Introduction

This version of Fuse is NOT a traditional admin template, it's an React app written entirely with Javascript and has no jQuery dependency.

While Fuse React is a great tool and source for learning React, it also requires at least an entry level of React, Redux knowledge so you can find your way within the source code.

Here you can find a list of core libraries, design specifications and coding standards that we use in Fuse React:

Google's Material Design

All libraries and custom made components are following **Google's Material Design Specifications.**

React

React is the core of our template. Fuse React is NOT a traditional admin template, it's an React app. If you don't know what React is or don't know how to use it, we strongly recommend checking the React before start doing anything with Fuse.

Material-UI

Material-UI is a react ui library that implement Google's Material Design specification.

Create React App (CLI)

Create React App is a tool built by developers at Facebook to help you build React applications. It saves you from time-consuming setup and configuration.

Installation

Prerequisites

This section will give you some information about what tools you will need.

Node.js

To install and use Fuse React, you will need **Node.js** installed to your computer. We won't get into too much detail about Node.js as it's out of the scope of this documentation. Also you won't need to actually use Node.js, it's only required for the development process.

Git

To be able to install and use Fuse, you will also need **Git** installed to your computer. Git is required for npm/yarn to work correctly.

Yarn - Package Manager

Fuse React uses **Yarn** package manager to install and manage 3rd party components and libraries.

Installation

A. Installing Prerequisites

1. Download and install **at least LTS** or the latest version of **Node.js** from its web site.
2. Download and install the latest version of **Git** from its web site.
3. Download and install the latest **Yarn** from its web site.

B. Installing Fuse React

1. Unzip the zip file that you have downloaded from Themeforest. Inside the zip file, you will find the Skeleton Project **(Fuse-react-x.x.x-skeleton.zip)** along with the Demo Project **(Fuse-react-x.x.x-demo.zip)**, PSD designs and a readme file.
2. Extract the contents of the zip file **(Fuse-react-x.x.x-skeleton.zip)** into a folder that you will work within. For this documentation, we will refer that as "your work folder".
3. Open your favorite console application (Terminal, Command Prompt etc.), navigate into your work folder, run the following command and wait for it to finish:

   ```
   yarn
   ```

   This command will install all the required Node.js modules into the node_modules directory inside your work folder.

   And now, you are ready to run the Fuse React for the first time. Please continue to the Working with Fuse React section.

Development

While still in your work folder, run the following command in the console application:

```
yarn start
```

And that's it. create-react-app will take care everything and start the Fuse React.

You can check out your console application to get further information about the server. By default, it will run on **http://localhost:3000** but it might change depending on your setup.

Production

```
yarn run build
```

compiles the application into `/build` directory

Production

Production

The following command builds the application into an output directory

```
yarn run build
```

