SWPP Practice Session #4
Introduction to Redux

2022 Sep 28

Announcement

- Project proposal due 9/28 6pm.

Recap on the last practice session

- We covered…
  - Frontend Basics(HTML/CSS/Type(Java)script)
  - React
  - React Router

Today’s Objective

- Redux Basics
  - Flux Design Pattern
- React + Redux
- HTTP Request
  - Axios(Promise Design Pattern)
  - Redux Middleware

Clone Repo

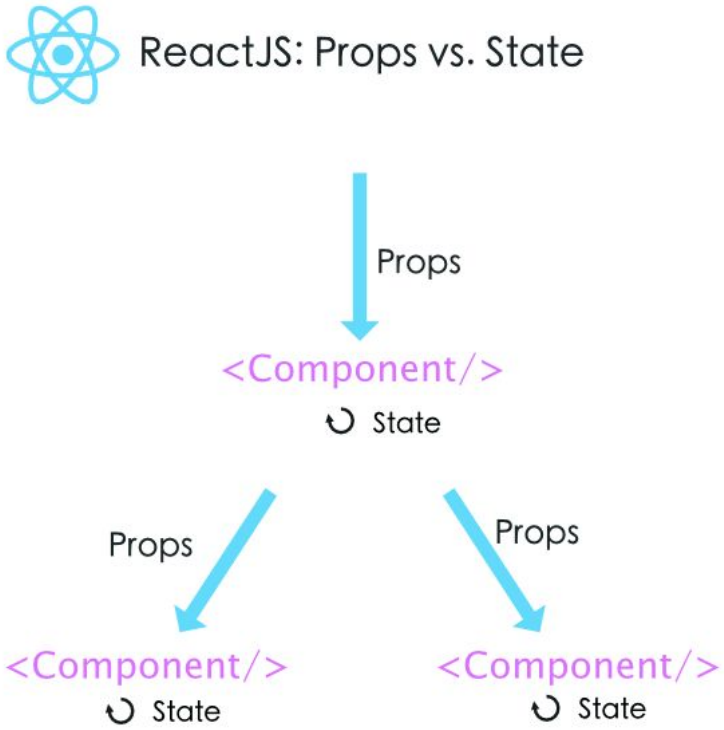
- Fork and clone the repo (We'll be using this regardless of the previous todo project. We've got something new in the repo.)
  - https://github.com/swpp22fall-practice-sessions/swpp-p4-redux-tutorial.git
- We have 4 branches ready. If you're in trouble and can't keep up, you can jump to the following branches.

```
● ~/jong/ta/swpp-p4-redux-tutorial (practice/start ✔) git branch
  practice/1-prepare_slice
  practice/2-react_redux
  practice/3-todo-detail
  practice/4-finish
  practice/redux-example
* practice/start
```

Redux Basics

[Recap from practice session #3]

Two informative objects in React component: Props and State



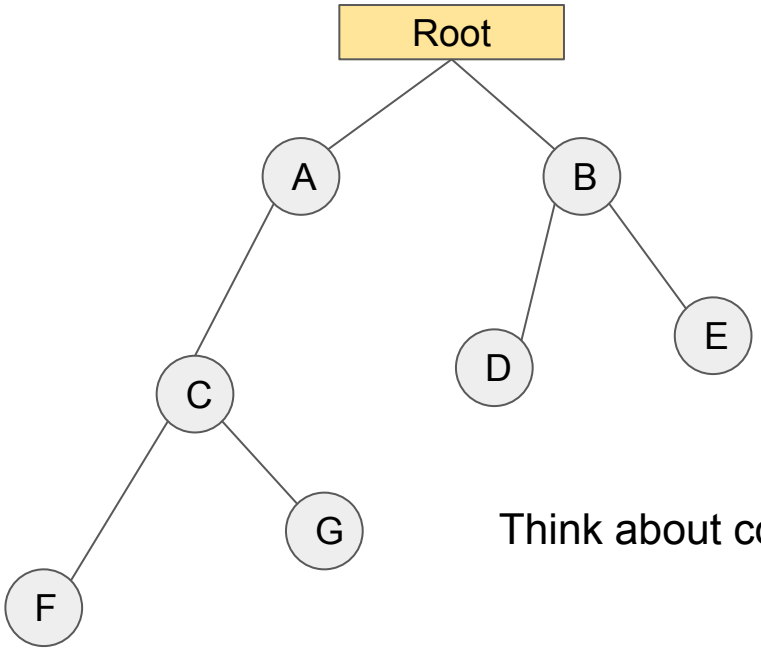
Changing *props* and *state*

-	*props*	*state*
Can get initial value from parent Component?	Yes	Yes
Can be changed by parent Component?	Yes	No
Can set default values inside Component?*	Yes	Yes
Can change inside Component?	No	Yes
Can set initial value for child Components?	Yes	Yes
Can change in child Components?	Yes	No

* Note that both *props* and *state* initial values received from parents override default values defined inside a Component.

Two-way binding is not enough

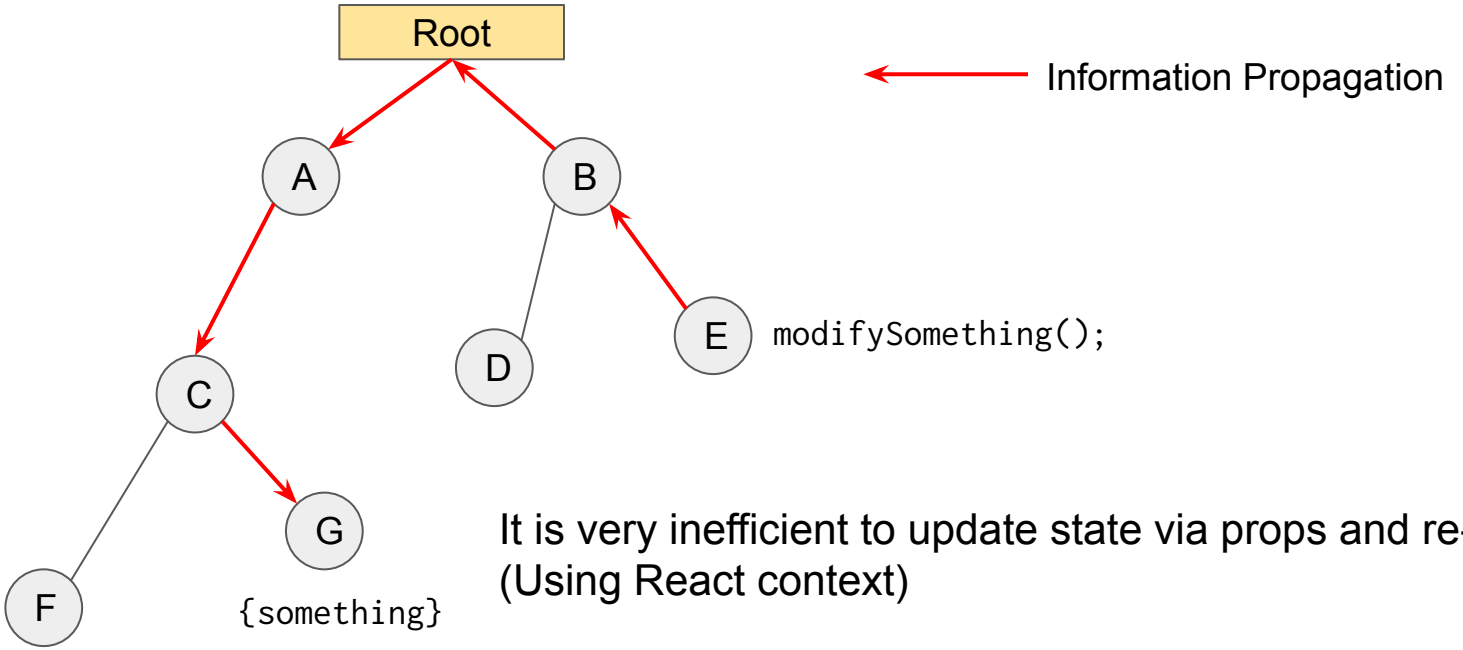
- What if multiple distant components had to share the same state?



