VR Applications using React 360

Transcripts for Tomasz Łakomy (https://egghead.io/instructors/tomasz-lakomy) course on egghead.io (https://egghead.io/courses/vr-applications-using-react-360).

Description

If you've used React, you know how it can provide smart solutions to complex problems. And how exciting that can be.

React 360 brings the same ease and enjoyability to the creation of 3D and VR experiences. It's built on top of React with an additional set of exciting, powerful tools like surfaces, events, and native modules.

In this course, Tomasz Łakomy will show you how to use React 360 to create amazing 3D and VR experiences. You'll build on your React foundation, using the component formatting you already know, and push it to another dimension to create web

apps that can be enjoyed across mobile, desktop, and VR headsets. And you won't need to use crazy complex tools like WebGL to do it — just React!

Following the course, you'll be ready to use React 360 to take an idea from creation to completion in a VR app — for instance, a VR image gallery, game, web browser, or interactive story.

If you're comfortable in React and you're ready for something new, this course is perfect for you.

Start a Virtual Reality project with React 360

Instructor: [0:00] In order to get started with React 360, first install `react-360-cli` from npm. Once this is done, create a new product. Use the `init` command. We're going to call our product `travelVR` because our app will allow people to travel.

```
npm install react-360-cli -g

react-360 init travelVR
```

or

```
npx react-36- init travelVR
```

[0:13] This is going to take a while because React 360 has a lot of dependencies. Once this is done, go to our product directory, and then run `npm start`.

```
cd travelVR

npm start
```

This is going to start React Native packager. This is going to tell us to open our browser at this URL.

[0:25] Once we do that, we need to wait a while in order for all dependencies to be loaded, but once it's done, we can see the result over here.

[0:31] We have this 360 view. We can look around. We have this text, "Welcome to React 360," in front of us. Let's take a look inside of the code. The important thing to know about React 360 is that even though it runs inside of the fancy VR environment, it's not that different from React that we know for the Web.

[0:46] We do have index.html. We also have things like React components. We have views, which you may know from React Native. We also have our stylesheet objects, which basically allows us to write CSS and JS in VR.

[0:59] In order to track whether our project has been set up correctly, change this text to, "`Welcome to Egghead`." Save and refresh that. We can see this change over here, which means that now we are ready to go and implement our app.

index.html

```
<Text style={styles.greeting}>
  Welcome to EggHead
</Text>
```

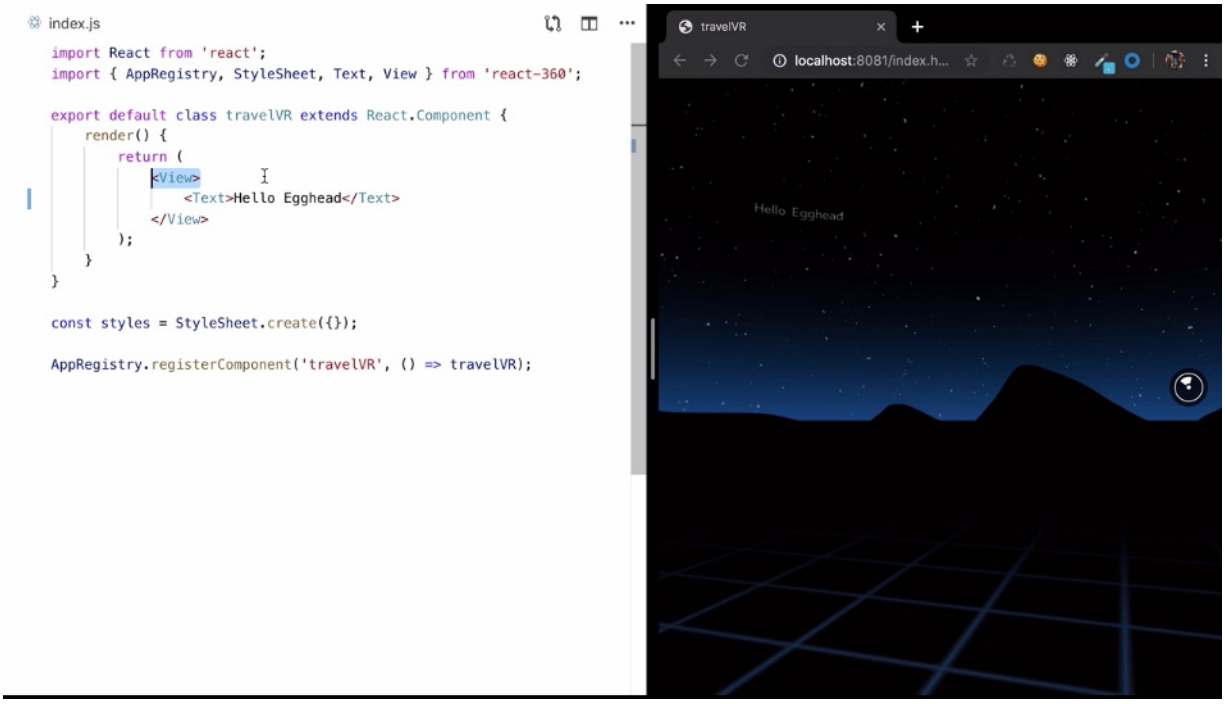
Write text using the React 360 Text component

Instructor: [00:00] Start by removing all the boilerplate. This travelVR component is going to render null, and we are not going to have any styles. Next, render a view component. Inside of this view, we're going to have a `Text` component, like this. I am going to type in `Hello Egghead` inside of this `Text` component.

index.js

```
export default class travelVR extends
React.Component {
  render() {
    return (
      <View>
        <Text>Hello Egghead</Text>
      </View>
    )
  }
}
```

[00:15] After we save and refresh this, we're going to see this text, "Hello, Egghead," appearing over here.



