# Redux

tsevdos.me / @tsevdos

# Agenda

All the content can be found here.

- FP concepts
- redux principles
- store/state
- actions and action creators
- reducers
- middleware
- selectors

# Rules

Feel free to interrupt me for:

- questions
- relevant comments

# FP concepts

- pure functions
- immutability
- currying
- higher-order functions

# Pure functions

- given the same input, will always return the same output
- produce no side effects (ex. API calls, updating DOM, subscribing to event listeners - anything where you want an "imperative" action to happen)

# Pure functions

```
function add(num1, num2) {
  return num1 + num2;
}

const add = (num1, num2) => num1 + num2;
```

# Pure functions?

```
let total = 5;
function add(num1, num2) {
  const result = total + num1 + num2;
  return result;
}

//----------------
function add(num1, num2) {
  console.log(num1, num2);
  return num1 + num2;
}
```

# Immutability

- once it's created it can’t be changed
- it will have the same properties and values forever
- makes our objects/state/values more predictable
- less bugs

# Currying

Is the process of transforming a function that takes multiple arguments into a series of functions that take one argument at a time.

# Currying

```
// Sum function
const sum = function (a, b) {
  return a + b;
};

// Curried sum function
const curriedSum = function (a) {
  return function (b) {
    return a + b;
  };
};

const result1 = sum(2, 3); // 5
const result2 = curriedSum(2)(3); // 2 + 3 // 5
```

# Currying

```
// Sum function
const sum = (a, b) => a + b;

// Curried sum function
const curriedSum = (a) => (b) => a + b;

const result1 = sum(2, 3); // 5
const result2 = curriedSum(2)(3); // 2 + 3 // 5

const addTwo = curriedSum(2); // 2 + b
const result3 = addTwo(5); // 7
```

# Higher-order function

A higher order function is a function that either takes one or more functions as arguments or returns a function as its result or both.

# Higher-order function

```
/// normal function
function add(x, y) {
  return x + y;
}

// HoF
function higherOrderFunction(x, callback) {
  return callback(x, 5);
}

higherOrderFunction(10, add);
```

# Higher-order function

```
// normal functions
const add = (...values) => values.reduce((a, b) => a + b);
const multiply = (...values) => values.reduce((a, b) => a * b);

// Calculator (higher-order function)
const calculator = (command) => (...args) => command(...args);

const addition = calculator(add);
const multiplication = calculator(multiply);

const total = addition(3, 6, 9, 12, 15, 18); // 63
const otherTotal = multiplication(2, 4, 3); // 24
```

# Redux: History

- created by Dan Abramov as a flux alternative
- inspired by ELM language (FP)