Think about component E tries to modify state of component G

Two-way binding is not enough

- What if multiple distant components had to share the same state?



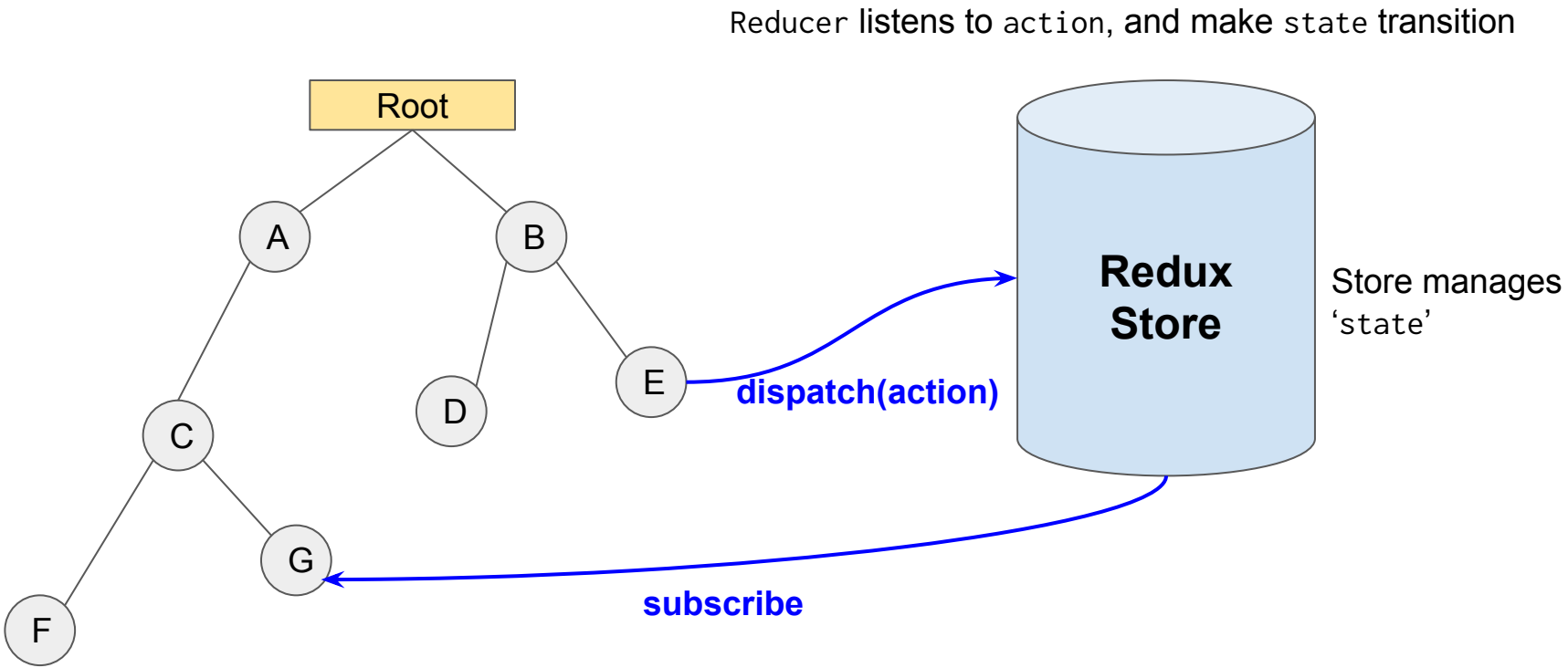
It is very inefficient to update state via props and re-render
(Using React context)

Why this happens?

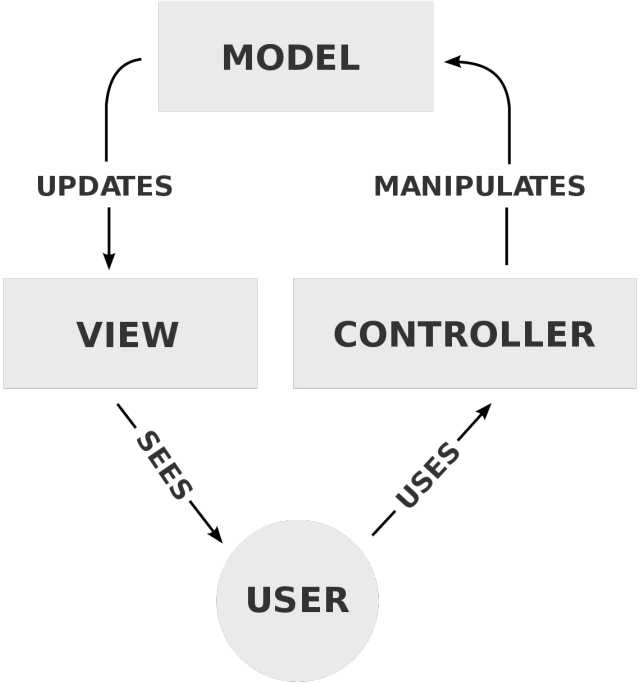
- State can have several scopes:
  - Local state: value of input field, boolean state if todo is checked
  - Page-wise state: Shopping cart items on cart page
  - App-wise state: Name of user currently signed in
- For local state, length of information flow is short.
- Page-wise state or application-wise state should flow through almost a whole component tree because it should be able to be accessed by any component.
- There is a need for storing global state somewhere

What's Redux

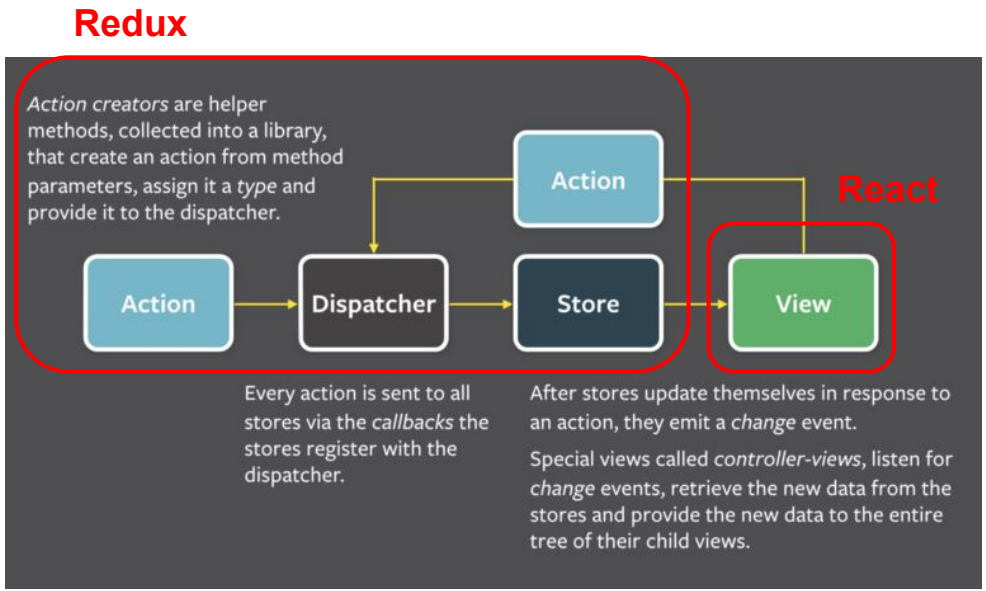
- Redux solves this problem!



