First Install Router:
npm install react-router-dom
npm install react-fontawesome react-simple-sidenav react-slick –save

Importing BrowserRouter in App.jsx

Creating *routes.jsx* file for managing all routes.

Creating class based component for routes.

Inside Switch, create route for *home* component.

home inside *Components/Home* folder.

Create high order component for header/footer etc.

Creating *Layout* as HOC in *hoc/layout* folder.

Layout is a class based component.

In routes, Switch is inside layout.

Create CSS for layout *layout.css*

Using Google Fonts Roboto & copy font family to *layout.css*

We're gonna be using CSS modules we need to eject: *npm run eject*

npm start

We need to enable the CSS modules go to *config/webpack.config.dev & config/webpack.config.prod*

In *webpack.config.dev* few down the line 160: see right above *importLoaders: modules: true*

This didn't work for my project & will come back later ……

Adding Sidenav

Now we will create sidenav. It needs a state to know if it is opened or closed.

Layout handels state for all it's children so *showNav* is in state for *true and false* and passed in *Header*.

Two props *onHideNav* and *onOpenNav* are also passed which takes *toggleSidenav* function.

Inside *Header* create *SideNav* where all props of header is passed as *...props*

SideNav is in different file inside Header folder *Header/SideNav/sideNav.jsx*

We have installed dependency for sidenav so *import SideNav from 'react-simple-sidenav'*

In *SideNav* component's children, we have all the options or menus like home, about, etc

In *SideNav,* we pass two arguments *showNav* and *onHideNav* which are predefined options.

For *onOpenNav,* we are catching this on *Header* where this FontAwesome bar is opening the *SideNav*

In *FontAwesome* tag for *bars* we have *onClick* function for passing *onOpenNav.*

Now we add style in *SideNav* which takes argument as *navStyle* where we set CSS properties like *background, maxWidth.*

Sidenav Items

Create a component *SideNavItems* rendering it as a children of *SideNav* inside SideNav folder as *sideNav_items.jsx.*

We render div with class *navitem-option* with property *font-weight: 300; font-size: 12px; color: #bababa; padding: 10px; border-top: 1px solid #404040;*

Import *font-awesome, Link* in *sideNav_items.jsx.*

Inside div there is *Link* with directory for home and inside *Link* there is *FontAwesome* which takes attribute *name=”home”* which generates home-like icon and *FontAwesome* is self closing tag.

navitem-option a => color: inherit; text-decoration: none;

navitem-option span => color: #fff; margin-right: 10px;

Now this div is a template for json like object which we will make a map inside a function *showItems.*

Make *item* which is array of objects & each object have attributes => *type: navitem-option(class name of div), icon: ‘home’(different icon for different object), text: ‘Home’, link: ‘/’*

Similarly: *newspaper, News, /news play, Videos, videos arrow-right, Sign In, sign-in arrow-left, Sign Out, /sign-out*

Note: For multiple lines of code we should add return for rendering anything we want.

For id, take second argument as i in map. When we loop arrays in react we need key.

Creating the Footer

components/Footer/footer.jsx and *footer.css*

It is stateless component.

Import *Footer* in *Layout.*

Import *Link* in *Footer.*

Div with class name *footer-footer*

Link for *Home* with classname *footer-logo.* Image inside link same as header.

Div with classname *footer-right* is sibling of Link with copyright text *@NBA 2018 All rights reserved.*

.footer-footer => margin-top: 20px; background: #242424; display: flex; justify-content: center; align-items: center;

.footer-logo img => height: 20px; padding: 10px;

.footer-right => flex-grow: 1; color: #878787; font-size: 12px;

src/config.jsx where global variables we can use in many places of our project.

CurrentYear = (new Date()).getFullYear();

Export *CurrentYear* as an object(not as default)

Replace hard coded year with *CurrentYear*

Install json server: sudo *npm install -g json-server*

Put json file inside the root directory.

Run json server: *json-server –watch db.json –port 3004*

Enter *localhost:3004/articles* in browser to see results.