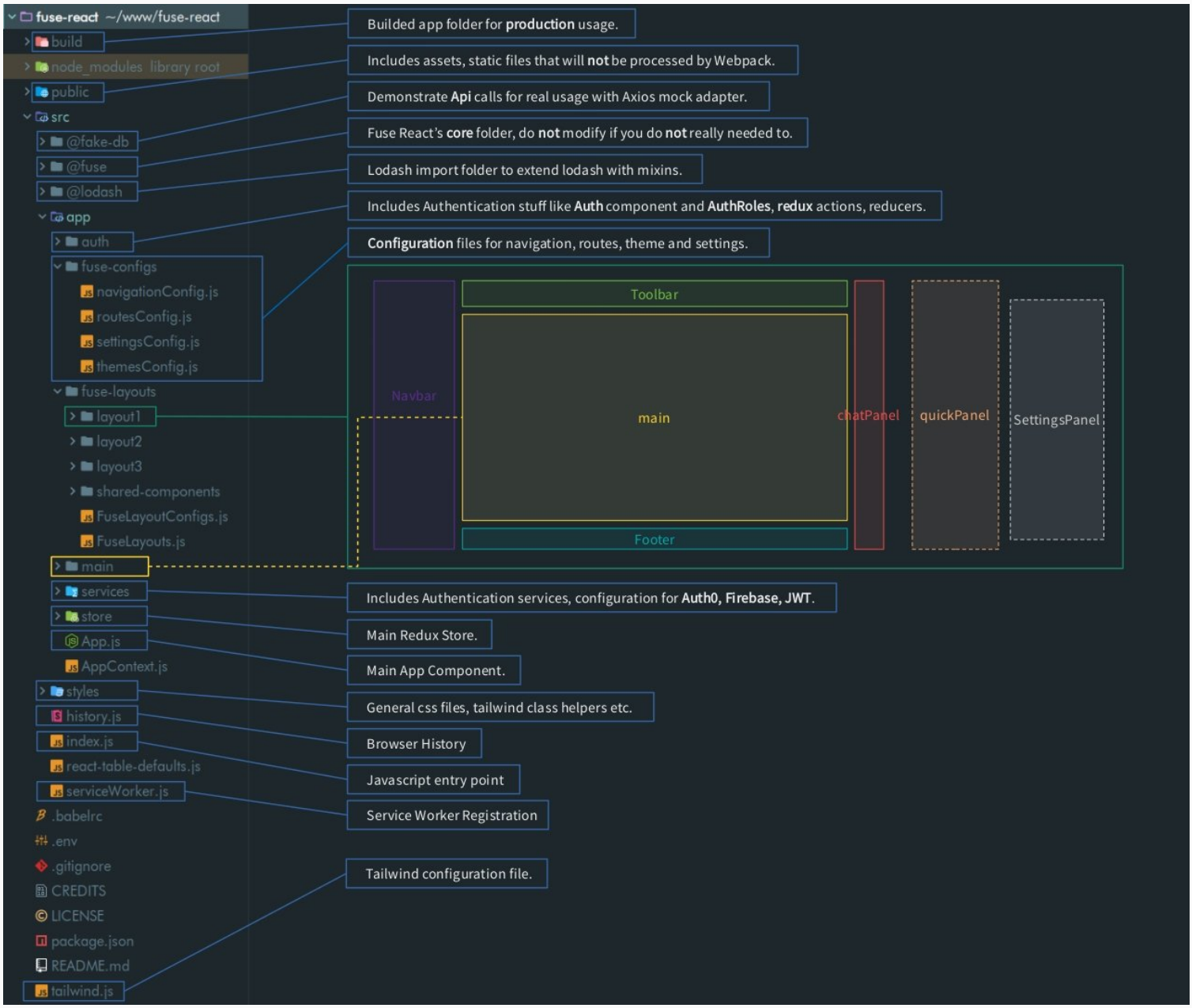
compiles the application into `/build` directory

Deployment

Checkout at Facebook's create-react-app documentation: **Deployment**

Project Structure

Here's the project structure of the Fuse React:



Updating Fuse React

Fuse React isn't end product or an extension so there is no easy way to update the project. Due to the nature of apps, we cannot give any set instructions for updating Fuse React. It heavily depends on your project and it's up to you to update your code. However, there are couple points that we want to put forward which might help you to keep Fuse React updated:

1. The most important one is not to touch the /@fuse directory but sometimes that is going to be inevitable and in those cases, try to keep the modifications minimal.

2. Usually React and Material UI libraries do some breaking changes and force our hands to change things. In those cases, it's always good to check their official Changelogs to see what they did. Usually they provide clear instructions and even helper tools to update your code.

3. Before starting your new project, **join our Github repo** , fork it and build your app on top of that fork. This way, in the future, you can easily merge or compare the changes from the main repo onto your fork. This will require merging a lot of changes manually, but it's the best way to keep the Fuse React updated.

Theming

The Fuse React uses material-ui's theming by default. You can can change the colors, the typography with defining theme configuration objects.

Configuration

For the configuration options checkout **Material UI's theme configuration options.**