MVC & Flux Design Pattern

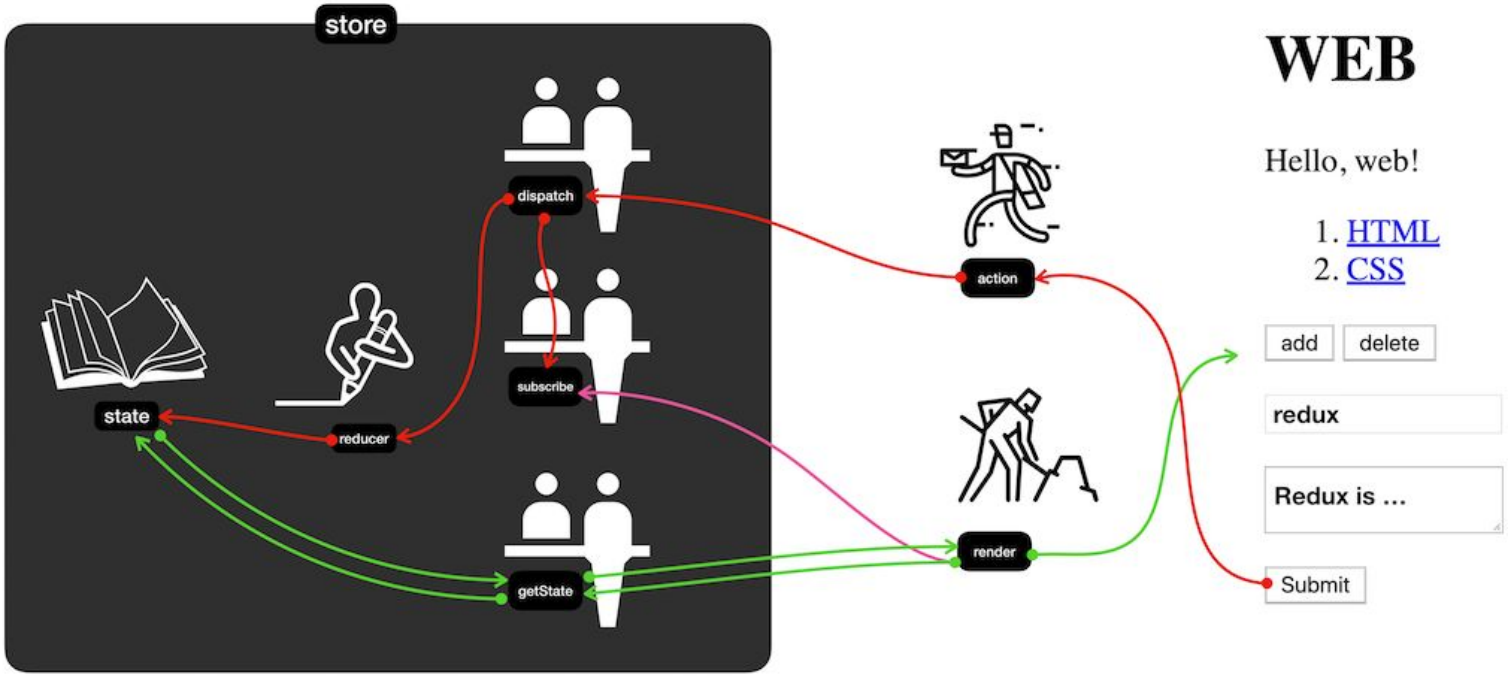


MVC Pattern



Flux Pattern

Redux main concept



Simplest Redux Example

- Let's understand the basic concept of redux
- Create javascript file `redux-basics.js`
- `# install package`
- `$ yarn add redux react-redux`

```
const { configureStore } = require('@reduxjs/toolkit');  // load module in Node.js
const initialState = { number: 0 }; // default state
// create identity reducer
const reducer = (state = initialState, action) => {
  return state;
}

// create redux store
const store = configureStore({ reducer: reducer });
console.log(store.getState());
```

```
# run
$ node redux-basics.js
{ number: 0 }
$
```

Simplest Redux Example

- Dispatch actions
  - Must set **type**, with some optional payloads
  - type naming convention == uppercase letter

```
store.dispatch({ type: 'ADD' });
store.dispatch({ type: 'ADD_VALUE', value: 5 });
console.log(store.getState());
```

- Now add behavior in reducer

```
const reducer = (state = initialState, action) => {
  if (action.type === 'ADD') { return ({ ...state, number: state.number + 1}); }
  else if (action.type === 'ADD_VALUE') {
    return ({...state, number: state.number + action.value});
  }
  return state;
}
```

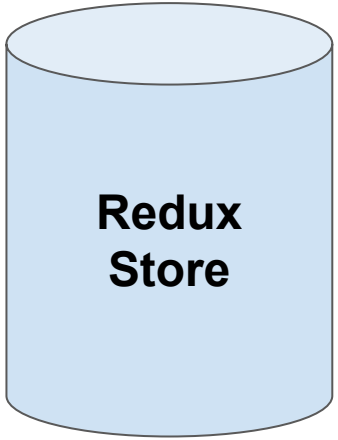
Simplest Redux Example

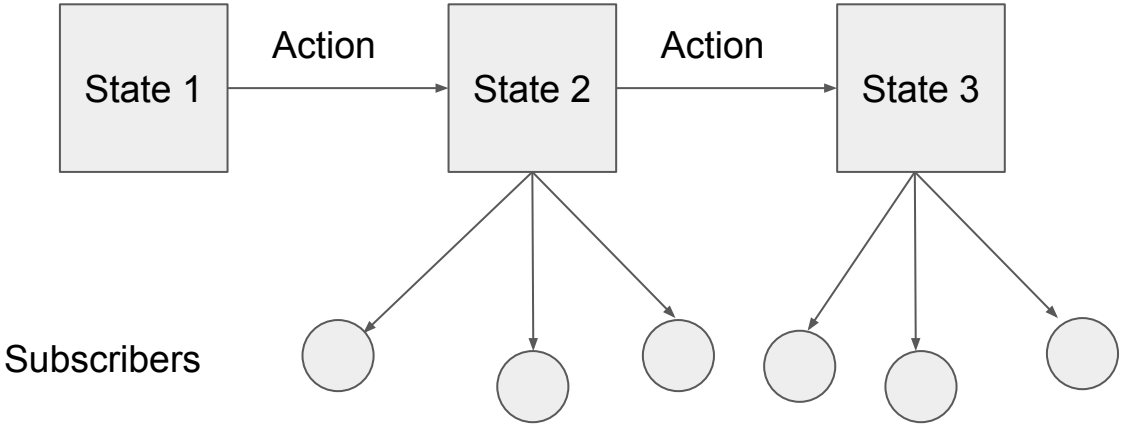
- # now we can make our number state changed
- $ node redux-basics.js
- { number: 6 }
- Let's subscribe store

```
store.subscribe(() => {
  console.log('[Subscription]', store.getState());
});
```

- Then it prints…
- $ node redux-basics.js
- [Subscription] { number : 1 }
- [Subscription] { number : 6 }
- { number: 6 }

Redux is a kind of state machine





A Redux store informs subscribers that state has changed by callback.

