

Front-End

- Naming Conventions
- Application Folder Structure
- Service(s)
- Component(s)
- Modal(s)
- Context(s)
- Constant(s)
- Hook(s)
- Util(s)
- Router(s)
- React Component Inside Structure
- Publishing: Scheduler: Post: Editor

Naming Conventions

- Camel Case: camelCase
- Snake Case: snake_case
- Kebab Case: kebab-case
- Pascal Case: PascalCase
- Upper Case: UPPERCASE
- Lower Case: lowercase

Application Folder Structure

- /.devops
- /public
- /src
 - /_metronic
 - /assets
 - /_components
 - /_constants
 - /_contexts
 - /_hooks
 - /_modals
 - /_services
 - /_redux
 - /_router
 - /_utils

Service(s)

Folder Name

[{ServiceName}] (PascalCase): Conversations

File Name

{ServiceName}+".js" (PascalCase): Conversations.js

React Functional Component Name

{ServiceName}+"Service"

```
function ConversationsService() {  
  ..  
}
```

Folder Structure

- /{ServiceName}
 - /_components (Optional)
 - /_constants (Optional)
 - /_contexts (Optional)
 - /_hooks (Optional)
 - /_modals (Optional)
 - /_redux (Optional)
 - /_router (Optional)
 - /_utils (Optional)
 - /{PartialServiceName}
 - {PartialServiceName}.js
 - {ServiceName}.js

Component(s)

Component Types

Global Component: Tüm uygulama içerisinde veya farklı servislerde birden fazla kere kullanılan component'lardır.

Service Component: Tek bir servis içerisinde bir veya birden fazla kere kullanılan component'lardır.

Partial Component: Bir component içerisinde kullanılan alt component'lardır.

Global Components Locations

Application Level: /components/app: Uygulamanın en üst seviyesinde bir kere kullanılan komponentlar.

Basic: /components/basic: Eğer bir komponent sadece prop olarak işlemler yapıyorsa basit bir komponenttir.

Advanced: /components/advanced: Eğer bir komponent kendi içerisinde Redux, Modal, Utils gibi özelliklere sahipse gelişmiş bir komponenttir.

Folder Name

[{ComponentName}] (PascalCase): Sidekick

File Name

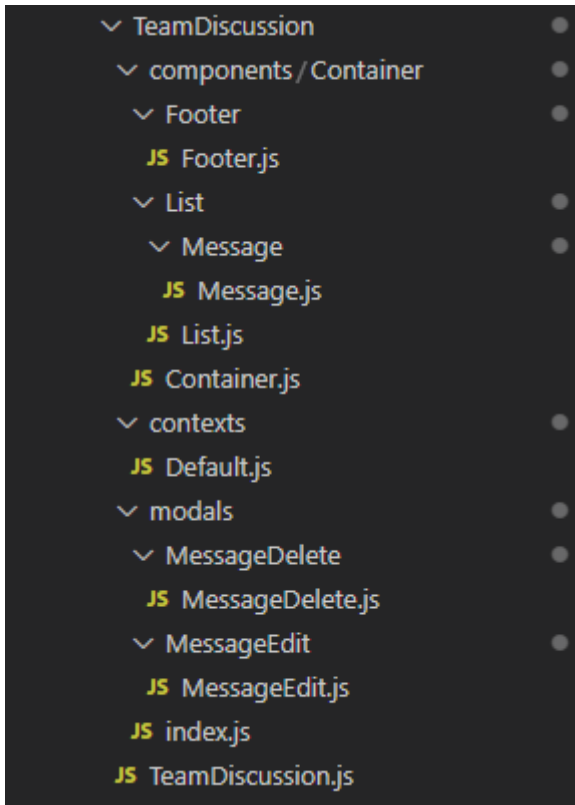
{ComponentName} + ".js" (PascalCase): Sidekick.js

Component Folder Structure

- /{ComponentName}
 - **/_constants** (Optional just for Global Components)
 - **/_contexts** (Optional just for Global Components)
 - **/_hooks** (Optional just for Global Components)
 - **/_modals** (Optional just for Global Components)
 - **/_redux** (Optional just for Global Components)
 - **/_utils** (Optional just for Global Components)
 - /{PartialComponentName}
 - {PartialComponentName}.js
 - {PartialComponentName}.test.js
 - {PartialComponentName}.styled.js
 - {PartialComponentName}.scss

- {ComponentName}.js
- {ComponentName}.test.js
- {ComponentName}.styled.js
- {ComponentName}.scss

Component Folder Structure Sample



React Functional Component Name

{ServiceName} + {ParentName} + {...} + {ParentName} + {ComponentName}

```
function ConversationsSidekickToolbar() {  
  ...  
}
```

Modal(s)

A modal is a dialog box/popup window that is displayed on top of the current page.

