<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>Lugia Mega</title>

<script>

(function() {

if (!process.env.HOT) {

const link = document.createElement('link');

link.rel = 'stylesheet';

link.href = './style.css';

// HACK: Writing the script path should be done with webpack

document.getElementsByTagName('head')[0].appendChild(link);

}

})();

</script>

</head>

<body>

<div id="root"></div>

<script>

{

const scripts = [];

// Dynamically insert the DLL script in development env in the

// renderer process

if (process.env.NODE\_ENV === 'development') {

scripts.push('../dll/renderer.dev.dll.js');

}

// Dynamically insert the bundled app script in the renderer process

const port = process.env.PORT || 1212;

scripts.push(

process.env.HOT

? 'http://localhost:' + port + '/dist/renderer.dev.js'

: './renderer.prod.js'

);

document.write(

scripts

.map(script => `<script defer src="${script}"><\/script>`)

.join('')

);

}

</script>

</body>

</html>

import React from 'react';  
import { render } from 'react-dom';  
import { AppContainer } from 'react-hot-loader';  
import Root from './windows/MainRoot';  
import { configureStore, history } from './models/configureStore';  
import { getHistoryProjects } from './services/api.sync';  
import '@lugia/lugia-web/dist/css/global.css';  
import './utils/customScrollbar.css';  
const store = configureStore({  
 historyProjects: getHistoryProjects()  
});  
render(  
 <AppContainer>  
 <Root store={store} history={history} />  
 </AppContainer>,  
 document.getElementById('root')  
);  
if (module.hot) {  
 module.hot.accept('./windows/MainRoot', () => {  
 const NextRoot = require('./windows/MainRoot'); // eslint-disable-line global-require  
 render(  
 <AppContainer>  
 <NextRoot store={store} history={history} />  
 </AppContainer>,  
 document.getElementById('root')  
 );  
 });  
}

import { join } from 'path';

import build from '@lugia/mega-scripts/lib/utils/build';

// 此函数执行时需要设置环境变量

// 无论是在父进程还是子进程

function singleCompile(cwd, entry, name = '\_\_SINGLE\_COMPILED\_\_') {

const applyConfig = config => ({

...config,

commons: [],

html: null,

hash: null,

manifest: null

// disableCssExtract: true

});

const applyWebpack = (webpackConfig, { merge }) =>

merge(webpackConfig, {

output: {

path: join(cwd, './.lugia/temp'),

library: name,

libraryTarget: 'window'

// filename: '[name].js'

},

externals: {

react: 'React',

'react-dom': 'ReactDom',

'@lugia/lugiax': 'lugiax'

}

});

return build({

cwd,

entry,

useMemoryFS: true,

applyConfig,

applyWebpack

});

}

function send(msg) {

process.send({ SINGLE\_COMPILED: true, ...msg });

}

process

.on('message', m => {

if (m && m.SINGLE\_COMPILE) {

const { cwd, entry, name } = m;

singleCompile(cwd, entry, name)

.then(({ assets, warnings }) => {

send({

warnings,

assets

});

return assets;

})

.catch(error => {

send({ error });

});

}

})

.on('exit', code => {

// 退出事件

if (code === 0) {

send({ info: 'exit' });

} else {

send({ error: `exit code ${code}` });

}

})

.on('error', error => {

// 报错事件

send({ error: `onError: ${error.message}` });

})

.on('SIGHUP', () => {

// 程序停止信号

send({ info: 'SIGHUP' });

})

.on('SIGTERM', () => {

// kill 默认参数信号

send({ info: 'SIGTERM' });

})

.on('SIGINT', () => {

// Ctrl + c 信号

send({ info: 'SIGINT' });

})

.on('uncaughtException', error => {

// 未捕获异常

send({ error: `onUncaughtException: ${error.message}` });

});

program

.version(pkg.version)

.description(

'Quickly create react apps. Based on redux, lugiax and react-router.',

)

.command(

'create <app-name> [options]',

'create a new application from a scaffolding',

)

.command('list', 'list available official scaffoldings')

.command(

'generate <type> [options]',

'generate pages / components / models to application',

)

.alias('g')

.command(

'build [entry] [options]',

'build a .js or .jsx file in production mode with zero config',

)

.command(

'serve [options] [entry]',

'serve a .js or .jsx file in development mode with zero config',

);

program.on('--help', () => {

console.log('');

console.log(

` All commands can be run ${chalk.green(

'mega <command> --help',

)} (or -h) for more information.`,

);

});

program.on('command:\*', cmd => {

cmd = cmd[0] || 'help';

if (allCommands.includes(cmd)) return;

program.outputHelp();

console.log('');

console.log(

` ${chalk.red('Unknown command')} ${chalk.yellow(cmd)}${chalk.red('.')}`,

);

console.log('');

});

process.on('SIGINT', () => {

if (program.runningCommand) program.runningCommand.kill('SIGKILL');

process.exit(0);

});

function validateBoolOption(name, value, defaultValue) {

if (typeof value === 'undefined') {

value = defaultValue;

}

if (typeof value !== 'boolean') {

throw new Error(

`@lugia/babel-preset-mega: '${name}' option must be a boolean.`,

);

}

return value;

}

const getCacheScaffoldingPath = post =>

join(CacheScaffoldingDir, post.replace(/[\/:]/g, '-').replace(/-+/g, '-'));

function fileFilter(appPath, scaffoldingPath, verbose) {

return through.obj(function(file, enc, cb) {

// file => https://github.com/gulpjs/vinyl

if (!file.stat.isFile()) {

return cb();

}

if (verbose) {

logger.create(file.path.replace(`${scaffoldingPath}/`, ''));

}

this.push(file);

cb();

});

}

function error(msg) {

logger.br();

logger.error(msg);

logger.br();

process.exit(1);

}

function printSuccess() {

const displayedCommand = useYarn ? 'yarn' : 'npm';

console.log(`

${chalk.green('Success!')} Created ${chalk.green(appName)} at ${chalk.green(

appPath,

)}

Inside that directory, you can run several commands:

${chalk.cyan(`${displayedCommand} start`)}

Starts the development server.

${chalk.cyan(`${displayedCommand} ${useYarn ? '' : 'run '}build`)}

Bundles the app into static files for production.

${chalk.cyan(`${displayedCommand} test`)}

Starts the test runner.

We suggest that you begin by typing:

${chalk.cyan(`cd ${appPath}`)}

${chalk.cyan(`${displayedCommand} start`)}

Happy hacking!

`);

}

// scaffolding (generate) => appPath

function g(appPath, scaffoldingPath) {

logger.info(`Creating a new React app in ${chalk.green(appPath)}.`);

vfs

.src(['\*\*/\*', '!node\_modules/\*\*/\*'], {

cwd: scaffoldingPath,

cwdbase: true,

dot: true,

})

.pipe(fileFilter(appPath, scaffoldingPath, verbose))

.pipe(vfs.dest(appPath))

.on('end', () => {

const gitignorePath = join(appPath, 'gitignore');

if (existsSync(gitignorePath)) {

verbose && logger.info('rename gitignore -> .gitignore'); // eslint-disable-line

renameSync(gitignorePath, join(appPath, '.gitignore'));

}

if (autoInstall) {

install({

cwd: appPath,

useYarn,

verbose,

})

.then(printSuccess)

.catch(e => error(e));

} else {

printSuccess();

}

})

.resume();

}

function generate(program, { cwd }) {

const defaultBase = 'src';

const rc = getBabelRc(cwd);

const base = program.base || rc.base || defaultBase;

const defaultEntry = `${base}/index.js`;

const defaultRouter = `${base}/router.js`;

const [type, name] = program.args;

try {

switch (type) {

case 'model':

(() => {

const modelPath = `./models/${name}`;

const filePath = `${base}/models/${name}.js`;

const entry = program.entry || defaultEntry;

info('create', `model ${name}`);

info('register', `to entry ${entry}`);

api('models.create', {

namespace: name,

sourcePath: cwd,

filePath,

entry,

modelPath,

});

})();

break;

case 'route':

(() => {

const componentName = upperCamelCase(name);

const componentPath = `${base}/routes/${componentName}.js`;

const componentCSSPath = `${base}/routes/${componentName}.css`;

const withCSS = program.css ? `, ${componentCSSPath}` : '';

info('create', `routeComponent ${componentPath}${withCSS}`);

api('routeComponents.create', {

sourcePath: cwd,

filePath: componentPath,

componentName,

css: program.css,

});

info('create', `route ${name} with ${componentPath}`);

api('router.createRoute', {

filePath: program.router || defaultRouter,

sourcePath: cwd,

path: `/${name}`,

component: {

componentName,

filePath: componentPath,

},

});

})();

break;

case 'component':

(() => {

const fileName = basename(name);

const fileDir = dirname(name);

const componentName = upperCamelCase(fileName);

const filePath = join(

`${base}/components`,

fileDir,

`${componentName}.js`,

);

const componentCSSPath = join(

`${base}/components`,

fileDir,

`${componentName}.css`,

);

const withCSS = program.css ? `, ${componentCSSPath}` : '';

info('create', `component ${filePath}${withCSS}`);

api('components.create', {

sourcePath: cwd,

filePath,

componentName,

css: program.css,

});

})();

break;

default:

error(`ERROR: uncaught type ${type}`);

break;

}

} catch (e) {

error(e.stack);

}

}

export default function getDefaultEntry(cwd) {

if (existsSync(join(cwd, './main.js'))) {

return './main.js';

}

if (existsSync(join(cwd, './index.js'))) {

return './index.js';

}

if (existsSync(join(cwd, './App.jsx'))) {

return './App.jsx';

}

console.log(`

Failed to locate entry file in ${cwd}.

Valid entry file should be one of: main.js, index.js or App.jsx.