Adding Slider

In *package.json* inside *scripts: “dev-serv”: “json-server –watch db.json –port 3004”*

Now run json server by *npm run dev-serv.*

components/widgets/NewsSlider/slider.jsx is a class based component *NewsSlider*.

Reference *NewsSlider* in *Home*

In *NewsSlider* empty *news* array inside state

After everything is ready(mounted) we will make a request to localhost/articles by lifecycle method *componentWillMount()*

Install *axios* which is a library to make request just like ajax.

In our directory: *npm install axios –save*

Will be importing *axios* in *slider.jsx: import axios from ‘axios’*

axios.get(`http://localhost:3004/articles`)

```
.then( response => {
    console.log(response);
} )
```

We have bunch of information we don’t really need. Info we want is inside *data* so *response.data*

Now we get array of objects

```
.then( response => {
    this.setState({
        news: response.data
    })
} )
```

Now when we console log news inside render and see, first it will display empty array and then array of articles. This is because when there is large data to load *componentWillMount()* will take some time. So it is waiting and go to *render.* It will not wait till the load is complete.
After some time *componentWillMount()* will change the state and *render* reloads when the state is changed.

Now we don’t wall all of the database to show because it is huge in general. Suppose we want only latest 3 articles we want to show. For that

axios.get(`http://localhost:3004/articles?_start=0&_end=3`)

We will get article from id: 0 to id: 3

NewsSlider will return *SliderTemplates* component with *data* as *news.*

Create *widgets/NewsSlider/slider_templates.jsx* where *SliderTemplates* is stateless component

Slider Part-2

Slick is slider dependency so in *slider_template: import Slick from ‘react-slick’*

SliderTemplates returns *Slick* component which needs some configuration

Create a variable *settings* where: *dots: true, infinite: true, arrows: false, speed: 500, slidesToShow: 1, slidesToScroll: 1*

In *Slick* pass settings as an object using sperad operator.

We want to create a template and pass it on *Slick* child as a variable.

We want to create different template according to the *type.* In *NewsSlider, SliderTemplates* has another atrribute *type=‘featured’*.

At first *template* is a variable which is defined null.

In *switch-case* statement we will define a condition for *featured* which will return specific template by mapping the data.

Map returns parent div with key then childdiv with classname *slider-featured-item* then it has a child div with classname *slider-featured-image* with inline styling:
background: `url(../images/articles/${item.image})`

This div has a sibling *Link to={`/articles/${item.id}`}*

Link has child div with classname *slider-featured-caption* which contains *item.title*

Create CSS with *.slider-featured-item=> position: relative;*

slider-featured-item a=> position: absolute; width: 100%; bottom: 0; right: 0; text-decoration: none;

slider-featured-image=> height: 330px; background-size: cover !important;

slider-featured-caption=> color: #fff; top: 0; width: 100%; padding: 20px; font-weight: 300; font-size: 28px; box-sizing: border-box;

Slider Part-3

In *Home* in *NewsSlider* pass argument *type='featured'* to define all the attributes in parent component rather than hardcoding in clild component. Then get the type in child component by *this.props.type*

Add more arguments in *NewsSlider* like: *start={0} amount={3}* so in axios we can use template strings and pass the variable. Another is we can add *settings* in *Home* just some properties as an object with double curly braces which we can override it's value in *settings* of *NewsSlider.* For this there is ES6 syntax where at the end we can specify by spread operator at the end of objects just add: *...props.settings*

In *Home* pass another *NewsSlider* and pass different arguments.

News List Cards

Now *NewsSlider* has a sibling *NewsList* component. File *widgets/NewsList/newsList.jsx* is class component *NewsList.*

We will have *load-more* button which requires transitions so import *CSSTransition and TransitionGroup.* Import *Link and axios.*

Npm install react-transition-group –save

Different *NewsList* template for different sections. We will pass arguments *type='card', loadmore= 'true'//for displaying or hiding load-more button, start={3}//remaining list from slider, amount={3}*

Inside *state=> items: [], start: this.props.start, end: this.props.start + this.props.amount, amount: this.props.amount*

Network request inside *componentWillMount()*

In *axios* when we pass url for database, the url can change within time. So save url in *config* and use as a variable in order to repeat this url in every network request in our project.