Folder Structure

- /{ModalName}
 - /_redux (Optional)
 - /_components (Optional)
 - /_constants (Optional)
 - /_contexts (Optional)
 - /_hooks (Optional)
 - /_utils (Optional)
 - {ModalName}.js
 - {ModalName}.test.js
 - {ModalName}.styled.js
 - {ModalName}.scss

Context(s)

Context provides a way to pass data through the component tree without having to pass props down manually at every level.

In a typical React application, data is passed top-down (parent to child) via props, but such usage can be cumbersome for certain types of props (e.g. locale preference, UI theme) that are required by many components within an application. Context provides a way to share values like these between components without having to explicitly pass a prop through every level of the tree.

Folder Name

/contexts

File Name

{ContextName} + ".js" (PascalCase): Default.js

React Functional Component Name

{ParentName} + {ContextName} + "ContextProvider"

```
export function ConversationsDefaultContextProvider({ children }){  
  ...  
}
```


Constant(s)

What is a constants file?

A constants file is a dedicated file to store declared constant properties. The beauty of this file is that it's accessible globally throughout the app.

Declare A Constant

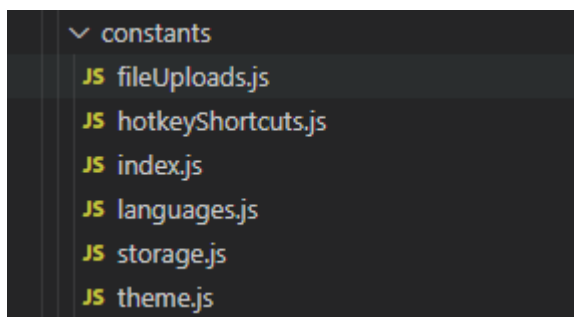
ES6 provides a new way of declaring a constant by using the `const` keyword. The `const` keyword creates a read-only reference to a value. By convention, the constant identifiers are in uppercase.

```
export const MAX_FILE_SIZE = 1048576 * 100;
```

Folder Structure

- /constants
 - {constantsGroupName}.js
 - index.js

Folder Structure Sample



Constants File Sample

```
export const MAX_FILE_SIZE = 1048576 * 100; // 100 MB

export const FILE_UPLOAD_ACCEPTS_IMAGES =
"image/jpg,image/jpeg,image/png,image/gif,image/tiff,image/webp";
export const FILE_UPLOAD_ACCEPTS_VIDEOS =
`video/mp4,video/quicktime,video/mov,video/avi,video/mpg,video/wmv,video/x-
m4v,video/webm,video/x-matroska,video/x-msvideo,video/x-ms-asf,video/x-ms-wmv,video/x-
flv,video/3gpp,video/ogg`;
export const FILE_UPLOAD_ACCEPTS_IMAGES_VIDEOS =
```

```
`${FILE_UPLOAD_ACCEPTS_IMAGES},${FILE_UPLOAD_ACCEPTS_VIDEOS}`;
```

Constant Files Grouping Sample: index.js

```
export * from "@constants/fileUploads";  
export * from "@constants/hotkeyShortcuts";  
export * from "@constants/languages";  
export * from "@constants/theme";
```

Hook(s)

Hooks are the new feature introduced in the React 16.8 version. It allows you to use state and other React features without writing a class. Hooks are the functions which "hook into" React state and lifecycle features from function components. It does not work inside classes.

Hooks are backward-compatible, which means it does not contain any breaking changes. Also, it does not replace your knowledge of React concepts.

When to use a Hooks

If you write a function component, and then you want to add some state to it, previously you do this by converting it to a class. But, now you can do it by using a Hook inside the existing function component.