`);

process.exit(1);

}

export default async function install({

cwd,

useYarn = false,

dependencies = [],

verbose,

}) {

assert(

is.array(dependencies),

`The dependencies config must be Array, but got ${dependencies}`,

);

useYarn = useYarn && canUseYarn(cwd, true);

const isonline = await isOnline();

const isEmptyDependencies = is.empty(dependencies);

return new Promise((resolve, reject) => {

let command;

let args;

if (useYarn) {

[command] = Object.keys(getYarnResolved());

args = [];

if (!isEmptyDependencies) {

args.push('add', ...dependencies, '--exact');

}

if (!isonline) {

args.push('--offline');

logger.warn(

'You appear to be offline. Falling back to the local Yarn cache.',

);

logger.br();

}

args.push('--cwd', cwd);

} else {

command = 'npm';

args = ['install'];

if (!isEmptyDependencies) {

args.push(...dependencies, '--save', '--save-exact');

}

args.push('--loglevel', 'error');

}

if (verbose) {

args.push('--verbose');

}

if (isEmptyDependencies) {

logger.loading(

`Installing all the dependencies using ${chalk.green(

command,

)}. This might take a couple of minutes`,

);

} else {

logger.loading(

`Installing ${dependencies.map(d => chalk.cyan(d)).join(', ')}}`,

);

}

logger.br();

runCommand(command, args, code => {

if (code !== 0) {

reject(new Error(`${command} ${args.join(' ')}`));

} else {

resolve({

code,

useYarn,

command,

args,

});

}

});

});

}

export function runCommand(command, args = [], close) {

const child = spawn(command, args, {

// keep color

stdio: 'inherit',

env: process.env,

});

child.on('close', code => close && close(code));

}

export function isOnline() {

const registry = 'registry.yarnpkg.com';

return new Promise(resolve =>

lookup(

registry,

{

all: true,

},

(err, address) => resolve(is.empty(err) && !is.empty(address)),

),

);

}

export function isLocalScaffolding(scaffolding) {

return /^[./]|(^[a-zA-Z]:)/.test(scaffolding);

}

export function getScaffoldingPath(scaffolding, cwd = process.cwd()) {

return isAbsolute(scaffolding)

? scaffolding

: normalize(join(cwd, scaffolding));

}

program.Command.prototype.missingArgument = function(name) {

this.outputHelp();

console.log();

console.log(

` ${chalk.red('Missing required argument')} ${chalk.yellow(

`<${name}>`,

)}${chalk.red('.')}`,

);

console.log();

process.exit(1);

};

export default () => {

program.\_args.forEach((arg, i) => {

if (arg.required && program.args[i] == null) {

program.missingArgument(arg.name);

}

});

};

export default function singleBuild(

entry = getDefaultEntry(cwd),

{ target = 'app', libName, dest = 'dist', sourceMap },

) {

const { name: entryName } = parse(entry);

const isLib = target === 'lib';

const name = libName || entryName;

const applyConfig = config => {

if (config.html.\_fromMegaScriptsDefault) {

config.html = {};

}

if (isLib) {

return {

devtool: sourceMap && 'source-map',

...config,

commons: [],

html: {

template: join(\_\_dirname, '../templates/html.ejs'),

title: 'Lugia Mega Demo',

filename: 'demo.html',

...config.html,

},

manifest: null,

};

}

return {

devtool: sourceMap && 'source-map',

...config,

html: {

template: join(\_\_dirname, '../templates/html.ejs'),

title: 'Lugia Mega App',

minify: {

collapseWhitespace: true,

removeComments: true,

removeRedundantAttributes: true,

removeScriptTypeAttributes: true,

removeStyleLinkTypeAttributes: true,

useShortDoctype: true,

},

...config.html,

},

};

};

const applyWebpack = (webpackConfig, { merge }) => {

return merge(webpackConfig, {

output: {

path: join(cwd, dest),

...(isLib

? {

library: name,

libraryTarget: 'umd',

filename: `${name}.umd.js`,

}

: {}),

},

resolve: {

alias: {

react: require.resolve('react', [cwd, join(\_\_dirname)]),

'react-dom': require.resolve('react-dom', [cwd, join(\_\_dirname)]),

},

},

});

};

build({

cwd,

entry,

applyConfig,

applyWebpack,

});

}

export default function singleBuild(

entry = getDefaultEntry(cwd),

{ open = false, sync = false, copy },

) {

const applyConfig = config => {

if (config.html.\_fromMegaScriptsDefault) {

config.html = {};

}

return {

...config,

html: {

template: join(\_\_dirname, '../templates/html.ejs'),

title: 'Lugia Mega App',

...config.html,

},

};

};

const applyWebpack = (webpackConfig, { merge }) => {

return merge(webpackConfig, {

resolve: {

alias: {

react: require.resolve('react', [cwd, join(\_\_dirname)]),

'react-dom': require.resolve('react-dom', [cwd, join(\_\_dirname)]),

},

},

});

};

dev({

cwd,

entry,

applyConfig,

applyWebpack,

\_cliEnv: {

BROWSER: open,

BROWSER\_SYNC: sync,

},

onOpenPort({ urls }) {

if (copy) {

clipboardy.writeSync(urls.localUrlForBrowser);

}

},

});

}

export default {

process(src, filename) {

return `module.exports = ${JSON.stringify(path.basename(filename))};`;

},

};

export default function(context, opts = {}) {

return {

presets: [

[

require.resolve('@lugia/babel-preset-mega'),

{

corejs: false,

helpers: false,

...opts,

},

],

],

};

}

export default function(opts = {}, cb) {

const {

cwd = process.cwd(),

watch,

entry,

useMemoryFS,

applyWebpack,

applyConfig,

configFile,

} = opts;

const babel = resolve(\_\_dirname, './babel.js');

const paths = getPaths(cwd);

return new Promise((resolve, reject) => {

// register babel for config files

registerBabel(babel, {

cwd,

configOnly: true,

});

// get user config

const { config } = getUserConfig({

cwd,

configFileName: configFile || CONFIG\_FILE\_NAME,

});

debug(`user config: ${JSON.stringify(config)}`);

// get webpack config

const webpackConfig = applyWebpackConfig(

applyWebpack,

getWebpackConfig(

{

cwd,

config,

babel,

paths,

entry,

},

applyConfig,

),

);

build({

webpackConfig,

watch,

useMemoryFS,

success: ({ stats, warnings, assets }) => {

if (cb) {

cb(undefined, { stats, warnings, assets });

}

resolve({ warnings, assets });

},

fail: err => {

if (cb) {

cb(err);

}

reject(err);

},

});

});

}

export default function runDev(opts = {}) {

const {

cwd = process.cwd(),

entry,

applyWebpack,

applyConfig,

onOpenPort,

configFile,

\_cliEnv = {},

} = opts;

const babel = resolve(\_\_dirname, './babel.js');

const paths = getPaths(cwd);

// register babel for config files

registerBabel(babel, {

cwd,

configOnly: true,

});

let isFirstCompile = true;

let bs = null;

function initBrowserSync({

appName,

urls,

HOST,

PROTOCOL,

cwd,

disableBrowserSync,

autoOpenBrowser = true,

}) {

disableBrowserSync =

process.env.BROWSER\_SYNC === 'none' ? true : disableBrowserSync;

autoOpenBrowser = process.env.BROWSER === 'none' ? false : autoOpenBrowser;

debug('disableBrowserSync', disableBrowserSync);

debug('autoOpenBrowser', autoOpenBrowser);

debug('isFirstCompile', isFirstCompile);

if (disableBrowserSync || !isFirstCompile) return;

if (browserSync.has(appName)) {

chalk.red(`[BROWSER\_SYNC] This project (${appName}) is using it.\n`);

return;

}

bs = browserSync.create(appName || undefined);

detect(DEFAULT\_BROWSER\_SYNC\_PORT).then(

port => {

debug('localUrlForBrowser', urls.localUrlForBrowser);

debug('BROWSER\_SYNC\_PORT', port);

bs.init({

open: autoOpenBrowser,

// ui: false,

notify: false,

proxy: {

target: urls.localUrlForBrowser,

ws: true,

},

cwd,

port,

});

if (onOpenPort) {

const urls = prepareUrls(PROTOCOL, HOST, port);

onOpenPort(

{

port,

urls,

appName,

HOST,

PROTOCOL,

},

'BROWSER\_SYNC',

);

}

},

err => {

chalk.red(

`[BROWSER\_SYNC] Could not find an open port.\nNetwork error message: ${err.message ||

err}\n`,

);

},

);

isFirstCompile = false;

}

// get user config

let config = null;

let userPKG = null;

let returnedWatchConfig = null;

try {

({ config, userPKG, watch: returnedWatchConfig } = getUserConfig({

cwd,

configFileName: configFile || CONFIG\_FILE\_NAME,

}));

debug(`user config: ${JSON.stringify(config)}`);

} catch (e) {

console.error(chalk.red(e.message));

debug(`Get ${CONFIG\_FILE\_NAME} config failed, watch config and reload`);

watchConfigs({ cwd, configFileName: CONFIG\_FILE\_NAME }).on(

'all',

(event, path) => {

debug(`[${event}] ${path}, unwatch and reload`);

bs && bs.active && bs.exit(); // eslint-disable-line

bs = null;

unwatchConfigs();

runDev(opts);

},

);

return;

}

// get webpack config

const webpackConfig = applyWebpackConfig(

applyWebpack,

getWebpackConfig(

{

cwd,

config,

babel,

paths,

entry,

},

applyConfig,

),

);

const {

openBrowser: autoOpenBrowser = \_cliEnv.BROWSER,

disableBrowserSync = is.undefined(\_cliEnv.BROWSER\_SYNC)

? undefined

: !\_cliEnv.BROWSER\_SYNC,

} = config;

dev({

webpackConfig,

userPKG,

autoOpenBrowser,

index: config.html && config.html.filename,

port: config.port,

proxy: config.proxy || {},

historyApiFallback: config.historyApiFallback,

beforeMiddleware(app) {

// This service worker file is effectively a 'no-op' that will reset any

// previous service worker registered for the same host:port combination.

// We do this in development to avoid hitting the production cache if

// it used the same host and port.

