Beezy

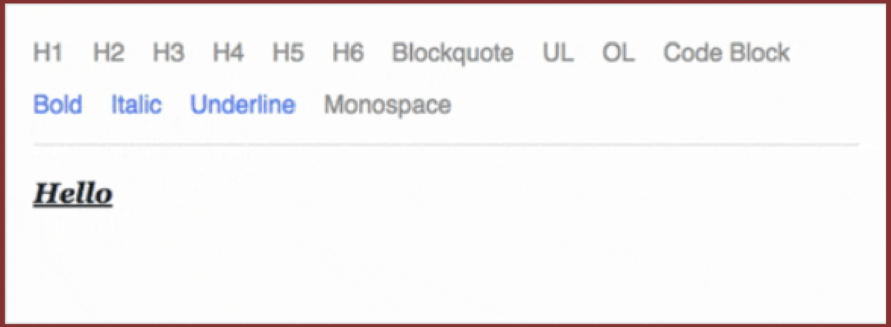
Working with Draft.js

Christian Ruiz & Mario Terrón

Draft.js

Rich Text Editor Framework for React

Get Started



Extensible and Customizable

We provide the building blocks to enable the creation of a broad variety of rich text composition experiences, from basic text styles to embedded media.

Declarative Rich Text

Draft.js fits seamlessly into React applications, abstracting away the details of rendering, selection, and input behavior with a familiar declarative API.

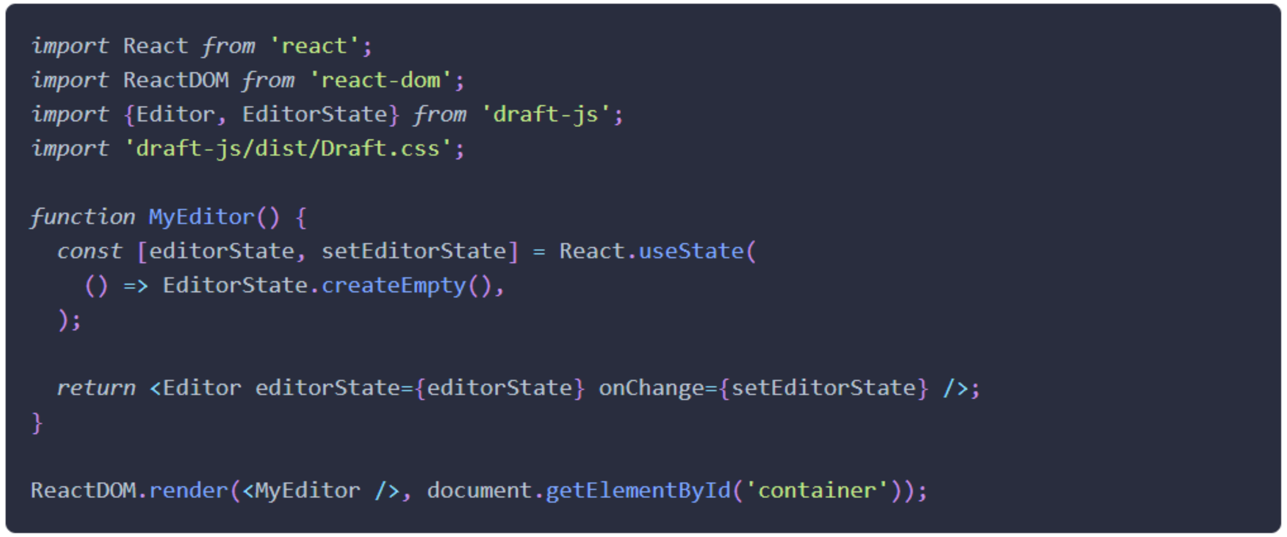
Immutable Editor State

The Draft.js model is built with immutable-js, offering an API with functional state updates and aggressively leveraging data persistence for scalable memory usage.

Beezy

EditorState

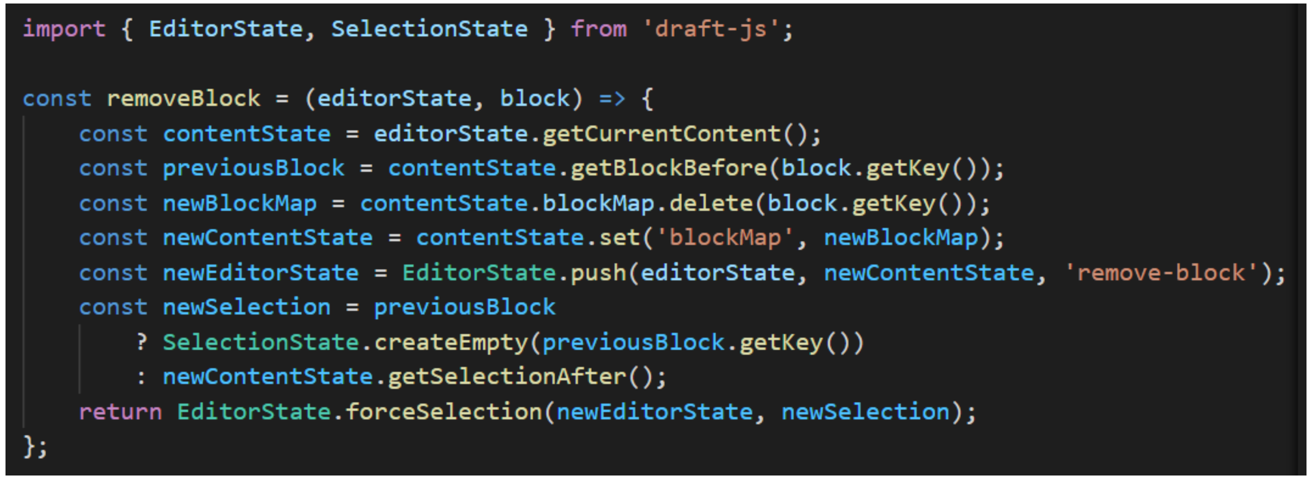
- Top level state for the editor, containing:
  - Current text content state
  - Current selection state
  - Fully decorated representation of contents
  - Undo/redo stacks
  - Most recent change type
- Important methods:
  - create
  - getCurrentContent
  - getCurrentSelection
  - push
  - undo/redo
  - forceSelection



```
import React from 'react';
import ReactDOM from 'react-dom';
import {Editor, EditorState} from 'draft-js';
import 'draft-js/dist/Draft.css';

function MyEditor() {
  const [editorState, setEditorState] = React.useState(
    () => EditorState.createEmpty(),
  );

  return <Editor editorState={editorState} onChange={setEditorState} />;
}

ReactDOM.render(<MyEditor />, document.getElementById('container'));
```

