

**A10 GUI – Framework**

***Engineering Design Document***

Version 1.0a

2018/1/10

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Author / Editor | Revision | Date | Comments |
| Zuoping | 0.1 | 1/6/2018 | draft |
| Zuoping/Peter | 0.1 | 1/6/2018 | Proof read and rewrite for style for formatting. |
| Zuoping/Peter | 0.3 | 1/8/2018 | Framework usage and action types namespace |
| Roll/Chris | 0.4 | 1/9/2018 | Proofreading and error adjustment. |
| Peter/Zuoping | 1.0a | 1/10/2018 | First alpha release |
| Roll | 1.1a | 8/22/2018 | Update |
|  |  |  |  |

Contents

[Figures 4](#_Toc503347406)

[Why A10 GUI Framework? 6](#_Toc503347407)

[Design Overview 6](#_Toc503347408)

[Packages 6](#_Toc503347409)

[Framework Package 7](#_Toc503347410)

[Widgets Package 7](#_Toc503347411)

[Commonly-Used Container Packages 7](#_Toc503347412)

[Applications 8](#_Toc503347413)

[A10 GUI Framework 8](#_Toc503347414)

[Component Work Flow 9](#_Toc503347415)

[A10Provider 9](#_Toc503347416)

[Object Definition and UML Diagram 10](#_Toc503347417)

[Use Case 10](#_Toc503347418)

[A10Root 10](#_Toc503347419)

[Object Definition and UML Diagram 11](#_Toc503347420)

[Use Case 12](#_Toc503347421)

[A10Router 12](#_Toc503347422)

[Object Definition and UML Diagram 12](#_Toc503347423)

[Use Case 13](#_Toc503347424)

[A10Container 14](#_Toc503347425)

[Class UML 15](#_Toc503347426)

[Object Relation UML 16](#_Toc503347427)

[Create Interface 16](#_Toc503347428)

[Create Container 16](#_Toc503347429)

[Use Case 17](#_Toc503347430)

[A10 Widgets 17](#_Toc503347431)

[Class UML 18](#_Toc503347432)

[Object Relation UML 19](#_Toc503347433)

[Create Widget 19](#_Toc503347434)

[Use Case 19](#_Toc503347435)

[Redux 20](#_Toc503347436)

[Data Categorization Based on Life Cycle 20](#_Toc503347437)

[Data Life Cycle 20](#_Toc503347438)

[Global Layer 21](#_Toc503347439)

[Container Layer 21](#_Toc503347440)

[Action Type Namespace 21](#_Toc503347441)

[CSS Infrastructure 23](#_Toc503347442)

[About Less Pre-processor 23](#_Toc503347443)

[A10 Predefined Less Library 23](#_Toc503347444)

[CSS In Projects 26](#_Toc503347445)

[Less Usage Example 28](#_Toc503347446)

[Appendix 30](#_Toc503347447)

[Technology Selection 30](#_Toc503347448)

[Framework Usage Example 30](#_Toc503347449)

[Folder Structures 30](#_Toc503347450)

[Future Enhancement (Not in the first release) 36](#_Toc503347451)

[Theme 36](#_Toc503347452)

[Locale 37](#_Toc503347453)

[Plugin Manager 39](#_Toc503347454)

[Utilities and Redux Related Code 39](#_Toc503347455)

# Figures

[Figure 1 Overview of the relationships between software packages/projects. 6](#_Toc503229855)

[Figure 2 Major building blocks of A10 GUI Framework 8](#_Toc503229856)

[Figure 3 Framework work flow UML 9](#_Toc503229857)

[Figure 4 Object Definition and UML Diagram for A10Provider. 10](#_Toc503229858)

[Figure 5 Object Definition and UML Diagram for A10Root 11](#_Toc503229859)

[Figure 6 Object Definition and UML Diagram for A10Router. 12](#_Toc503229860)

[Figure 7 Object Definition and UML Diagram for THFormContainer. 15](#_Toc503229861)

[Figure 8 Object Relation UML for A10Container. 16](#_Toc503229862)

[Figure 9 Object Definition and UML Diagram for A10Form. 18](#_Toc503229863)

[Figure 10 Object Relation UML for A10Widget. 19](#_Toc503229864)

[Figure 11 Three difference categories of data in Redux Store. 21](#_Toc503229865)

[Figure 12 Less Library Files 26](#_Toc503229866)

[Figure 13Pre-defined less library use case 27](#_Toc503229867)

[Figure 14 Less in applicatoin folders 29](#_Toc503229868)

[Figure 15 Less Definition in your application 30](#_Toc503229869)

[Figure 16 Less file import into your code 31](#_Toc503229870)

[Figure 17Theme data flow 39](#_Toc503229871)

# Why A10 GUI Framework?

To provide consistency, reusability and cost savings for the construction of web UI across A10’s product portfolio, we need a common set of technologies, design guidelines, and software libraries.

A10 GUI Framework is based on modern web design. At the heart of it, we leverage a component based front-end JavaScript framework called React.js[[1]](#endnote-2). Together with Redux[[2]](#endnote-3) and our proposals embodied in a set of design patterns, we believe this will go a long way to providing a set of tools to build appealing, modern, and consistent web UI with relative ease and simplicity.

# Design Overview

## Packages

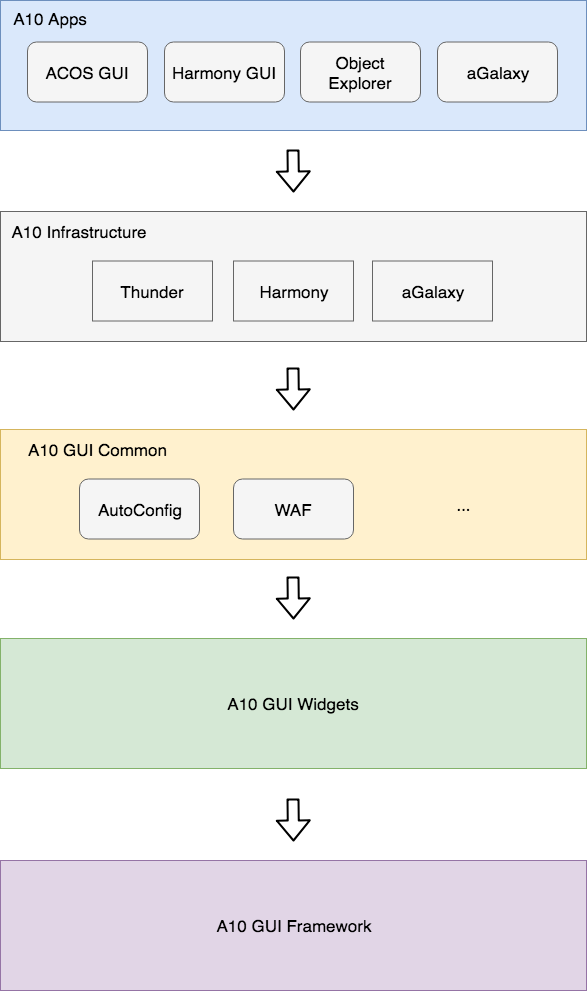


Figure 1 Overview of the relationships between software packages/projects.

We propose the following organization.

### Framework Package

This is a foundation for web UI. All other packages such as Router, Provider, and Redux will depend on this. This package will be more tightly controlled with more stringent design reviews and guidelines for code contribution.