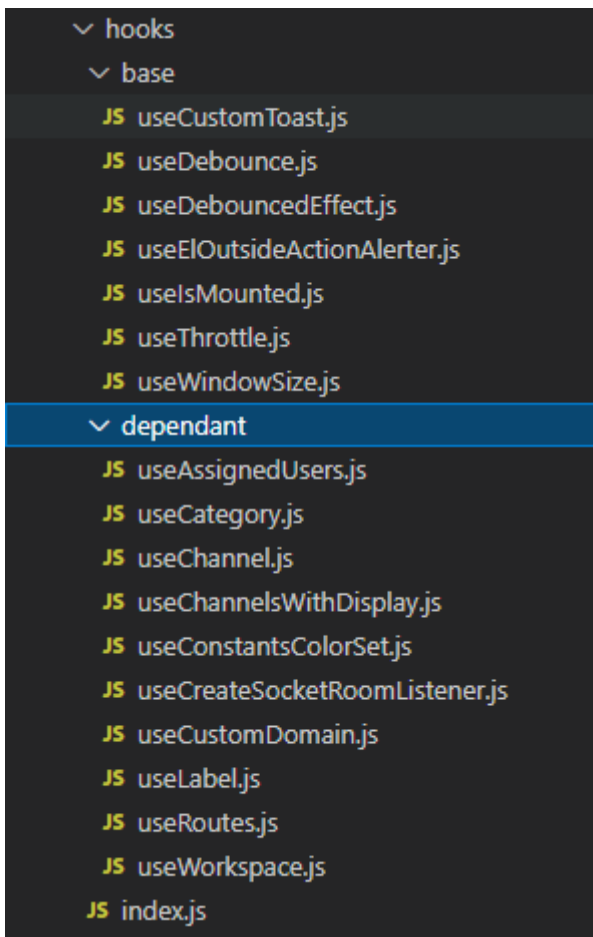
Rules of Hooks

Hooks are similar to JavaScript functions, but you need to follow these two rules when using them. Hooks rule ensures that all the stateful logic in a component is visible in its source code. These rules are:

Folder Structure

- /hooks
 - /category-name (Optional)
 - use{HookName}.js
 - use{HookName}.js
 - index.js

Folder Structure Sample



React Hook File Sample

```
import { useEffect, useState } from "react";

function useWindowSize() {
  [] ..
}

export default useWindowSize;
```

React Hook Grouping Sample: index.js

```
export { default as useDebounceEffect } from "@hooks/base/useDebounceEffect";
export { default as useIsMounted } from "@hooks/base/useIsMounted";
..
```

Util(s)

- /utils
 - {utilsGroupName}.js
 - index.js

Router(s)

In a typical React application, data is passed top-down (parent to child) via props, but such usage can be cumbersome for certain types of props (e.g. locale preference, UI theme) that are required by many components within an application. Context provides a way to share values like these between components without having to explicitly pass a prop through every level of the tree.

React Functional Component Name

{Name} + "Router": AppRouter

```
function AppRouter() {  
  ...  
}  
  
export default AppRouter;
```

Folder Structure

- /router
 - index.js

React Component Inside Structure

- **base hooks (not custom)**
 - useTranslate
 - useHistory
 - useLocation
 - etc.
- **states**
- **querys**
- **mutations**
- **selectors**
- **actions & events / actions / events**
- **renders**

Example:

src > pages > Billing > pages > Subscription > components > layout > Main > Main.js > SubscriptionLayoutMain

You, 2 minutes ago | 4 authors (You and others)

```
1 import React, { useState } from "react";
2 import { useTranslation } from "react-i18next";
3 import { isEmpty } from "lodash";
4 import { searchData, toCurrency } from "@utils";
5 import { Button, Icon, LoaderButton, MomentInline, NoRecords, ScrollableContainer, SearchBox } from "@components";
6 import { useDispatch } from "react-redux";
7 import { useQuery } from "@redux-requests/react";
8 import { uiFCR, workspaceFCR } from "@redux/common";
9 import { subscriptionPlanTotal } from "@redux/common/workspace/utils";
10
11 function SubscriptionLayoutMain() {
12   const { t } = useTranslation();
13   const dispatch = useDispatch();
14
15   // states
16   const [searchText, setSearchText] = useState("");
17
18   // querys
19   const workspacesQuery = useQuery({ type: workspaceFCR.FETCH_WORKSPACES });
20
21   // mutations
22
23   // selectors
24
25   // events
26   const onClickWorkspaceDelete = (workspace) => dispatch(uiFCR.showModal("workspace.delete.confirmation", workspace));
27
28   // renders
29   > const renderCardToolbar = () => { ...
41   };
42
43   > const renderWorkspaces = () => { ...
158   };
159
160   return (
161   >   <>...
214   </>
215   );
216 }
217
218 export default SubscriptionLayoutMain;
```


Publishing: Scheduler: Post:
Editor