// https://github.com/facebookincubator/create-react-app/issues/2272#issuecomment-302832432

app.use(noopServiceWorkerMiddleware());

},

beforeServer(devServer) {

try {

applyMock(devServer);

} catch (e) {

console.log(e);

}

},

afterServer(devServer, urlsInfo) {

if (onOpenPort) {

onOpenPort(urlsInfo, 'DEV\_SERVER');

}

returnedWatchConfig(devServer);

},

onCompileDone({ urls, appName, HOST, PROTOCOL }) {

if (isFirstCompile) {

initBrowserSync({

urls,

appName,

HOST,

PROTOCOL,

cwd,

autoOpenBrowser,

disableBrowserSync,

});

isFirstCompile = false;

} else {

bs && bs.active && bs.reload(); // eslint-disable-line

}

},

});

}

export default function(opts = {}) {

const { cwd = '', entry, isBuild } = opts;

let entryObj = null;

if (is.undefined(entry)) {

entryObj = getEntry(getExistsDefaultEntry(cwd));

} else if (is.string(entry)) {

const files = getFiles(entry, cwd);

entryObj = getEntries(files);

} else if (is.array(entry)) {

const files = entry.reduce((memo, entryItem) => {

return memo.concat(getFiles(entryItem, cwd));

}, []);

entryObj = getEntries(files);

} else if (is.plainObject(entry)) {

entryObj = entry;

} else {

throw new Error(

`entry should be String, Array or Plain Object, but got ${entry}`,

);

}

if (!isBuild) {

entryObj = Object.keys(entryObj).reduce((memo, key) => {

return !is.array(entryObj[key])

? {

...memo,

[key]: [webpackHotDevClientPath, entryObj[key]],

}

: {

...memo,

[key]: entryObj[key],

};

}, {});

}

// add setPublicPath

if (process.env.SET\_PUBLIC\_PATH) {

const setPublicPathFile = join(

\_\_dirname,

'../template/setPublicPath.tpl.js',

);

entryObj = Object.keys(entryObj).reduce((memo, key) => {

return {

...memo,

[key]: [

setPublicPathFile,

...(is.array(entryObj[key]) ? entryObj[key] : [entryObj[key]]),

],

};

}, {});

}

return entryObj;

}

function getEntry(filePath) {

const key = basename(filePath).replace(/\.(j|t)sx?$/, '');

return {

[key]: filePath,

};

}

function getFiles(entry, cwd) {

const files = glob.sync(entry, {

cwd,

});

return files.map(file => {

return file.charAt(0) === '.' ? file : `.${sep}${file}`;

});

}

function getEntries(files) {

return files.reduce((memo, file) => {

return {

...memo,

...getEntry(file),

};

}, {});

}

function getExistsDefaultEntry(cwd) {

if (existsSync(join(cwd, './src/index.js'))) {

return './src/index.js';

}

if (existsSync(join(cwd, './src/index.jsx'))) {

return './src/index.jsx';

}

if (existsSync(join(cwd, './src/index.ts'))) {

return './src/index.ts';

}

if (existsSync(join(cwd, './src/index.tsx'))) {

return './src/index.tsx';

}

// default

return './src/index.js';

}

function resolveOwn(relativePath) {

return resolve(\_\_dirname, relativePath);

}

export default function(cwd) {

const appDirectory = realpathSync(cwd);

function resolveApp(relativePath) {

return resolve(appDirectory, relativePath);

}

return {

appBuild: resolveApp('dist'),

appPublic: resolveApp('public'),

appPackageJson: resolveApp('package.json'),

appSrc: resolveApp('src'),

appNodeModules: resolveApp('node\_modules'),

ownNodeModules: resolveOwn('../../node\_modules'),

resolveApp,

resolveOwn,

appDirectory,

};

}

export default function(opts = {}, applyConfig) {

const { cwd, config, babel, paths, entry } = opts;

const browserslist = config.browserslist || defaultBrowsers;

debug(`babel: ${babel}`);

debug(`browserslist: ${browserslist}`);

if (!config.html) {

config.html = { \_fromMegaScriptsDefault: true };

const appPublicHtmlPath = resolve(paths.appPublic, 'index.html');

if (existsSync(appPublicHtmlPath)) {

config.html.template = appPublicHtmlPath;

}

const appSrcHtmlPath = resolve(paths.appSrc, 'index.html');

if (existsSync(appSrcHtmlPath)) {

config.html.template = appSrcHtmlPath;

}

if (!isDev) {

config.html.minify = {

collapseWhitespace: true,

removeComments: true,

removeRedundantAttributes: true,

removeScriptTypeAttributes: true,

removeStyleLinkTypeAttributes: true,

useShortDoctype: true,

};

}

}

if (!is.function(applyConfig)) {

applyConfig = c => c;

}

return getConfig(

applyConfig({

cwd,

hash: true,

manifest: {},

commons: [

{

name: 'vendors',

minChunks(module) {

// 把node\_modules中的模块提取到vendors.js中

return (

module.resource &&

(/\.js$/.test(module.resource) ||

/\.css$/.test(module.resource)) &&

module.resource.indexOf('node\_modules') > -1

);

},

},

],

...config,

entry: getEntry({

cwd: paths.appDirectory,

entry: entry || config.entry,

isBuild: !isDev,

}),

babel: config.babel || {

presets: [

[babel, { browsers: browserslist }],

...(config.extraBabelPresets || []),

],

plugins: config.extraBabelPlugins || [],

},

browserslist,

}),

config.applyWebpack,

);

}

function getConfig() {

if (existsSync(mockConfigFile)) {

// disable require cache

Object.keys(require.cache).forEach(file => {

if (file === mockConfigFile || file.indexOf(mockDir) > -1) {

debug(`delete cache ${file}`);

delete require.cache[file];

}

});

return require(mockConfigFile); // eslint-disable-line

} else {

return {};

}

}

function createMockHandler(method, path, value) {

return function mockHandler(...args) {

const res = args[1];

if (is.function(value)) {

value(...args, { sheet2json });

} else {

res.json(value);

}

};

}

function createProxy(method, path, target) {

return proxy(target, {

filter(req) {

return method ? req.method.toLowerCase() === method.toLowerCase() : true;

},

proxyReqPathResolver(req) {

let matchPath = req.originalUrl;

const matches = matchPath.match(path);

if (matches.length > 1) {

[, matchPath] = matches;

}

return winPath(join(url.parse(target).path, matchPath));

},

});

}

export function applyMock(devServer) {

try {

realApplyMock(devServer);

error = null;

} catch (e) {

console.log(e);

error = e;

console.log();

outputError();

const watcher = chokidar.watch([mockConfigFile, mockDir], {

ignored: /node\_modules/,

ignoreInitial: true,

});

watcher.on('change', path => {

console.log(

chalk.green('CHANGED'),

path.replace(paths.appDirectory, '.'),

);

watcher.close();

applyMock(devServer);

});

}

}

function realApplyMock(devServer) {

const config = getConfig();

const { app } = devServer;

devServer.use(bodyParser.json({ limit: '5mb', strict: false }));

devServer.use(

bodyParser.urlencoded({

extended: true,

limit: '5mb',

}),

);

Object.keys(config).forEach(key => {

const keyParsed = parseKey(key);

const { method } = keyParsed;

let { path } = keyParsed;

assert(!!app[method], `method of ${key} is not valid`);

assert(

is.function(config[key]) ||

is.object(config[key]) ||

is.array(config[key]) ||

is.string(config[key]),

`mock value of ${key} should be function or object or string, but got ${is(

config[key],

)}`,

);

if (is.string(config[key])) {

if (/\(.+\)/.test(path)) {

path = new RegExp(`^${path}$`);

}

app.use(path, createProxy(method, path, config[key]));

} else {

app[method](path, createMockHandler(method, path, config[key]));

}

});

let lastIndex = null;

app.\_router.stack.forEach((item, index) => {

if (item.name === 'webpackDevMiddleware') {

lastIndex = index;

}

});

const mockAPILength = app.\_router.stack.length - 1 - lastIndex;

if (lastIndex && lastIndex > 0) {

const newStack = app.\_router.stack;

newStack.push(newStack[lastIndex - 1]);

newStack.push(newStack[lastIndex]);

newStack.splice(lastIndex - 1, 2);

app.\_router.stack = newStack;

}

const watcher = chokidar.watch([mockConfigFile, mockDir], {

ignored: /node\_modules/,

persistent: true,

});

watcher.on('change', path => {

console.log(chalk.green('CHANGED'), path.replace(paths.appDirectory, '.'));

watcher.close();

app.\_router.stack.splice(lastIndex - 1, mockAPILength);

applyMock(devServer);

});

}

function parseKey(key) {

let method = 'get';

let path = key;

if (key.indexOf(' ') > -1) {

const splited = key.split(' ');

method = splited[0].toLowerCase();

[, path] = splited;

}

return { method, path };

}

export function outputError() {

if (!error) return;

const filePath = error.message.split(': ')[0];

const relativeFilePath = filePath.replace(paths.appDirectory, '.');

const errors = error.stack

.split('\n')

.filter(line => line.trim().indexOf('at ') !== 0)

.map(line => line.replace(`${filePath}: `, ''));

errors.splice(1, 0, ['']);

console.log(chalk.red('Failed to parse mock config.'));

console.log();

console.log(`Error in ${relativeFilePath}`);

console.log(errors.join('\n'));

console.log();

}

export default function(babelPreset, opts) {

const { configOnly, disablePreventTest, ignore, cwd } = opts;

const files = [

CONFIG\_FILE\_NAME,

MOCK\_CONFIG\_FILE,

MOCK\_CONFIG\_DIR,

CONFIG\_DIR,

// 'src',

].map(file => excapeRegExp(join(cwd, file)));

const only = configOnly ? [new RegExp(`(${files.join('|')})`)] : null;

registerBabel({

only,

ignore,

babelPreset,

disablePreventTest,

});

}

export function sheet2json(filePath, sheetname, opts = {}) {

return new Promise((resolve, reject) => {

if (!filePath) {

return reject(error(500, 'must specify a filePath'));

}

filePath = pathResolve(process.cwd(), filePath);

if (!existsSync(filePath)) {

return reject(error(510, `${filePath}: No such file or directory`));

}

if (is.object(sheetname)) {

opts = sheetname;

sheetname = null;

}

sheetname = sheetname || opts.sheetname;

delete opts.sheetname;

const { outputPath, readAll, arrays, rawJs, ...readOpts } = opts;

let wb = null;

try {

wb = X.readFile(filePath, readOpts);

