Basics

Basics

Once you have everything setup (if you haven't go back to the Installation (/app/docs/installation) page and finish the installation) you'll notice a file structure similar to this:

```
.
├── CHANGELOG.md
├── bower.json
├── globals.js
├── gulpfile.js
├── package.json
├── prebuild
├── public
│   ├── bower_components
│   ├── css
│   │   ├── app
│   │   ├── demo
│   │   ├── fonts
│   │   └── vendor
│   ├── favicons
│   ├── fonts
│   ├── imgs
│   ├── js
│   │   ├── app
│   │   ├── common
│   │   ├── demo
│   │   └── vendor
│   ├── locales
│   │   └── demo
│   └── video
│       └── homepage
├── scaffold.js
├── server.js
├── src
│   ├── global
│   │   ├── enhance.jsx
│   │   ├── jsx
│   │   ├── router.jsx
│   │   ├── sass
│   │   └── vendor
│   ├── jsx
│   │   ├── app
│   │   └── demo
│   └── sass
│       ├── app
│       └── demo
└── test
    ├── babel-setup.js
    ├── components
    └── setup.js
```

The **src** directory is where your source files are located and it contains three folders:

- **global**: This directory contains files shared by all projects. Do not alter these files unless you know what you are doing.
- **jsx**: This directory contains project sources (JSX files). By default there are two projects that ship with your purchase (Rails users have only 1 folder **app**. To create multiple projects copy the rails-seed directory):
    - **app**: this is a blank starter project
    - **demo**: this is the demo project which can be used for reference
- **sass**: This directory contains project sources (SASS files) and has two folders:
    - **app**: this is a blank starter project

- **demo**: this is the demo project which can be used for reference

main.jsx

This file is the starting point of your app. Look for it in the `src/jsx/app` folder.

The first few lines of the file contain a snippet of code that initializes Mozilla L20n. The first parameter passed to the locale is your project name (this variable is replaced at compilation time with your project name). So if your project's name is ap, then all your locales are stored in `public/locales/app` (For reference see the locales stored in `public/locales/demo`). You can pass your locales to the `locales` option and also set the `default` locale.

```
l20n.initializeLocales(__APPNAME__, {
  'locales': ['en-US'],
  'default': 'en-US'
}});
```

Store all your routes in `src/jsx/app/routes` folder and require them in src/jsx/app/routes.jsx. An example route pointing to a blank page is show in the routes.jsx file:

```
import Blank from 'routes/blank';
```

For routing we make use of the excellent **react-router (http://rackt.github.io/react-router/tags/v1.0.0-beta3.html)** library. Its advised that you go through the documentation for react-router before reading this section.

Now, we define routes to the blank page. You can see from this snippet that we have referenced the variable to the blank page we required earlier.

```
export default (withHistory, onUpdate) => {
  const history = withHistory? new BrowserHistory : null;
  return (
    <Router history={history} onUpdate={onUpdate}>
      <Route path='/' component={Blank} />
    </Route>
  );
);
```

The page itself is rendered within the **div#app-container** element.

blank.jsx

This is an example file and should serve as a starting point for creating various routes in your app. When you open the file you'll immediately notice that there are 5 files required:

```
import classNames from 'classnames';
import SidebarMixin from 'global/jsx/sidebar_component';

import Header from 'common/header';
import Sidebar from 'common/sidebar';
import Footer from 'common/footer';
```

Of the above 5, three files (header.jsx, sidebar.jsx and footer.jsx) are stored in the **common** folder for the app. All of the above files are optional and are only required if you want a full blown dashboard layout. For instance, when designing a homepage you wouldn't need any of the above files.

Then we have a Body component which contains a Container#body component. All your main application code should be written within this component.

```
class Body extends React.Component {
  render() {
    return (
      <Container id='body'>
        <Grid>
          <Row>
            <Col sm={12}>
              <PanelContainer>
                <Panel>
                  <PanelBody className='text-center'>
                    <p>BLANK PAGE</p>
                  </PanelBody>
                </Panel>
              </PanelContainer>
            </Col>
          </Row>
        </Grid>
      </Container>
    );
  }
}
```

Finally we have a default component class which renders the entire page. It contains a Container#container component which has Sidebar, Header, Body and Footer components.

It is important to note that we also include a SidebarMixin which takes care of all the boilerplate code required to show/hide the sidebar on smaller viewport. The variable 'classes' stores the state of the Sidebar and is used for toggling the Sidebar.

```
@SidebarMixin
export default class extends React.Component {
  render() {
    var classes = classNames({
      'container-open': this.props.open
    });

    return (
      <Container id='container' className={classes}>
        <Sidebar />
        <Header />
        <Body />
        <Footer />
      </Container>
    );
  }
}
```

sidebar.jsx

sidebar.jsx file contains the Sidebar section of the page. The sidebar section consits of a div#avatar container, SidebarControls component and the div#sidebar-container.

The SidebarControls component is optional (if you're going to have only 1 sidebar) and can be removed. If you are going to be removing it, you need to also make a small change in **src/global/sass/rubix/overrides/_variables.scss** by making sure the variable `$sidebar-controls-visibility` is set to `hidden` . If you don't want a global setting that affects all your projects you can add it to the top of your **src/sass/app/main.scss** file which restricts the setting to the specific project.

The sidebar props passed to each SidebarControlBtn controls the relevant Sidebar component.

```
export default class extends React.Component {
  render() {
    return (
      <div id='sidebar' {...props}>
        <div id='avatar'>
          ...
        </div>
        <SidebarControls>
          <SidebarControlBtn bundle='fontello' glyph='docs' sidebar={0} />
        </SidebarControls>
        <div id='sidebar-container'>
          <Sidebar sidebar={0} active>
            <ApplicationSidebar />
          </Sidebar>
        </div>
      </div>
    );
  }
}
```

Here is an example of a Sidebar navigation component defined in ApplicationSidebar component. You can nest multiple SidebarNav's to have multiple menu levels.

```
<SidebarNav>
  <SidebarNavItem glyph='icon-fontello-gauge' name='Blank' href='/' />
  <SidebarNavItem glyph='icon-feather-mail' name={<span>Menu <BLabel className='bg-darkgr
een45 fg-white'>3</BLabel></span>}>
    <SidebarNav>
      <SidebarNavItem glyph='icon-feather-inbox' name='Inbox' href='#' />
      <SidebarNavItem glyph='icon-outlined-mail-open' name='Mail' href='#' />
      <SidebarNavItem glyph='icon-dripicons-message' name='Compose' href='#' />
    </SidebarNav>
  </SidebarNavItem>
</SidebarNav>
```