# Routing

The entire routing is based on the react-router-dom module. If you want to use our project in a NextJS application, you have to update the replace our router Link component to use next/link. Also routing in NextJS has to be done using pages folder structure.

## Route config

All our routes can be found in the src/routes.js (src/routes.tsx for TS version). This folder contains renderRoutes method that allows us to pass an array of route configs and convert them in React Nodes wrapped in Suspense, Switch, Guard and Layout components. Sounds complicated but is really simple.

The routes config has the following structure:

type Routes = {

exact?: boolean;

path?: string | string[];

guard?: any;

layout?: any;

component?: any;

routes?: Routes;

}[];

This allows you to have routes in routes. For example, lets say that you have a Blog Layout and you want to have multiple pages to use the same layout, if this is the case you need to do the following strucuture:

// Import your views

const routes = [

{

path: '/blog',

layout: BlogLayout,

routes: [

{

exact: true,

path: '/blog/articles',

component: ArticleListView

},

{

exact: true,

path: '/blog/articles/:slug',

component: ArticleDetailView

}

]

}

];

## Lazy loading routes

Most React apps will have their files “bundled” using tools like Webpack, Rollup or Browserify. Bundling is the process of following imported files and merging them into a single file: a “bundle”. This bundle can then be included on a webpage to load an entire app at once.

To reduce bundle size, we use the new lazy and Suspense React methods with the dynamic import() syntax.

import React, {

lazy,

Suspense

} from 'react';

import LoadingScreen from 'src/components/LoadingScreen';

const App = () => {

return (

<Suspense fallback={<LoadingScreen />}>

<Switch>

<Route

exact

path="/404"

component={lazy(() => import('src/views/pages/ErrorView'))}

/>

</Switch>

</Suspense>

);

}

lazy takes a function that must call a dynamic import(). This must return a Promise which resolves to a module with a default export containing a React component.

The lazy component should then be rendered inside a Suspense component, which allows us to show some fallback content (such as a loading indicator) while we’re waiting for the lazy component to load.

To use lazy loaded components in our render function instead of importing the views using the standard import syntax, remove it and setup the config as:

{

exact: true,

path: '/blog/articles/:slug',

component: lazy(() => import('src/views/blog/ArticleDetailView'))

}

## Protecting routes

We created route wrappers (Guards) to check the user credentials before displaying the route content. For example, we only allow the dashboard rendering only if the user is logged in and login/register pages for guest users (not authenticated). This can be extended and add one extra layer of security by checking for user role. It’s up to you to decide your protection method