} catch (e) {

return reject(error(520, `${filePath} error parsing: ${e}`));

}

if (!wb) {

return reject(error(530, `${filePath} error parsing: empty workbook`));

}

const targetSheet = sheetname || (wb.SheetNames || [''])[0];

let ws;

try {

ws = wb.Sheets[targetSheet];

if (!ws) {

return reject(error(540, `Sheet ${targetSheet} cannot be found`));

}

} catch (e) {

return reject(

error(550, `error parsing ${filePath} ${targetSheet}: ${e}`),

);

}

const jsonOpts = {

header: arrays ? 1 : undefined,

raw: rawJs ? true : undefined,

};

const sheetToJson = ws => X.utils.sheet\_to\_json(ws, jsonOpts);

let result = {};

if (readAll) {

wb.SheetNames.forEach(s => {

result[s] = sheetToJson(wb.Sheets[s]);

});

} else {

result = sheetToJson(ws);

}

if (outputPath) {

try {

writeFileSync(

pathResolve(process.cwd(), outputPath),

JSON.stringify(result),

'utf8',

);

} catch (e) {

return reject(error(560, e));

}

}

resolve(result);

});

}

export default function(path) {

return path.replace(/\\/g, '/');

}

describe('sheet2json', () => {

test('error()', () => {

expect(error(1, 'err info')).toEqual(new Error('[sheet2json 1]: err info'));

});

test('500 error', async () => {

try {

await sheet2json();

} catch (error) {

expect(() => {

throw error;

}).toThrow('[sheet2json 500]: must specify a filePath');

}

});

test('510 error', async () => {

const fileName = './xxxx.csv';

try {

await sheet2json(fileName);

} catch (error) {

expect(() => {

throw error;

}).toThrow(

`[sheet2json 510]: ${resolve(

process.cwd(),

fileName,

)}: No such file or directory`,

);

}

});

test('540 error', async () => {

const filePath = resolve(\_\_dirname, './fixtures/sheet/A4X\_2013.xls');

let error;

try {

await sheet2json(filePath, 's1');

} catch (e) {

error = e;

}

expect(() => {

throw error;

}).toThrow('[sheet2json 540]: Sheet s1 cannot be found');

expect(await sheet2json(filePath)).toEqual([]);

});

test('sheetName', async () => {

const filePath = resolve(\_\_dirname, './fixtures/sheet/AutoFilter.xlsx');

expect((await sheet2json(filePath))[0]).toEqual({

Code: '5',

Format: 'XLSX (Excel 2007+)',

Importance: '1',

Library: 'js-xlsx',

Notes: 'Strict Open XML Spreadsheet, Excel Workbook',

});

});

test('readAll', async () => {

const filePath = resolve(\_\_dirname, './fixtures/sheet/AutoFilter.xlsx');

const result = await sheet2json(filePath, { readAll: true });

expect(Object.keys(result)).toEqual([

'No Filter',

'Just Filter',

'One Cond',

'Two Cond',

'Top10',

'Bot10',

'Average',

'NE',

'GT',

'AND Bounding',

'OR Range',

]);

});

test('output', async () => {

const filePath = resolve(\_\_dirname, './fixtures/sheet/AutoFilter.xlsx');

const outputPath = resolve(\_\_dirname, './fixtures/sheet/AutoFilter.json');

const result = await sheet2json(filePath, {

readAll: true,

outputPath,

});

expect(readFileSync(outputPath, 'utf8')).toBe(JSON.stringify(result));

rm.sync(outputPath);

});

});

export default function getGitUser() {

let name = null;

let email = null;

try {

name = execSync('git config --get user.name');

email = execSync('git config --get user.email');

} catch (e) {} // eslint-disable-line

name = name && name.toString().trim();

email = email && email.toString().trim();

return { name, email };

}

describe('build', () => {

test('useMemoryFS', done => {

build({

webpackConfig: getConfig({

cwd: join(\_\_dirname, './fixtures/normal'),

entry: join(\_\_dirname, './fixtures/normal/index.js'),

}),

useMemoryFS: true,

success({ assets }) {

expect(assets[0]).toMatchSnapshot({ path: expect.any(String) });

done();

},

fail(err) {

throw new Error(err);

},

});

});

});

export async function ensureMegaProject(projectPath) {

try {

await ensureDir(join(projectPath, DOT\_LUGIA\_DIR));

await ensureFile(join(projectPath, PAGES\_CONFIG\_PATH));

await ensureDir(join(projectPath, PROJECT\_PAGES\_DIR));

await ensureDir(join(projectPath, PROJECT\_MODELS\_DIR));

} catch (err) {

log('ensureMegaProject error', projectPath, err);

throw new Error(err);

}

}

export async function createWorkSpace() {

log('createWorkSpace');

log('homeOrTmp', homeOrTmp);

try {

if (!(await existsSync(LUGIA\_HOME\_DIR))) {

await mkdirSync(LUGIA\_HOME\_DIR);

log('mkdirSync homeOrTmp', LUGIA\_HOME\_DIR);

}

if (!(await existsSync(MEGA\_DIR))) {

await mkdirSync(MEGA\_DIR);

log('mkdirSync MEGA\_DIR', MEGA\_DIR);

}

if (!(await existsSync(CACHE\_DIR))) {

await mkdirSync(CACHE\_DIR);

log('mkdirSync CACHE\_DIR', CACHE\_DIR);

}

if (!(await existsSync(WORKSPACE\_DIR))) {

await mkdirSync(WORKSPACE\_DIR);

log('mkdirSync WORKSPACE\_DIR', WORKSPACE\_DIR);

}

const materialsString = JSON.stringify(materialsjson);

log('write materials.json', materialsString);

await writeFileSync(

join(CACHE\_DIR, './official-react-materials.json'),

materialsString

);

} catch (err) {

log('createWorkSpace error', err.stack);

}

}

function getMsg(msg) {

if (!msg) {

return '';

}

return typeof msg === 'object' ? JSON.stringify(msg) : msg.toString();

}

function getMsgs(...msgs) {

return msgs.map(msg => `${getMsg(msg)}`).join(' ');

}

module.exports = LOG\_FILE => {

let isFirst = true;

return topic =>

async function(...msgs) {

const isExistLog = await fs.existsSync(LOG\_FILE);

if (!isExistLog) {

isFirst = false;

await fs.writeFileSync(LOG\_FILE, `init log\n${new Date()}`);

}

if (isFirst) {

const line = `${'#'.repeat(40)}`;

isFirst = false;

await fs.appendFileSync(

LOG\_FILE,

`\n\n${line} - ${moment().format('YYYY-MM-DD HH:mm:ss')} - ${line}\n`

);

}

await fs.appendFileSync(

LOG\_FILE,

`${moment().format('YYYY-MM-DD HH:mm:ss')} - ${topic}: (${getMsgs(

...msgs

)})\n`

);

};

};

function createChildProcess(

type: Function,

command: string,

args?: Array<string>,

options?: Object = {}

) {

const { mid, ...useoptions } = options;

let useid;

if (mid && !hasChildProcess(mid)) {

useid = mid;

} else {

useid = shortid.generate();

}

const subprocess = type(command, args, useoptions);

subprocess.on('exit', () => {

if (hasChildProcess(useid)) {

delete childProcess[useid];

}

});

subprocess.mid = useid;

childProcess[mid] = subprocess;

return subprocess;

}

export function spawn(...args) {

return createChildProcess(crossSpawn, ...args);

}

export function fork(...args) {

return createChildProcess(\_fork, ...args);

}

export function hasChildProcess(mid: string) {

return !!childProcess[mid];

}

export function getChildProcess(mid?: string) {

if (!mid) {

return childProcess;

}

return childProcess[mid];

}

export function unChildProcess(mid: string, cb?: Function) {

if (hasChildProcess(mid)) {

const subprocess = getChildProcess(mid);

if (cb) {

subprocess.on('exit', (code, signal) => {

cb(undefined, code, signal);

});

}

kill(subprocess.pid, error => cb(error));

delete childProcess[mid];

} else if (cb) {

cb();

}

}

export function unAllChildProcess() {

log('Main Process exit. unAllChildProcess.');

Object.keys(getChildProcess()).forEach(mid => unChildProcess(mid, () => {}));

}

function titleCase(str) {

return str.toLowerCase().replace(/( |^)[a-z]/g, L => L.toUpperCase());

}

export default function pages2routingConfig(pages, pagesDir) {

const res = [];

const info = getInfoByPages(pages);

res.push(`${getImportByInfo(info, pagesDir)}

export default [`);

info.forEach(item => {

const { children } = item;

if (children) {

const { name, text, icon } = item;

let html = `

{

value: '${titleCase(name)}',

text: '${text}',

icon: '${icon}',

children: [

`;

children.forEach(childrenItem => {

const { name, text, icon } = childrenItem;

html += `

{

value: '/${name}',

text: '${text}',

icon: '${icon}',

component: ${titleCase(name)},

},

`;

});

html += ']},';

res.push(html);

} else {

const { name, text, icon } = item;

res.push(`

{

value: '/${name}',

text: '${text}',

icon: '${icon}',

component: ${titleCase(name)},

},

`);

}

});

res.push(']');

return res.join('');

}

function getInfoByPages(pages) {

return pages.map(page => ({

name: page.name,

text: page.title,

icon: page.icon

}));

}

const getImportByInfo = (info, pagesDir = '') => {

let importInfo = '';

info.forEach(item => {

const { children } = item;

if (children) {

children.forEach(childrenItem => {

const { name } = childrenItem;

importInfo = `${importInfo} import ${titleCase(name)} from '${join(

pagesDir,

`./${name}.lugiad`

).replace(/\\/g, '\\\\')}'; `;

});

} else {

const { name } = item;

importInfo = `${importInfo} import ${titleCase(name)} from '${join(

pagesDir,

`./${name}.lugiad`

).replace(/\\/g, '\\\\')}'; `;

}

});

return importInfo;

};

export default function() {

if (mainWindow) {

log('mainWindow is exist');

return mainWindow;