We would like to be able to style both view -- view is basically like a div -- and a `Text` component. To do that, we're going to have a `mainView` and a `greetings` object.

```
const styles = StyleSheet.create({
  mainView: {},
  greetings: {}
})
```

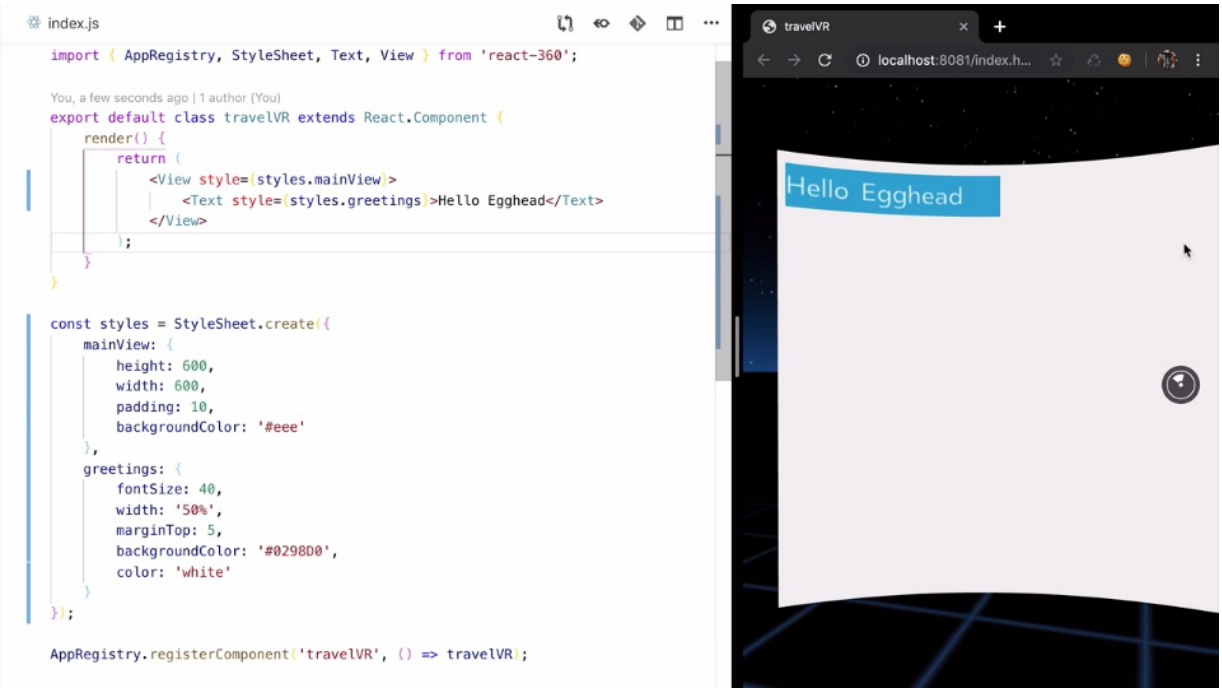
[00:31] Inside of the mainView, we're going to have a `height` of `600`, `width` of `600`, as well as `padding` of `10`, and we're going to set the `backgroundColor` to this shade of gray. Inside of the `greetings` object, `fontSize` of `40`. We're going to set the `width` to `50` percent, `marginTop` to `5`, `backgroundColor` to this shade of blue, and the `color` of the text to `white`.

```
const styles = StyleSheet.create({
  mainView: {
    height: 600,
    width: 600,
    padding: 10,
    backgroundColor: "#eee"
  },
  greetings: {
    fontSize: 40,
    width: "50%",
    marginTop: 5,
    backgroundColor: "#0298D0",
    color: "white"
  }
})
```

[00:59] Each React 360 component takes a style property. We're going to set the style of this view to be equal to `styles.mainView`. We're going to set the style of this text to `styles.greetings`.

