

WHAT I LEARNED DURING THE  
PAST MONTH

# AGENDA

- Overview of React
- React-Redux
- React Loadable
- Caching using Service Worker
- Server Side Rendering in React
- React-Router

# REACT OVERVIEW

- React → makes creating interactive UI painless, by building components that have states and props

```
import React from 'react';
import ReactDOM from 'react-dom';
import Clock from './Clock';

ReactDOM.render(
  <Clock name="Dito"/>,
  document.getElementById('root')
);
```

index.js

```
import React, { Component } from 'react';

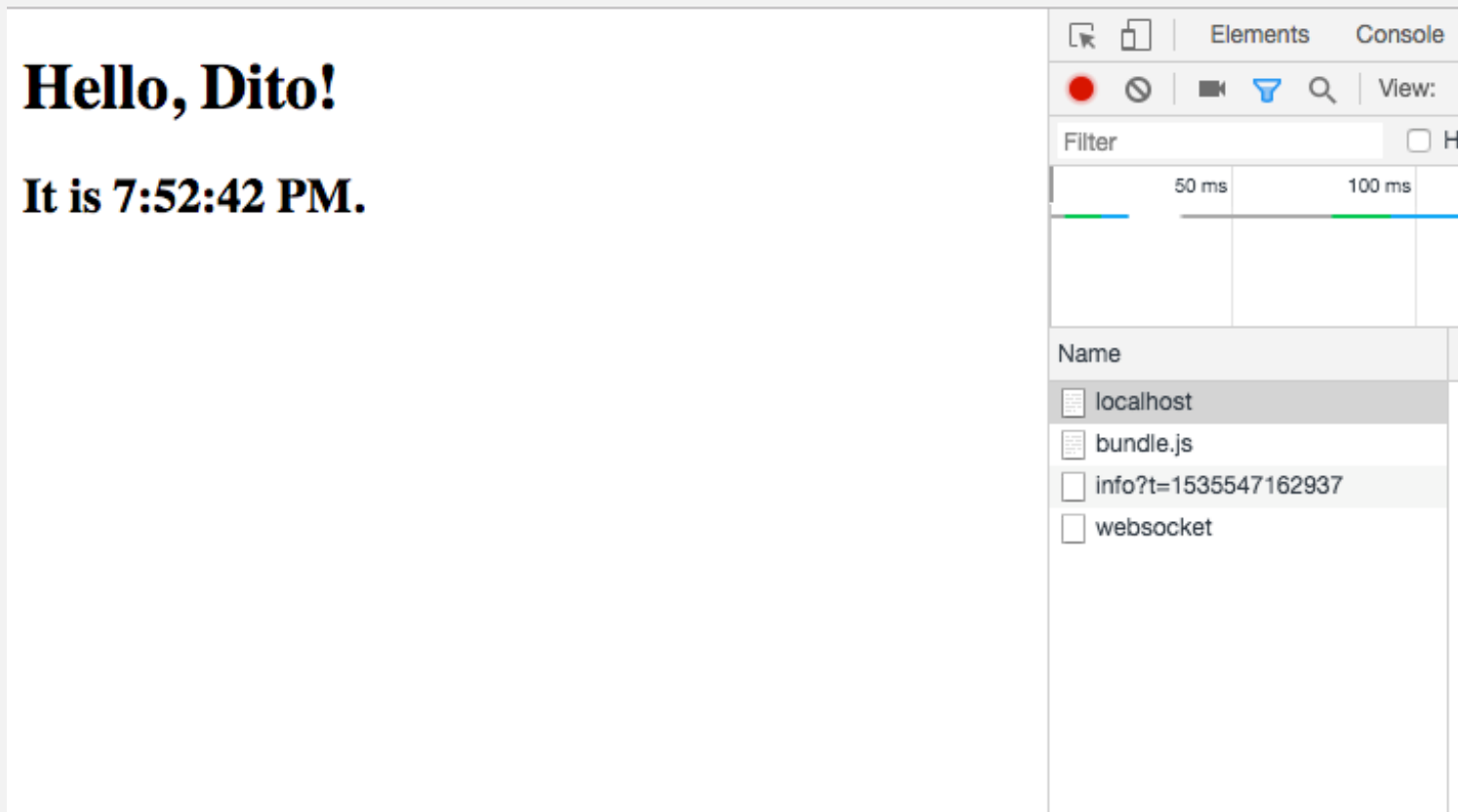
class Clock extends Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  render() {
    return (
      <div>
        <h1>Hello, {this.props.name}!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}

export default Clock;
```