}

mainWindow = new BrowserWindow({

show: false,

...MAIN\_SIZE

});

log('mainWindow is create');

const url = `file://${HTML\_PATH}/app.html`;

mainWindow.loadURL(url);

log(url);

// @TODO: Use 'ready-to-show' event

// https://github.com/electron/electron/blob/master/docs/api/browser-window.md#using-ready-to-show-event

mainWindow.webContents.on('did-finish-load', () => {

if (!mainWindow) {

throw new Error('"mainWindow" is not defined');

}

log('did-finish-load');

if (process.env.START\_MINIMIZED) {

mainWindow.minimize();

} else {

mainWindow.maximize();

mainWindow.show();

log('show');

mainWindow.focus();

}

});

log('webContents on did-finish-load');

mainWindow.on('closed', () => {

mainWindow = null;

});

log('mainWindow on closed');

log('mainWindow state', !!mainWindow);

const menuBuilder = new MenuBuilder(mainWindow);

log('menuBuilder create');

menuBuilder.buildMenu();

log('menuBuilder.buildMenu()');

return mainWindow;

}

ipcMain.on('buildProject-async', (e, { params: { project } }) => {

const { workspacePath, path } = project;

const buildSpawn = crossSpawn('yarn', ['build'], {

cwd: workspacePath,

encoding: 'utf-8',

env: getEnv()

});

function send(result) {

e.sender.send('buildProject-reply', result);

}

buildSpawn.stdout.on('data', chunk => {

const data = chunk

.toString()

.replace(/\u001b\[\d;?\d{2,3}m|\u001b\[0m/g, '') // eslint-disable-line

.replace(/\u001b\[\d[A-Z]\u001b\[\d[A-Z]\u001b\[\d{1,2}m/g, '') // eslint-disable-line

.replace(/\u001b\[\d{2,3}m/g, '') // eslint-disable-line

.replace(/\u001b\[\d[A-Za-z]/g, ''); // eslint-disable-line

send({

error: null,

data

});

});

buildSpawn.on('error', error => {

send({

error,

data: null

});

});

buildSpawn.on('exit', async code => {

buildSpawn.stdin.end();

if (code === 0) {

try {

await copy(join(workspacePath, 'dist'), join(path, 'dist'));

} catch (err) {

send({

error: err.message,

data: null

});

}

send({ error: null, data: 'success', exit: true });

} else {

send({ error: `code: ${code}`, data: null, exit: true });

}

});

});

export function delMaterials(name: string): Result<MaterialsType> {

return getMaterials(() => {

db.get('materials')

.remove({ name })

.write();

});

}

export function resetMaterials(): Result<MaterialsType> {

return getMaterials(() => {

db.set('materials', OFFICIAL\_MATERIALS).write();

});

}

const subMenuHelp = {

label: 'Help',

submenu: [

{

label: 'Learn More',

click() {

shell.openExternal('http://electron.atom.io');

}

},

{

label: 'Documentation',

click() {

shell.openExternal(

'https://github.com/atom/electron/tree/master/docs#readme'

);

}

},

{

label: 'Community Discussions',

click() {

shell.openExternal('https://discuss.atom.io/c/electron');

}

},

{

label: 'Search Issues',

click() {

shell.openExternal('https://github.com/atom/electron/issues');

}

}

]

};

const subMenuView =

process.env.NODE\_ENV === 'development' ? subMenuViewDev : subMenuViewProd;

return [subMenuAbout, subMenuEdit, subMenuView, subMenuWindow, subMenuHelp];

}

buildDefaultTemplate() {

const templateDefault = [

{

label: '&File',

submenu: [

{

label: '&Open',

accelerator: 'Ctrl+O'

},

{

label: '&Close',

accelerator: 'Ctrl+W',

click: () => {

this.mainWindow.close();

}

}

]

},

{

label: '&View',

submenu:

process.env.NODE\_ENV === 'development'

? [

{

label: '&Reload',

accelerator: 'Ctrl+R',

click: () => {

this.mainWindow.webContents.reload();

}

},

{

label: 'Toggle &Full Screen',

accelerator: 'F11',

click: () => {

this.mainWindow.setFullScreen(

!this.mainWindow.isFullScreen()

);

}

},

{

label: 'Toggle &Developer Tools',

accelerator: 'Alt+Ctrl+I',

click: () => {

this.mainWindow.toggleDevTools();

}

}

]

: [

{

label: 'Toggle &Full Screen',

accelerator: 'F11',

click: () => {

this.mainWindow.setFullScreen(

!this.mainWindow.isFullScreen()

);

}

}

]

},

{

label: 'Help',

submenu: [

{

label: 'Learn More',

click() {

shell.openExternal('http://electron.atom.io');

}

},

{

label: 'Documentation',

click() {

shell.openExternal(

'https://github.com/atom/electron/tree/master/docs#readme'

);

}

},

{

label: 'Community Discussions',

click() {

shell.openExternal('https://discuss.atom.io/c/electron');

}

},

{

label: 'Search Issues',

click() {

shell.openExternal('https://github.com/atom/electron/issues');

}

}

]

}

];

return templateDefault;

}

}

export default async function returnResult(cb: () => any): Result {

let error = null;

let data = null;

try {

data = await cb();

log('data', data);

} catch (e) {

const { name, message } = e;

error = JSON.stringify({

name,

message,

ErrorToString: e.toString(),

extra: { ...e },

stack: String(e.stack)

});

log('error', e.stack);

}

return { error, data };

}

describe('reducers', () => {

describe('counter', () => {

it('should handle initial state', () => {

expect(counter(undefined, {})).toMatchSnapshot();

});

it('should handle INCREMENT\_COUNTER', () => {

expect(counter(1, { type: INCREMENT\_COUNTER })).toMatchSnapshot();

});

it('should handle DECREMENT\_COUNTER', () => {

expect(counter(1, { type: DECREMENT\_COUNTER })).toMatchSnapshot();

});

it('should handle unknown action type', () => {

expect(counter(1, { type: 'unknown' })).toMatchSnapshot();

});

});

});

describe('main window', function spec() {

beforeAll(async () => {

this.app = new Application({

path: electronPath,

args: [path.join(\_\_dirname, '..', '..', 'app')]

});

return this.app.start();

});

afterAll(() => {

if (this.app && this.app.isRunning()) {

return this.app.stop();

}

});

const findCounter = () => this.app.client.element('[data-tid="counter"]');

const findButtons = async () => {

const { value } = await this.app.client.elements('[data-tclass="btn"]');

return value.map(btn => btn.ELEMENT);

};

it('should open window', async () => {

const { client, browserWindow } = this.app;

await client.waitUntilWindowLoaded();

await delay(500);

const title = await browserWindow.getTitle();

expect(title).toBe('Lugia Mega');

});

it("should haven't any logs in console of main window", async () => {

const { client } = this.app;

const logs = await client.getRenderProcessLogs();

// Print renderer process logs

logs.forEach(log => {

console.log(log.message);

console.log(log.source);

console.log(log.level);

expect(log.level).not.toEqual('SEVERE');

});

// @NOTE: Temporarily have to disable this assertion because there are some warnings in

// electron@2. Loading files from localhost in development uses http and this causes

// electron to throw warnings

// expect(logs).toHaveLength(0);

});

it('should to Counter with click "to Counter" link', async () => {

const { client } = this.app;

await client.click('[data-tid=container] > a');

expect(await findCounter().getText()).toBe('0');

});

it('should display updated count after increment button click', async () => {

const { client } = this.app;

const buttons = await findButtons();

await client.elementIdClick(buttons[0]); // +

expect(await findCounter().getText()).toBe('1');

});

it('should display updated count after descrement button click', async () => {

const { client } = this.app;

const buttons = await findButtons();

await client.elementIdClick(buttons[1]); // -

expect(await findCounter().getText()).toBe('0');

});

it('shouldnt change if even and if odd button clicked', async () => {

const { client } = this.app;

const buttons = await findButtons();

await client.elementIdClick(buttons[2]); // odd

expect(await findCounter().getText()).toBe('0');

});

it('should change if odd and if odd button clicked', async () => {

const { client } = this.app;

const buttons = await findButtons();

await client.elementIdClick(buttons[0]); // +

await client.elementIdClick(buttons[2]); // odd

expect(await findCounter().getText()).toBe('2');

});

it('should change if async button clicked and a second later', async () => {

const { client } = this.app;

const buttons = await findButtons();

await client.elementIdClick(buttons[3]); // async

expect(await findCounter().getText()).toBe('2');

await delay(3000);

expect(await findCounter().getText()).toBe('3');

});

it('should back to home if back button clicked', async () => {

const { client } = this.app;

await client.element('[data-tid="backButton"] > a').click();

expect(await client.isExisting('[data-tid="container"]')).toBe(true);

});

});

function setup(initialState) {

const store = configureStore(initialState);

const history = createBrowserHistory();

const provider = (

<Provider store={store}>

<ConnectedRouter history={history}>

<CounterPage />

</ConnectedRouter>

</Provider>

);

const app = mount(provider);

return {

app,

buttons: app.find('button'),

p: app.find('.counter')

};

}

describe('containers', () => {

describe('App', () => {

it('should display initial count', () => {

const { p } = setup();

expect(p.text()).toMatch(/^0$/);

});

it('should display updated count after increment button click', () => {

const { buttons, p } = setup();

buttons.at(0).simulate('click');

expect(p.text()).toMatch(/^1$/);

});

it('should display updated count after decrement button click', () => {

const { buttons, p } = setup();

buttons.at(1).simulate('click');

expect(p.text()).toMatch(/^-1$/);

});

it('shouldnt change if even and if odd button clicked', () => {

const { buttons, p } = setup();

buttons.at(2).simulate('click');

expect(p.text()).toMatch(/^0$/);

});

it('should change if odd and if odd button clicked', () => {

const { buttons, p } = setup({ counter: 1 });

buttons.at(2).simulate('click');

expect(p.text()).toMatch(/^2$/);

});

});

});

function setup() {

const actions = {

increment: spy(),

incrementIfOdd: spy(),

incrementAsync: spy(),

decrement: spy()

};

const component = shallow(<Counter counter={1} {...actions} />);

return {

component,

actions,

buttons: component.find('button'),

p: component.find('.counter')

