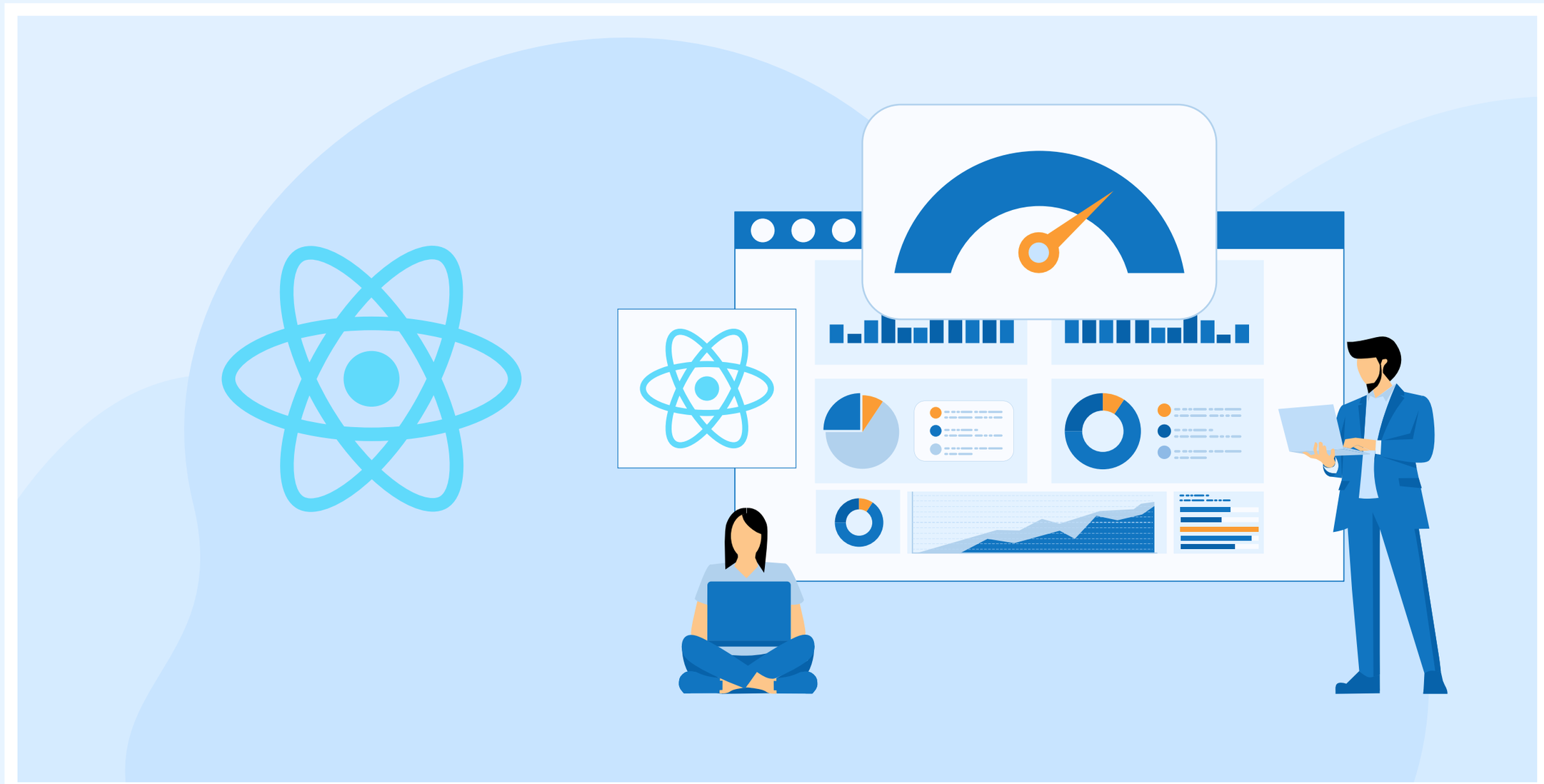
ReactJS Performance Optimization Techniques



TatvaSoft
sculpting thoughts...