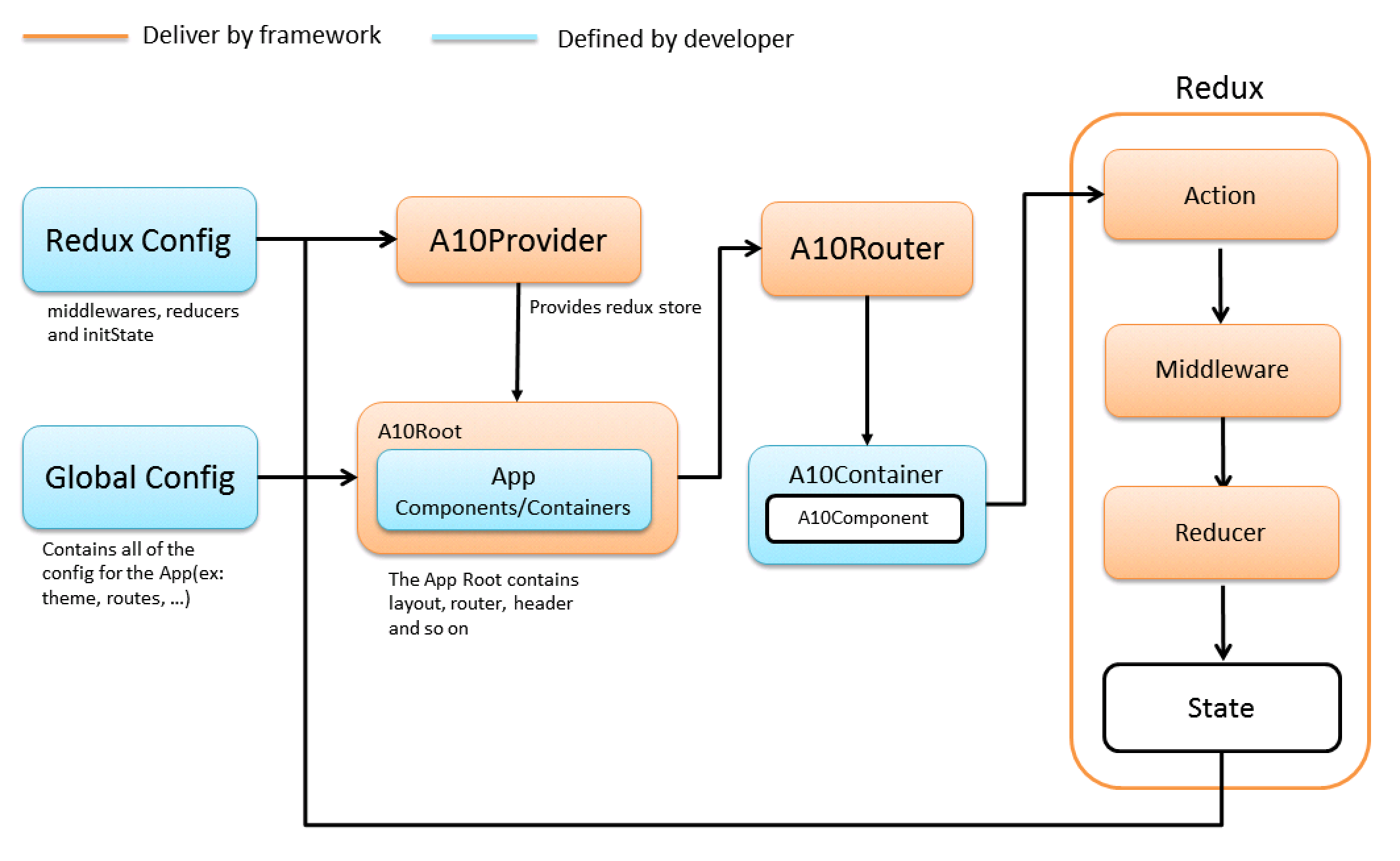


Figure Framework workflow

### Widgets Package

Abstracted from Framework. As time goes on, the volume will grow as more developers contribute increasing variety of Widgets.

### Commonly-Used Container Packages

As different Applications (see below) are built, we see a possibility that certain components will be commonly implemented throughout A10 product portfolio. These components lend themselves to be refactored into a set of Commonly-Used Container Packages so they can be reused.

### Applications

Framework’s main components like A10Provider, A10Root, A10Router and A10Container are used to build a basic scaffolding. Addition Application-specific Containers and Widgets are added to complete the Application.

# A10 GUI Framework

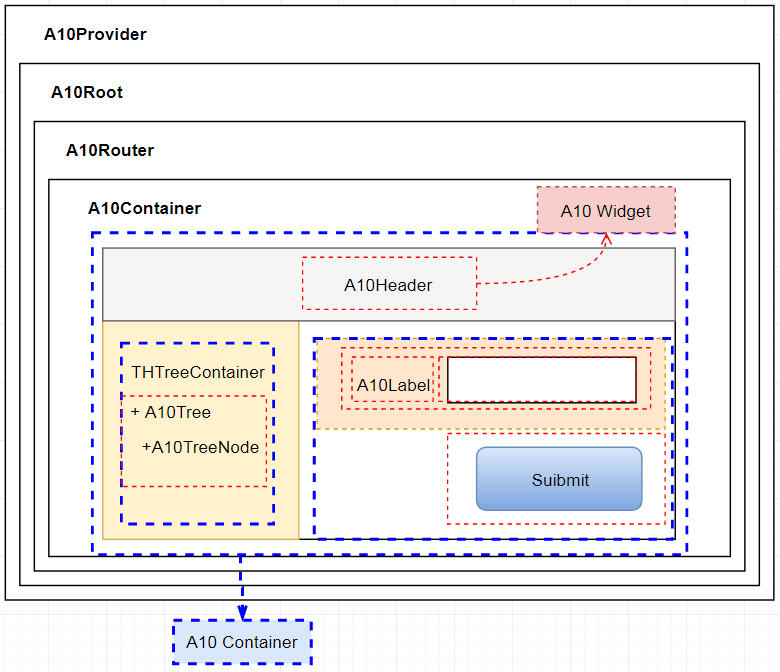


Figure 3 Major building blocks of A10 GUI Framework

Figure 3 depicts an application skeleton which includes major building blocks defined in the package **a10-gui-framework**.