Redux in React

The `src` folder (reference)

- Recommended structure
- `src`
  - `containers`
    - Stateful components
  - `components`
    - Render related components
  - `store`
    - all store-related items

React + Redux

- Let’s integrate redux with our TODO app
- First of all, make store in src/store/index.ts
- “store” contains our global state.

```
// src/store/index.ts
import { configureStore } from "@reduxjs/toolkit";

export const store = configureStore(
  { reducer: (state = {}, action) => state }
); // TODO

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

React + Redux

- Now wrap our `App` with `Provider`
  - Provider makes the Redux `store` available to any nested components.

```
import { Provider } from "react-redux";
import { store } from "./store";
...
root.render(
 <React.StrictMode>
   <Provider store={store}><App /></Provider>
 </React.StrictMode>
);
```

Prepare slice

- To manage large root state, we separate into functional scope slice.
- `@reduxjs/toolkit` help us implement slice easily.
- Reducer is a function that takes in a state and an action as an argument
- Reducer is responsible for changing the current state depending on the specific action.

```
// src/store/slices/todo.ts
import { createSlice, PayloadAction } from "@reduxjs/toolkit";
import { RootState } from "..";
import { Root } from "@reduxjs/toolkit";
export interface TodoType { id: number; title: string; content: string; done: boolean; }
export interface TodoState { todos: TodoType[]; selectedTodo: TodoType | null; }
const initialState: TodoState = {
 todos: [
   { id: 1, title: "SWPP", content: "take swpp class", done: true },
   { id: 2, title: "Movie", content: "watch movie", done: false },
   { id: 3, title: "Dinner", content: "eat dinner", done: false },
 ], selectedTodo: null,
}; // continue
```

Prepare slice

```
...
export const todoSlice = createSlice({
 name: "todo",
 initialState,
 reducers: {
   getAll: (state, action: PayloadAction<{ todos: TodoType[] }>) => {},
   getTodo: (state, action: PayloadAction<{ targetId: number }>) => {},
   toggleDone: (state, action: PayloadAction<{ targetId: number }>) => {},
   deleteTodo: (state, action: PayloadAction<{ targetId: number }>) => {},
   addTodo: (state, action: PayloadAction<{ title: string; content: string }>) => {},
 },
});

export const todoActions = todoSlice.actions;
export const selectTodo = (state: RootState) => state.todo;


export default todoSlice.reducer;
```

Implement addTodo

- `addTodo` function in reducers will make a new Todo using title and content in payload, and put it todos

```
// src/store/slices/todo.ts
   addTodo: (
     state,
     action: PayloadAction<{ title: string; content: string }>
   ) => {
     const newTodo = {
       id: state.todos[state.todos.length - 1].id + 1, // temporary
       title: action.payload.title,
       content: action.payload.content,
       done: false,
     };
     state.todos.push(newTodo);
   },
```

Store combined with reducer

- Return to `store/index.ts`

```
import { configureStore } from "@reduxjs/toolkit";
import todoReducer from "./slices/todo";

export const store = configureStore({
 reducer: {
   todo: todoReducer,
 },
});

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

useSelector & useDispatch

- useSelector
  - Allows you to extract data from the Redux store state, using a selector function.
  - Subscribe

- useDispatch
  - This hook returns a reference to the dispatch function from the Redux store.
  - You may use it to dispatch actions as needed.
  - Publish

Connect React Component to Redux Store

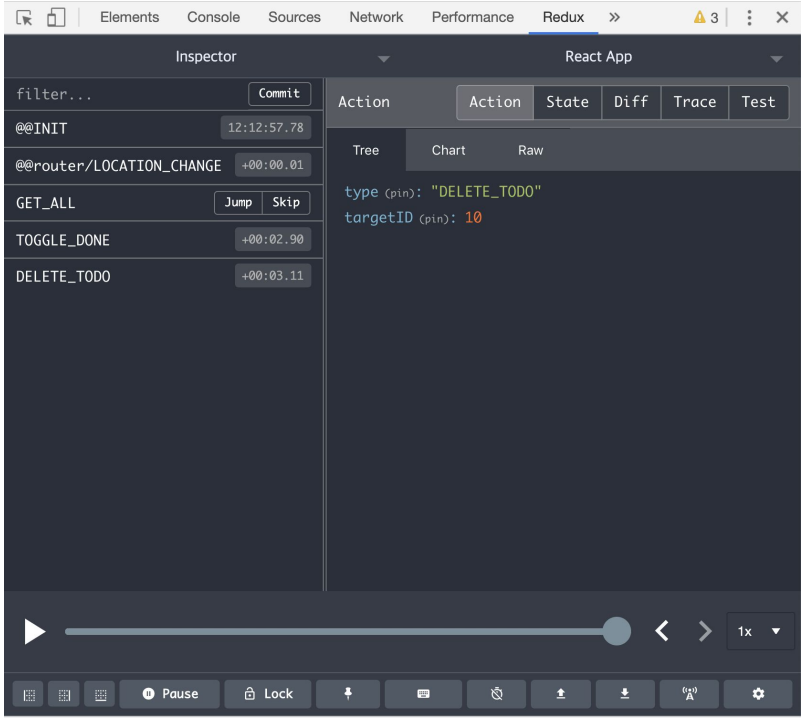
- Let’s go back to our NewTodo.tsx component
- Publish addTodo action when postTodoHandler is called

```
import { useDispatch } from "react-redux";
import { todoActions } from "../../../store/slices/todo";
...
export default function NewTodo() {
...
 const dispatch = useDispatch()
...
 const postTodoHandler = () => {
   const data = { title: title, content: content };
   dispatch(todoActions.addTodo(data))
   setSubmitted(true);
 };
...
};
```

ProTip: Redux Chrome Extension

- Now you can monitor your state transitions!
- Use this extension along with React extension we had installed in the last session.

https://chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklioeibfkpmmfibljd



At TodoList Component

```
import { useDispatch, useSelector } from "react-redux";
import { selectTodo, todoActions } from "../../store/slices/todo";

// Inside function
  const todoState = useSelector(selectTodo);
  const dispatch = useDispatch();
  ...
```

- Now use `todoState.todos` as our todo items

```
return (
  ...
  {todoState.todos.map((td) => {
);
```

- It now works!
- We can remove [todo, setTodo] with `useState`

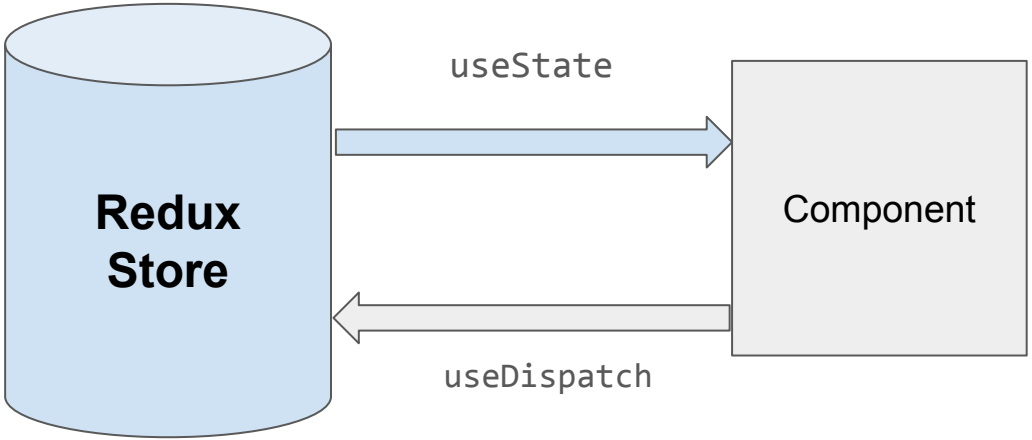
Redux State vs Component State

- We manage `todos` as redux state
- We manage `title, content, submitted` (NewTodo) as component state

Type	Example	Redux? Component?
Local UI State	Show / Hide / Input …	Mostly within component
Persistent State	All Todos, All Users…	Stored on Server Managed by Redux
Client State	Authenticated?	Managed by Redux

Recap: Two-way binding

- `useState` and `useDispatch` can be thought of as two-way binding between a component connected to store and the store itself.



