Settings

Currently we only have implemented text direction and theme selection. We decided to use React Context API to make the settings available in the entire app by wrapping it into SettingsProvider. We could have used Redux, but we want to give you more options.

Before the app is being loaded, we try to extract stored settings (from localStorage) and display the app using the restored settings.

Using settings data

This can done by using useSettings hook:

import useSettings from 'src/hooks/useSettings';

const MyComponent = () => {

const { settings } = useSettings();

return (

// use settings

);

};

Or by SettingsConsumer:

import { SettingsConsumer } from 'src/settings/SettingsContext';

const MyComponent = () => {

return (

<SettingsConsumer>

{({settings}) => (

// use settings

)}

</SettingsConsumer>

);

};

Updating settings

It can be implemented in any component within seconds.

import useSettings from 'src/hooks/useSettings';

const MyComponent = () => {

const { saveSettings } = useSettings();

const [values, setValues] = useState({});

const handleSave() {

const nextSettings = {}; // Get values from form

saveSettings(nextSettings);

};

return (

<div>

<form onSubmit={handleSave}>

{/\* form fields \*/}

<Button>Save</Button>

</form>

</div>

);

};

Using settings to control component rendering or any business logic

Lets say that you want to display or hide a component based on user settings. All you have to do is to make sure you update the src/contexts/SettingsContext.js file. It contains a default settings object that has to be updated with your new settings options, for example: displayPromo: true. After that it will be available in your entire app.

import useSettings from 'src/hooks/useSettings';

const MyComponent = () => {

const { settings } = useSettings();

return (

<div>

{settings.displayPromo && (

<Promo />

)}

</div>

);

};