Theme configurations are located at `src/app/fuse-configs/themesConfig.js` file.

```
import {fuseDark} from '@fuse/fuse-colors';
import lightBlue from '@material-ui/core/colors/lightBlue';
import red from '@material-ui/core/colors/red';

const themesConfig = {
    default    : {
        palette: {
            type     : 'light',
            primary  : fuseDark,
            secondary: {
                light: lightBlue[400],
                main : lightBlue[600],
                dark : lightBlue[700]
            },
            error    : red
        },
        status : {
            danger: 'orange'
        }
    },
    sunset     : {
        palette: {
            type     : 'light',
            primary  : {
                light: '#ff908b',
                main : '#d0605e',
                dark : '#9b3134'
            },
            secondary: {
                light       : '#c76a1d',
                main        : '#ff994c',
                dark        : '#ffca7b',
                contrastText: '#fff'
            },
            error    : red
        },
        status : {
            danger: 'orange'
        }
    },
    greeny     : {
```

```
        palette: {
            type     : 'light',
            primary  : {
                light: '#6cabd4',
                main : '#387ca3',
                dark : '#005074'
            },
            secondary: {
                light       : '#89f6cf',
                main        : '#55c39e',
                dark        : '#159270',
                contrastText: '#fff'
            },
            error    : red
        },
        status : {
            danger: 'orange'
        }
    },
    beach      : {
        palette: {
            type     : 'light',
            primary  : {
                light       : '#c4d8dd',
                main        : '#93a7ab',
                dark        : '#65787c',
                contrastText: '#fff'
            },
            secondary: {
                light       : '#ffb281',
                main        : '#f18153',
                dark        : '#ba5228',
                contrastText: '#fff'
            }
        }
    },
    tech       : {
        palette: {
            type     : 'light',
            primary  : {
                light       : '#87efff',
                main        : '#4dbce9',
                dark        : '#008cb7',
                contrastText: '#fff'
            },
            secondary: {
                light: '#ffff83',
                main : '#d1e751',
                dark : '#9db516'
            }
        }
    },
    sweetHues  : {
        palette: {
            type     : 'light',
            primary  : {
```

```
            light       : '#d5c1eb',
            main        : '#a391b9',
            dark        : '#746389',
            contrastText: '#fff'
        },
        secondary: {
            light: '#90afd4',
            main : '#6080a3',
            dark : '#325474'
        }
    }
},
defaultDark: {
    palette: {
        type     : 'dark',
        primary  : fuseDark,
        secondary: {
            light: lightBlue[400],
            main : lightBlue[600],
            dark : lightBlue[700]
        },
        error    : red
    },
    status : {
        danger: 'orange'
    }
},
deepOcean  : {
    palette: {
        type     : 'dark',
        primary  : {
            light: '#8f53e7',
            main : '#5a24b4',
            dark : '#1e0083'
        },
        secondary: {
            light       : '#ff61ff',
            main        : '#fe00e9',
            dark        : '#c600b6',
            contrastText: '#fff'
        }
    }
},
slate      : {
    palette: {
        type     : 'dark',
        primary  : {
            light: '#86fff7',
            main : '#4ecdc4',
            dark : '#009b94'
        },
        secondary: {
            light       : '#ff9d99',
            main        : '#ff6b6b',
            dark        : '#c73840',
            contrastText: '#fff'
```

```
            }
        }
    }
};

export default themesConfig;
```

Theme Layouts

Fuse comes with variety of different Theme Layouts which you can see and try them from the configuration sidebar (Click on the animated, spinning cog button from the right side of your screen).

These layouts are accessible from `/src/app/fuse-layouts` directory and you can modify them however you like. Each layout has its own options. Those options allow you to configure the layout elements such as Toolbar, Footer and Navbar.

File Structure

Inside the `/src/app/fuse-layouts` directory

- **/layout-1** :
  - **/components** : Contains Layout elements such as Toolbar, Footer and Navbar.
  - **/Layout1.js** : Layout 1 component
  - **/Layout1.config.js** : Contains title, default configs and form options of the layout.
- **/layout-2**
- **/layout-3**
- **/shared components** : Each layouts shares the components of this directory
- **/FuseLayoutConfigs.js** : Imports all layout configs.
- **/FuseLayouts.js** : Imports all layout components.

Configuring

Fuse React has a powerful layout system which allows you to configure and use a different layout per route.

Each route can have its own layout configuration meaning that it's very easy to have pages like login page where there isn't any toolbar or navbar showing, without leaving the Fuse React.

You can get more information about route configuration and its usage from Routing documentation page.

Page Layouts

One of the strong sides of the Fuse React is its Page layout components.

Every single app, pre-built page and even this documentation page uses the Fuse React's page layout component.

Simply, page layout components are pre-built layouts which you can simply copy/paste and start building your own page or app based on it.

Because page layout components, it's very easy to replicate any page style that you can find in Fuse React.