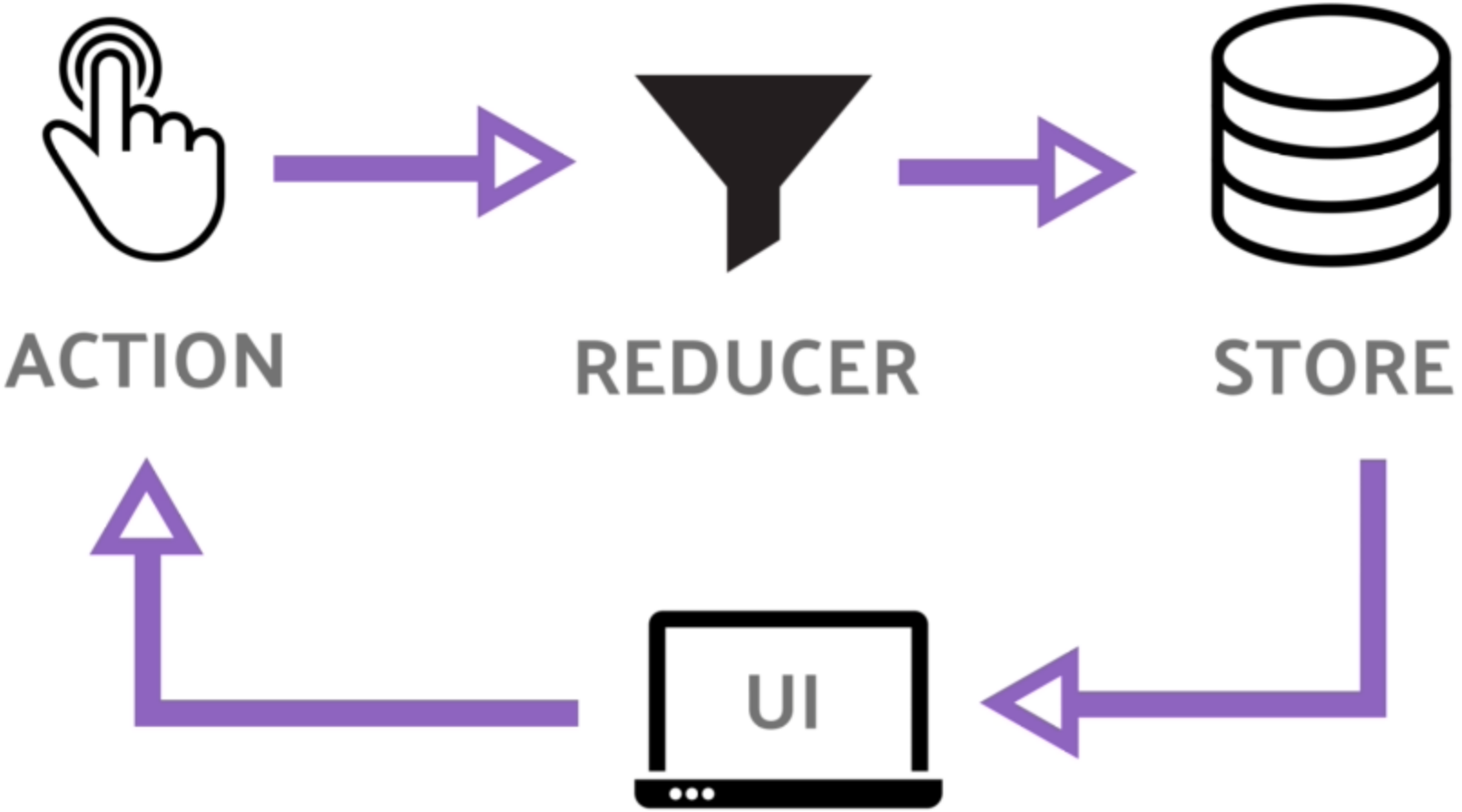
# Redux: Principles

- single source of truth
- state is read-only
- changes are made with pure functions

# Redux: Principles

- the state of your whole application is stored in an object tree within a single store
- the only way to change the state is to emit an action, an object describing what happened
- to specify how the state tree is transformed by actions, you write pure reducers

# Unidirectional data flow



# Redux: Terminology

- actions and action creators
- reducers
- store
- middleware
- selectors

# Actions

- are payloads of information that send data from your application to your store
- they are the only source of information for the store

# Actions

- must contain a type (types should typically be defined as string constants)
- optional payload

# Actions

```
{
  type: "ADD_TODO",
  payload: {
    id: "0",
    title: "learn redux",
    completed: false
  }
}
```

# Action creators

```
const ADD_TODO = "ADD_TODO";

function addTodo(id, title) {
  return {
    type: ADD_TODO,
    payload: {
      id,
      title,
      completed: false,
    },
  };
}
```

# Action creators

```
const ADD_TODO = "ADD_TODO";

const addTodo = (id, title) => ({
  type: ADD_TODO,
  payload: {
    id,
    title,
    completed: false,
  },
});
```

# Reducers

- specify how the application's state changes in response to actions sent to the store
- reducers never modify the state! They always create a new copy with the needed modifications
- root reducer function that will call additional reducer functions to calculate the new state

# Reducers must be pure and NOT:

- mutate its arguments
- perform side effects like API calls and routing transitions
- call non-pure functions, e.g. Date.now() or Math.random()

# Reducers

```
(previousState, action) => newState;
```

# Reducers

```
function rootReducer(state, action) {
  switch (action.type) {
    case "ADD_TODO":
      return [...state, action.payload];
    default:
      return state;
  }
}
```

# Store

- holds application state
- allows access to state via getState()
- allows state to be updated via dispatch(action)
- registers listeners via subscribe(listener)

# Store

```
const store = createStore(
  rootReducer,
  initialState,
  applyMiddleware(...middlewares)
);
```

# What to put in the state:

- data used in different parts of the application
- data used to drive multiple components
- data used to create further derived data

# ...and what to leave out:

- currently selected tab in a tab control on a page
- hover visibility/invisibility on a control
- modal being open/closed
- currently displayed errors

# Redux hooks

- useSelector
- useDispatch

# Redux

Examples 1-4

# Redux exercise 1

(./workshop/src/examples/exercise1)