};

}

describe('Counter component', () => {

it('should should display count', () => {

const { p } = setup();

expect(p.text()).toMatch(/^1$/);

});

it('should first button should call increment', () => {

const { buttons, actions } = setup();

buttons.at(0).simulate('click');

expect(actions.increment.called).toBe(true);

});

it('should match exact snapshot', () => {

const { actions } = setup();

const counter = (

<div>

<Router>

<Counter counter={1} {...actions} />

</Router>

</div>

);

const tree = renderer.create(counter).toJSON();

expect(tree).toMatchSnapshot();

});

it('should second button should call decrement', () => {

const { buttons, actions } = setup();

buttons.at(1).simulate('click');

expect(actions.decrement.called).toBe(true);

});

it('should third button should call incrementIfOdd', () => {

const { buttons, actions } = setup();

buttons.at(2).simulate('click');

expect(actions.incrementIfOdd.called).toBe(true);

});

it('should fourth button should call incrementAsync', () => {

const { buttons, actions } = setup();

buttons.at(3).simulate('click');

expect(actions.incrementAsync.called).toBe(true);

});

});

describe('actions', () => {

it('should increment should create increment action', () => {

expect(actions.increment()).toMatchSnapshot();

});

it('should decrement should create decrement action', () => {

expect(actions.decrement()).toMatchSnapshot();

});

it('should incrementIfOdd should create increment action', () => {

const fn = actions.incrementIfOdd();

expect(fn).toBeInstanceOf(Function);

const dispatch = spy();

const getState = () => ({ counter: 1 });

fn(dispatch, getState);

expect(dispatch.calledWith({ type: actions.INCREMENT\_COUNTER })).toBe(true);

});

it('should incrementIfOdd shouldnt create increment action if counter is even', () => {

const fn = actions.incrementIfOdd();

const dispatch = spy();

const getState = () => ({ counter: 2 });

fn(dispatch, getState);

expect(dispatch.called).toBe(false);

});

// There's no nice way to test this at the moment...

it('should incrementAsync', done => {

const fn = actions.incrementAsync(1);

expect(fn).toBeInstanceOf(Function);

const dispatch = spy();

fn(dispatch);

setTimeout(() => {

expect(dispatch.calledWith({ type: actions.INCREMENT\_COUNTER })).toBe(

true

);

done();

}, 5);

});

});

export default class Root extends Component<Props> {

render() {

const { store, history } = this.props;

return (

<Provider store={store}>

<ConnectedRouter history={history}>

<Routes />

</ConnectedRouter>

</Provider>

);

}

}

export default () => (

<Fragment>

<Switch>

<Route exact path={routes.HOME} component={Home} />

<Route path={routes.COUNTER} component={Counter} />

<Route path={routes.CREATE} component={Create} />

<Route path={`${routes.PROJECT}/:projectID`} component={Project} />

<Route path={routes.MORE} component={More} />

</Switch>

</Fragment>

);

function configureStore(initialState?: InitialStateType) {

return createStore(

createRootReducer(history), // new root reducer with router state

initialState,

compose(

applyMiddleware(thunk, routerMiddleware(history), loadingMiddleware)

)

);

}

const configureStore = (initialState?: InitialStateType) => {

// Redux Configuration

const middleware = [];

const enhancers = [];

// Thunk Middleware

middleware.push(thunk);

// Logging Middleware

const logger = createLogger({

level: 'info',

collapsed: true

});

// Skip redux logs in console during the tests

if (process.env.NODE\_ENV !== 'test') {

middleware.push(logger);

}

// Router Middleware

const router = routerMiddleware(history);

middleware.push(router);

// Loading Middleware

middleware.push(loadingMiddleware);

// Redux DevTools Configuration

const actionCreators = {

...counterActions,

...routerActions

};

// If Redux DevTools Extension is installed use it, otherwise use Redux compose

/\* eslint-disable no-underscore-dangle \*/

const composeEnhancers = window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_COMPOSE\_\_

? window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_COMPOSE\_\_({

// Options: http://extension.remotedev.io/docs/API/Arguments.html#actioncreators

actionCreators

})

: compose;

/\* eslint-enable no-underscore-dangle \*/

// Apply Middleware & Compose Enhancers

enhancers.push(applyMiddleware(...middleware));

const enhancer = composeEnhancers(...enhancers);

// Create Store

const store = createStore(

createRootReducer(history), // new root reducer with router state

initialState,

enhancer

);

if (module.hot) {

module.hot.accept(

'./reducers',

() => store.replaceReducer(require('./reducers')(history)) // eslint-disable-line global-require

);

}

return store;

};

export const getHistoryProjectsSelector: AllStateType => HistoryProjectsType = state =>

state.historyProjects;

export const getCurrentPorjectIDSelector: AllStateType =>

| null

| string = state => state.projectManage.projectID;

export const getProjectByIDSelector: (

AllStateType,

string

) => HistoryProjectType = (state, projectID) => {

const projects = getHistoryProjectsSelector(state);

return [

projects.findIndex(({ id }) => id === projectID),

projects.find(({ id }) => id === projectID)

];

};

export const getSwitchProjectIDSelector: (AllStateType, Props) => string = (

state,

props

) => props.match.params.projectID;

export const sortProjectsSelector: AllStateType => HistoryProjectType = createSelector(

getHistoryProjectsSelector,

historyProjects =>

[...historyProjects].sort((pre, next) => {

if (pre.favorite !== next.favorite) {

return pre.favorite ? -1 : 1;

}

return next.openDate - pre.openDate;

})

);

export const getCurrentProjectSelector: AllStateType => HistoryProjectType = createSelector(

getHistoryProjectsSelector,

getCurrentPorjectIDSelector,

(historyProjects, currentID) =>

historyProjects.find(({ id }) => id === currentID)

);

export const switchProjectSelector: (

AllStateType,

Props

) => boolean = createSelector(

getCurrentPorjectIDSelector,

getSwitchProjectIDSelector,

(currentID, switchID) => {

return switchID !== currentID;

}

);

export const existsProjectSelector: (

AllStateType,

Props

) => boolean = createSelector(

getHistoryProjectsSelector,

getSwitchProjectIDSelector,

(historyProjects, switchID) =>

historyProjects.some(({ id }) => switchID === id)

);

export const getManageHistorysSelector: AllStateType => ProjectManageHistorysType = state =>

state.projectManage.historys;

export const getProjectManageInfoByIDSelector: (

AllStateType,

string

) => HistoryProjectType = (state, projectID) => {

const historys = getManageHistorysSelector(state);

return historys[projectID];

};

export const getCurrentProjectManageInfoSelector: AllStateType => ProjectManageInfoType = createSelector(

getManageHistorysSelector,

getCurrentPorjectIDSelector,

(historys, currentID) => historys[currentID]

);

export const getCurrentPageIdSelector: AllStateType => null | string = state =>

state.projectManage.currentPage;

export const getCurrentPageSelector: AllStateType => ProjectPageType = createSelector(

getCurrentProjectManageInfoSelector,

getCurrentPageIdSelector,

(manageInfo, pageId) =>

manageInfo && manageInfo.pages.find(p => p.id === pageId)

);

export const createMaterialsSelector: (

'blocks' | 'layouts' | 'scaffoldings'

) => AllStateType =>

| BlocksDataType

| LayoutsDataType

| ScaffoldingsDataType = type =>

createSelector(

materialsSelector,

materials => materials.map(m => m.cacheData[type])

);

export default function bindActionCreators(

actionCreators: Object,

dispatch: Dispatch

): ActionCreators {

const filterActionCreators: ActionCreators = {};

Object.keys(actionCreators).forEach(k => {

if (k !== 'default' && is.function(actionCreators[k])) {

filterActionCreators[k] = actionCreators[k];

}

});

return bind(filterActionCreators, dispatch);

}

export const loadingMiddleware = ({ dispatch }) => next => action => {

const { type, PENDING, RESOLVED, REJECTED } = action;

if (type) {

const [pending, resolved, rejected] = ACTION\_TYPE\_SUFFIXES;

const isPending = new RegExp(`${pending}$`).test(type) || !!PENDING;

const isResolved = new RegExp(`${resolved}$`).test(type) || !!RESOLVED;

const isRejected = new RegExp(`${rejected}$`).test(type) || !!REJECTED;

const scope = PENDING || RESOLVED || REJECTED || action.scope;

if (isPending) {

dispatch(showLoading(scope));

} else if (isResolved || isRejected) {

dispatch(hideLoading(scope));

}

}

return next(action);

};

export function showLoading(scope?: string = GLOBAL\_LOADING\_KEY) {

return {

type: SHOW,

scope

};

}

export function hideLoading(scope?: string = GLOBAL\_LOADING\_KEY) {

return {

type: HIDE,

scope

};

}

/\*\*

\* process asyncAction

\*/

export function withLoading(

asyncAction: AsyncAction,

scope?: string

): AsyncAction {

return async (dispatch, getState) => {

let result;

if (getState().loading[GLOBAL\_LOADING\_KEY]) {

dispatch(hideLoading());

}

dispatch(showLoading(scope));

try {

result = await asyncAction(dispatch, getState);

} catch (e) {

dispatch(showLoading());

if (process.env.NODE\_ENV !== 'production') {

console.error(`[redux-loading ${GLOBAL\_LOADING\_KEY}]`, e);

}

}

dispatch(hideLoading(scope));

return result;

};

}

// create loading reducer

export const loadingReducerCreater = initLoadingState => (

state = initLoadingState,

action

) => {

const scope = action.scope in state ? action.scope : GLOBAL\_LOADING\_KEY;

switch (action.type) {

case SHOW:

return {

...state,

[scope]: true

};

case HIDE:

return {

...state,

[scope]: false

};

default:

return state;

}

};

// connect combineReducers and loadingReducer

export const connectReducers = (

reducers,

opts: { stateKey: string } = { stateKey: 'loading' }

) => {

const loadingState = {};

Object.keys(reducers).forEach(k => {

loadingState[k] = false;

});

return {

...reducers,

[opts.stateKey]: loadingReducerCreater({

[GLOBAL\_LOADING\_KEY]: false,

...loadingState

})

