynchronous actions

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Redux Thunk

```
export const useAppDispatch: () => AppDispatch &
  ThunkDispatch<RootState, unknown, AnyAction> = useDispatch;

export type AppThunk<ReturnType = void> = ThunkAction<
  ReturnType, RootState, unknown, AnyAction
>;
```

```
export function delayedAction(delayMs = 1000): AppThunk {
  return (dispatch, getState) => {
    setTimeout(() => {
      dispatch({
        type: "some action type",
      });
    }, delayMs);
  };
}
```

An async action creator that returns a thunk function

- Redux Thunk is a middleware function that adds support for asynchronous actions
- Thunk functions can dispatch zero or more actions when they are called by the middleware

- Docs: https://github.com/reduxjs/redux-thunk

But what is a thunk?

Wikipedia: "Thunks are primarily used to **delay a calculation** until its result is needed ..."

Maybe a better name for the lib would have been Redux Delay...

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Asynchronous action creator

Instead of returning a simple action object, an action creator can now return a function. It gets two parameters: ***dispatch()*** and ***getState()***

dispatch() is called within the function as many times as is needed.

In this example, a Promise chain is returned that allows dispatching status actions once the HTTP request progresses.

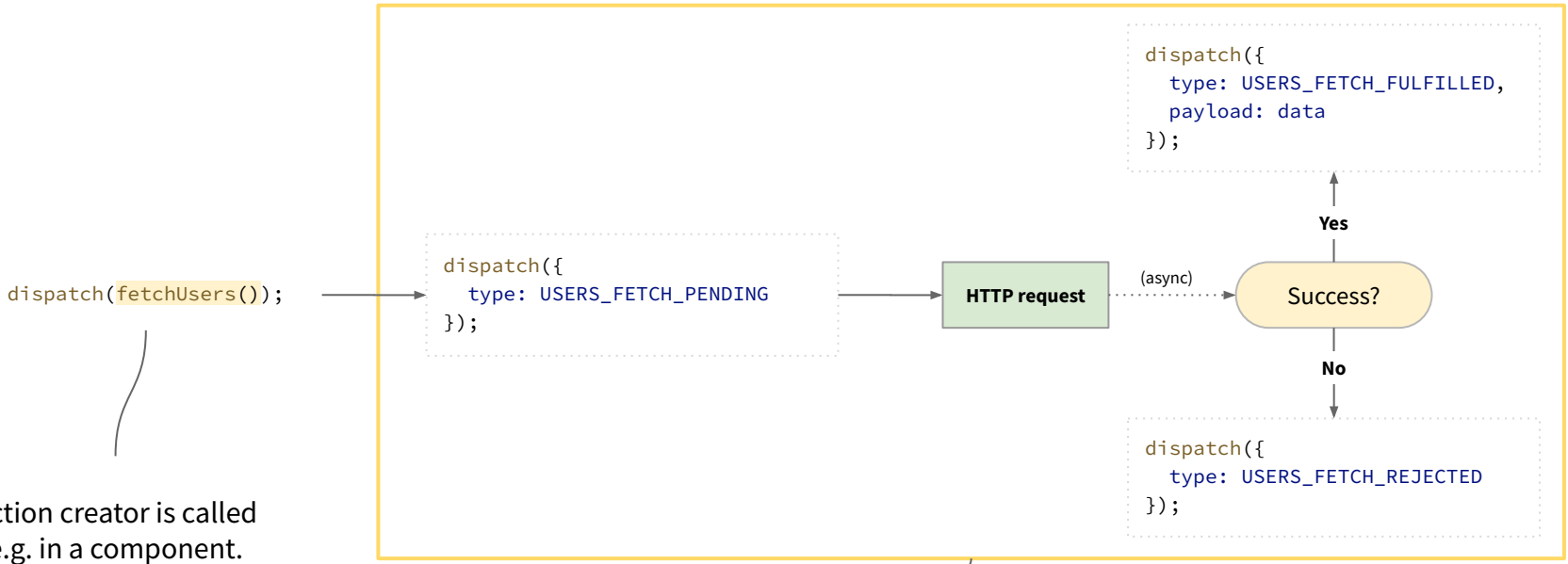
Thunk's return value can be used in e.g. a component that calls the action creator.

```
export function fetchUsers(): AppThunk<Promise<UserItem[]>> {
  return (dispatch, getState) => {
    dispatch({
      type: USERS_FETCH_PENDING
    });

    return axios
      .get<UserItem[]>('http://localhost:3001/users')
      .then(response => response.data)
      .then(data => {
        dispatch({
          type: USERS_FETCH_FULFILLED,
          payload: data
        });

        return data;
      })
      .catch(() => {
        dispatch({
          type: USERS_FETCH_REJECTED
        });

        return [];
      });
  };
}
```

The received data is carried in **action.payload** to the reducers

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

dispatch() calls during an HTTP request



Action creator is called e.g. in a component.