Add Other Actions

- At `TodoList.tsx` component, pass callbacks to Todo

```
{todoState.todos.map((td) => {
  return (
    <Todo
      title={td.title}
      done={td.done}
      clickDetail={() => clickTodoHandler(td)}
      clickDone={() => dispatch(todoActions.toggleDone({targetId: td.id}))}
      clickDelete={() => dispatch(todoActions.deleteTodo({targetId: td.id}))}
    />
  );
})}
```

Add Buttons in Todo Component

- At Todo.tsx, change IProps to match type

```
interface IProps {
 title: string;
 clickDetail?: React.MouseEventHandler<HTMLDivElement>; // Defined by React
 clickDone?: () => void;
 clickDelete?: () => void;
 done: boolean;
}
```

Add Buttons in Todo Component

- At Todo.tsx, change return() to add button and connect callbacks.
- Now you can see action is published when you click button in redux devtools

```
const Todo = (props: IProps) => {
 return (
   <div className="Todo">
     <div className={`text ${props.done && "done"}`} onClick={props.clickDetail} >
       {props.title}
     </div>
     {props.done && <div className="done-mark">&#x2713;</div>}
     <button onClick={props.clickDone}>{(props.done) ? 'Undone' : 'Done'}</button>
     <button onClick={props.clickDelete}>Delete</button>
   </div>
 );
};
```

```
todo/toggleDone
▼ action: {} 1 key
  ▶ payload: {} 1 key
▼ state: {} 1 key
  ▶ todo: {} 2 keys
todo/deleteTodo
▼ action: {} 1 key
  ▶ payload: {} 1 key
▼ state: {} 1 key
  ▶ todo: {} 2 keys
```

`toggleDone` & `deleteTodo`

- Now, these actions should be handled in store/slices/todo.ts

```
toggleDone: (state, action: PayloadAction<{ targetId: number }>) => {
  const todo = state.todos.find(
    (value) => value.id === action.payload.targetId
  );
  if (todo) {
    todo.done = !todo.done;
  }
},
deleteTodo: (state, action: PayloadAction<{ targetId: number }>) => {
  const deleted = state.todos.filter((todo) => {
    return todo.id !== action.payload.targetId;
  });
  state.todos = deleted;
},
```

Modify `TodoDetail` as page

- Now, we will modify `TodoDetail` as page
- Modify `clickTodoHandler` in `TodoList.tsx` as

```
import { NavLink, useNavigate } from "react-router-dom";
...

 const navigate = useNavigate()

 const clickTodoHandler = (td: TodoType) => {
   navigate('/todos/' + td.id)
 };
```

Modify `TodoDetail` as page

- Now you will be at separated page when you click one of the todos
- Due to
  - `<Route path='/todos/:id' component={RealDetail} />`
- Now implement getTodo reducer in store/slices/todo.ts

```
getTodo: (state, action: PayloadAction<{ targetId: number }>) => {
  const target = state.todos.find((td) => td.id === action.payload.targetId);
  state.selectedTodo = target ?? null
},
```

Modify `TodoDetail` as page

- Import required

```
import { useEffect } from "react";
import { useDispatch, useSelector } from "react-redux";
import { useParams } from "react-router";
import { selectTodo, todoActions } from "../../store/slices/todo";
```

Modify `TodoDetail` as page

- Delete Props and get url parameter using `useParams`
  - You can get url parameter using useParams
  - url parameter is parameter in path.
  - When path is ‘todo/:id’ and real url is ‘todo/4’, parameter id will be ‘4’ (**string**)
- Get Selected todo using useEffect
  - useEffect will run whenever any dependency array item is changed

```
const { id } = useParams();
const dispatch = useDispatch();
const todoState = useSelector(selectTodo);
useEffect(() => {
  dispatch(todoActions.getTodo({ targetId: Number(id) }));
}, [dispatch, id]);
```

Modify `TodoDetail` as page

- Use `Optional Chaining` for null check
  - to prevent trying to get null's attribute error, use optional chaining

```
return (
      ...
      <div className="right">{todoState.selectedTodo?.title}</div>
      ...
      <div className="right">{todoState.selectedTodo?.content}</div>
      ...
);
```

Redux and Immutability - legacy

- Redux state has to be *immutable*. That is, you should not directly modify its content.
- You should do it in “constructive” manner, not with in-place operation.
- **@redux/toolkit will handle this.**

BAD

```
const prime = [2, 3];

const reducer = (state = {prime: prime},
action) => {
  if (action.type === ‘ADD_PRIME’) {
    state.prime.push(action.value);
    return state;
  }
  return state;
};
```

GOOD

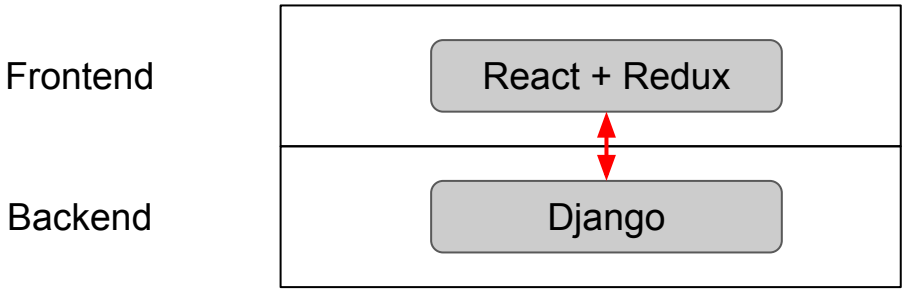
```
const prime = [2, 3];

const reducer = (state = {prime: prime}, action) => {
  if (action.type === ‘ADD_PRIME’) {
    return {prime: [...state.prime, action.value]};
  }
  return state;
};
```

HTTP Request

Our App’s Problem

- Refresh your app right after making / deleting many TODOs
- Our modifications are gone when we refresh page!
- We want to store them in **server**
  - Let’s connect our store to server!

HTTP Communication Overview



- Service will communicate with the backend that we have prepared for the session.
- Using HTTP, we will **C**reate, **R**ead, **U**pdate, and **D**elete our TODOs.
  We call this **CRUD**

Prepare (Server)

- Modify `package.json`

```
{
  "name": "todo",
  "version": "0.1.0",
  "private": false,
  "proxy": "http://127.0.0.1:8000",
  "dependencies": {
      ...
  }
      ...
}
```

- Run django

```
$ pip install django==4.1.1
$ cd backend
$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, todo
Running migrations:
  Applying auth.0012_alter_user_first_name_max_length... OK

$ python manage.py runserver &
System check identified 1 issue (0 silenced).
September 15, 2022 - 14:59:29
Django version 3.2.6, using settings 'todo_backend.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Prepare (Package)

- # download and install package
- $ yarn add axios
- # now you can use as