## Component Work Flow

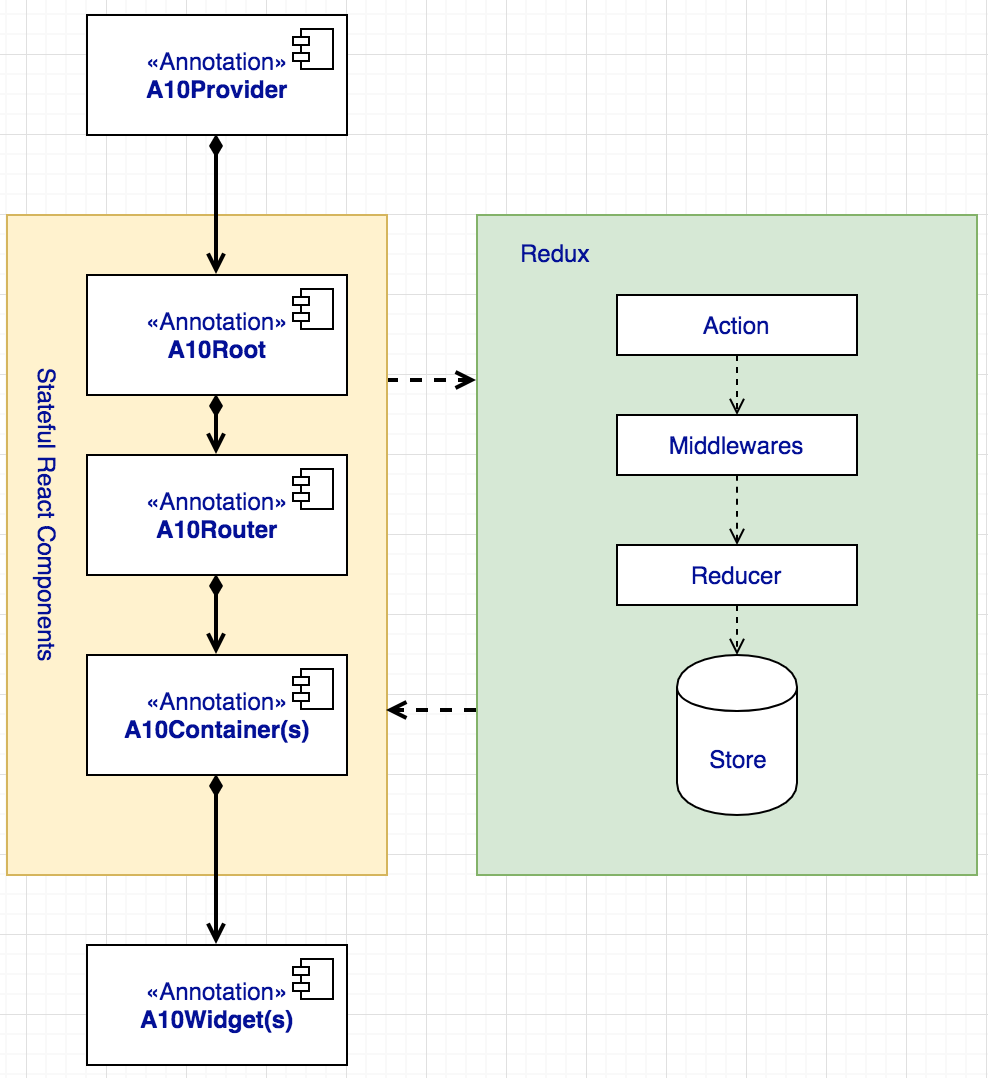


Figure 4 Framework work flow UML

This diagram depicts the main components of the framework, and which framework objects work with Redux (stateful).

## A10Provider

A10Provider is a wrapper for React-Redux Provider[[3]](#endnote-4). A10Provider makes the Redux store available to redux.connect() calls in the component hierarchy below it. We use this wrapper to wire up some global middleware to help system monitor global data (theme, debug) or to log critical information.

In addition, we may need some global actions and reducers to handle global attributes or behaviors.

### Object Definition and UML Diagram

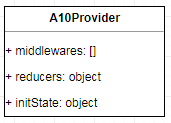


Figure 5 Object Definition and UML Diagram for A10Provider.

### Use Case

import middlewares from ‘approot/redux/middlewares’

import reducers from ‘approot/redux/reducers’

import { A10Provider } from ‘a10-gui-framework’

Const initState = {}

ReactDom.render(

<A10Provider

middlewares={middlewares}

reducers={reducers}

initState={initState}

config={}

>

</A10Provider>

, documentRoot)

## A10Root

We need a place where we can define some global variables like theme, debug toggle, locale etc. and pass them to the whole Virtual-DOM context. A10Root is the ideal place to accept initial configurations defined in a JS file and sent into 1) Redux, and 2) *component context* so that every descendent can access the global variables.

### Object Definition and UML Diagram

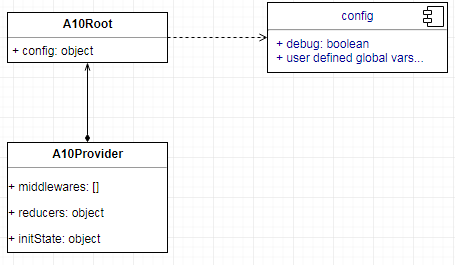


Figure 6 Object Definition and UML Diagram for A10Root

A10Root currently only has a prop named config, which is a JS object. We recommend that the JS object be defined at a specific location: approot/settings/configs.ts.

Currently the Framework defines only one environment variable ̶ debug, user can help himself debug A10Container or A10Component to show some debug log on each React Life Cycle, so the config file looks like:

export default {

debug: true,

user\_env\_var\_1: 'any',

user\_env\_var\_2: 'any',

….

}

In the future, by the framework enhancement, the framework will use more global variables like theme, locale, the config file looks like:

export default {

theme: 'dark',

locale: 'en\_US',

debug: true,

user\_env\_var\_1: 'any',

user\_env\_var\_2: 'any',

….

}

In the code, the user\_env\_var\_1 and user\_env\_var\_2 are framework user defined variables.

### Use Case

import configurations from ‘approot/settings/configs’

import { A10Root } from ‘a10-gui-framework’

ReactDom.render(<A10Provider middlewares={} reducers={} initState={} config={ configurations }>

</A10Provider>, documentRoot)

## A10Router

A10Router is a React Router[[4]](#endnote-5) wrapper, simplified from React Router for ease of use.

### Object Definition and UML Diagram

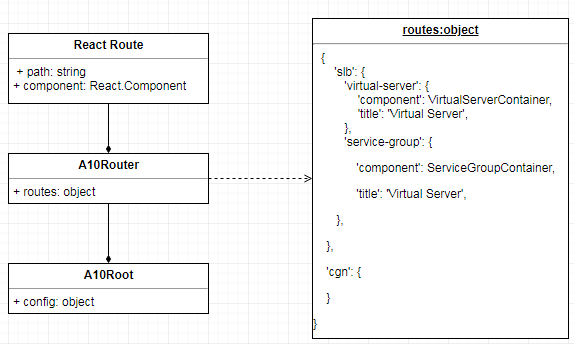


Figure 7 Object Definition and UML Diagram for A10Router.

### Use Case

import routes from ‘approot/settings/routes’

import { A10Router } from ‘a10-gui-framework’

ReactDom.render(<A10Provider middlewares={} reducers={} initState={} config={}>

<A10Router routes={routes}>

<Redirect from="messages/:id" to="/messages/:id" />

</A10Router>

</A10Provider>, documentRoot)

In this case, we imported the routes from app/settings/routes.tsx, the file looks like:

export default {

‘slb’: {

‘virtual-server’: {

component: THVirtualServerContainer,

params: [‘name’],

**index: true**,

title: ‘Virtual Server’,

template: {

component: THVirtualServerContainer,

title: ‘Virtual Server’,

params: [‘name’],

…

}

},

‘virtual-service’: {

component: THVirtualServiceContainer,

title: ‘Virtual Server’,

params: [‘ip’, ‘port’]

…

},

},

‘cgn’: {}

}

On above definition, the ‘slb’, ‘virtual-server’ will combine to a route path /slb/virtual-server, and the path will route to an A10Container called THVirtualServerContainer. params also is a Frameworks preserved option, with the params option, an example virtual-service URL looks may be of the form /slb/virtual-service/:ip/:port.

We will also support some react-router key features like Redirect Router and Index Router. They allow us to write our own routes directly inside of A10Router if necessary. For example we can put RedirectRoute inside A10Router, and Framework provides an index configuration to help user set IndexRoute. In the above example, /slb/virtual-server is an IndexRoute.