1. add a "delete" button next to each todo entry
2. add a new "deleteTodo" action (type + action creator)
3. think if this action needs a payload
4. bind the action to the "TodoApp" component
5. pass it to "TodoItem" component
6. handle the deleteTodo action in "TodosReducer"

# Redux middleware

the concept of middleware allows us to add code that will run before the action is passed to the reducer

# Redux middleware signature

```
const logger = (store) => (next) => (action) => {
  // our code!
};
```

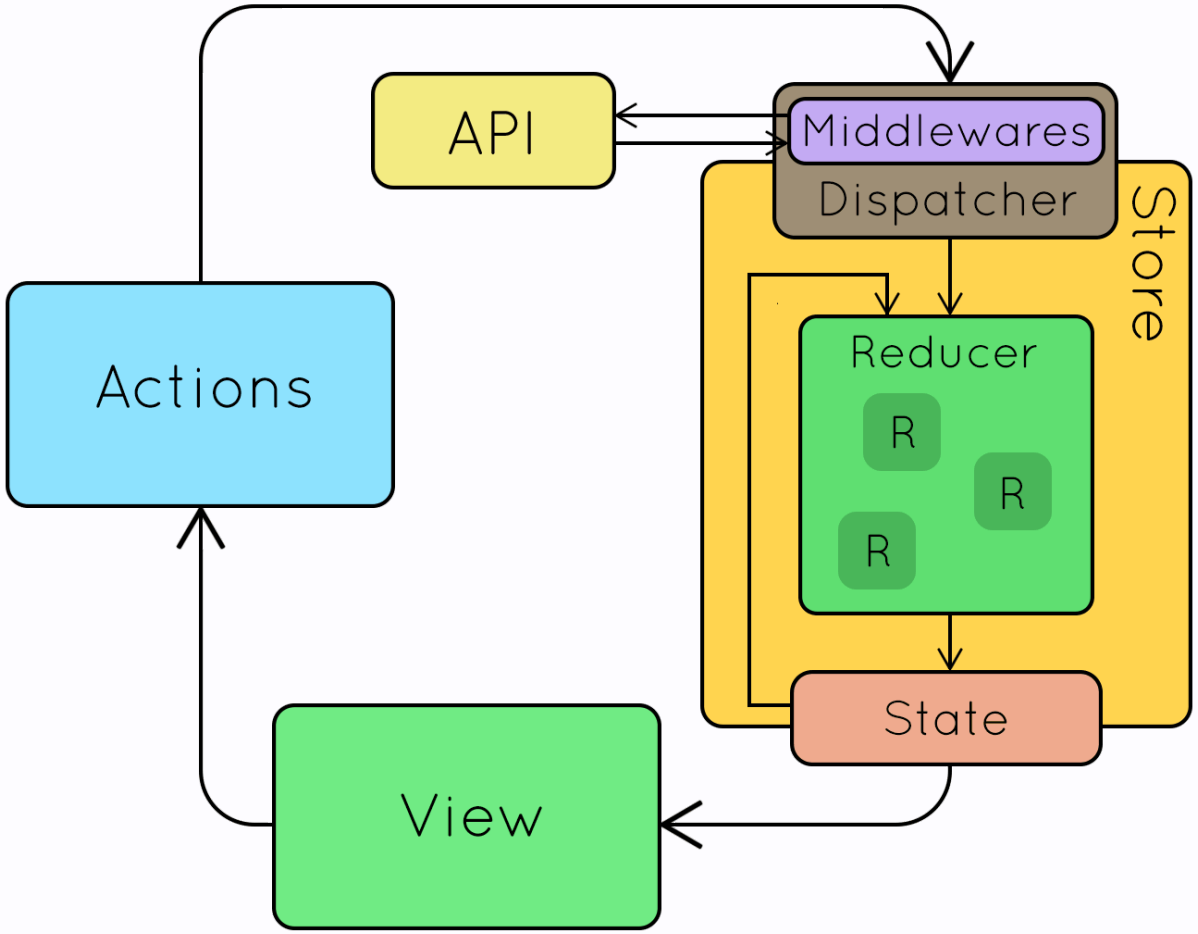
# Redux

Example 5

# Redux middleware

- redux-thunk
- redux-logger

# Redux flow


API
Middlewares
Dispatcher
Store
Actions
Reducer
R
R
R
State
View

# Redux resources

redux.js.org

# That's all folks

## Questions / Discussions?