Inside the thunk function, HTTP request is created and actions are dispatched when the request is created, and when it succeeds or fails.

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Exercise

Fetch users

1. **In store.ts,** import the users reducer function and add it to the rootReducer named as **users**
2. Open Redux DevTools' log and inspect the data structure the reducer created
3. Open **UsersRoute.ts** and dispatch the users fetch action there when the component mounts
4. Open **src/data/slices/users.ts** and implement handling for the users fetch action

   Note that users **are stored in an object by ID**.
5. Go back to the UsersRoute.ts and select the loaded data from the store

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Application state in Redux

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

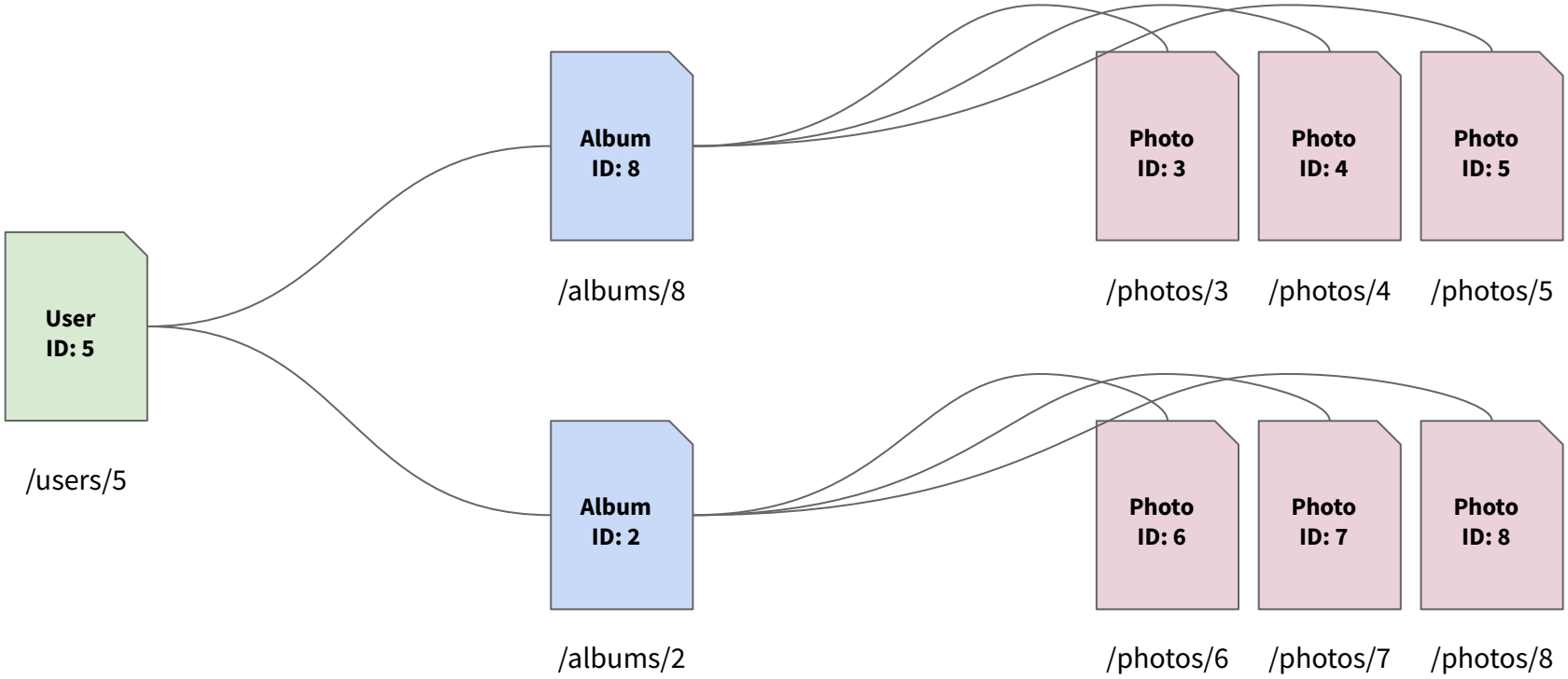
Application state in Redux

https://redux.js.org/recipes/structuring-reducers/prerequisite-concepts#normalizing-data
https://redux.js.org/recipes/structuring-reducers/normalizing-state-shape
https://redux.js.org/recipes/structuring-reducers/basic-reducer-structure#basic-state-shape

- Redux's state can be compared to **a normalized database**
  - Data is stored by primary IDs (cf. primary key)
  - Relations between data is modeled with IDs (cf. foreign key)
  - Reducers should not contain overlapping data (cf. BCNF, normalization)
    - If a resource can be fetched from multiple API endpoints, it should still be handled by the same reducer function
    - Read more: https://en.wikipedia.org/wiki/Boyce%E2%80%93Codd_normal_form

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Modeling relational data with Redux



INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Modeling relational data with Redux

```
// User object from API
{
  id: 5,
  name: 'Leanne Graham',
  username: 'Bret',
  email: 'Sincere@april.biz'
}
```

```
// Album object from API
{
  id: 8,
  userId: 5,
  title: 'quidem molestiae'
}
```