The title can be used for Menu. Framework provides a static method named A10Router.getMenu() to fetch the menu.

The routes can be defined recursively. See template in virtual-server in the above example. The object structure can be overridden by user. The key “component” is required, the rest are optional.

## A10Container

A10Container is the heavy-hitter. It is a stateful component which is connected to Redux store. A10Container contains page elements, and could include another A10Container, or A10Widgets, or external (external to this Framework) React Components.

A10Container is an abstract class that must be implemented.

A10Container can be routed. Whether it is a page container or a commonly–used container, just add it into router for it to be routed.

For reusability, we separate A10Container into two packages. The first part contains A10Containers that are specific to an Application (Thunder, aGalaxy, Harmony, etc); the second part contains sharable A10Containers, organized into a commonly-used application library. The name could be **a10-gui-thunder-com** for example.

For detailed definition about A10Container please see a separate documentation on “A10Container”.

### Class UML

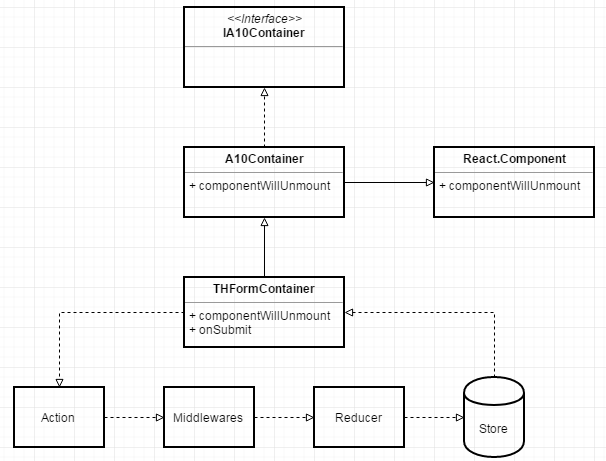


Figure 8 Object Definition and UML Diagram for THFormContainer.

This is the example UML for writing THFormContainer.

### Object Relation UML

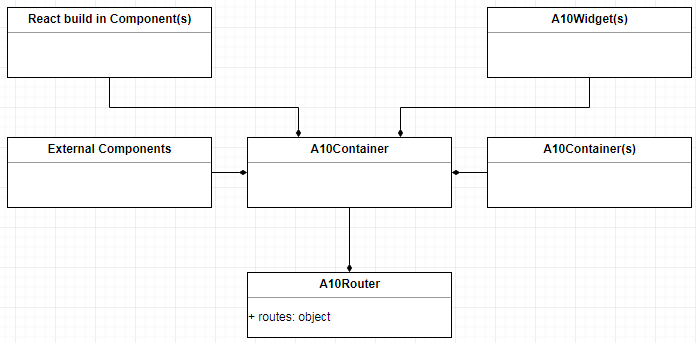


Figure 9 Object Relation UML for A10Container.

### Create Interface

setupA10Container(component, mapStateToProps={}, mapDispatchToProps={}, options={}) : function

### Create Container

import { setupA10Container } from ‘a10-gui-framework’

import { A10Button, A10Input, A10Form } from ‘a10-gui-widgets’

import { THFormContainer } from ‘a10-gui-thunder-com’

Class LoginContainer extends A10Container {

Render() {

return (

<THFormContainer>

<A10Form.Item><A10Input title=”username” /></A10Form.Item>

<A10Form.Item><A10Input title=”password” type=”password” /><A10Form.Item>

<A10Form.Item><A10Button>Submit</A10Button></A10Form.Item>

</THFormContainer>

)

}

}

function mapStateToProps(state) {

return { todos: state.todos }

}

function mapDispatchToProps(dispatch) {

return bindActionCreators(Object.assign({}, todoActionCreators, counterActionCreators), dispatch)

}

Const options = {

redux: { … },

isWithRouter: true,

autoPurge: true,

}

Export default setupA10Container(LoginContainer, mapStateToProps, mapDispatchToProps, options)

### Use Case

import { A10Router, A10Provider, A10Root } from ‘a10-gui-framework’

import LoginContainer from ‘./containers/LoginContainer’

const routes = {

‘login’: LoginContainer

}

ReactDom.render(<A10Provider middlewares={} reducers={} initState={} config={}>

<A10Router routes={routes}/>

</A10Provider>, documentRoot)

## A10 Widgets

A10 Widgets is a stateless React UI Component library. It is *not* connected with Redux store and is only concerned with UI ̶ i.e., they are mainly concerned with how things look. Stateless here means that it has nothing to do with data or business logic.

It is contained in A10Container or another A10Widget. Generally we can use any possible design pattern to build a A10Widget, example, we can use Adapter design pattern to adapt to Ant Design UI, or use Composite design pattern to compose several other React components or A10Widgets.

For a more in-depth treatment on A10Widget please refer to a separate documentation, “A10Widget Design”.

### Class UML

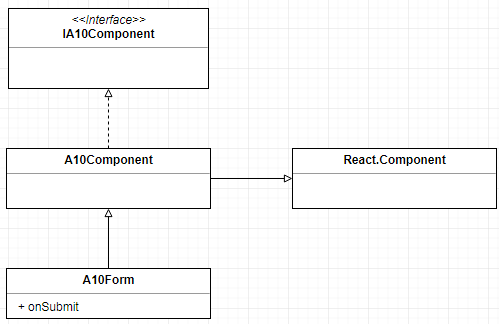


Figure 10 Object Definition and UML Diagram for A10Form.

This is the example UML for writing A10Form.

We extend all our widgets from A10Component which implements the interface IA10Component and extends from React.Component.

### Object Relation UML

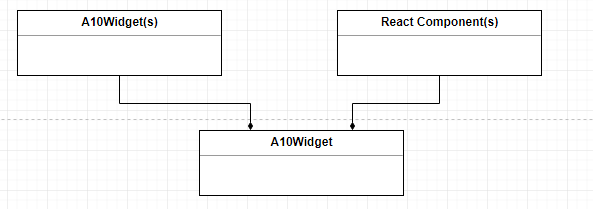


Figure 11 Object Relation UML for A10Widget.

### Create Widget

import { A10Component,createA10Widget } from ‘a10-gui-framework’

import { A10Button } from ‘a10-gui-widgets’

Class A10SubmitButton extends A10Component {

Render() {

return (

<div>

<A10Button>Submit</A10Button>

</div>

)

}

}

export default createA10Widget(A10SubmitButton)

In this trivial example, we wrap a <div> around an A10Button, which is also an A10Widget itself.

### Use Case

See above A10Container and Create Widget Case.

# Redux

## Data Categorization Based on Life Cycle

We separate our Redux store into distinct parts to better help developers distinguish how long their data can stay at the store.