};

};

function processReactElement(

elementsTree: React.Element<any>,

props

): React.Element<any> {

const elementsTreeProps = elementsTree.props;

const style = elementsTreeProps.style || props.style;

const className = elementsTreeProps.className || props.className;

const withProps: {

style?: CSSProperties,

className?: string

} = {

...(style ? { style } : {}),

...(className ? { className } : {})

};

const newProps = Object.assign({}, elementsTreeProps, withProps);

return React.cloneElement(

elementsTree,

newProps,

elementsTree.props.children

);

}

export default function withStyleAndClassNameHOC(

Target: React.ComponentType<any>

): React.ComponentType<any> {

let WithStyleAndClassName;

if (is.undefined(Target.prototype.render)) {

WithStyleAndClassName = props =>

processReactElement(Target({ ...Target.defaultProps, ...props }), props);

} else {

WithStyleAndClassName = class extends Target {

render() {

return processReactElement(super.render(), this.props);

}

};

}

if (process.env.NODE\_ENV !== 'production') {

WithStyleAndClassName.displayName = wrapDisplayName(

Target,

'withStyleAndClassNameHOC'

);

}

return WithStyleAndClassName;

}

const colorMap: { [key: typeValue]: string } = {

success: COMMON\_STYLE.successColor,

info: COMMON\_STYLE.themeColor,

warning: COMMON\_STYLE.warningColor,

error: COMMON\_STYLE.dangerColor

};

const getTypeColor = (type: typeValue) => {

const color = colorMap[type];

return `

background-color: ${changeColor(color, 0, 0, 20).rgba};

border-left-color: ${color};

`;

};

const Container = styled.div`

${props => getTypeColor(props.type)} border-left-width: 0.4rem;

border-left-style: solid;

border-radius: 0.4rem;

padding: 1rem;

font-size: 1.4rem;

line-height: 1.5;

color: ${COMMON\_STYLE.blackColor};

word-wrap: break-word;

`;

const Message = styled.span`

display: block;

font-size: 1.8rem;

margin-bottom: 0.4rem;

`;

const Description = styled.span`

display: block;

color: ${COMMON\_STYLE.darkGreyColor};

`;

function Alert({ type, message, description }: Props) {

if (description) {

return (

<Container type={type}>

{message ? <Message>{message}</Message> : null}

{(Array.isArray(description) ? description : [description]).map(des => (

<Description

key={is.string(des) ? des.length + des.substr(1, 3) : is(des)}

>

{des}

</Description>

))}

</Container>

);

}

return <Container type={type}>{message}</Container>;

}

Alert.defaultProps = {

type: 'info',

message: null,

description: null

};

const AlertHOC = withStyleAndClassNameHOC(Alert);

const AlertQueryDataError = props => (

<AlertHOC

message={ALERT.QUERY\_DATA\_ERROR.message}

description={ALERT.QUERY\_DATA\_ERROR.description}

type="error"

{...props}

/>

);

const Span = styled.span`

display: inline-block;

font-size: 1.2rem;

& > button {

display: block;

width: 100%;

font-family: inherit;

font-size: inherit;

}

`;

function ButtonWrap(props) {

return (

<Span>

<Button {...props} />

</Span>

);

}

const CardContainer = styled.div`

position: relative;

display: flex;

flex-direction: column;

height: 100px;

${props => props.bordered && 'border: 1px solid #e8e8e8;'}

${props =>

props.checked && 'border: 1px dashed #4D63FF;'}

border-radius: 2px;

font-size: 1rem;

text-align: center;

background-color: #fff;

transition: all 0.3s;

${props =>

props.hoverable &&

`

&:hover{

box-shadow: 0 2px 8px rgba(0,0,0,.09);

border-color: rgba(0,0,0,.09);

}`}

`;

const Content = styled.div`

position: relative;

flex: 1;

overflow: hidden;

background-color: #ecf1fb;

border-radius: 2px;

${props => props.checked && 'border: 1px dashed #4D63FF;'}

${props => props.pointer && 'cursor: pointer;'}

&:hover {

&:after {

bottom: 1em;

}

}

&:after{

position: absolute;

bottom: -2.2em;

left: 25%;

display: ${props => (props.showClickMsg ? 'block' : 'none')};

content: '${props => props.clickMsg}';

width: 50%;

height: 2em;

font-size: 1.2em;

line-height: 2em;

color: #fff;

background-image: linear-gradient(-270deg, #4D63FF 3%, #6F81FF 100%);

box-shadow: 0 2px 6px 1px rgba(77,99,255,0.45);

border-radius: 1em;

cursor: pointer;

transition: all 0.4s ease;

}

`;

const Title = styled.span`

display: block;

text-overflow: ellipsis;

overflow: hidden;

white-space: nowrap;

margin-top: 0.5em;

font-size: 1.4em;

line-height: 1.5em;

color: #747e90;

`;

const Description = styled(Title)`

margin-top: 0;

margin-bottom: 0.4em;

font-size: 1.2em;

line-height: 1;

color: #a6aab2;

`;

function Card(props: Props) {

const {

onClick,

title,

description,

children,

clickMsg,

bordered,

hoverable,

checked

} = props;

const showClickMsg = !!clickMsg;

const showDescription = !!description;

const pointer = hoverable || onClick !== noop;

return (

<CardContainer

bordered={bordered}

hoverable={hoverable}

checked={checked && bordered}

>

<Content

pointer={pointer}

checked={checked && !bordered}

onClick={onClick}

showClickMsg={showClickMsg}

clickMsg={clickMsg}

>

{children}

</Content>

<Title>{title}</Title>

{showDescription && <Description>{description}</Description>}

</CardContainer>

);

}

const CollapsePanelContainer = styled.div`

& > div {

padding-left: 0;

line-height: 1.5;

& > div {

border-color: #ccc;

& > div:first-child {

color: #50575d;

line-height: 1.5em;

padding: 0.6em 0;

${props => props.showArrow && 'padding-right: 1.5em;'}

& > i {

left: auto;

right: 0;

top: 50%;

color: #ccc;

font-size: ${1.2 / 1.4}em;

margin-top: -0.5em;

&:before {

content: '${allIconName['icon-direction\_right']}';

}

}

& > div {

display: none;

}

& + div > div {

padding: 0 0 1em;

}

}

}

}

`;

function CollapsePanel(props: { showArrow?: boolean }) {

const { showArrow } = props;

return (

<CollapsePanelContainer showArrow={showArrow}>

<Panel {...props} />

</CollapsePanelContainer>

);

}

export default class ColorInput extends React.Component<

ColorInputProps,

ColorInputState

> {

props: ColorInputProps;

state: ColorInputState;

static defaultProps = {

value: undefined,

notBlock: false,

defaultValue: '#ccc',

placeholder: '',

onChange: () => 1

};

constructor(props: ColorInputProps) {

super(props);

this.state = {

value: props.defaultValue

};

}

static getDerivedStateFromProps(

props: ColorInputProps,

state: ColorInputState

) {

const { value } = props;

if ('value' in props && value !== undefined) {

return { value };

}

if (!state) {

const { defaultValue = '#ccc' } = props;

return { value: defaultValue };

}

}

onChangeComplete = (color: Object) => {

const { hex: value } = color;

this.triggerChange(value);

this.setState({

value

});

};

triggerChange(value: string) {

const { onChange } = this.props;

const { value: oldValue } = this.state;

onChange({ newValue: value, oldValue });

}

render() {

const { value } = this.state;

const { placeholder, notBlock } = this.props;

const config = {

[Widget.Trigger]: {

zIndex: 9999999,

color: value

}

};

return (

<Theme config={config}>

<Trigger

offsetX={-60}

popup={

<SketchPicker

onChangeComplete={this.onChangeComplete}

color={value}

/>

}

action="click"

createPortal

>

<Input

value={value}

placeholder={placeholder}

prefix={notBlock ? null : <Color color={value} />}

/>

</Trigger>

</Theme>

);

}

}

const FormItemContainer = styled.div`

margin-bottom: ${props => (props.validateStatus ? 0.2 : 2.3)}rem;

width: 100%;

font-size: 1.4rem;

color: #50575d;

text-align: right;

`;

const FormItemLabel = styled.div`

vertical-align: middle;

line-height: ${props => props.lineHeight};

overflow: hidden;

text-overflow: ellipsis;

white-space: nowrap;

`;

const getLabelFix = ({ required, colon }) => {

const requiredCSS = required

? `

&:before {

display: inline-block;

margin-right: 0.4rem;

content: '\*';

font-family: SimSun;

line-height: 1;

color: #f5222d;

}`

: '';

const colonCSS = colon

? `

&:after {

content: ':';

margin: 0 0.8rem 0 0.2rem;

}`

: '';

return requiredCSS + colonCSS;

};

const Label = styled.label`

${getLabelFix};

`;

const FormItemWrapper = styled.div`

text-align: left;

line-height: ${props => props.lineHeight};

${props =>

props.fullWrapper

? `

& > span {

width: 100%;

vertical-align: middle;

}`

: ''};

`;

function validate(

value: any,

rules: ValidationRule | Array<ValidationRule>

): Promise<any, Array> {

const validator = new ValidatorSchema({

value: rules

});

return new Promise((resolve, reject) => {

validator.validate({ value }, (errors, fields) => {

if (errors) {

return reject(errors.map(({ message }) => ({ message, value })));

}

return resolve(fields);

});

});

}

function hasRequiredRule(rules: Array<ValidationRule>): boolean {

return rules.some(rule => rule.required);

}

class FormItem extends React.Component<Props, State> {

props: Props;

state: State;

static defaultProps = {

label: null,

notFixCss: false,

labelCol: 8,

wrapperCol: 16,

fullWrapper: null,

height: '3.2rem',

required: false,

colon: true,

help: null,

rules: []

};

state = {

validateStatus: undefined,

message: null

};

onChange = (...args) => {

const { children } = this.props;

const { onChange: childrenOnChange } = children.props;

const [{ newValue }] = args;

const rules = this.getRules();

const validateChange = errors => {

if (is.function(childrenOnChange)) {

childrenOnChange(Object.assign(...args, { validate: errors }));