Clock.js

For further reference: <https://reactjs.org/>

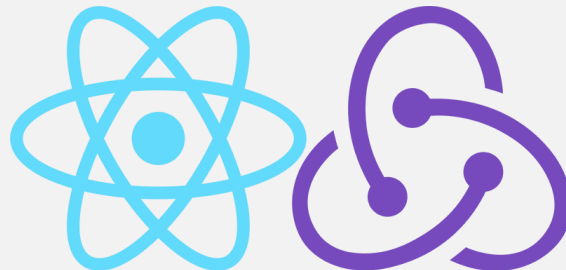


This example can be found on:

<https://github.com/albertusandito-olx/simple-react-app>

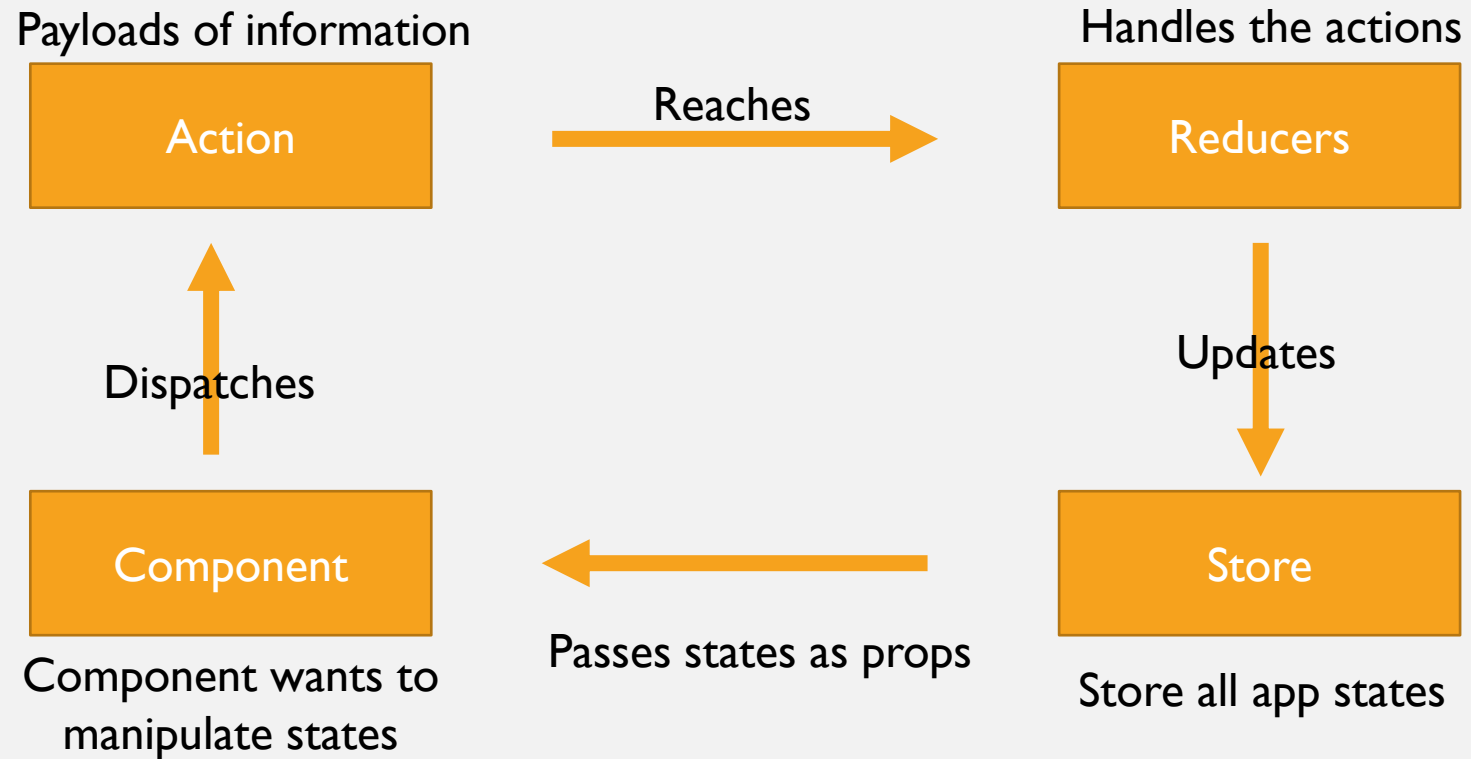
# REACT-REDUX

- React app contains many states → **messy and difficult to manage**
- + With Redux, **state management is easy!**
- Only need one STORE to keep all of the states
- Redux uses ACTIONS to update the states, and REDUCER to handle the actions and connect to the store