In *config.jsx: const URL = 'http://localhost:3004'* use this in all network request.

In *axios getting data from network.then setState* using spread operator because when we click *load more* we want to add another list of database in current list: *items: [...this.state.items, ...response.data]*

News List Load More

There is different type of news template so inside *newsList, renderNews(this.props.type).* Inside *renderNews* there will be switch case statement to set template according to type.

When making template main div with key to wrap up all element. Then div with classname *news-newsList-item.* Child *Link* to with template strings for *articles/item.id.* Child *h2* having title & create *newsList.css: news-newsList-item => border: 1px solid #f2f2f2; background: #fff; margin-top: 0; padding: 8px 5px 0 5px; news-newsList-item h2 => font-size: 13px; line-height: 21px; color: #525252 news-newsList-item a => text-decoration: none; flex-grow: 1;*

Create a button *Load More* and generate function to add items.

Put axios inside of a function *request(start, end)* because the request is changing with time.

Add a div with *Load More* button with *onClick* for *loadMore()*

News List Transitions

Each of the news comes with some transitons so *renderNews* inside *TransitionGroup* where *component='div' className='list'.*

Now when templating put everything inside *CSSTransition.* Set classnames only when the item is entered. Setting classnames as an object=> *enter: news-newsList-wrapper, enterActive: news-newsList-wrapper-enter* Timeout is 500 Key is *I*

CSS:: *.news-newsList-wrapper { box-sizing: border-box; opacity: 0; transform: translateX(-100%);*

transition: all .5s ease-in; } .news-newsList-wrapper-enter { opacity: 1; transform: translateX(0%); }

There is repeated button so lets make a component for button:

Here we have button component with *type= 'loadmore' loadmore={this.loadMore()} btnText= 'Load More News'.*

File *widgets/Buttons/buttons.jsx:* functional component B*utton* where templating according to switch case. Give classname *btn-blue-btn* and onClick to parent div. Style it with *text-align center background 2196f3 border-bottom 1px solid d7d7d7 font-weight 500 color eee padding 9px cursor pointer font-size 15px*

In database we have teams but in articles database team is only given a number. So we have to fetch details from team and match the number.

News List Teams

We will have a object in State called *teams* and we will only assign once from the database only when the length of team is zero. So in *request*, put a conditional if the length of team is less than one then use axios to get value.

Create *widgets/CardInfo/cardInfo.js* where *CardInfo* is stateless functional component. It has div with classname card-*info* has child span with classname *team-name* has a sibling span with classname *team-date* has child FontAwesone with *name= "clock-o"* which contains date in it.

Styles:: *.card-info { font-size: 11px; } .card-info i { margin-right: 5px; } .team-name { margin-right: 7px; background: #d1d1d1; color: #fff; padding: 2px 5px; } .team-date { color: #2196f3;}*

In *NewsList, CardInfo* component is just ahead of *title* where we pass arguments like *teams:* whole team array, *team:* team object of article, *date:* date object of article.

Now to figure out the team name, we create arrow function *teamName* with two arguments *teams* and *team.* We will use ES6 *find()* function :

```
let data = teams.find(( item ) => {
    return item.id === team;
} );
return data.name;
```

Videos List

Component *VideosList* in *Home* with props *type=card, title=true(*we can choose to have to not have title*), loadmore=false(*we can choose to display or not display loadmore button*), start=0, amount=3.*

Widgets/VideosList/videosList.jsx has Class Component *VideosList.* In State it has *teams, videos, start, end* is start + amount, *amount.*

In *VideosList div* with class name *videos-list-wrapper* with child title where we pass *renderTitle(this.props.title)* for logic to pass or not pass title according to props. Use ternary operator.