```
import axios from 'axios';
```

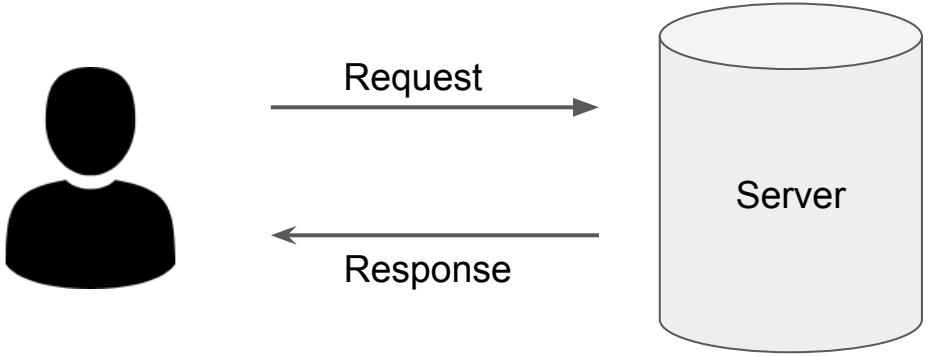
What’s AXIOS

- Promise based HTTP client for node.js
- Features
  - Make http requests from node.js
  - Supports the Promise API
- Example

```
const axios = require(‘axios’);

axios.get('/api/todos')
   .then(response => console.log(response))
   .catch(error => console.log(error));
```

Promise Pattern

- Promise is an object that may produce a single value sometime in the future
- Using when handling async
- Useful in client - server communication
  - We cannot claim that server sends response directly

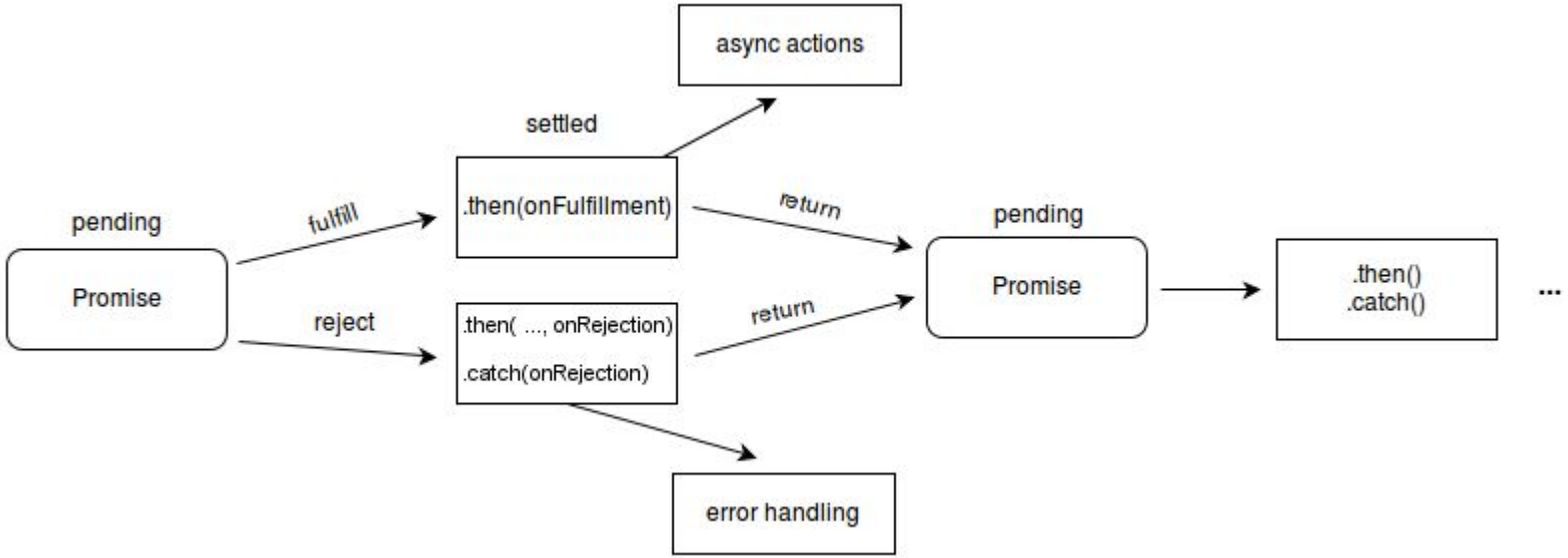


Promise Pattern

- Simple example

```
const sayHello = () => {
  const number = Math.random();
  return (new Promise<string>((resolve, reject) => {
    if (number > 0.5) { resolve(“Hello”); }
    else { reject(“Bye”); }
  }));
};

sayHello()
  .then(res => console.log(res))
  .catch(err => console.log(err));
```

Promise Pattern



Async Function

- Promise is redundant and Hard to read
- Async-Await is accepted in javascript;

```
const sayHelloAwait = () => {
 const number = Math.random();
 return new Promise((resolve, reject) => {
   if (number > 0.5) { resolve("Hello");}
   else { reject("Bye"); }
 });
};
const main = async () => {
 const result = await sayHelloAwait();
 console.log(result);
};
main();
```

- “await” has to exist in async function
- async is same with Promise

new Promise(res => res(“hi”))
async () => {return “hi”}

Simple Test

- inside `TodoList.tsx` add

```
useEffect(() => {
  axios.get('/api/todo')
      .then(result => console.log(result));
})
```

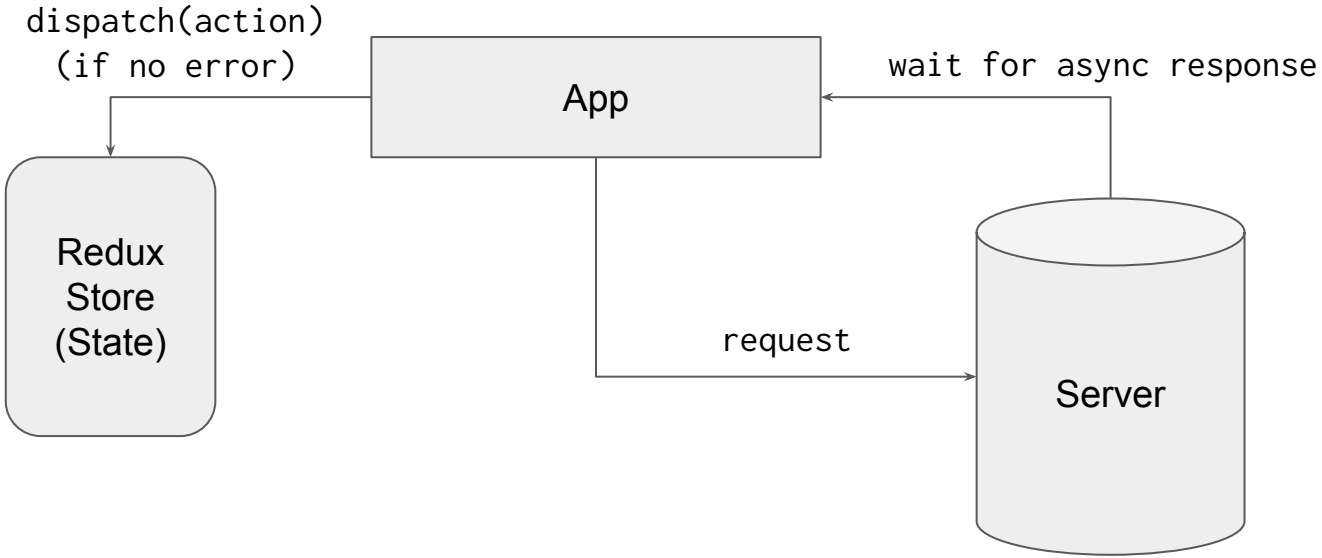
asynchronous

- We can catch error via