For further reference: <https://redux.js.org/basics>

# REDUX FLOWCHART



# REACT LOADABLE

- React app's bundle is often so big, especially *Single Page Application* (SPA) → slow things down
- React-Loadable code-splits the app into several chunks.
- By using dynamic import, it only loads the components when needed.
- + Advantage? Faster! Lighten up the initial load!

**REACT LOADABLE**

ROUTE-CENTRIC CODE SPLITTING IS SHIT. COMPONENT-CENTRIC SPLITTING IS COOL AS SHIT

- Ken Wheeler... probably

For further reference: <https://github.com/jamiebuilds/react-loadable>

# CACHING USING SERVICE WORKER

- Caching → Providing offline support!
- Caching is all done in the Service Worker
- Files are cached in the Cache Storage in Browser
- What to cache? App Shell!
- Multiple strategies to cache: I used network with cache fallback

For further reference: <https://developers.google.com/web/ilt/pwa/caching-files-with-service-worker>



# REACT SERVER-SIDE RENDERING

- Normal React: send blank HTML to client → render on client
- React SSR: render on server → send full HTML to client
- Using Node and Express
- Pros:
  - + Could improve the SEO
  - + Displaying content to user faster
- Cons:
  - - More work for the server
  - - Can't be used with other languages other than Node (with Babel transpiler)

For further reference: <https://naspers.udemy.com/server-side-rendering-with-react-and-redux/learn/v4/content>

## GITHUB REPO LINK

The following example can be found on:

<https://github.com/albertusandito-olx/react-ssr-example>

# NORMAL REACT

**Hello, Dito!**

**It is 7:52:42 PM.**

The screenshot shows the Network tab of a web browser's developer tools. The left pane lists the following resources: localhost, bundle.js, info?t=1535547162937, and websocket. The right pane shows the response for the 'localhost' resource, which is an HTML document. The HTML content is as follows:

```
7
8 <link rel="manifest" href="/manifest.json">
9 <link rel="shortcut icon" href="/favicon.ico">
10
11 <title>Simple React App</title>
12 </head>
13 <body>
14 <noscript>
15   You need to enable JavaScript to run this app.
16 </noscript>
17 <div id="root"></div>
18 <script type="text/javascript" src="/static/js/bundle.js"></script></body>
19 </html>
20
```

A red box highlights the closing tag `</div>` of the `<div id="root">` element on line 17.

# REACT SERVER-SIDE RENDERING

I'm the home component

Press me!

The screenshot displays the Chrome DevTools interface with the Network tab selected. The top toolbar shows various icons for navigation and filtering. The 'View' section includes options for 'Group by frame', 'Preserve log', 'Disable cache' (checked), 'Offline', and 'Online'. The 'Filter' section shows 'Hide data URLs' and 'All' selected, with other filter categories like XHR, JS, CSS, etc. The main area shows a timeline of requests. The selected request is from 'localhost' to 'bundle.js'. The 'Response' tab is active, showing the rendered HTML. A red box highlights the main content of the HTML response: `<div id="root"><div data-reactroot=""><div>I'm the home component</div><button>Press me!</button></div></div>`. The bottom panel shows the Console tab with the message 'Hi there!' and the source 'bundle.js:21282'.

Elements Console Sources Network Performance Memory Application Security Audits React

View: [Icons] [ ] Group by frame [ ] Preserve log [x] Disable cache [ ] Offline Online ▼

Filter [ ] Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other

10 ms 20 ms 30 ms 40 ms 50 ms 60 ms 70 ms 80 ms 90 ms 100 ms 110 ms 120 ms 130 ms 140 ms 150 ms 160 ms 170

Name x Headers Preview Response Cookies Timing

localhost  
bundle.js

```
1 <html>  
2 <head></head>  
3 <body>  
4 <div id="root"><div data-reactroot=""><div>I'm the home component</div><button>Press me!</button></div></div>  
5 <script src="bundle.js"></script>  
6 </body>  
7 </html>  
8  
9
```