ContentState

- Immutable record that represents the state of:
  - Editor contents (text, blocks, inline styles, entities)
  - Selection state before and after rendering the content
- Important methods:
  - createFromText
  - convertToRaw / convertFromRaw
  - getBlockForKey
  - getBlockBefore / getBlockAfter
  - getEntity
  - blockMap.delete

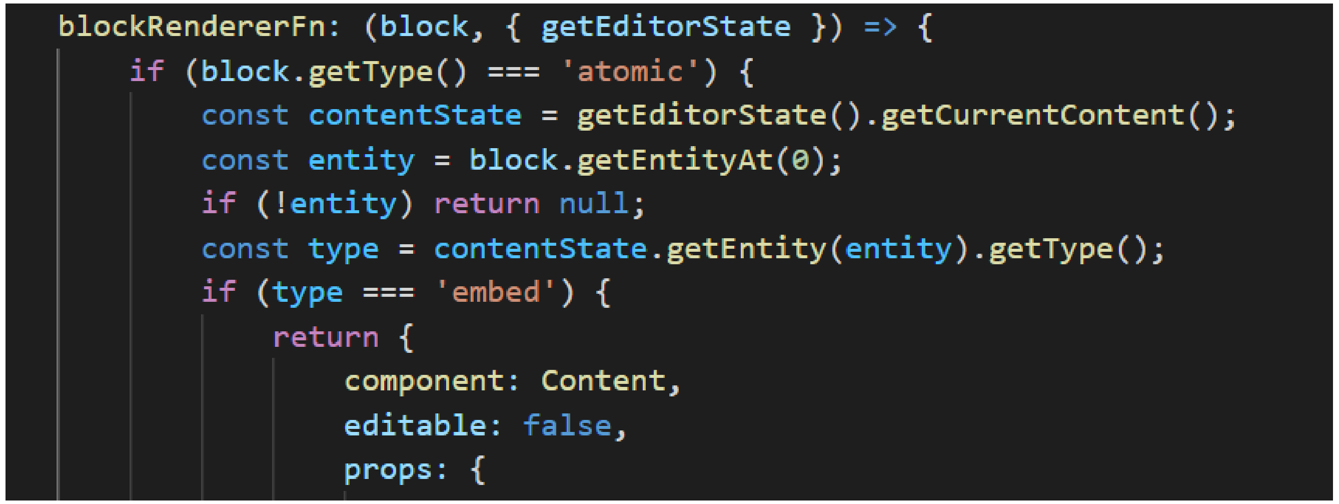


```
import { EditorState, SelectionState } from 'draft-js';

const removeBlock = (editorState, block) => {
    const contentState = editorState.getCurrentContent();
    const previousBlock = contentState.getBlockBefore(block.getKey());
    const newBlockMap = contentState.blockMap.delete(block.getKey());
    const newContentState = contentState.set('blockMap', newBlockMap);
    const newEditorState = EditorState.push(editorState, newContentState, 'remove-block');
    const newSelection = previousBlock
        ? SelectionState.createEmpty(previousBlock.getKey())
        : newContentState.getSelectionAfter();
    return EditorState.forceSelection(newEditorState, newSelection);
};
```