Page Layout Components

- FusePageSimple
- FusePageCarded

Settings

Default Settings

You can set default layout, theme settings of your app at `src/app/fuse-configs/settingsConfig.js`

```
const settingsConfig = {
    layout          : {
        style : 'layout1', // layout-1 layout-2 layout-3
        config: {} // checkout layout configs at app/fuse-configs/layout-1/Layout1Config.js
    },
    customScrollbars: true,
    theme           : {
        main   : 'default',
        navbar : 'mainThemeDark',
        toolbar: 'mainThemeLight',
        footer : 'mainThemeDark'
    }
};

export default settingsConfig;
```

Fuse Routing

Fuse React routing system based on **react-router** and its package **react-router-config**

For the modular approach and route based Fuse settings, we are using config files and generate routes from that files.

For example, have a look at the code below `MailAppConfig.js`. You can override all settings for a particular route as `/apps/mail` for this example.

```
import MailApp from './MailApp';
import React from 'react';
import {Redirect} from 'react-router-dom';

export const MailAppConfig = {
    settings: {
        layout          : {
            style : 'layout1',
            config: {
                scroll : 'content',
                navbar : {
                    display : true,
                    folded  : false,
                    position: 'left'
                },
                toolbar: {
                    display : true,
                    style   : 'fixed',
                    position: 'below'
                },
                footer : {
                    display : true,
                    style   : 'fixed',
                    position: 'below'
                },
                mode    : 'fullwidth'
            }
        },
        customScrollbars: true,
        theme           : {
            main   : 'default',
            navbar : 'mainThemeDark',
            toolbar: 'mainThemeLight',
            footer : 'mainThemeDark'
        }
    },
    routes  : [
        {
            path     : '/apps/mail/label/:labelHandle/:mailId?',
            component: MailApp
        },
        {
            path     : '/apps/mail/filter/:filterHandle/:mailId?',
```

```
            component: MailApp
        },
        {
            path     : '/apps/mail/:folderHandle/:mailId?',
            component: MailApp
        },
        {
            path     : '/apps/mail',
            component: () => <Redirect to="/apps/mail/inbox"/>
        }
    ]
};
```

Then we import and generate routes from that file in `fuse-configs/fuseRoutes`

```
import {appsRoutes} from 'app/main/apps/mail/MailAppConfig.js';
import {FuseUtils} from '@fuse/index';
import {Redirect} from 'react-router-dom';
import React from 'react';

const routeConfigs = [
    MailAppConfig
];

export const routes = [
    ...FuseUtils.generateRoutesFromConfigs(routeConfigs),
    {
        path     : '/',
        component: () => <Redirect to="/pages/errors/error-404"/>
    }
];
```

Code Splitting (Lazy loading)

Code-splitting your app can help you “lazy-load” just the things that are currently needed by the user, which can dramatically improve the performance of your app. While you haven’t reduced the overall amount of code in your app, you’ve avoided loading code that the user may never need, and reduced the amount of code needed during the initial load.

Route-based code splitting

We are using **React Loadable** library and created FuseLoadable higher order component for to avoid repetition of Loadable component defaults.
Checkout the examples below to see dynamically or regular way of importing the components.

Lazy Loaded Component:

```
import {FuseLoadable} from '@fuse';

export const AnalyticsDashboardAppConfig = {
    settings: {
        layout: {
            config: {}
        }
    },
    routes  : [
        {
            path     : '/apps/dashboards/analytics',
            component: FuseLoadable({
                loader: () => import('./AnalyticsDashboardApp')
            })
        }
    ]
};
```

Regular Loaded Component:

```
import AnalyticsDashboardApp from './AnalyticsDashboardApp';

export const AnalyticsDashboardAppConfig = {
    settings: {
        layout: {
            config: {}
        }
    },
    routes  : [
        {
            path     : '/apps/dashboards/analytics',
            component: AnalyticsDashboardApp
        }
```

```
    ]
};
```

Code splitting the Redux reducers (Dynamically loaded reducers)

We created Higher Order Component `withReducer` to load redux reducer before the component render.
You just need to pass **key** and the **reducer** to the component.

```
import withReducer from 'app/store/withReducer';
import reducer from './store/reducers';
.
.
export default withReducer('analyticsDashboardApp', reducer)(AnalyticsDashboardApp);
```