Lazy Loading Images

Implement lazy loading to improve loading time by loading images only when they appear on the user's screen. Use libraries like react-lazy-load-image-component.

```
import { LazyLoadImage as LazyImage } from "react-lazy-load-image-component";
import "react-lazy-load-image-component/src/effects/blur.css";
export default function App() {
  return (
    <div className="App">
      <LazyImage
        src={"https://placedog.net/500/300"}
        alt="lazyimage alt"
        effect="blur"
      />
    </div>
  );
}
```

Reduce Unnecessary Renders

Utilize React.memo and useCallback to reduce unnecessary re-renders and improve performance.

```
const MemoizedComponent = React.memo(ComplexComponent);
```

```
const memoizedCallback = useCallback(
  () => {
    doSomething(a, b);
  },
  [a, b],
);
```

TatvaSoft
sculpting thoughts...

Avoid Spreading Props on DOM Elements

Avoid unknown HTML attributes by specifying only necessary attributes.

Avoid:

```
<MessageLabel { ... props}>
```

Use:

```
const MessageText = props ⇒ {
  return (
    <MessageLabel specificAttr={props.specificAttr}>
      {props.message}
    </MessageLabel>
  );
};
```

TatvaSoft

sculpting thoughts...

Server-Side Rendering (SSR)

- Implement SSR to deliver content faster and improve SEO.
- Consider using frameworks like Next.js to simplify SSR setup.

Dependency Optimization

- Analyze dependencies to reduce bundle size by removing unused code.
- Use plugins like moment-locales-webpack-plugin and lodash-webpack-plugin.

Avoid Props in Initial States

Do not initialize the state with props in the constructor. Use props directly in the render method instead.

```
class MessageComponent extends Component {
  constructor(props) {
    super(props);
    this.state = { isEnabled: true };
  }

  render() {
    return <div>{this.props.messageText && <>Enter Message:
    <Input type="text" name="message" /></>}</div>;
  }
}
```

TatvaSoft
sculpting thoughts...

Memoize React Components

Use memoization to cache expensive function calls and prevent unnecessary re-renders.

```
import moize from 'moize';
const UserDetails = ({user, onEdit}) =>{
    const {titleText, fullName, profileImg} = user;
    return (
        <div className="user-detail-wrapper">
            <img src={profileImg} alt={fullName} />
            <span>{fullName}</span>
            <span>{titleText}</span>
        </div>
    )
}
export default moize(UserDetails,{
    isReact: true
});
```

TatvaSoft
sculpting thoughts...

React.PureComponent

- Optimize component updates by using PureComponent for shallow prop comparisons.
- Ensure state/props are immutable and not deeply nested.

```
class Greeting extends PureComponent {
  render() {
    return <h1>Hello World, {this.props.name}!</h1>;
  }
}
```

TatvaSoft
sculpting thoughts...

Avoid Inline Functions

Avoid using inline functions in render methods to prevent unnecessary re-renders.

```
class MessageList extends React.Component {
  onMessageClick = (messageId) => {
    this.setState({ selectedMessageId: messageId });
  }

  render() {
    return messages.map((message) => (
      <Message onClick={this.onMessageClick} message={message}
       key={message.id} />
    ));
  }
}
```

Use a Function in setState

Use a function in setState to ensure state updates are correctly based on previous state.

```
this.setState((prevState) ⇒ ({ correctData: !prevState.correctData }));
```

List Virtualization in React Applications

- Implement list virtualization to render only visible items, improving performance.
- Leverage libraries like react-virtualized or react-window.

CSS Animations Instead of JS Animations

- Prefer CSS animations for simple, state-based animations.
- Reserve JS animations for complex effects requiring fine control.

Want to add your insights?

Leave your optimization strategies in the comments.

TatvaSoft
sculpting thoughts...