Style:: *.videos-list-wrapper { text-align: center; margin: 20px; color: #353535; font-weight: 300; font-size: 21px; }*

Along with *renderTitle* we pass *renderButton()* where ternary operator asks if the type is *loadmore* then pass *Button* component with *type=loadmore, loadmore=(returns loadMore function defined above), btnText=Load More Videos.* Else button *type=linkTo, btnText=More Videos linkTo=/videos* route. Define this in switch case function in *Button* component.

Template for *linkTo: Link* with to is *props.linkTo,* class name *.btn-blue-btn. Link* is not *div* so doesn't contain full width so go to class and add *display: block; text-decoration: none;*

Videos List 2

In *VideosList* make a function that takes the request for teams and videos same as we did in *NewsList.* Function *renderVideos* between *title* and *loadmore.* For *VideosList* we currently have type *card* but we can have other types as well so in *renderVideos* function we will use switch case to return template.

In case *card* we will pass a component *VideosListTemplate* with props data=this.state.videos teams=same

Widgets/VideosList/videosListTemplate.js where *VideosListTemplate* is stateless function where it maps data and returns: *Link* with *to=/videos/item.id key=i* then div with class name *videos-list-item-wrapper* has a child *div* with class *left* which contains image for videos. It has inline styling with background image:*DataUrl/images/videos/ $item.image.* This div has a sibling div with class *right* has a child *h2* with title.

Styles:: *.videos-list-item-wrapper { display: flex; border-bottom: 2px solid #d5d5d5; background: #fff; } .videos-list-item-wrapper h2 { font-size: 13px; color: #525252; } .videos-list-left { background-size: cover !important; } .left div { width: 90px; height: 90px; background-size: 40px !important; background-repeat: no-repeat !important; background-position: center center !important; background: url('/images/play.png'); } .videos-list-right { flex-grow: 1; padding: 10px; }*

Videos List 3

We will use *CardInfo* when templating in *VideosListTemplate* and pass *teams, team=item.team, date=item.date* .

Add functionality to *loadMore* in *VideosList* same as we did to *loadMore* in *NewsList.*

When getting *request* by clicking *Load More Videos* or *Load More News* we are not updating *start* & *end* of *state.* So we are getting same repeated data. So update them on *setState* of axios.

Article Post View

Add *Route* in *routes.js* for single page view of articles as *Route* path*=/articles/:id exact component=NewsArticle*

Create *components/Articles/News/Post/index.jsx* has class *NewsArticles* with state *article[], team[]*

Call *axios* in *ComponentWillMount* request to *DataUrl/articles?id=${this.props.match.params.id}* Information we're gonna fetch is going to be data of a particular article like *articles/idOfThisArticle*

Inside *then,* if we console log *response.data,* we will see object like 0: *data we required.* So *let article = response.data[0].* We will require another network request for teams because in news there will be team id and from that id we will fetch team name and logo. So *axios.get URL/teams?id=${article.team} then* response will set state of *article, team: response.data* If we console log *this.state* inside render then we will see article and team objects.

Inside render lets redefine variables like *article=this.state.article* and same for *team.* We return div with class *article-wrapper* which returns two reusable components *Header* and *Body. Header* & *Body* is in same folder.

Header.jsx & Body.jsx are stateless functional component.

In *Header* component, we pass props like *teamData={team[0]} date={article.date} author*

Article Post View 2

In *header.js,* f*unctio*n *teamInfo(props.teamData)* inside div where it returns *TeamInfo* component on ternary operator and pass it's argument as props.