## Data Life Cycle

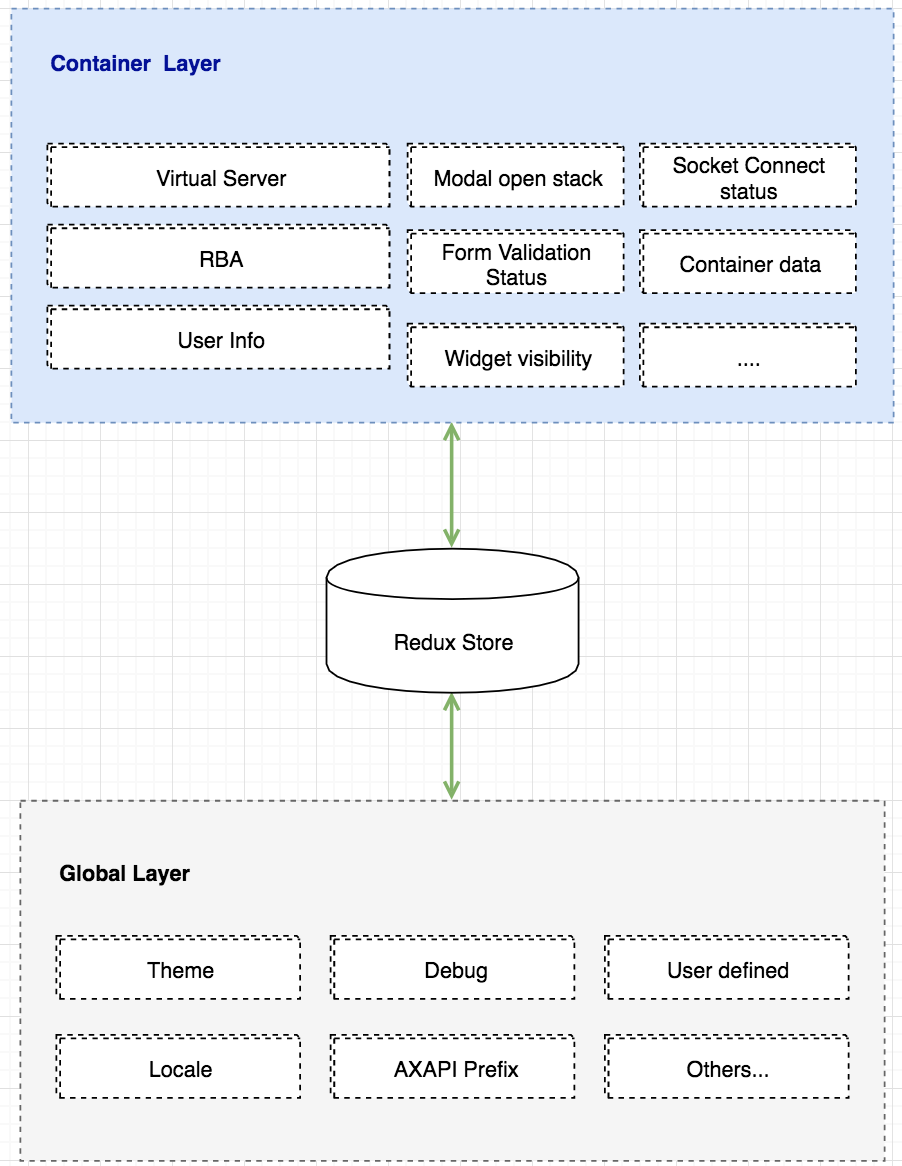


Figure 12 Two difference categories of data in Redux Store.

Figure 12 depicts two categories of data in Redux Store based on the data’s life cycle.

### Global Layer

Global Layer stores the global variables, they are two types of data that can be global layer data. These data will be kept in the Redux store until the application shuts down.

1. A10Root's config object, example the locale, theme, debug etc.
2. Some data we have to store as global, example, when crossing pages, some info have to stored globally

### Container Layer

A Container will control data in its own scope. It could be a sub-component’s data, or data from external source like from restful API, schema, or other data. The Container’s life cycle will be destroyed once the Container is unmounted.

## Action Type Namespace

As the Framework is being used, both the Framework itself and the Applications that make use of the Framework will implement many more Actions. Each action has to be given it an action type name and stored in a file, e.g., *redux/actionTypes.ts*. Its content may be the following:

export const TABLE\_LOADING = 'TABLE\_LOADING'; // Shows a loading indicator.

export const TABLE\_DATA = 'TABLE\_DATA'; // Stores the table data.

export const TABLE\_QUERY\_SORT = 'TABLE\_QUERY\_SORT'; // Sets a query parameter to sort the data.

export const TABLE\_LIMIT = 'TABLE\_QUERY\_LIMIT'; // Sets a limit on the number of records to return.

export const TABLE\_SEARCH = 'TABLE\_QUERY\_SEARCH'; // Sets a search term.

To better organize action type names, the Framework imports an action type name manager called **redux-action-namespacer**[[5]](#endnote-6). This NPM package allows developers to structure action types in a logical way:

*approot/actionTypes/actionTypes.js*

import { createA10ActionTypes } from 'a10-gui-framework'

const actionTypes = [

'table', [

'loading',

'data',

'query', [

'sort',

'limit',

'search'

]

]

];

export const ACTIONS = createA10ActionTypes(actionTypes);

In the above code a10-gui-framework exports a function createA10ActionTypes, this is an alias function to redux-action-namesapcer’s nsActionTypes. For more information about this function, please refer the official site.

Then, in your reducers file, you import these action types before using them in your reducers:

*approot/reducers/reducers.js*

import ACTIONS from 'approot/actionTypes/actionTypes.js';

const tablesReducer = (state = initialState, action) => {

switch (action.type) {

case ACTIONS.table.loading: // Here is where your action type structure works nicely.

...

}

}

export default tablesReducer;

# CSS Infrastructure

To make all GUI products have the unified looking and make CSS writing easy, share the CSS infrastructure became important. A10 GUI framework has a commonly-used Less CSS (Referred as Less) library allows you to customize some basic design aspects to meet the needs of UI diversity. UI units include primary color, border radius, border color, etc.

## About Less Pre-processor

Less is a CSS pre-processor, meaning that it extends the CSS language, adding features that allow variables, mixins, functions and many other techniques that allow you to make CSS that is more maintainable, theme-able and extensible.

Less runs inside Node, in the browser and inside Rhino. There are also many 3rd party tools that allow you to compile your files and watch for changes.

## A10 Predefined Less Library

On Framework side, we defined a global Less library (referred as library), which can be reused on each package, like a10-gui-widgets, applications, etc.

The library folder looks like the following:

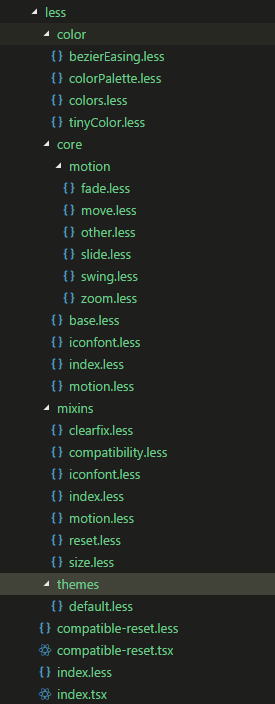


Figure Less Library Files

1. color: Defined color variables to help unify color across applications is an important aspect in the Framework.
2. core: It is mandatory to import this in every application. It contains base HTML styles (e.g., body, div, h1, …), CSS motions, icon fonts (i.e., Ant Icon), etc.
3. mixins: Less function utilities.
4. themes: Default theme Less files, basically you can copy and paste it into your application’s theme folders and modify Less files to define your own themes.
5. index.less: If we want use the entire Less library, it is recommended to import this file into your application.
6. index.tsx: Typescript way to import index.less. Webpack takes this file as the entry point.
7. compatible-reset.less: Resets all default CSS rendering behavior across browsers.
8. compatible-reset.tsx: Typescript way to import compatible-reset.less.