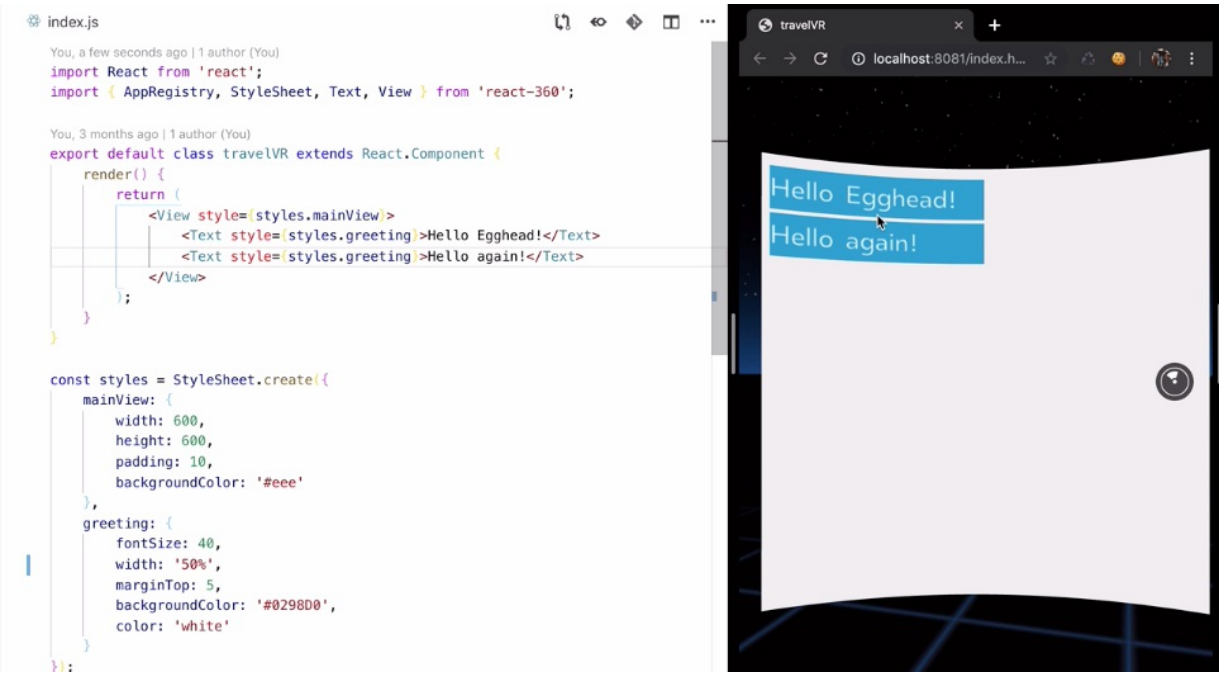
```
export default class travelVR extends
React.Component {
  render() {
    return (
      <View style={styles.mainView}>
        <Text Style={styles.greetings}>Hello
Egghead</Text>
      </View>
    )
  }
}
```

After I save and refresh that, we're going to see the result over here.



[01:16] We can see both the view and the `Text` component styled using the CSS that we've defined over here. We can use the same styles for multiple components. If I were to have another `Text`

component over here, and I would set the text to "`Hello again!`," after I save, refresh that, we're going to have two `Text` components with exactly the same styles applied.



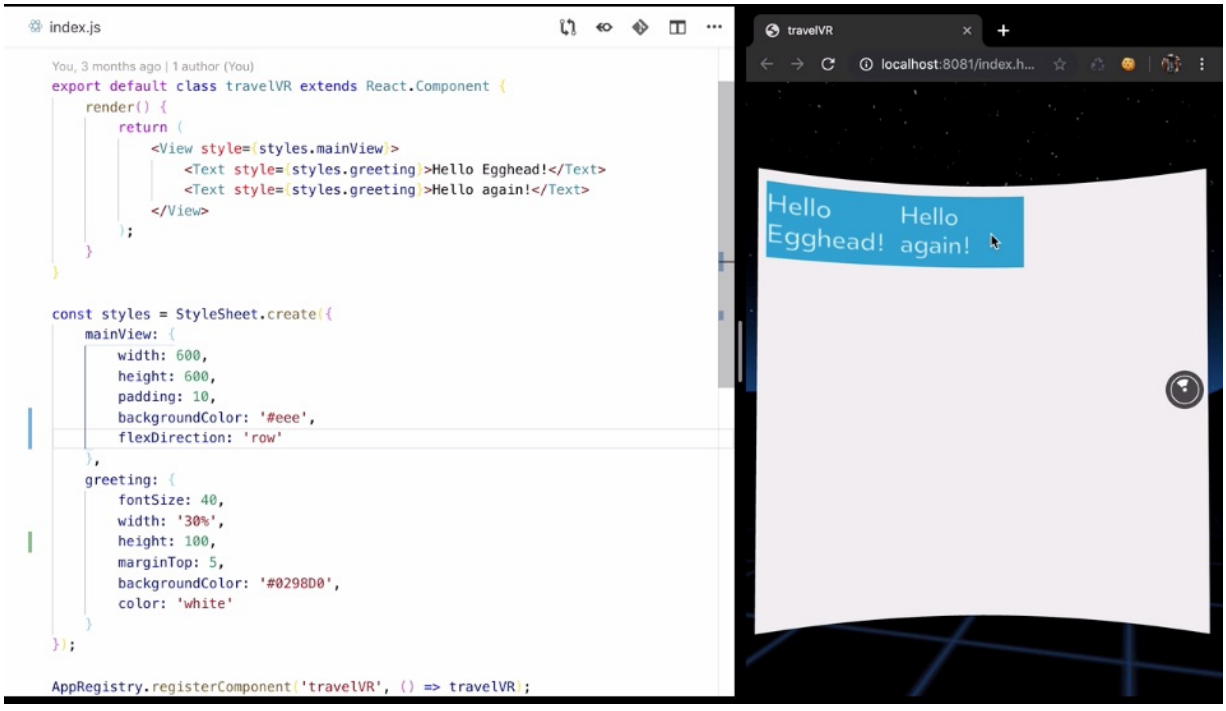
Use flexbox to create layouts in React 360

Instructor: [0:00] We have two text components which are displayed over here. They have a width specified to 30 percent. Even though they could fit in a single line, they have displayed one under another.