Beezy

ContentBlock

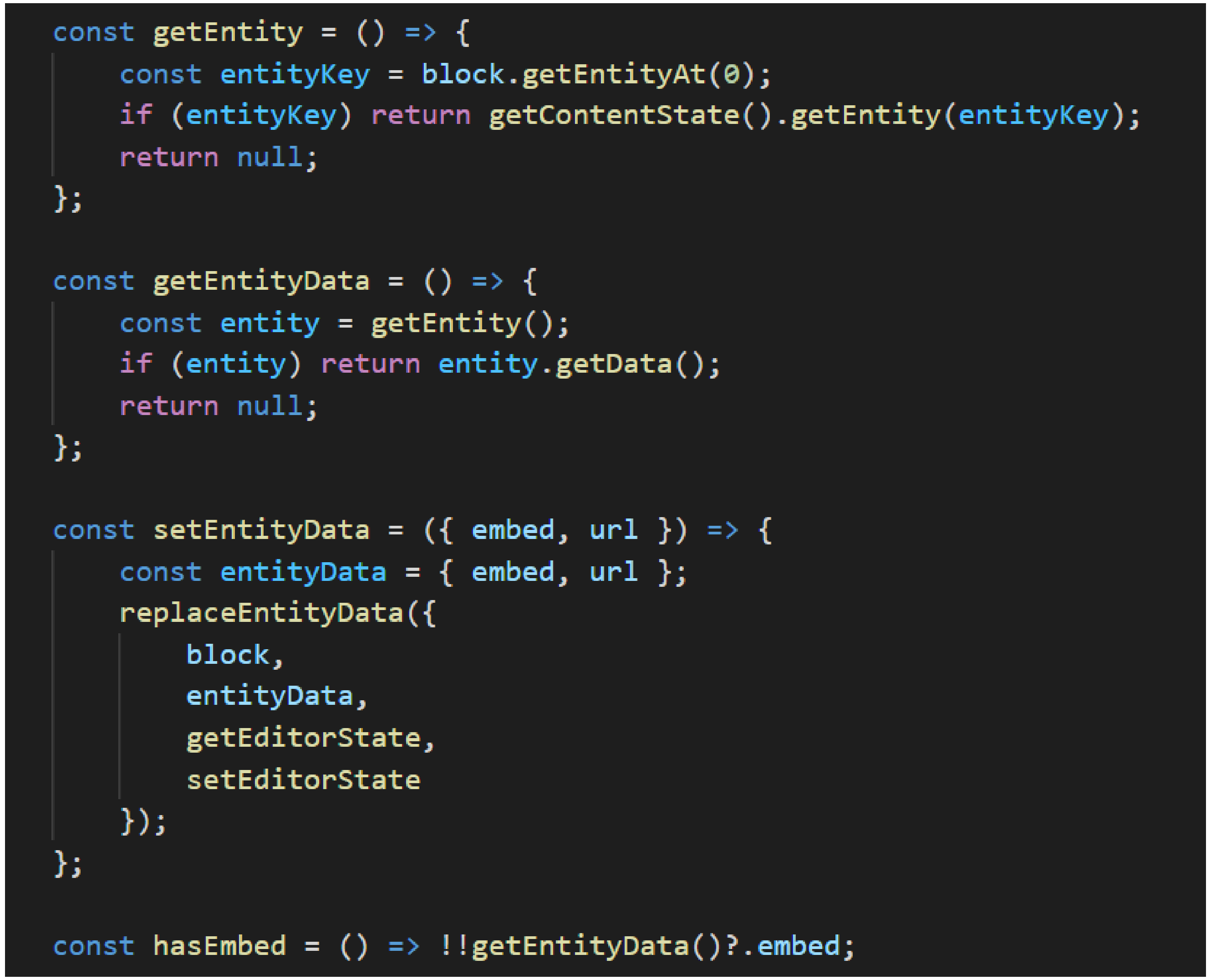
- ContentBlock or just "Block" represents the full state of a single block of editor content, including:
  - Plain text contents of the block
  - Type, e.g. paragraph, header, list item
  - Entity, inline style, and depth information
- Important methods
  - getKey
  - getType
  - getText
  - getInlineStyleAt
  - getEntityAt



```
blockRendererFn: (block, { getEditorState }) => {
    if (block.getType() === 'atomic') {
        const contentState = getEditorState().getCurrentContent();
        const entity = block.getEntityAt(0);
        if (!entity) return null;
        const type = contentState.getEntity(entity).getType();
        if (type === 'embed') {
            return {
                component: Content,
                editable: false,
                props: {
```

Entity

- Allow us to anotate a range of text or a block with metadata. Allows us to add complex types to the editor beyond text styles, like links, mentions, images, embeds, etc.
- Important methods:
  - create
  - getData
  - mergeData
  - replaceData



```
const getEntity = () => {
    const entityKey = block.getEntityAt(0);
    if (entityKey) return getContentState().getEntity(entityKey);
    return null;
};

const getEntityData = () => {
    const entity = getEntity();
    if (entity) return entity.getData();
    return null;
};

const setEntityData = ({ embed, url }) => {
    const entityData = { embed, url };
    replaceEntityData({
        block,
        entityData,
        getEditorState,
        setEditorState
    });
};

const hasEmbed = () => !!getEntityData()?.embed;
```