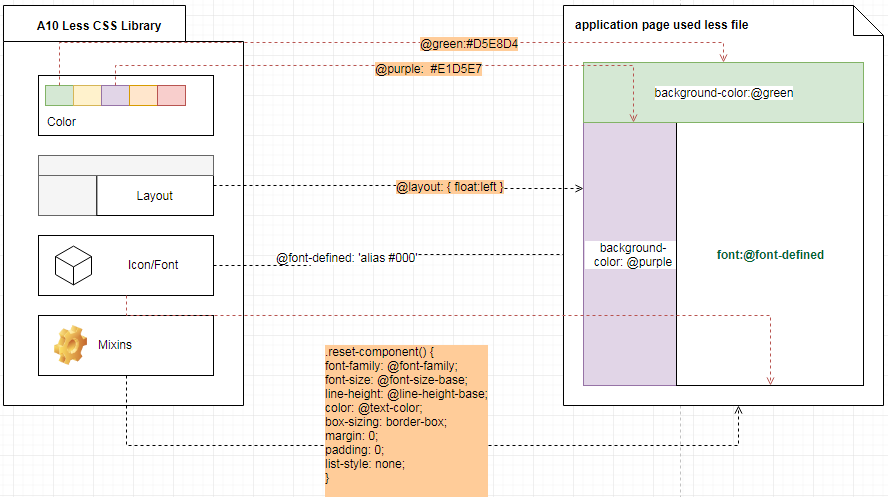


Figure Pre-defined less library use case

Example in your any default.less

@import "a10-gui-framework/less/color/colors";

.anyClassname {

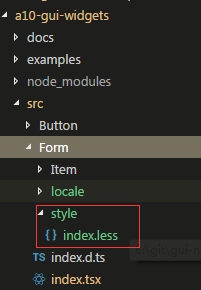
background-color: @purple-6

color: @green-1

}

## CSS In Projects

1. In A10 Widgets Library



For each of our Widgets folder, we need to define a style folder, and put all Less files inside it. Webpack will preprocess and bundle all the Less files into a widgets.bundle.css.

Then, when needed in an application, a single bundled CSS file is imported.

1. In Application

In Application, we can also import the Less library to write module’s Less, here pasted the application style related folders:

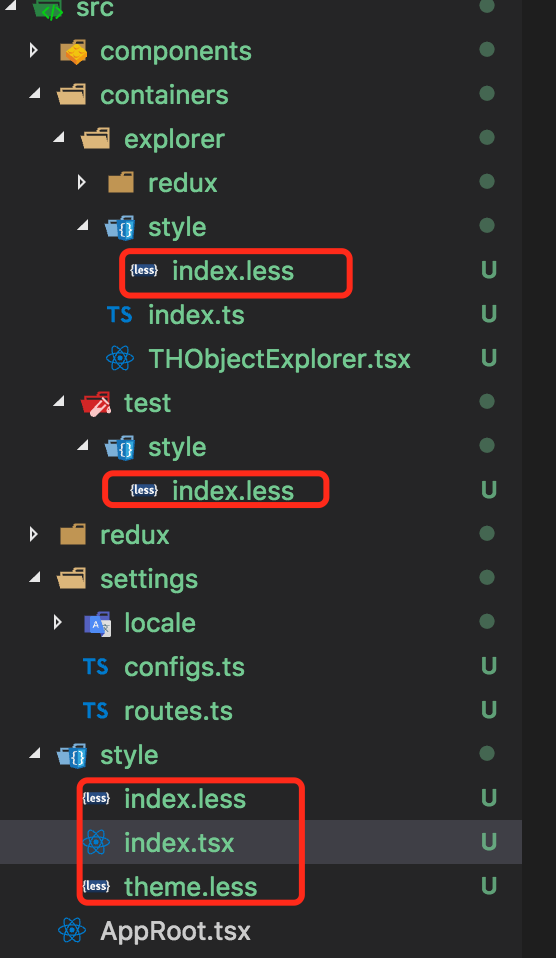


Figure Less in applicatoin folders

In above figure, our Webpack entry is index.tsx, theme.less could cover framework defined default theme. All the modules Less files should registered into index.less, example:

// approot/style/index.less

import '../containers/explorer/style/index.less'

import '../containers/test/style/index.less'

