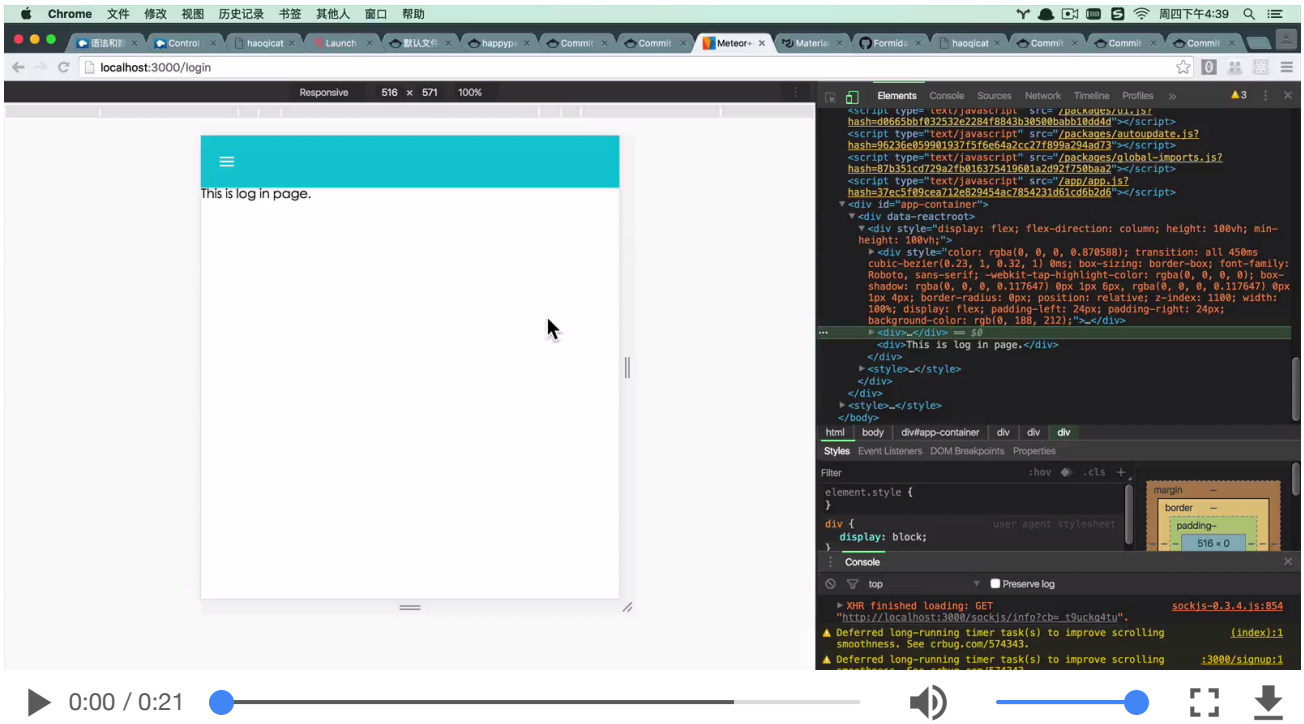

响应式导航栏

导航栏使用的是 Material-UI 的 Tabs 组件，Material-UI 组件不支持 Radium，它使用自己的机制控制组件在不同屏幕设备上的显示样式，所以本节将用另外一种方法完成导航栏的响应式布局。当在小屏幕设备上的时候，用 Material-UI 的 AppBar 组件代替 NavBar 组件。

使用 AppBar 组件

我们将在 App 组件中使用 AppBar 组件，所以要修改 `App.jsx` 文件。首先把问题简化一下，通过判断一个布尔类型的变量，决定显示 NavBar 组件，还是 AppBar 组件，怎样用代码表达出来呢？如下：

```
renderNavBar ? <NavBar /> : <AppBar />
```