DraftJS Plugins

High quality plugins with great UX

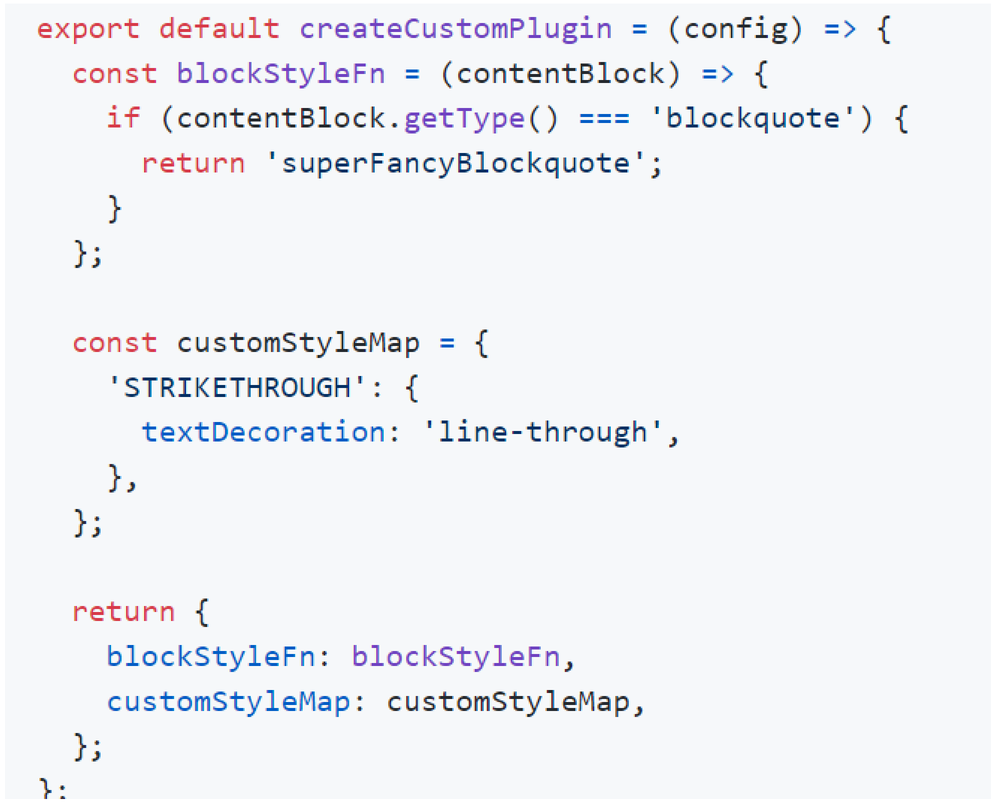
Github 3,792

Mention	Emoji	Image	Video	
Sticker	Hashtag	Inline Toolbar	Side Toolbar	
Static Toolbar	Undo	Counter	Anchor	Linkify
Focus	Alignment	Resizeable	Drag'n'Drop	Divider

Beezy

createPlugin

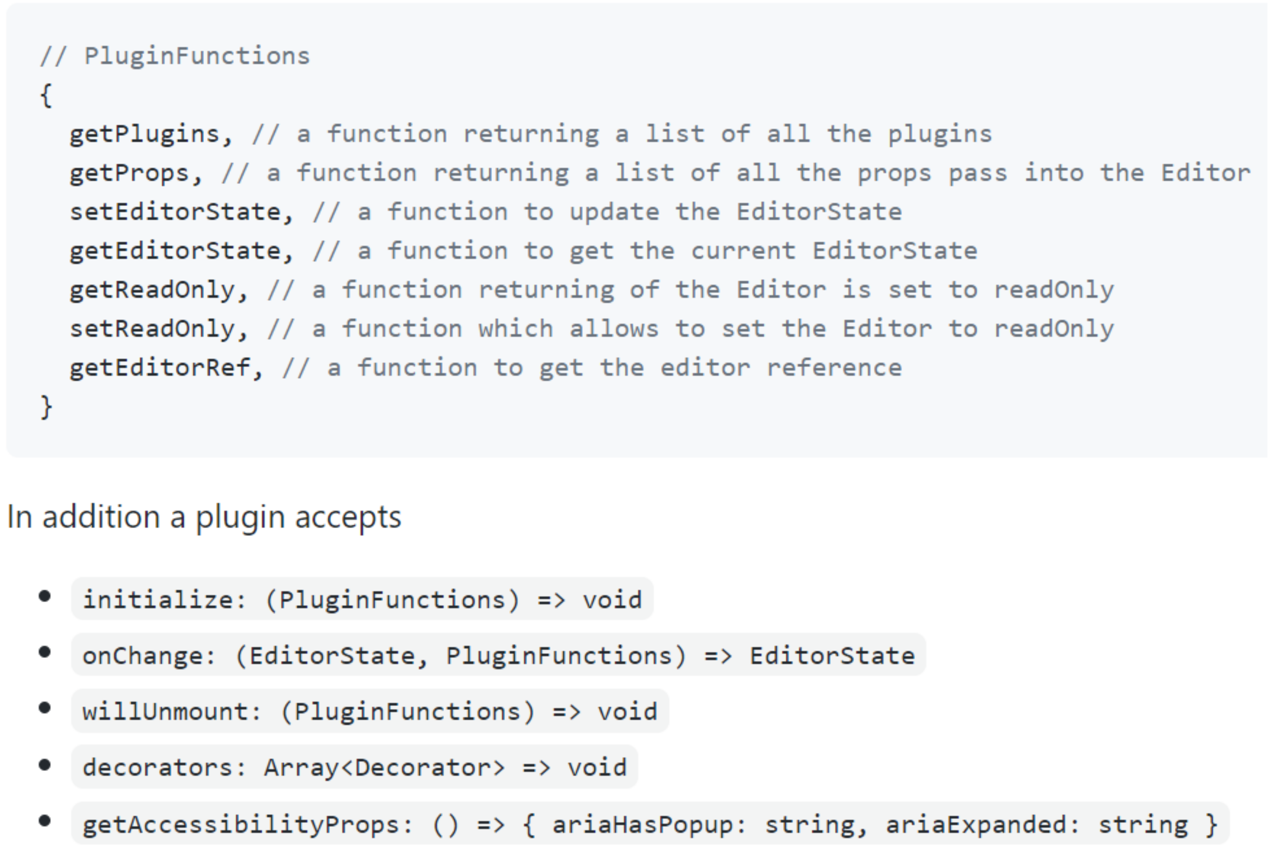
- We should return all the props and handlers we want to pass to the editor for a plugin
- Important props and handlers:
  - blockRendererFn
  - keyBindingFn
  - blockStyleFn
  - handleReturn
  - handleBeforeInput
  - handlePastedText
  - handlePastedFiles
  - handleDroppedFiles
  - onFocus
  - onBlur



```
export default createCustomPlugin = (config) => {
  const blockStyleFn = (contentBlock) => {
    if (contentBlock.getType() === 'blockquote') {
      return 'superFancyBlockquote';
    }
  };

  const customStyleMap = {
    'STRIKETHROUGH': {
      textDecoration: 'line-through',
    },
  };

  return {
    blockStyleFn: blockStyleFn,
    customStyleMap: customStyleMap,
  };
};
```

PluginFunctions

- Some methods pass extra functions to provide some common Draft.js functionality to the plugins:
- Important methods:
  - initialize
  - decorators
  - onChange
- Important functions:
  - getEditorState
  - setEditorState
  - getReadOnly
  - setReadOnly
  - getEditorRef



```
// PluginFunctions
{
  getPlugins, // a function returning a list of all the plugins
  getProps, // a function returning a list of all the props pass into the Editor
  setEditorState, // a function to update the EditorState
  getEditorState, // a function to get the current EditorState
  getReadOnly, // a function returning of the Editor is set to readOnly
  setReadOnly, // a function which allows to set the Editor to readOnly
  getEditorRef, // a function to get the editor reference
}
```