## Less Usage Example

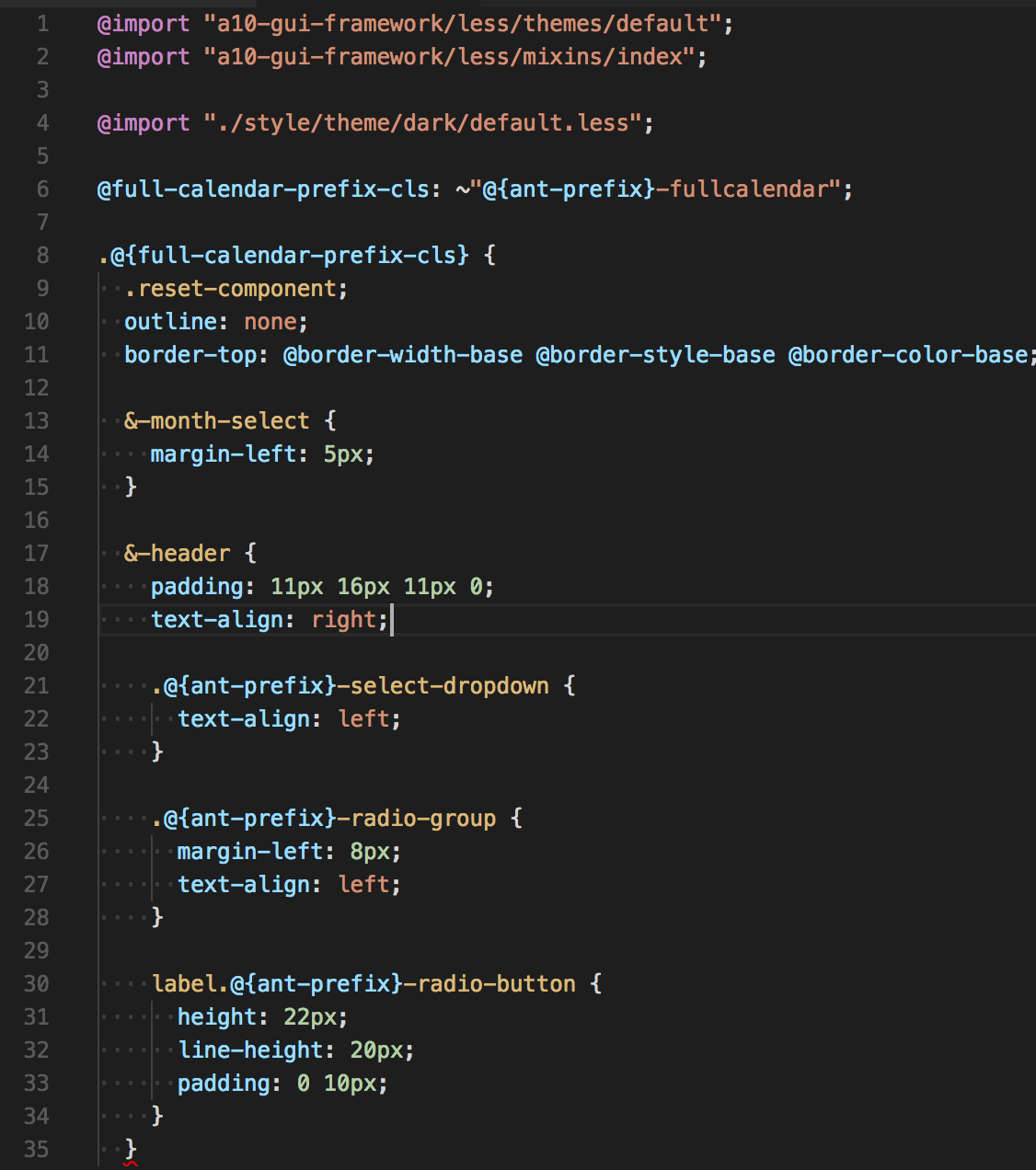


Figure Less Definition in your application

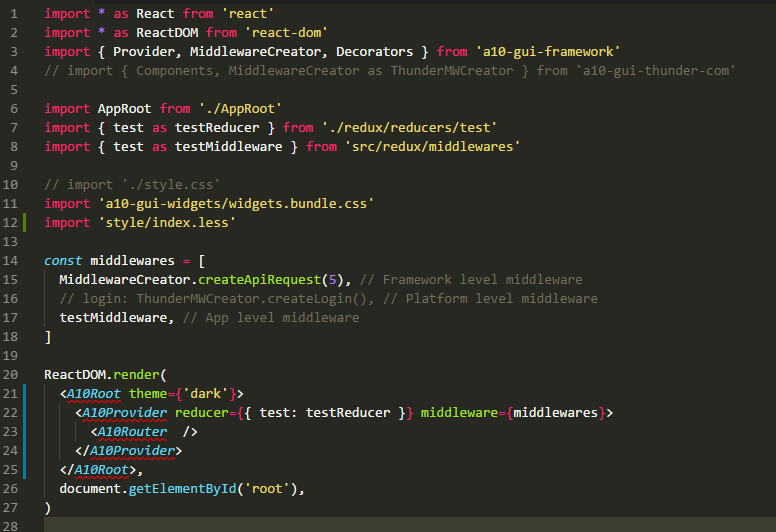


Figure Less file import into your code

After the Less file imported in Typescript, Webpack will extract those Less file and compress them into a CSS file, here I paste the Webpack settings

// webpack.­­config.js

{

…

plugins: [

new webpack.optimize.ModuleConcatenationPlugin(),

new ExtractTextPlugin({

filename: 'reactgui.bundle.css',

allChunks: true,

}),

…

}

[How to use styles after importing.]

# Appendix

## Technology Selection

Yarn(NPM), React.JS, Redux, AntDesign UI, Less, Webpack, TypeScript, Redux-action-namespacer, react-intl.

## Framework Usage Example

To illustrate how the Framework is used, we take thunder-explorer as an example.

### Folder Structures

*Note: By convention, in the project folder, there are two types of file extensions for Typescript files, namely .ts and .tsx. The .ts file extension means it is without JSX support, while the .tsx supports JSX.*

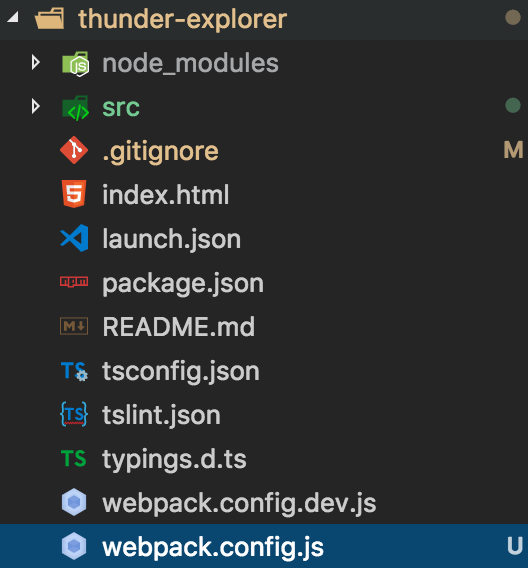


Figure 12 Root of the application

On above image, other than files for editor and Typescript settings, some important files/folders are listed below:

1. src (will show detail later)
2. index.html

The only HTML file in the application. The application JS bundle will be sourced into this file, and the application entry will be loaded and bootstrapped from here.

1. Webpack.config.js (for production) and webpack.dev.js (for development).

Settings for webpack to bundle all JS file into one file, the file will loaded in index.html (see above item 2). More detailed example will be provided later. We envision that the framework will provide a working example file and communicate later on how to customize it for application development.

1. package.json

Declares all external package dependencies and define application commands (npm-script) entries. For detail see NPM package.json[[6]](#endnote-7). More detailed example will be provided later. We envision that the framework will provide a working example file and communicate later on how to customize it for application development.

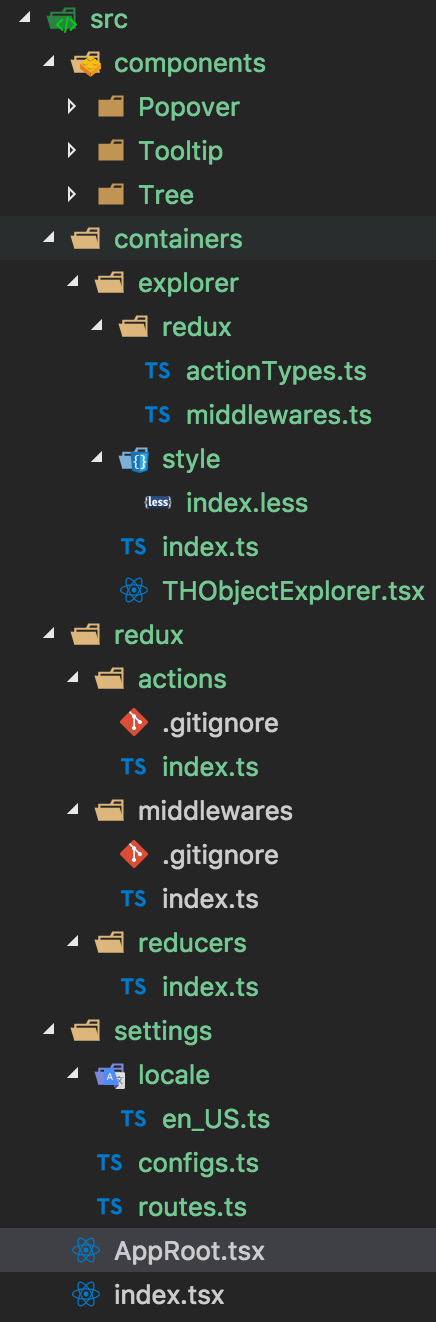


Figure 11 Source code of the application

The "src" folder is where all our source code resides:

1. Components

Application developer can put his/her own widgets and containers into this folder. For containers that should be shared with other applications, it is strongly advisable that they be moved into thunder-component package. For widgets, developers can contribute them to a10-gui-widgets.

1. Containers

Contains all modules stuff. Here we take explorer for example.

* 1. redux

Manages its own redux related files.

* 1. style (optional)

If we need to define some CSS, we can add our Less files here, the Less files can import Framework provided global functions and variables to be customized further.

* 1. index.ts

Exports all containers in this file.

* 1. THObjectExplorer.tsx

Defines the page container in this file, needs a router setting at src/settings/routes.ts so that page can be routed to this page container.

1. redux

Exports middlewares, reducers, and actions from all modules to the global scope.

1. settings
   1. locale (Not supported in the first release)

Defines language files following react-intl[[7]](#endnote-8)’s defined rules.

* 1. configs.ts (See A10Root)

Defines global environment variables. A10Root needs this to setup application runtime environment.

* 1. routes.ts (See A10Router)

Defines routes. A10Router needs this to setup application router.