```
// Photo object from API
{
  id: 3,
  albumId: 8,
  title: 'accusamus similique qui sunt',
  url: '/full/3.jpg',
  thumbnailUrl: '/thumbnails/3.jpg'
}
```

```
// User reducer's output
{
  byId: {
    5: {
      id: 5,
      name: 'Leanne Graham',
      username: 'Bret',
      email: 'Sincere@april.biz'
    }
  }
}
```

```
// Albums reducer's output
{
  byId: {
    8: {
      id: 8,
      userId: 5,
      title: 'quidem molestiae'
    }
  },
  byUserId: {
    5: [8]
  }
}
```

```
// Photos reducer's output
{
  byId: {
    3: {
      id: 3,
      albumId: 8,
      title: 'accusamus similique qui sunt',
      url: '/full/3.jpg',
      thumbnailUrl: '/thumbnails/3.jpg'
    }
  },
  byAlbumId: {
    8: [3]
  }
}
```


INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Retrieving relational data from Redux

Reducer setup

```
combineReducers({
  photos: photosReducer,
  albums: albumsReducer,
  users: usersReducer
});
```

Retrieving relational data for components

```
// Getting the user object (ID is obtained from a URL parameter)
const user = state.users.byId[5];

// User's albums
const albumIds = state.albums.byUserId[user.id];
const albums = albumIds.map(albumId => state.albums.byId[albumId]);

// Photos of an album
const albumId = albums[0].id;
const photoIds = state.photos.byAlbumId[albumId];
const albumPhotos = photoIds.map(photoId => state.photos.byId[photoId]);
```

User object:

```
{
  id: 5,
  name: 'Leanne Graham',
  username: 'Bret',
  email: 'Sincere@april.biz'
}
```

Album IDs related to the User object:

```
[8]
```

After mapping album IDs to album objects:

```
[
  {
    id: 8,
    userId: 5,
    title: 'quidem molestiae'
  }
]
```

Album object:

```
{
  id: 8,
  userId: 5,
  title: 'quidem molestiae'
}
```

Photo IDs related to the Album object:

```
[3]
```

After mapping photo IDs to photo objects:

```
[
  {
    id: 3,
    albumId: 8,
    title: 'accusamus similique qui sunt',
    url: '/full/3.jpg',
    thumbnailUrl: '/thumbnails/3.jpg'
  }
]
```


INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

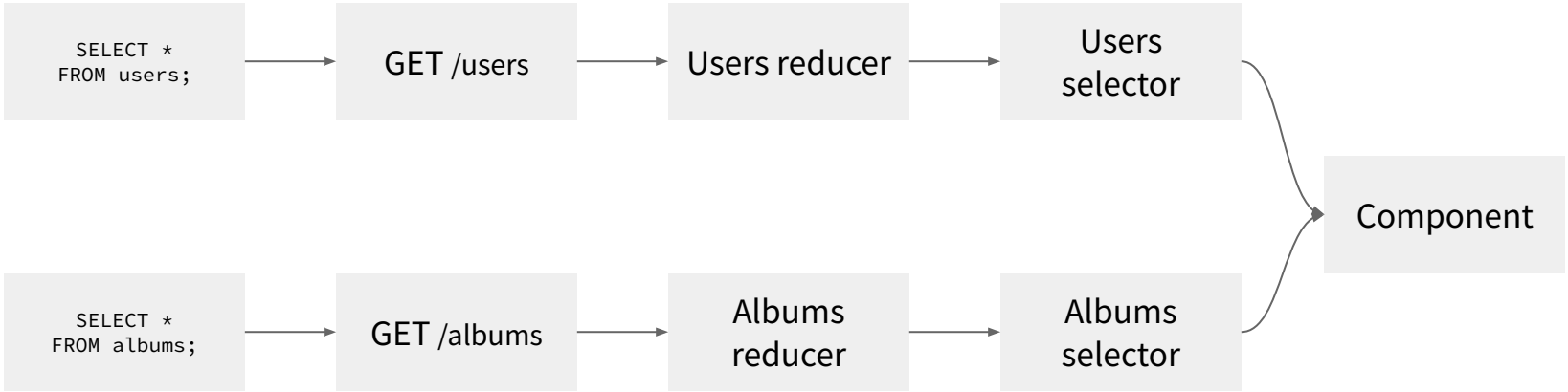
What to store in Redux

- **Domain data**
  - Any data that is received from an API and is used by many components
- **UI state**
  - Formed by user's actions e.g. "sort a list by name", "sort list by date"
- **Other app state**
  - E.g. undo history

- Try to figure out the lifespan of the data you have
- Does the data have to exist after a component is unmounted (that used the data)?
  - Yes --> Redux / No --> E.g. component's local state
- Is the data used by multiple components (at the same time)?
  - Yes --> Redux / No -> E.g. component's state

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

How to organize data with reducers



There is a ~1:1 relationship between database tables/entities and reducers!

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Optimizing Redux

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Things to consider

- Every action goes through all reducer functions

- Every state update can update every connected component
  - Use "Highlight updates" in React Developer Tools (under the Settings cogwheel) to see what changed and how frequently

- ***mapStateToProps() and useSelector() will be called a lot so keep those simple***

- The **Reselect** library creates selector functions with cache
  - Use selector functions in ***mapStateToProps()*** / ***useSelector()*** when getting data from Redux

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Reselect https://github.com/reactjs/reselect

```
import { createSelector } from 'reselect';

export const getAlbum = createSelector(
  [state => state.albums.byId, (state, albumId) => albumId],
  (byId, albumId) => byId[albumId] || {}
);

// How to use a selector
const album = getAlbum(state, albumId);
```

- A selector can pick values from any data structure (e.g. Redux store)
  - Selector implements caching; the output changes only if the input changes
  - Selectors can be composed of other selectors
  - A selector can e.g. map, filter, sort or transform data

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Sorting users in a selector function

```
import { createSelector } from 'reselect';
import _ from 'lodash';

// How to call this function e.g. in a component: getUsers(state, 'name', 'asc')
export const getUsers = createSelector(
  // createSelector()'s first argument is an array of functions.
  // These functions take care of providing arguments to the actual selector function below.
  // First function's return value becomes the first argument of the selector function, and so on.
  [
    state => state.users.byId,     // Users by ID, picked from Redux's state
    (state, key) => key,           // Name of the field that is used in sorting the result (e.g. name, email)
    (state, key, order) => order   // Sort order, ascending or descending
  ],

  // This is the actual selector function that is the second argument of createSelector()
  (usersById, key = 'name', order = 'asc') => {
    if (!(order === 'asc' || order === 'desc')) {
      throw new Error(`Invalid sort order: ${order}`);
    }

    const usersList = _.values(usersById);
    return _.orderBy(usersList, [key], [order]);
  }
);
```


INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi

Redux Toolkit

- Setting up Redux and Redux Thunk can be a lot of work especially in TypeScript
- Redux Toolkit is the official "framework" built on top of Redux that simplifies many aspects of traditional Redux
- Docs: https://redux-toolkit.js.org/

INSTRUCTOR Jukka Tupamäki
jukka@lemanse.fi | www.lemanse.fi