```
useEffect(() => {
  axios.get('/api/todoerror')
      .then(result => console.log(result))
      .catch(err => console.log(err));
})
```

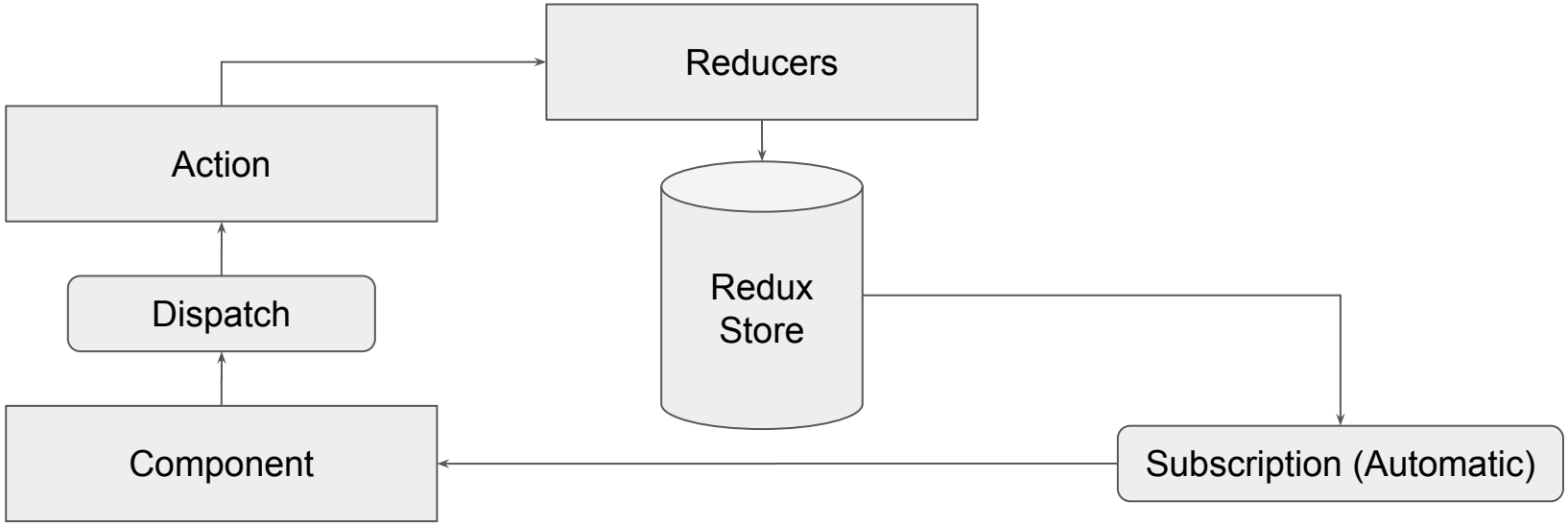
Integrate With Redux

- What we want to do is …



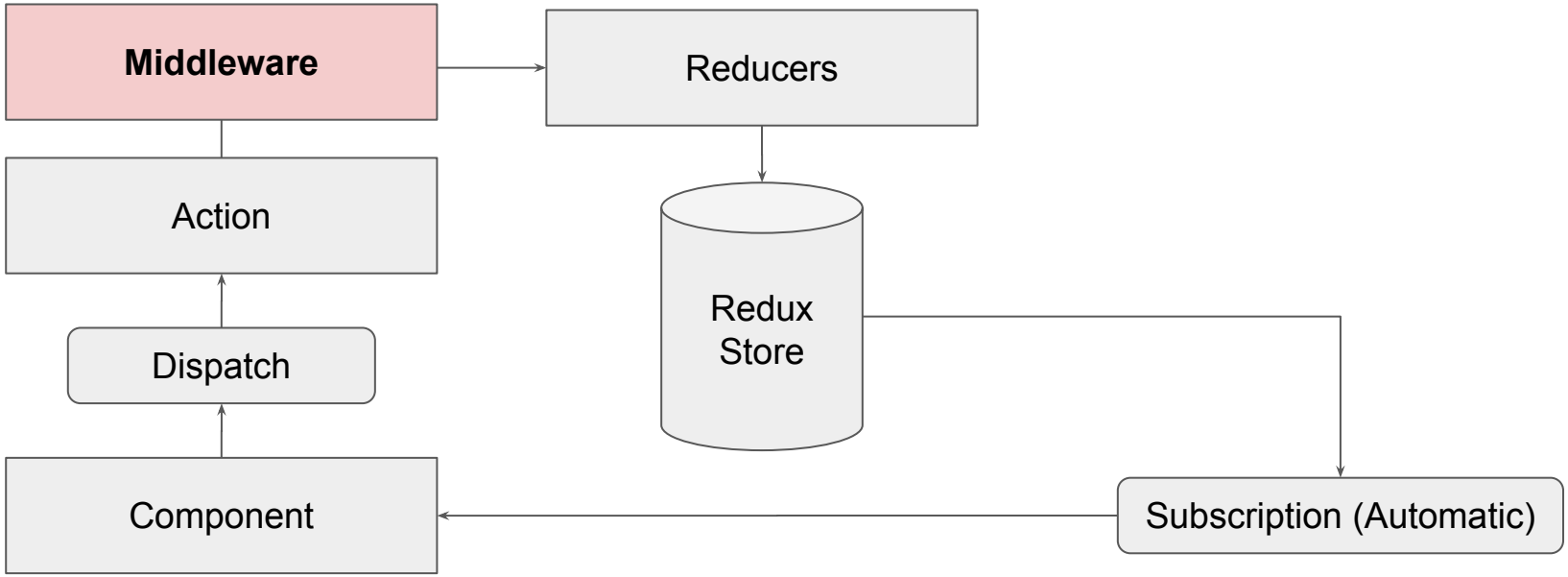
Redux Middleware

- We want to communicate and get some data from the server before or after action dispatched
- We want to dispatch different actions depending on the response of server

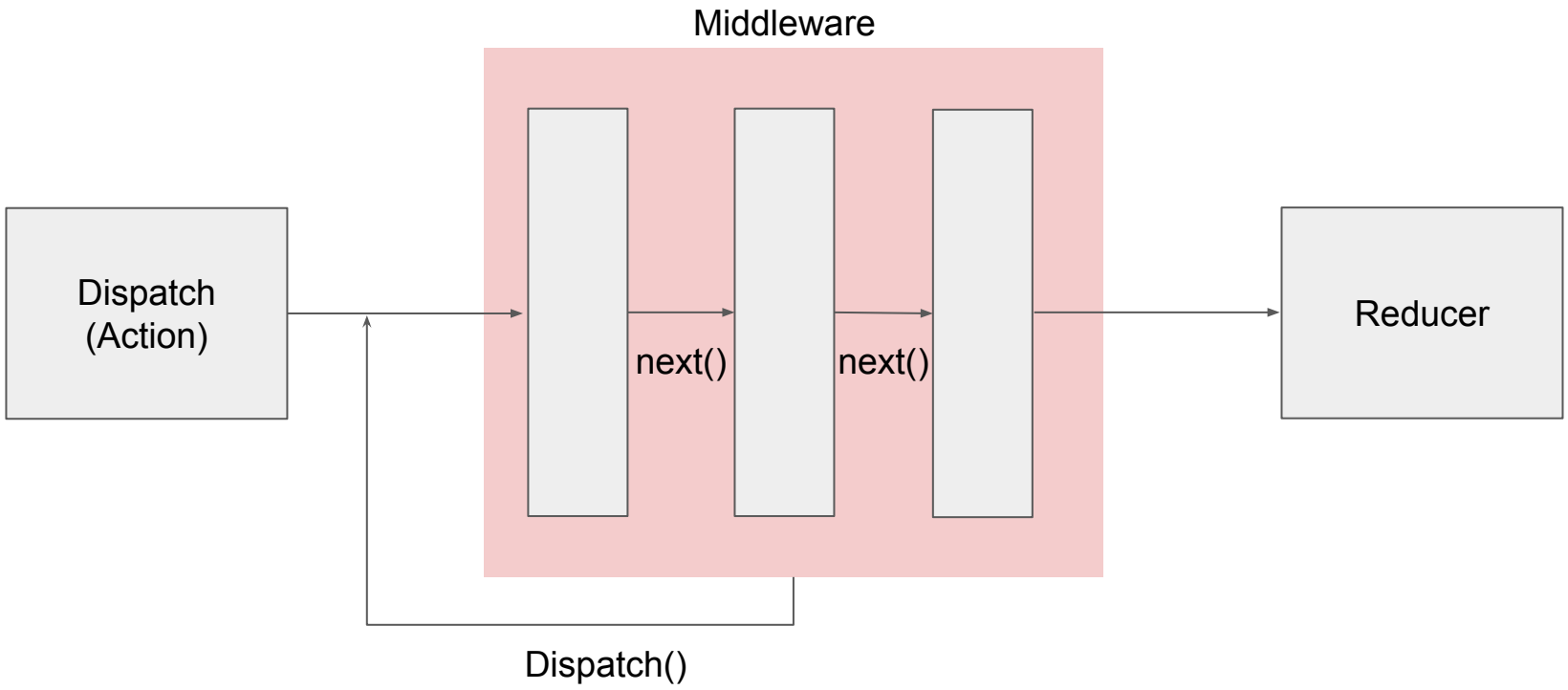


Redux Middleware

- Middleware provides an extension between dispatching an action, and the moment it reaches the reducer



Redux Middleware



Axios in Middleware

- Redux-Thunk enables insert an asynchronous job in the middle of component-store chain.
- Redux-Thunk is included in @redux/toolkit

Axios in Middleware