[0:10] The reason that it happens is that, by default, React 360 uses Flexbox and the default flex direction is set to column. If I were to change the `flexDirection` of this major component to row, and after I save and refresh that, we're going to see those two text components in a single line.

index.js

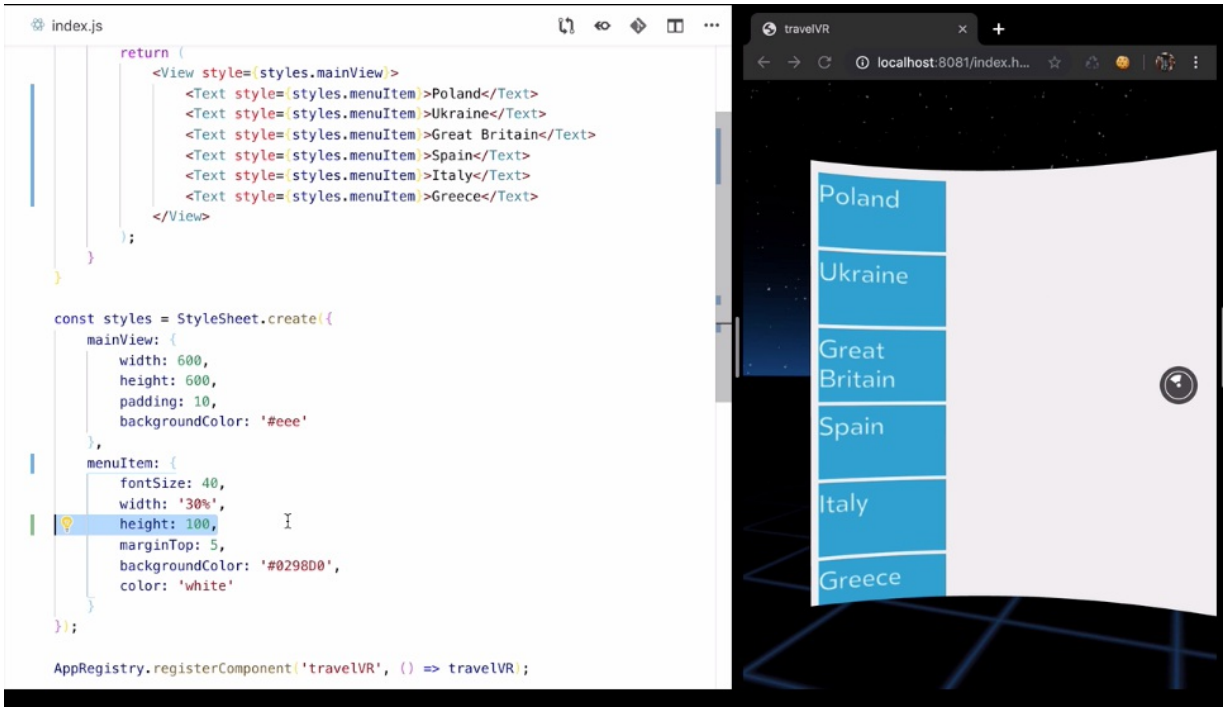
```
const styles = StyleSheet.create({
  mainView: {
    height: 600,
    width: 600,
    padding: 10,
    backgroundColor: "#eee",
    flexDirection: "row"
  },
  greetings: {
    fontSize: 40,
    width: "50%",
    marginTop: 5,
    backgroundColor: "#0298D0",
    color: "white"
  }
})
```



[0:26] Next, we're going to build a list of countries that we would like to visit. I'm going to copy and paste some text components. We're going to change the greeting to `menuItem`, as well as we're

going to remove the `flexDirection` set to `row`. After I save and refresh that, we're going to see all those countries displayed over here.

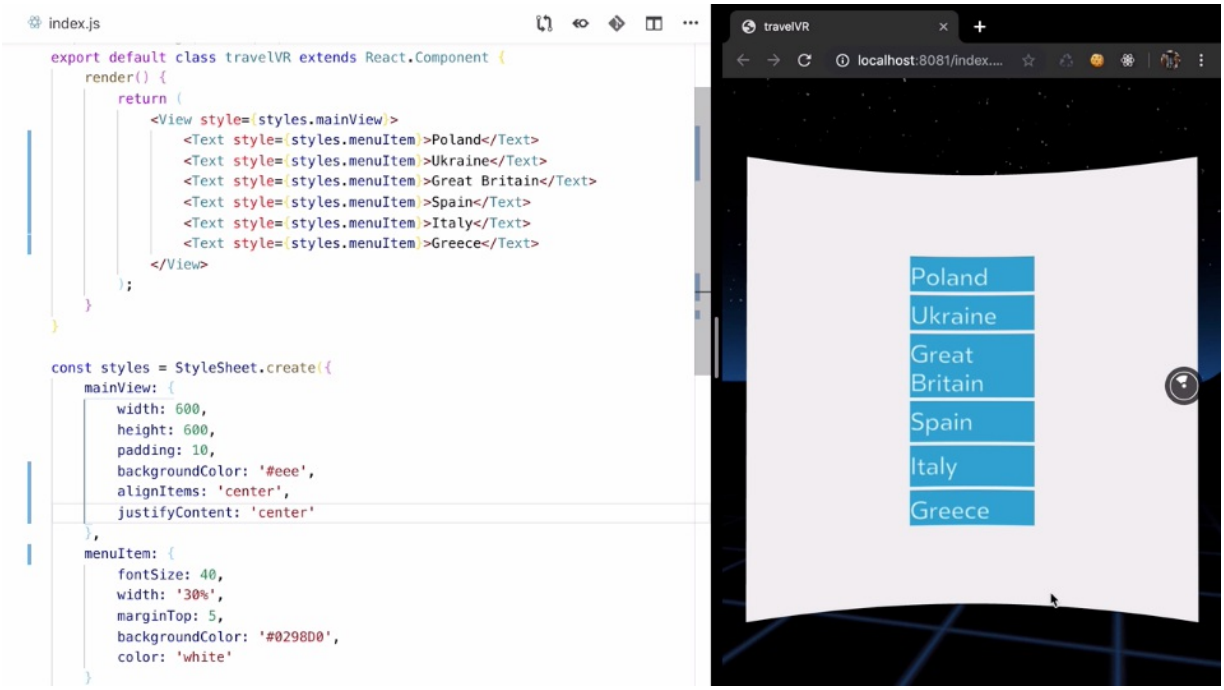
```
<View style={styles.mainView}>
  <Text style={styles.menuItem}>Poland</Text>
  <Text style={styles.menuItem}>Ukraine</Text>
  <Text style={styles.menuItem}>Great
Britain</Text>
  <Text style={styles.menuItem}>Spain</Text>
  <Text style={styles.menuItem}>Italy</Text>
  <Text style={styles.menuItem}>Greece</Text>
</View>
```