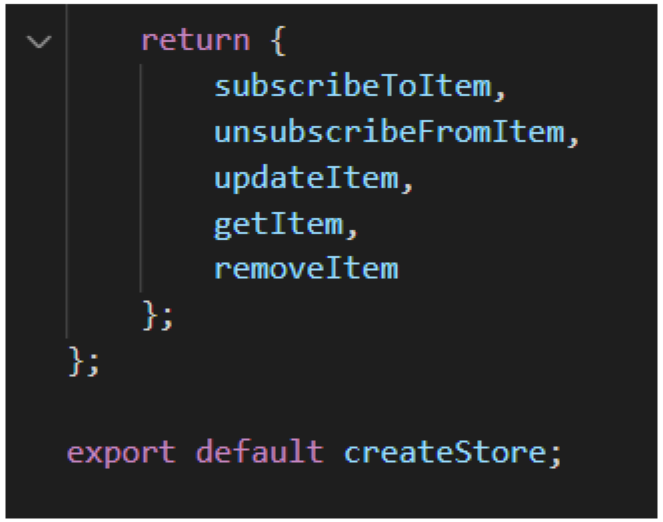
In addition a plugin accepts

- `initialize: (PluginFunctions) => void`
- `onChange: (EditorState, PluginFunctions) => EditorState`
- `willUnmount: (PluginFunctions) => void`
- `decorators: Array<Decorator> => void`
- `getAccessibilityProps: () => { ariaHasPopup: string, ariaExpanded: string }`

Beezy

Plugin store

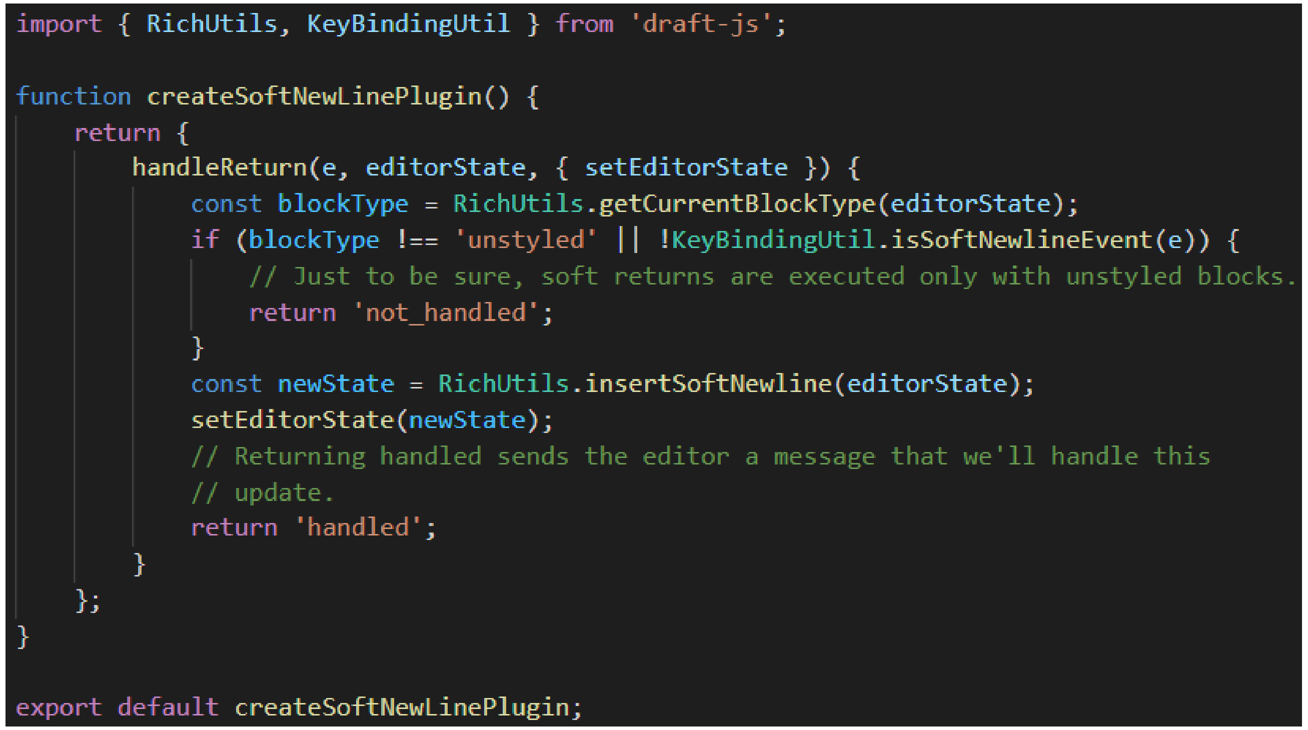
- Util that should be instanciated in the createPlugin and allow communication between the plugin different components/methods/handlers.
- Should never be shared across plugins.



```
        return {
            subscribeToItem,
            unsubscribeFromItem,
            updateItem,
            getItem,
            removeItem
        };
    };

    export default createStore;
```

Handler plugins

- Listens to specific handlers (handleReturn, handlePastedText, etc.) and updates the editorState.