1. index.tsx

This is the application’s bootstrapping Typescript file. All of A10Provider and A10Root should to be imported in this file. Example code in this file

import \* as React from 'react'

import \* as ReactDOM from 'react-dom'

import { A10Provider, A10Root} from 'a10-gui-framework'

import AppRoot from './AppRoot'

import reducers from './redux/reducers'

import middlewares from 'src/redux/middlewares'

import config from './settings/configs'

import './style.css'

import 'a10-gui-widgets/widgets.bundle.css'

ReactDOM.render(

<A10Provider reducers={reducers} middlewares={middlewares}>

<AppRoot config={configs} />

</A10Provider>,

document.getElementById('root'),

)

1. AppRoot.tsx

In index.tsx, we used AppRoot component, in AppRoot, we will import the A10Router to setup the application routes and A10Container(s) to implement our business.

import \* as React from "react"

import { A10Router, A10Container, setupA10Container } from "a10-gui-framework"

import { A10Layout, A10Menu, A10Icon, A10Breadcrumb } from "a10-gui-widgets"

const { A10SubMenu } = A10Menu

const { A10Header, A10Sider, A10Content } = A10Layout

import routes from "settings/routes"

interface IAppRoot {

explorer: any

}

class AppRoot extends A10Container<IAppRoot, any> {

constructor(props: any) {

super(props)

}

render() {

const { explorer } = this.props

const { breadcrumbs } = explorer

return (

<A10Layout>

<A10Layout className="layout">

<A10Header>

<h1>Thunder</h1>

</A10Header>

</A10Layout>

<Sider>

<div>left side bar</div>

</Sider>

<A10Layout>

<A10Breadcrumb>

{breadcrumbs.map((crumb: string, key: any) => {

return <A10Breadcrumb.Item key={key}>{crumb}</A10Breadcrumb.Item>

})}

</A10Breadcrumb>

<A10Content>

<A10Router routes={routes} />

</A10Content>

</A10Layout>

</A10Layout>

)

}

}

function mapStateToProps(state: any) {

return {

explorer: state.explorer.toJS()

}

}

function mapDispatchToProps(dispatch: any) {

return {}

}

export default setupA10Container(AppRoot, mapStateToProps, mapDispatchToProps)

## Future Enhancement (Not in the first release)

### Theme

We talk about the Theme from React side, not CSS side.

In A10Root, the passed in props which we can defined a theme variable to initial the theme, the variable also will stored into Redux store, so we can access it in any stateful containers, also we can use React context way to pass it to stateless components like A10Widget. Here is the theme data flow diagram:

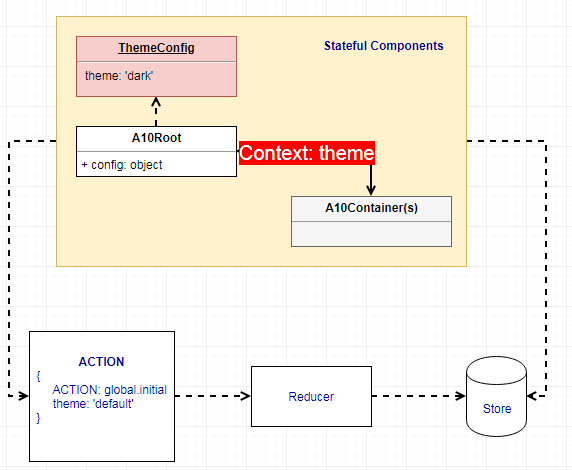
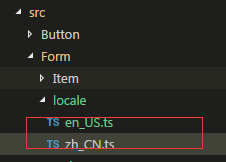


Figure Theme data flow

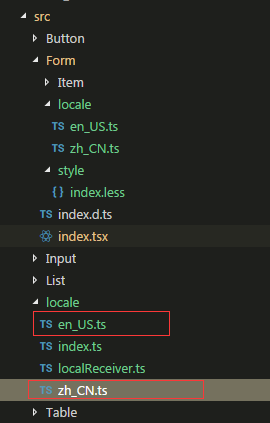
### Locale

To implement localization, we should collect all of a language’s translations into a single file. The system then provides a mechanism to transform them to the selected language at run-time.

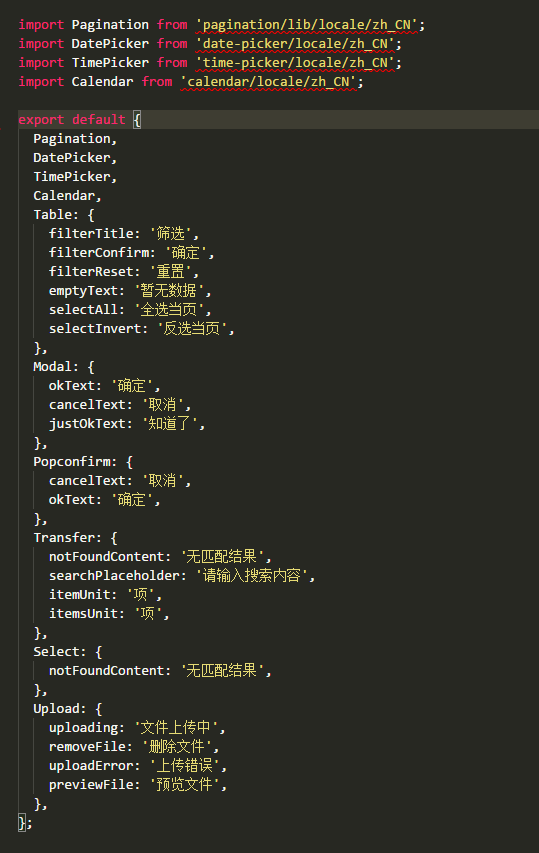
On Widget layer, if it is needed to be translate some words into other languages, we should have a folder named locale under your Widget lib, like below:



Those widgets’ locale files should also import into upper layer folder with the same names. This upper layer locale file includes localization content for all widgets.



The localization file looks like the following:



#### Using Localization

GUI Framework uses react-intl to manage localization and export this library as A10Intl which provides React components from react-intl and related APIs to format dates, numbers, and strings, including pluralization and handling translations. A10Provider comprises IntlProvider so that developers do not need to integrate IntlProvider themselves. Just passing the settings intl as part of global config to A10Provider.

**function** **changeLocale**(locale: string, url: string) **void**



### Plugin Manager

Extending the create interface using Command design pattern, it goes a long way to enhance our framework layer components.

### Redux Related Code

Providing more reusable utilities functions, code, and redux relevant actions/epics/reducers for each A10 packages and apps.

More details please refer to:

<https://a10networks.sharepoint.com/:w:/r/sites/UGF/_layouts/15/Doc.aspx?action=edit&sourcedoc=%7BAF24566D-BCD2-4EED-AB4C-3E8DE2AA2200%7D>

1. <https://reactjs.org/> [↑](#endnote-ref-2)
2. <https://redux.js.org/> [↑](#endnote-ref-3)
3. <https://github.com/reactjs/react-redux/blob/master/docs/api.md#provider-store> [↑](#endnote-ref-4)
4. <https://github.com/ReactTraining/react-router/tree/v3/docs> [↑](#endnote-ref-5)
5. <https://www.npmjs.com/package/redux-action-namespacer> [↑](#endnote-ref-6)
6. <https://docs.npmjs.com/files/package.json> [↑](#endnote-ref-7)
7. <https://github.com/yahoo/react-intl> [↑](#endnote-ref-8)