[0:42] Next, let's remove this hard coded hide. We're going to center all those countries in the middle of this component. We're going to use Flexbox. I'm going to set the `alignItems` to `center`, as well as `justifyContent` to `center`, as well.

```
const styles = StyleSheet.create({
  mainView: {
    height: 600,
    width: 600,
    padding: 10,
    backgroundColor: '#eee',
    alignItems: 'center',
    justifyContent: 'center'
    },
```

[0:57] After I save and refresh that, we're going to see the result over here. We have the list of all the countries centered both horizontally and vertically inside of this view component.



Assign multiple styles to a component in React 360

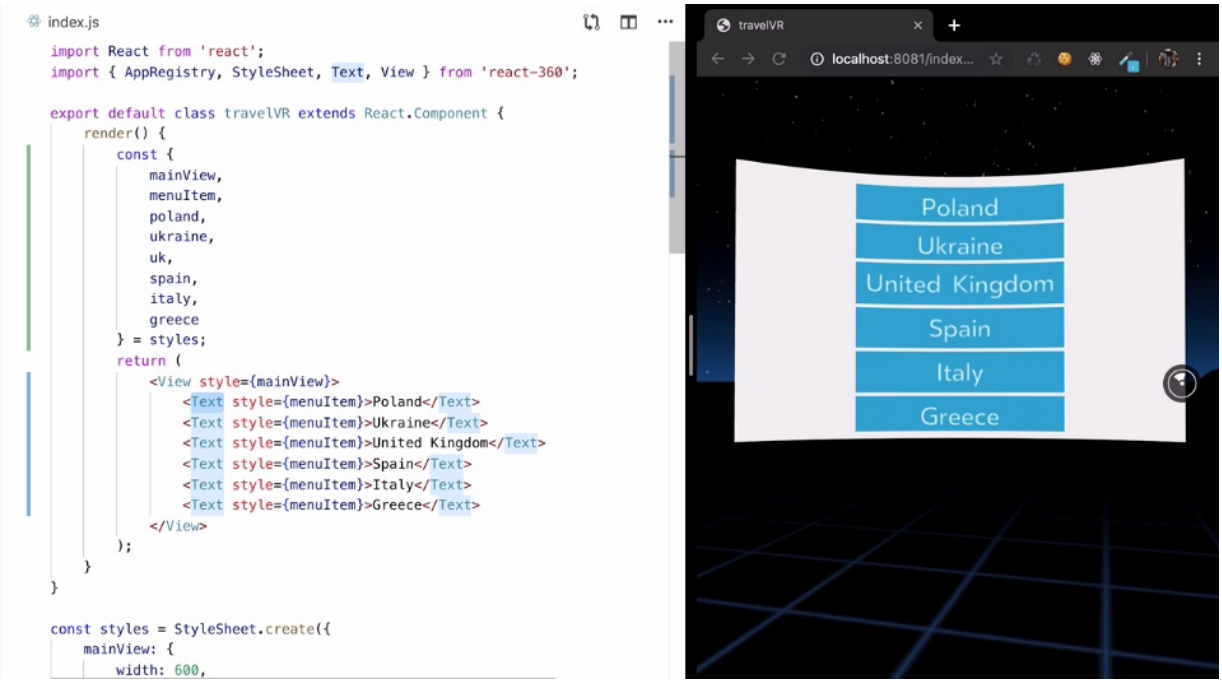
Instructor: [0:00] We have a viewer with the list of the countries that we would like to visit. As well over here in the side sheets, we have a couple of objects per country. Inside of those objects, we

have a background color with the color taken from the flag of each country. What we would like to do is to apply this color to each of the countries in addition to this many item style.

[0:18] First, destructure all the styles from the styles object. We're going to destructure `mainView`, `menuItem`, `poland`, `ukraine`, `uk`, `spain`, and `italy`, as well as `greece`. We're going to destructure those from the `styles` object. Let me save that. Now we're going to remove all of those styles, save, refresh. Obviously, it works. There we go. It said OK. Let me format that.