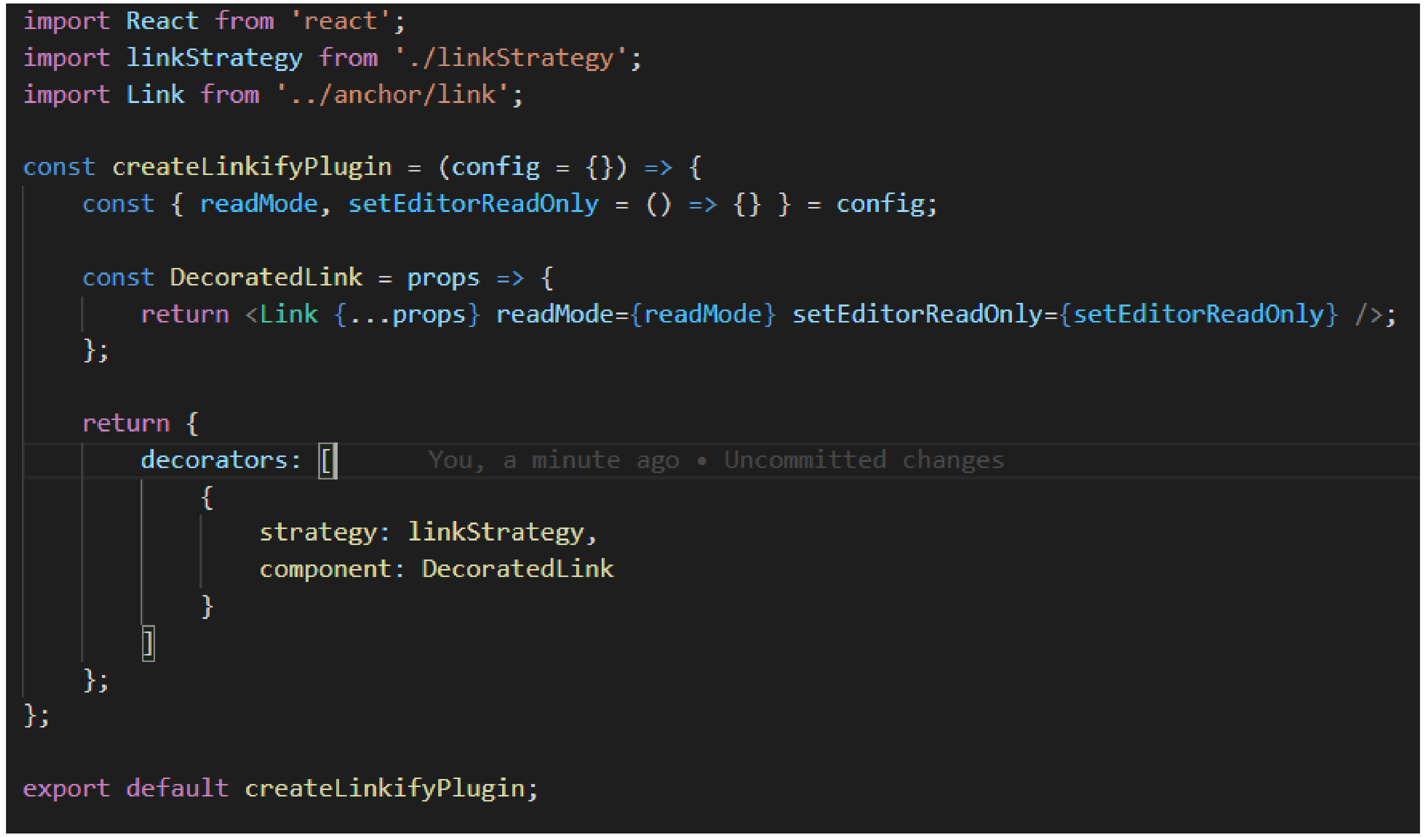


```
import { RichUtils, KeyBindingUtil } from 'draft-js';

function createSoftNewLinePlugin() {
    return {
        handleReturn(e, editorState, { setEditorState }) {
            const blockType = RichUtils.getCurrentBlockType(editorState);
            if (blockType !== 'unstyled' || !KeyBindingUtil.isSoftNewlineEvent(e)) {
                // Just to be sure, soft returns are executed only with unstyled blocks.
                return 'not_handled';
            }
            const newState = RichUtils.insertSoftNewline(editorState);
            setEditorState(newState);
            // Returning handled sends the editor a message that we'll handle this
            // update.
            return 'handled';
        }
    };
}

export default createSoftNewLinePlugin;
```