```
// Add to store/slices/todo.ts
...
export const fetchTodos = createAsyncThunk(
  "todo/fetchTodos",
  async () => {
    const response = await
axios.get<TodoType[]>("/api/todo/");
    return response.data;
  }
);
```

```
export const todoSlice = createSlice({
...
 reducers: {
  ...
  extraReducers: (builder) => {
  // Add reducers for additional action types
here, and handle loading state as needed
    builder.addCase(fetchTodos.fulfilled, (state,
action) => {
     // Add user to the state array
     state.todos = action.payload
   })
 }
});
```

Axios in Middleware

- Update useEffect in `TodoList` Component

```
import { fetchTodos, selectTodo, todoActions } from "../../store/slices/todo";

export default function TodoList(props: IProps) {
  const dispatch = useDispatch<AppDispatch>();

  useEffect(() => {
    dispatch(fetchTodos())
  // eslint-disable-next-line react-hooks/exhaustive-deps
  },[]);
...
```

Axios in Middleware

- Post new todo

```
// store/slices/todo.ts
export const postTodo = createAsyncThunk(
 "todo/postTodo",
 async (td: Pick<TodoType, "title" | "content">, { dispatch }) => {
   const response = await axios.post("/api/todo/", td);
   dispatch(todoActions.addTodo(response.data));
 }
);
```

```
// NewTodo.tsx
import { AppDispatch } from "../../../store";
import { postTodo } from "../../../store/slices/todo";


...
 const postTodoHandler = () => {
   const data = { title: title, content: content };
   dispatch(postTodo(data));
   setSubmitted(true);
 };
...
```

Axios in Middleware (example)

- Error handling included. (optional)

```
// NewTodo.tsx
 const postTodoHandler = async () => {
   const data = { title: title, content: content };
   const result = await dispatch(postTodo(data));
   if (result.payload) {
     setSubmitted(true);
   } else {
     alert("Error on post Todo");
   }
 };
```

```
// slices/todo.ts
 extraReducers: (builder) => {
   builder.addCase(fetchTodos.fulfilled, (state, action) => {
     state.todos = action.payload;
   });
   builder.addCase(postTodo.rejected, (_state, action) =>{
     console.error(action.error);
   })
 },
```

Axios in Middleware

- Delete Todo

```
// store/slices/todo.js
export const deleteTodo = createAsyncThunk(
 "todo/deleteTodo",
 async (id: TodoType["id"], { dispatch }) => {
   await axios.delete(`/api/todo/${id}/`);
   dispatch(todoActions.deleteTodo({ targetId: id }));
 }
);
```

```
// TodoList.js
import {
 ...
 deleteTodo,
} from "../../store/slices/todo";

// Update <Todo /> in return
   clickDelete={() => dispatch(deleteTodo(td.id))}
```

Axios in Middleware

- Toggle Todo

```
// store/slices/todo.js
export const toggleDone = createAsyncThunk(
 "todo/toggleDone",
 async (id: TodoType["id"], { dispatch }) => {
   await axios.put(`/api/todo/${id}/`);
   dispatch(todoActions.toggleDone({ targetId: id }));
 }
);
```

```
// TodoList.js
import {
 ...
 toggleDone,
} from "../../store/slices/todo";

// Update <Todo /> in return
   clickDone={() => dispatch(toggleDone(td.id))}
```

Axios in Middleware

- Get Single Todo (in `TodoDetail` Component)

```
// store/slices/todo.js
export const fetchTodo = createAsyncThunk( "todo/fetchTodo",
 async (id: TodoType["id"], { dispatch }) => {
   const response = await axios.get(`/api/todo/${id}/`);
   return response.data ?? null;
 }
);
...
 extraReducers: (builder) => {
   builder.addCase(fetchTodo.fulfilled,
     (state, action) => {
     state.selectedTodo = action.payload;
   });
 },
...
```

```
// TodoDetail.js
import { AppDispatch } from "../../store";
import { selectTodo, fetchTodo } from "../../store/slices/todo";

 // update useEffect
 useEffect(() => {
   dispatch(fetchTodo(Number(id)));
   // eslint-disable-next-line react-hooks/exhaustive-deps
 }, [id]);
```

Wrap up(1/2)

- A Redux store is one single store in which your *global state* is stored.
  - `dispatch`
  - `slice`
  - `createAsyncThunk`
- Reducer receives an action and then returns a new state according to the action's type. You shouldn't modify received state *in-place*!
- For communication with server, we can utilize Axios package in HTTPRequest
  - Promise Pattern
- You can make `dispatch(..)` accept an asynchronous code with `redux-thunk`.

Wrap up(2/2)

- Like the last practice session, there is no additional exercise.
- Please make a Pull Request of your works to the original repo.
  - into `swpp-p4-redux-tutorial:master` from `{yours}:main`
  - until tomorrow 9PM for your attendance check.

Next Week

- Frontend Code testing
  - Basic Concept of testing
  - frontend testing frameworks (Jest + Testing Library)

Q & A

Useful References

- https://redux.js.org/
- https://www.freecodecamp.org/news/an-introduction-to-the-flux-architectural-pattern-674ea74775c9/
- https://medium.com/@madasamy/flux-vs-mvc-design-pattern-de134dfaa12b
- https://blog.isquaredsoftware.com/2016/10/idiomatic-redux-why-use-action-creators/
- https://medium.com/dailyjs/asynchronous-adventures-in-javascript-promises-1e0da27a3b4
- https://velopert.com/3401
- https://stackoverflow.com/questions/35411423/how-to-dispatch-a-redux-action-with-a-timeout/35415559#35415559
- https://opentutorials.org/module/4078/24935
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Optional_chaining