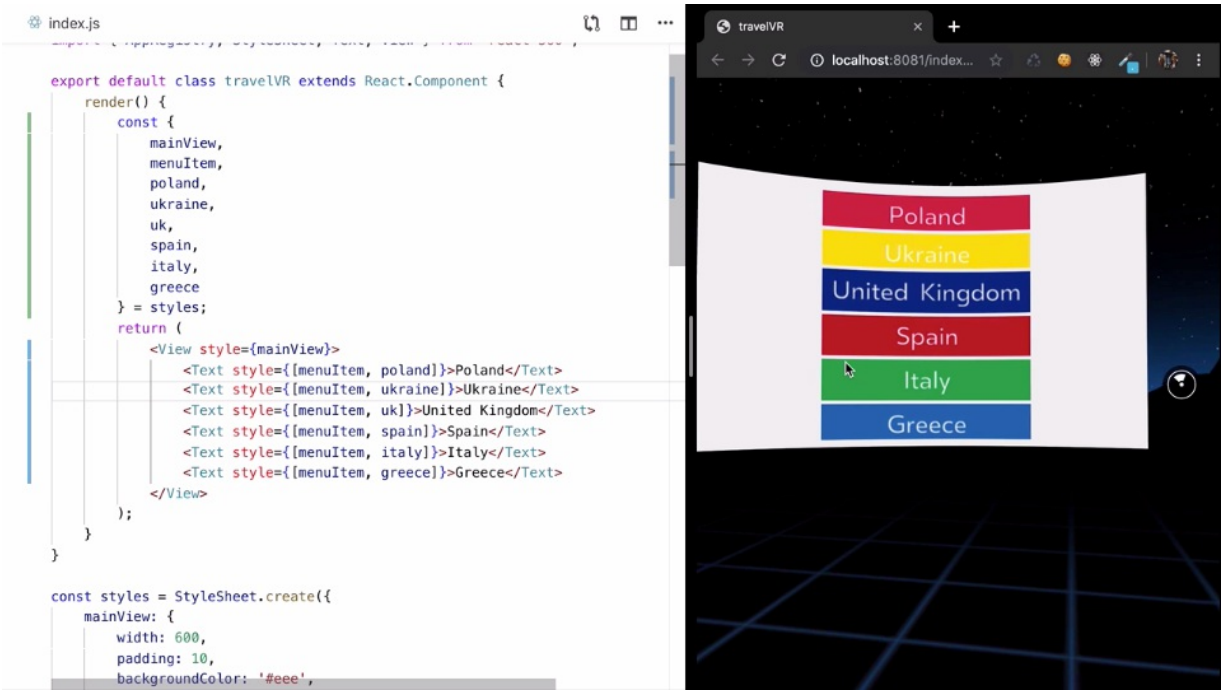
index.js

```
const { mainView, menuItem, poland, ukraine, uk,
spain, italy, greece } = styles
return (
  <View style={styles.mainView}>
    <Text style={menuItem}>Poland</Text>
    <Text style={menuItem}>Ukraine</Text>
    <Text style={menuItem}>Great Britain</Text>
    <Text style={menuItem}>Spain</Text>
    <Text style={menuItem}>Italy</Text>
    <Text style={menuItem}>Greece</Text>
  </View>
)
```



[0:39] Now, in order to provide multiple styles for those text components, what we need to do is to provide an array of styles to the style property. I'm going to wrap all of those inside of an array. We're going to do `poland`, `ukraine`, `uk`, `spain`, `italy`, and `greece`. Then save and refresh that.

```
return (
  <View style={styles.mainView}>
    <Text style={[menuItem,
poland]}>Poland</Text>
    <Text style={[menuItem,
ukraine]}>Ukraine</Text>
    <Text style={[menuItem, uk]}>Great
Britain</Text>
    <Text style={[menuItem, spain]}>Spain</Text>
    <Text style={[menuItem, italy]}>Italy</Text>
    <Text style={[menuItem,
greece]}>Greece</Text>
  </View>
)
```