Create *Articles/Elements/teamInfo.jsx* where *TeamInfo* is functional component returns div with class *article-team-header* has child div with class *left* has sibling div with class *right.*

Div *left* is displaying t*hum*b*nai*l so i*nli*n*e* b*ackgrou*n*d* of image *url:/images/teams/${props.team.logo}*

Div *right* has child div with two spans one with *props.team.city* other with *props.team.name.* This div has a sibling div with child *strong* has game stats:
W{*props.team.stats[0].wins*} – L*{props.teams.stats[0].defeats}*

Styles:: *.article-team-header { border: 1px solid #c5c5c5; border-bottom: 0px; background: #fff; margin: 5px 5px 0 5px; display: flex; justify-content: center; align-items: center; } .article-header-left { width: 57px; height: 50px; background-size: 65% !important; background-repeat: no-repeat ! important; background-position: center center !important; } .article-header-right { flex-grow: 1; padding-left: 10px; font-size: 10px; font-weight: 300; }*

In *header* inside return *teamInfo* has another sibling *postData* with *props.date, props.author.* Function *postData* returns component *PostData* with props *data(date, author)*

Elements/postData.js have f*unctio*n*al co*m*pone*n*t PostData* returns div with class *article-post-data* has child div with text *Date: span props.data.date* . This div has sibling div with text *Author: span props.data.author*

Style:: *.article-post-data { border: 1px solid #c5c5c5; border-bottom: 0px; background: #fff; margin: 5px 5px 0 5px; padding: 10px; font-size: 13px; font-weight: 300; } .article-post-data span { font-weight: 500; }*

Video Post View

Delete *body.jsx* it’s not reusable we will make body directly in *index.jsx*

Div with class *article-body* child *h1* with title: *article.title . A*nother child div with class *article-image* with style: background image *url:/images/articles/$article.image. Article-image* has sibling *article-text* having *article.body.*

Style:: *.article-body { background: #fff; margin: 0 5px; border: 1px solid #c5c5c5; padding: 10px; } .article-body h1 { font-size: 21px; font-weight: 400; color: #4d4d4d; margin: 15px 0; } .article-image { background-size: cover !important; width: 100%; height: 200px; background-position: center center !important; background-repeat: no-repeat !important; } .article-text { font-weight: 300; color: #666666; line-height: 23px; margin: 15px 0; }*

In *routes.jsx* add route to path:*/videos/:id exact component=VideoArticle.*

Create *Articles/Videos/Video/index.jsx* is a Class component *VideoArticle* . In the wrapper of Video Article we have logo with info, title, video and related videos. For related videos: inside db.json if we see in every video list there is tags we will compare main video tag with other videos in the list.

This class will have state *article[], team[].* If we see index of Post we will have same logic of *componentWillMount().* In render console log state and see data is actually saved or not.

In render redefine article and team. In return: inside div make a reference to *Header* component receiving *teamData=team[0]*

Related Videos

In *index, Header* has sibling div with class *single-video-wrapper* with *h1 article.title.* We need *iframe* to render video:

<iframe

title= ‘videoplayer’

width= ‘100%’

height= ‘300px’

src={`https://www.youtube.com/embed/${article.url}`}

/>

Style:: *.single-video-wrapper { background: #fff; margin: 0 5px; border: 1px solid #c5c5c5; } .single-video-wrapper h1 { font-size: 25px; font-weight: 400; color: #4d4d4d; margin: 15px 0; padding: 0 10px; } .single-video-wrapper iframe { border: 0; }*

Now in this case we’re gonna fetch videos related to city. To display cards we already have reusable component. For header we need to fetch the list of all the teams. So after *setState* of *axios*, we call function *getRelated* which wraps all related videos.

Function *getRelated* will get the list of all the teams from *axios* and save as a variable *teams.* Inside *getRelated* if we console log state we will have *article* and *team* so inside *team* we will get city of the particular post. We will use name of this city to use search query in url: *axios.get(URL:/videos?q=${ this.state.team[0].city }&_limit=3)* then setState *teams* and *related*

Now we need card to display related videos, for which we have *VideosList/videosListTemplate.jsx*

After *iframe* we have div with class *related-wrapper* and inside we call *VideosListTemplate* with props *data=props.data, teams=props.teams*

Style:: *.related-wrapper { margin: 0 5px; }*

For News and Videos Page do it yourself.