}

};

let validateChangeState = null;

if (is.empty(rules)) {

validateChangeState = null;

} else {

validateChangeState = null;

validate(newValue, rules)

.then(() =>

this.setState({

validateStatus: undefined

})

)

.catch(errors =>

this.setState(

{

validateStatus: 'error',

message: errors[0].message

},

() => {

validateChangeState = errors;

}

)

);

}

validateChange(validateChangeState);

};

getRules(): Array<ValidationRule> {

const { rules, required } = this.props;

let rulesResult = rules;

if (is.plainObject(rulesResult)) {

rulesResult = [rulesResult];

}

if (is.array(rulesResult)) {

if (required && !hasRequiredRule(rulesResult)) {

rulesResult = [{ required }, ...rulesResult];

}

return rulesResult;

}

return [];

}

isRequired(): boolean {

const { required } = this.props;

if (required) {

return true;

}

return hasRequiredRule(this.getRules());

}

isFullWrapper(): boolean {

const { fullWrapper, children } = this.props;

if (is.boolean(fullWrapper)) {

return fullWrapper;

}

if (FilterFullWrapperDisplayName.includes(children.type.displayName)) {

return false;

}

return true;

}

getChildren(): React.Node {

const { children, help } = this.props;

const { validateStatus, message } = this.state;

return React.cloneElement(children, {

validateStatus,

help: help || message,

validateType: 'bottom',

onChange: this.onChange

});

}

packFormItemContainer() {

const { height, notFixCss } = this.props;

if (notFixCss) {

const config = { [Widget.Input]: { width: 281.25 } };

return <Theme config={config}>{this.getChildren()}</Theme>;

}

const fullWrapper = this.isFullWrapper();

return (

<FormItemWrapper lineHeight={height} fullWrapper={fullWrapper}>

{this.getChildren()}

</FormItemWrapper>

);

}

render(): React.Node {

const { label, labelCol, wrapperCol, height, colon } = this.props;

const { validateStatus } = this.state;

const required = this.isRequired();

return (

<FormItemContainer validateStatus={validateStatus}>

{label ? (

<Row>

<Col span={labelCol}>

<FormItemLabel lineHeight={height}>

<Label required={required} colon={colon}>

{label}

</Label>

</FormItemLabel>

</Col>

<Col span={wrapperCol}>{this.packFormItemContainer()}</Col>

</Row>

) : (

this.packFormItemContainer()

)}

</FormItemContainer>

);

}

}

const getIconStyle = ({ placement, onlyIcon }) => {

switch (placement) {

case 'bottom':

return `

display:block;

font-size: 2em;

${onlyIcon && 'margin-top: 0.2em;'}

`;

case 'left':

return `

font-size: 1em;

vertical-align: bottom;

${onlyIcon && 'margin-right: 0.3em;'}

`;

case 'right':

return `

font-size: 1em;

vertical-align: bottom;

${onlyIcon && 'margin-left: 0.3em;'}

`;

default:

return `

display:block;

font-size: 2em;

${onlyIcon && 'margin-bottom: 0.2em;'}

`;

}

};

const Button = styled.button`

display: inline-block;

${props => props.block && 'width:100%;'}

padding: 0.5em;

font-size: 1.2rem;

line-height: 1;

color: #50575d;

background-color: #fff;

border: 0 none;

border-radius: 0.4em;

text-align: center;

white-space: nowrap;

touch-action: manipulation;

outline: 0 none;

text-overflow: ellipsis;

overflow: hidden;

cursor: pointer;

transition: color 0.3s cubic-bezier(0.645, 0.045, 0.355, 1);

&[disabled] {

color: rgba(0, 0, 0, 0.25);

background-color: #f5f5f5;

cursor: not-allowed;

}

&:not([disabled]):hover {

color: #4d63ff;

}

& > span {

${getIconStyle}

}

`;

function IconButton(props: Props) {

const {

type,

placement,

onClick,

children,

loading,

disabled,

block

} = props;

const isTopOrLeft = placement === 'top' || placement === 'left';

const onlyIcon = Boolean(children);

return (

<Button

type="button"

onClick={onClick}

placement={placement}

onlyIcon={onlyIcon}

disabled={disabled}

block={block}

>

{isTopOrLeft

? [<LugiaIcon type={type} spin={loading} />, children]

: [children, <LugiaIcon type={type} spin={loading} />]}

</Button>

);

}

const LoggerContainer = styled.div`

background: #fff;

height: ${props => props.height};

`;

const LoggerHeader = styled.div`

padding: 0.6rem 0.6rem 0.6rem 1.6rem;

font-size: 1.6rem;

line-height: 3.8rem;

& > i {

vertical-align: text-bottom;

}

`;

const LoggerTitle = styled.div`

margin-left: 0.6rem;

display: inline-block;

`;

const Logs = styled.div`

height: calc(${props => props.height} - 5rem);

overflow-x: hidden;

overflow-y: auto;

padding: 0 1.6rem;

font-size: 1.4rem;

font-family: Consolas, Menlo, Courier, monospace;

pre {

white-space: pre-wrap;

word-wrap: break-word;

}

`;

const LoggerMessage = styled.div`

padding: 0.2rem 0.4rem;

&:hover {

background-color: rgba(104, 79, 255, 0.05);

}

`;

export default class Logger extends React.Component<Props> {

props: Props;

static defaultProps = {

height: '20rem',

title: '日志'

};

render() {

const { height, title, messages } = this.props;

return (

<LoggerContainer height={height}>

<LoggerHeader>

<Icon iconClass="lugia-icon-financial\_credit\_card" />

<LoggerTitle>{title}</LoggerTitle>

</LoggerHeader>

<Logs height={height}>

{messages.map(message => (

<LoggerMessage>{message}</LoggerMessage>

))}

</Logs>

</LoggerContainer>

);

}

}

const loadingCircle = keyframes`

0% {

transform-origin: 50% 50%;

transform: rotate(0deg);

}

100% {

transform-origin: 50% 50%;

transform: rotate(360deg);

}

`;

const IconWrapper = styled.span`

display: inline-block;

& > i {

display: block;

cursor: inherit;

font-size: 1em;

${({ spin }) =>

spin &&

`animation: ${loadingCircle} ${

is.number(spin) ? spin : 1

}s infinite linear;`}

}

`;

function LugiaIcon(props: Props) {

const { type, spin } = props;

return (

<IconWrapper spin={spin}>

<Icon iconClass={`lugia-icon-${type}`} />

</IconWrapper>

);

}

const MaskContainer = styled.div`

position: relative;

${props => props.cover && 'height: 100%;'}

`;

const MaskContent = styled.div`

display: flex;

align-items: center;

justify-content: center;

position: absolute;

top: 0;

bottom: 0;

left: 0;

right: 0;

color: #1890ff;

`;

const MaskBlur = styled.div`

${props =>

!props.disable &&

`

pointer-events: none;

user-select: none;

overflow: hidden;

opacity: 0.5;

filter: blur(0.5px);

`}

height: 100%;

zoom: 1;

transition: opacity 0.3s;

`;

type Props = {

children: React.Node,

content?: React.Node,

disable?: boolean,

cover?: boolean

};

function Mask(props: Props) {

const { children, content, disable, cover } = props;

return (

<MaskContainer cover={cover}>

{!disable && <MaskContent>{content}</MaskContent>}

<MaskBlur disable={disable}>{children}</MaskBlur>

</MaskContainer>

);

}

Mask.defaultProps = {

content: null,

disable: false,

cover: false

};

const MaskHOC = withStyleAndClassNameHOC(Mask);

function MaskLoading(props) {

return (

<MaskHOC

content={

<LugiaIcon

type="financial\_loading\_o"

spin

style={{ fontSize: '4rem' }}

/>

}

{...props}

/>

);

}

const MenuTabsContainer = styled.div`

overflow: hidden;

& > div {

display: flex;

${props =>

['top', 'bottom'].includes(props.tabPosition) &&

'flex-direction: column;'}

height: 100%;

& > div:first-child {

height: auto;

line-height: 3.4rem;

margin: 0;

${props => ['top', 'bottom'].includes(props.tabPosition) && 'padding: 0;'}

border: 0 none;

${props => props.type !== 'line' && 'background-color: #f2f2f3;'}

& > span {

${props =>

['top', 'bottom'].includes(props.tabPosition) && 'display: none;'}

line-height: 2em;

}

& > span:first-child {

top: 0;

}

& > span:last-child {

bottom: 0;

transform: none;

}

& > div {

display: block;

& > div {

display: flex;

${props =>

['left', 'right'].includes(props.tabPosition) &&

'flex-direction: column;'}

height: auto;

& > div {

flex: 1;

left: 0;

bottom: 0;

padding: 0;

line-height: inherit;

border-radius: 0;

box-shadow: none;

& > div:first-child {

display: block;

text-align: center;

padding: 0;

height: auto;

line-height: inherit;

}

& > span {

display: none;

}

}

& > div:last-child {

${props =>

props.type !== 'line' &&

['top', 'bottom'].includes(props.tabPosition) &&

'display: none;'}

}

& > div > div > span {

text-overflow: inherit;

width: 6rem;

overflow: auto;

}

& > div:first-child {

${props =>

['left', 'right'].includes(props.tabPosition) &&

`

left: auto;

bottom: auto;

width: 0.1rem;

z-index: 99;

background-color: #4D63FF;

`}

}

}

}

& > div:last-child {

display: none;

}

& + div {

flex: 1;

display: block;

position: relative;

overflow: auto;

}

}

}

`;

function MenuTabs(props: MenuTabsProps) {

const { type, paneType, tabPosition, children } = props;

const tabsChildren = React.Children.map(children, pane => {

if (React.isValidElement(pane)) {

const { key } = pane;

const { tab, children: paneChildren } = pane.props;

return {

title: tab,

content: paneChildren,

activityKey: key

};

}

return pane;

});

return (

<MenuTabsContainer type={type} tabPosition={tabPosition}>

<Tabs {...props} tabType={type} pagedType={paneType} data={tabsChildren}>

{/\* {tabsChildren} \*/}

</Tabs>

</MenuTabsContainer>

);

}