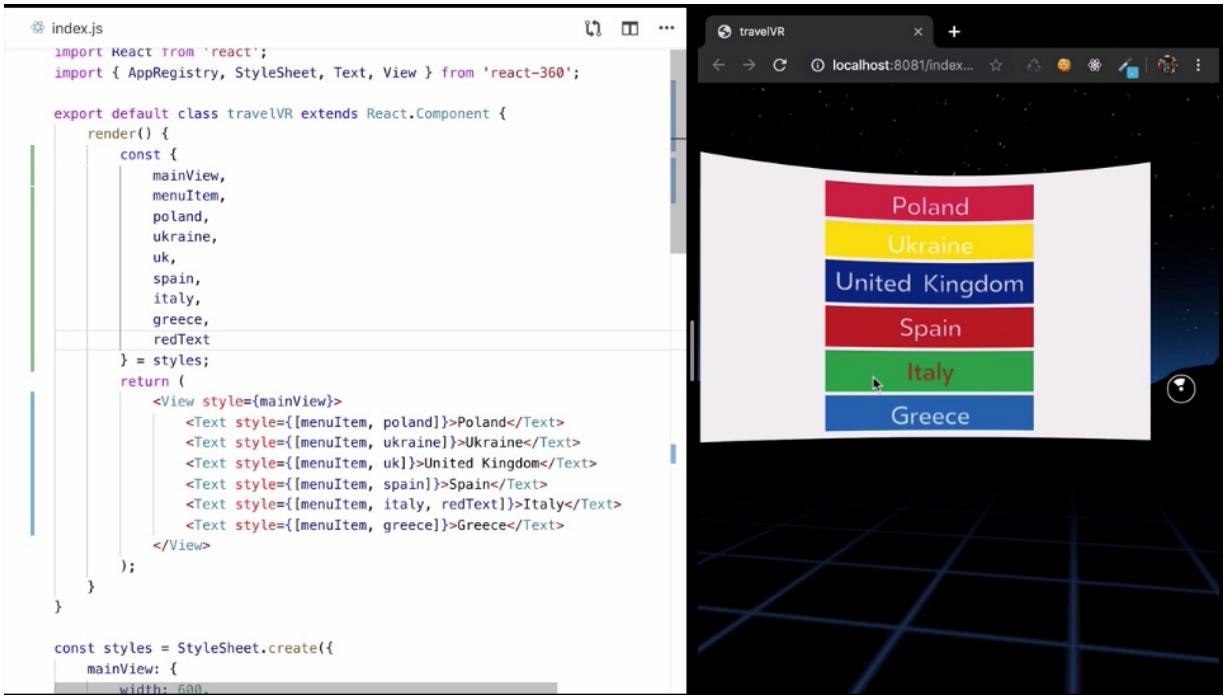
[0:56] Right now, we have the desired effect. Each text element has a background color of the flag of its country. Suppose I would like to have a `redText` style. I'm going to create a `redText` object, set the color to `red`.

```
  menuItem: {
    fontSize: 40,
    width: '50%',
    marginTop: 5,
    backgroundColor: '#0298D0',
    color: 'white',
    textAlign: 'center'
  },
  poland: {
    backgroundColor: '#DC143C'
  },
  ukraine: {
    backgroundColor: '#FFD500'
  },
  uk: {
    backgroundColor: '#00247D'
  },
  spain: {
    backgroundColor: '#C60D1F'
  },
  italy: {
    backgroundColor: '#029246'
  },
  greece: {
    backgroundColor: '#0D5EAF'
  },
  redText: {
    color: 'red'
  }
```

[1:08] We would like to apply this color to Italy. I can just add a `redText` over here and destructure it from styles as well.

```
const {
  mainView,
  menuItem,
  poland,
  ukraine,
  uk,
  spain,
  italy,
  greece,
  redText
} = styles
return (
  <View style={styles.mainView}>
    <Text style={[menuItem,
poland]}>Poland</Text>
    <Text style={[menuItem,
ukraine]}>Ukraine</Text>
    <Text style={[menuItem, uk]}>Great
Britain</Text>
    <Text style={[menuItem, spain]}>Spain</Text>
    <Text style={[menuItem, italy,
redText]}>Italy</Text>
    <Text style={[menuItem,
greece]}>Greece</Text>
  </View>
)
```

After I save and refresh that, Italy is going to get additional style. It's going to have a green background and a red text inside of it.



Display images using the Image component in React 360

Instructor: [00:00] We have a view with the text saying that we should add an image over here. In order to do that, first import `image` from React 360.

index,js

```
import React from "react"
import { AppRegistry, StyleSheet, View, Image }
from "react-360"
```

Next, we move this text component, we move the styles for the text, and we're going to display two flags.

[00:12] First, let me specify some styles. We're going to have a `width` of `50` percent and the `height` of `40` percent.

```
export default class travelVR extends
React.Component {
  render() {
    const { mainView, text } = styles;

    return (
      <View style={mainView}>

      </View>
    );
  }
}

const styles = StyleSheet.create({
  mainView: {
    width: 600,
    height: 600,
    padding: 10,
    backgroundColor: '#eee',
    alignItems: 'center',
  };
  flag: {
    width: '50%',
    height: '40%'
  }
})
```

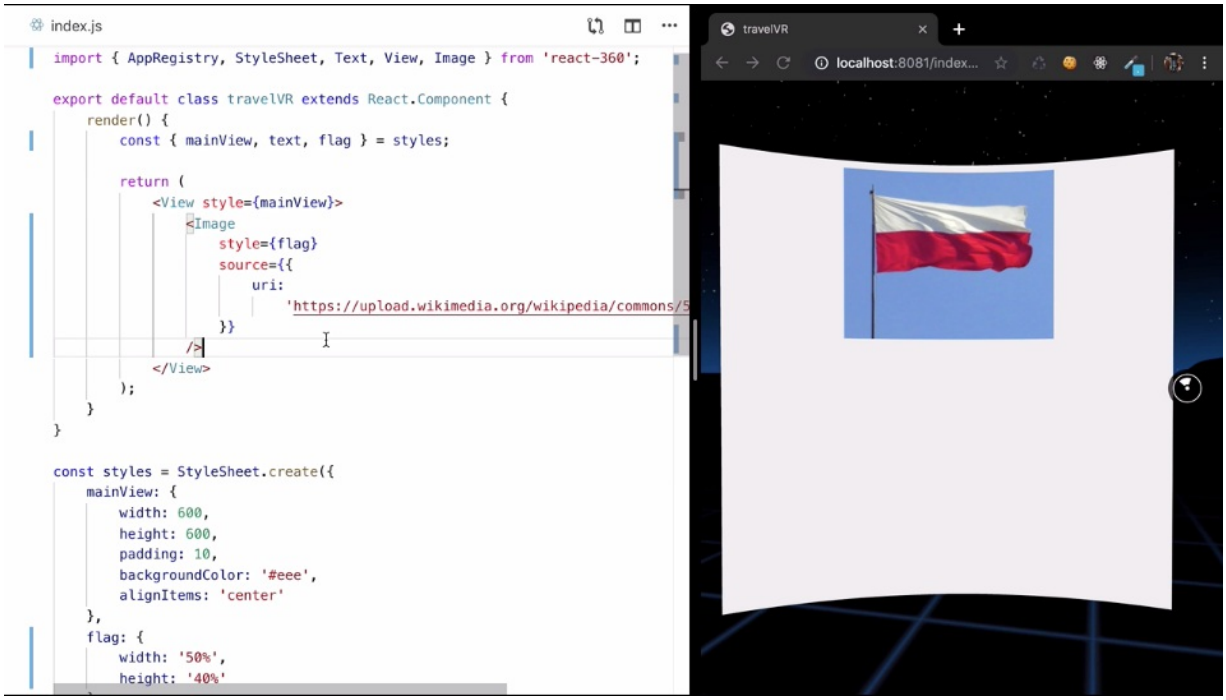
In React 360, we can display images from either the Internet or from assets stored within our project.