Beezy

Decorator plugins

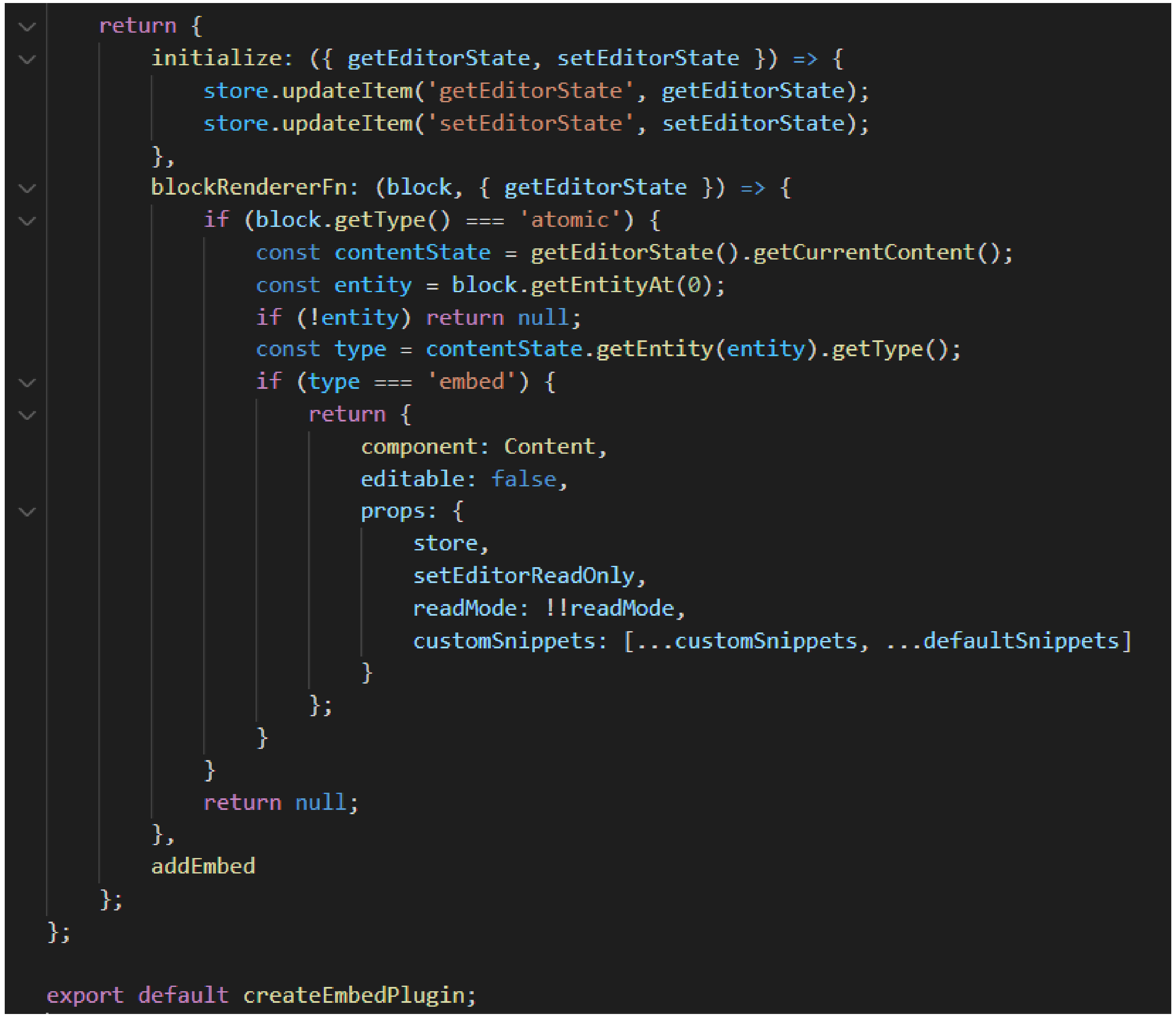
- Returns a Draft.js decorator
- Decorator props
  - strategy: (contentBlock, callback(start, end))
  - component



```
import React from 'react';
import linkStrategy from './linkStrategy';
import Link from '../anchor/link';

const createLinkifyPlugin = (config = {}) => {
    const { readMode, setEditorReadOnly = () => {} } = config;

    const DecoratedLink = props => {
        return <Link {...props} readMode={readMode} setEditorReadOnly={setEditorReadOnly} />;
    };

    return {
        decorators: [      You, a minute ago • Uncommitted changes
            {
                strategy: linkStrategy,
                component: DecoratedLink
            }
        ]
    };
};

export default createLinkifyPlugin;
```

Component plugins

- Uses the blockRendererFn to render a component inside the editor.
- Draft.js recommends the use of "atomic" blocks for this kind of components.
- Add an entity to only use the component for specific entity types ("image", "embed", "video"...)
- Uses the entity data to store persistent component state (avoid useState).

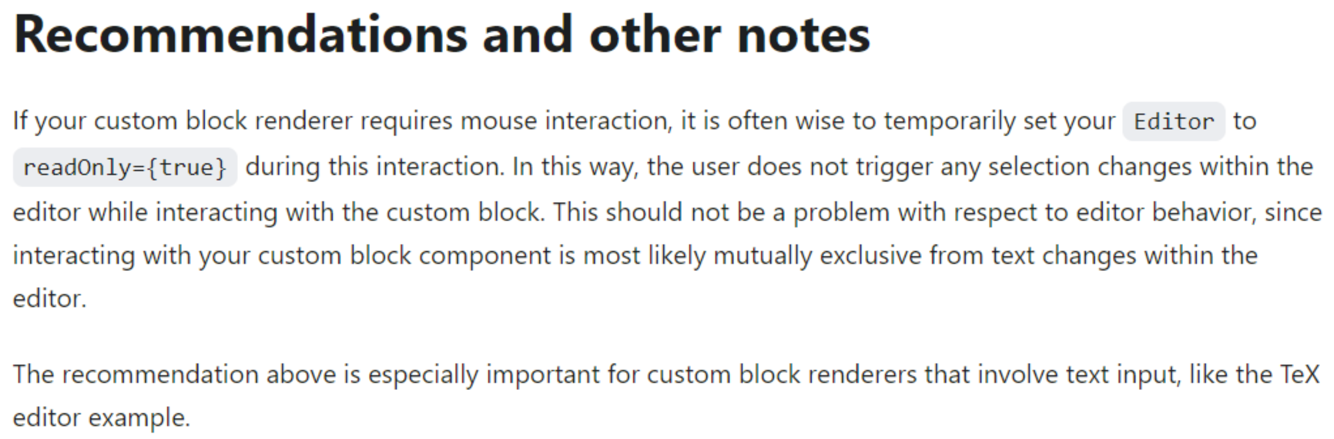


```
    return {
        initialize: ({ getEditorState, setEditorState }) => {
            store.updateItem('getEditorState', getEditorState);
            store.updateItem('setEditorState', setEditorState);
        },
        blockRendererFn: (block, { getEditorState }) => {
            if (block.getType() === 'atomic') {
                const contentState = getEditorState().getCurrentContent();
                const entity = block.getEntityAt(0);
                if (!entity) return null;
                const type = contentState.getEntity(entity).getType();
                if (type === 'embed') {
                    return {
                        component: Content,
                        editable: false,
                        props: {
                            store,
                            setEditorReadOnly,
                            readMode: !!readMode,
                            customSnippets: [...customSnippets, ...defaultSnippets]
                        }
                    };
                }
            }
            return null;
        },
        addEmbed
    };
};

export default createEmbedPlugin;
```

Beezy

readOnly

- Allows complex interactions in components inside the editor without Draft.js handling them. Mouse events, text input, buttons, etc.
- Should set the editor readOnly with onMouseEnter or onFocus and restore it to not readOnly onMouseLeave or onBlur.