2 requests | 81... { } Line 5, Column 31

Console What's New Remote devices

[ ] [ ] top ▼ Filter Default levels ▼ [x] Group similar

Hi there! bundle.js:21282

>

## SERVER – index.js

```
import path from 'path';
import fs from 'fs';

import express from 'express';
import React from 'react';
import { renderToString } from 'react-dom/server';
import Home from '../client/components/Home';

const app = express();

app.use(express.static('public'));

app.get('/', (req, res) => {
  const content = renderToString(<Home />);

  const indexFile = path.resolve('./public/index.html');
  fs.readFile(indexFile, 'utf8', (data) => {
    return res.send(
      data.replace('<div id="root"></div>', `<div id="root">${content}</div>`)
    );
  });
});

app.listen(3000, () => {
  console.log('Listening on port 3000.');
```

## CLIENT – client.js

```
// Startup point for the client side application

import React from 'react';
import ReactDOM from 'react-dom';
import Home from './components/Home';

ReactDOM.hydrate(<Home />, document.querySelector('#root'));
```

# REACT-ROUTER

- Beneficial for SPA → gives Multi-Page-Feeling
- Navigating in SPA ≠ go to new page → the views load within the same page
- + the URL can be updated, reflecting the views
- + user can use the browser's back and forward buttons
- + easy to handle nested views because it's component based

For further reference: <https://reacttraining.com/react-router/>



localhost:3001/posts/

Posts

New Post

**sunt aut facere repellat  
provident occaecati  
excepturi optio  
reprehenderit**

Max

**qui est esse**

Max

**ea molestias quasi  
exercitationem repellat  
qui ipsa sit aut**

Max

**eum et est occaecati**

Max





localhost:3001/posts/1

Posts

New Post

**sunt aut facere repellat  
provident occaecati  
excepturi optio  
reprehenderit**

Max

**qui est esse**

Max

**ea molestias quasi  
exercitationem repellat  
qui ipsa sit aut**

Max

**eum et est occaecati**

Max

**sunt aut facere repellat provident  
occaecati excepturi optio reprehenderit**

quia et suscipit suscipit recusandae consequuntur expedita et cum reprehenderit molestiae  
ut ut quas totam nostrum rerum est autem sunt rem eveniet architecto

← → ↻ ⓘ localhost:3001/new-post

Posts **New Post**

## Add a Post

Title

Content

Author

Max

Add Post

The page isn't reloaded!!!

# App.js

```
import React, { Component } from 'react';
import { BrowserRouter } from 'react-router-dom';

import Blog from './containers/Blog/Blog';

class App extends Component {
  render() {
    return (
      <BrowserRouter>
        <div className="App">
          <Blog />
        </div>
      </BrowserRouter>
    );
  }
}

export default App;
```

# Blog.js

```
import React, { Component } from 'react';
import { Route, NavLink, Switch, Redirect } from 'react-router-dom';

import './Blog.css';
import Posts from './Posts/Posts';
import NewPost from './NewPost/NewPost';

class Blog extends Component {
  state = {
    auth: true
  }

  render () {
    return (
      <div className="Blog">
        <header>
          <nav>
            <ul>
              <li><NavLink
                to="/posts/"
                exact
                activeClassName="my-active"
                activeStyle={{
                  color: '#fa923f',
                  textDecoration: 'underline'
                }}>Posts</NavLink></li>
              <li><NavLink to="/new-post">New Post</NavLink></li>
            </ul>
          </nav>
        </header>
      </div>
    );
  }
}
```

## Blog.js

```
        <Switch>
          {this.state.auth ? <Route path="/new-post" component={NewPost} /> : null}
          <Route path="/posts" component={Posts} />
          <Redirect from="/" to="/posts" />
        </Switch>
      </div>
    );
  }
}

export default Blog;
```

## Posts.js

```
render () {  
  let posts = <p style={{textAlign: 'center'}}>Something went wrong!</p>;  
  if (!this.state.error) {  
    posts = this.state.posts.map(post => {  
      return (  
        <Link to={'/posts/' + post.id} key={post.id} >  
          <Post  
            key={post.id}  
            title={post.title}  
            author={post.author}  
          />  
        </Link>  
      );  
    });  
  }  
  
  return (  
    <div>  
      <section className="Posts">  
        {posts}  
      </section>  
      <Route path={this.props.match.url + '/:id'} exact component={FullPost} />  
    </div>  
  );  
};
```

THANK YOU.  
ANY QUESTIONS?