`renderNavBar` 值为真时渲染 NavBar 组件，为假时渲染 AppBar 组件。显然，`renderNavBar` 变量要与设备屏幕相关联起来，当在小屏幕设备上时，它的值为假；在大屏幕设备上时，它的值为真。这样，`renderNavBar` 变量值应该随着屏幕尺寸的改变而变化，在 React 组件中，`state` 状态值是可以改变的，所以 `renderNavBar` 实际上是一个 `state` 变量。

```
this.state.renderNavBar ? <NavBar /> : <AppBar />
```

此时，上面这行代码只会渲染 AppBar 组件。下一个要解决的问题就是设置 `state` 变量 `renderNavBar`，最恰当的时刻是在 App 组件将要挂载的时候设置 `renderNavBar` 变量，所以这次又要用到 React 的生命周期方法 `componentWillMount`，代码如下：

```
componentWillMount(){
  this.setState({renderNavBar: window.innerWidth > 700});
}
```

其中，`window.innerWidth` 可以获取浏览器视窗大小，文档参考这里，由此判断设备屏幕尺寸。

到浏览器中访问页面，打开 Chrome 开发者工具，点击**设备**标签，选择要测试的手机设备，刷新页面，就可以看到原先的导航栏消失了，换成了一个带有汉堡包图标的蓝色长条，说明 AppBar 组件已经生效了。不过，关闭开发者工具后，调整浏览器窗口宽度，当宽度大于700像素的时候，页面中渲染的仍然是 AppBar 组件，而不是 NavBar 组件，怎么解决这个问题呢？当浏览器窗口大小改变的时候会执行 `resize` 事件，我们可以在 `resize` 事件对应的事件处理器中设置 `renderNavBar` 的状态值：

```
window.onresize = () => {
  this.setState({renderNavBar: window.innerWidth > 700});
};
```

把上述代码放到 `componentWillMount` 生命周期方法中，完成对 `window.onresize` 事件处理器的定义。

查看更改：使用 AppBar 组件

这些修改都保存之后，再去浏览页面，随意改变浏览器的窗口大小或者刷新页面，导航栏区域会适应屏幕尺寸渲染不同的组件。

添加 AppDrawer 组件

虽然在小屏幕设备上，导航栏区域已经换成了 AppBar 组件，但是还没有实际用处。下面我们将添加一个新的组件文件 `AppDrawer.jsx`，还会用到 Material-UI 的 Drawer 抽屉组件以及 List 组件。

查看更改：添加 AppDrawer 组件

添加的新代码完成的功能是，当点击 AppBar 组件中的汉堡包图标的时候会触发 `onLeftIconButtonTouchTap` 属性对应的 `handleTouchTap` 事件处理器，进一步会执行 AppBar 组件中的 `handleToggle` 方法，从而让抽屉组件显示出来，它默认是隐藏的。当抽屉组件显示出来之后，如何再隐藏它呢？在页面中点击除抽屉组件的其它部分或者按下键盘左上角的 `esc` 按键的时候就可以隐藏它了，这个功能是由抽屉组件的 `onRequestChange` 属性控制的。

另外，这里还涉及到了 React 组件的 Refs 功能。当给一个组件实例添加一个 `ref` 属性之后，比如说本节中的 `ref='drawer'`，就可以通过 `this.refs.drawer` 获取组件实例，进而访问在组件中定义的方法。

实现 AppDrawer 组件的导航功能

现在抽屉组件中的每一个条目，还没有起到导航的作用。下面就参考 Material-UI 的 List 件文档中所列举的 `Selectable list` 例子，实现类似 NavBar 组件中用到的可控模式的 Tabs 组件的功能，当点击某个条目的时候会渲染相应的组件，并让该条目高亮显

示，从而实现导航功能，当然不可避免的又要用到 React Router 的 `context.router` 功能和 React 组件的生命周期函数。

查看更改：实现 AppDrawer 组件的导航功能

对于 `SelectableList` 组件中用到的 `selectedItemStyle` 属性，文档中并没有列出来，但确实是可用的属性，可以参考 Material-UI 功能模块 `MakeSelectable` 的源码

到此为止，我们就直接通过 JS 代码制作了一个响应式的导航栏。