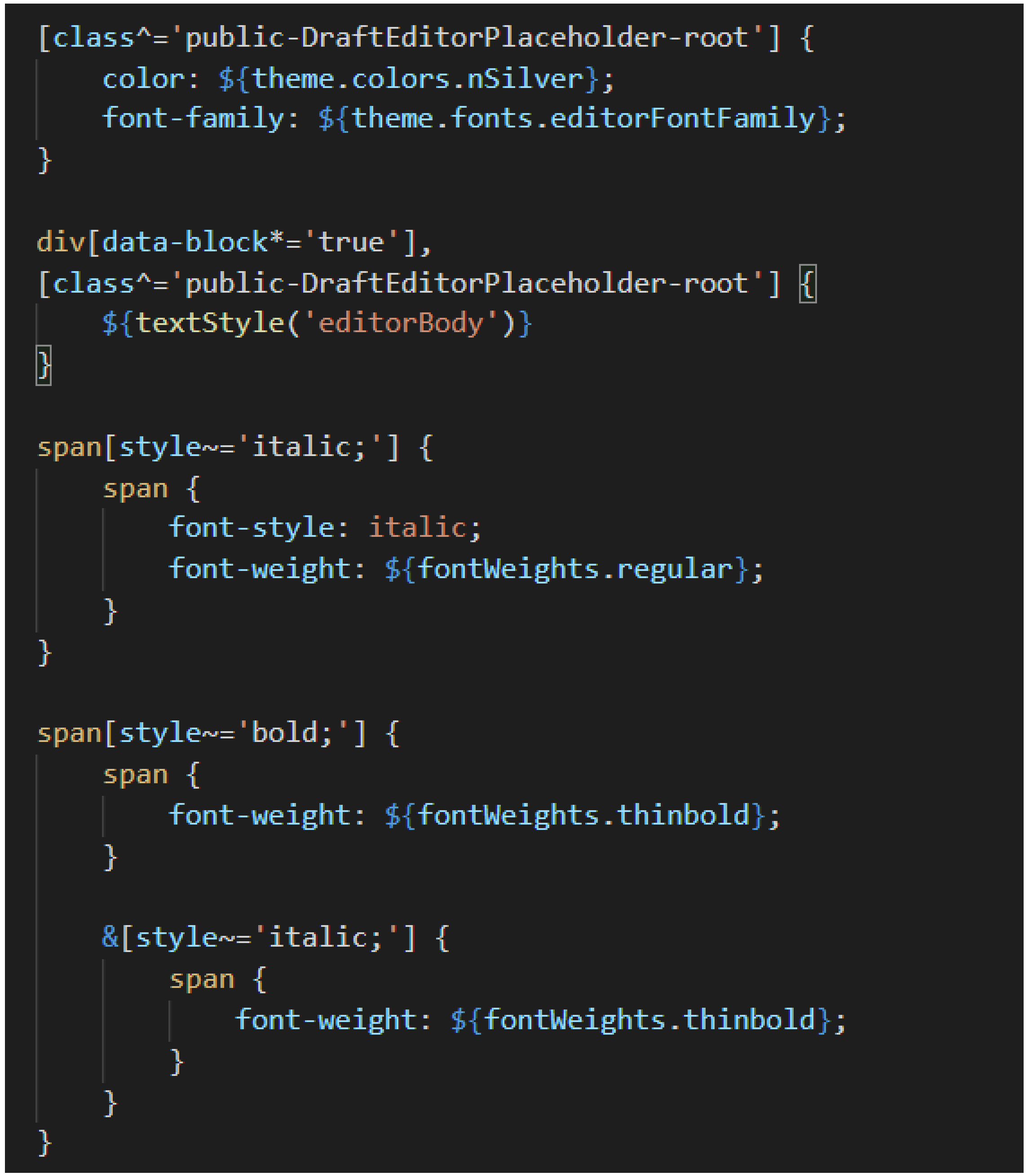
Recommendations and other notes

If your custom block renderer requires mouse interaction, it is often wise to temporarily set your `Editor` to `readOnly={true}` during this interaction. In this way, the user does not trigger any selection changes within the editor while interacting with the custom block. This should not be a problem with respect to editor behavior, since interacting with your custom block component is most likely mutually exclusive from text changes within the editor.

The recommendation above is especially important for custom block renderers that involve text input, like the TeX editor example.

Styling

- Draft.js is packaged with a css file.
- In uielements, we choose not to include the default css file and implement Product own design from scratch.
- For you own plugins, you can chose how to style your components (as regular React components).



```
[class^='public-DraftEditorPlaceholder-root'] {
    color: ${theme.colors.nSilver};
    font-family: ${theme.fonts.editorFontFamily};
}

div[data-block*='true'],
[class^='public-DraftEditorPlaceholder-root'] {
    ${textStyle('editorBody')}
}

span[style~='italic;'] {
    span {
        font-style: italic;
        font-weight: ${fontWeights.regular};
    }
}

span[style~='bold;'] {
    span {
        font-weight: ${fontWeights.thinbold};
    }

    &[style~='italic;'] {
        span {
            font-weight: ${fontWeights.thinbold};
        }
    }
}
```

Beezy