[00:26] First, I'm going to just enter this flag style, as well as I'm going to provide an image. I'm going to set the `style` to `flag`. We need to specify the `source` for this image. Let me just copy and

paste that, so the source property takes an object and inside of this object we are specifying that we would like to display this flag component taken from Wikipedia.

```
return (
  <View style={mainView}>
    <Image
      style={flag}
      source={{
        url:
'https://upload.wikimedia.org/wikipedia/commons/5
/5a/Flag_of_Poland.jpg'
      }}
    />
  </View>
```

[00:43] After I save and refresh that, we're going to see the result over here.



We have this flag component displayed. In order to use our own assets, we need to take a look inside of the React 360 project directory.

[00:54] Each React 360 project has a `static_assets` folder. This is where you keep all your static assets. First, on this `360_world_jpg` image, this is what we see when we look around inside of our app. We're going to put this flag into the PNG over here.

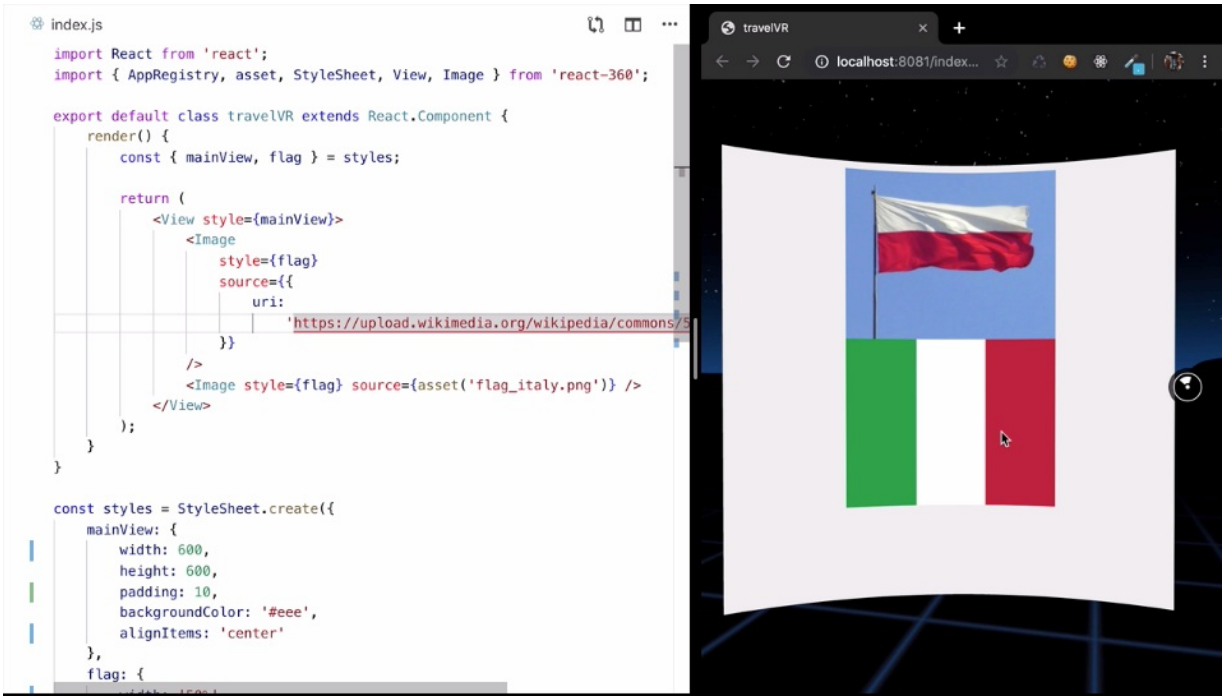
[01:08] Let's go back to `index.js`. In order to use static assets, we need to import `asset` from `react-360`.

```
import React from "react"
import { AppRegistry, asset, StyleSheet, View,
Image } from "react-360"
```

We're going to create another `image` component. We're going to set this `style` to `flag` as well and we're going to provide the source. We're going to set it to `asset flag_italy.png`.

```
return (
  <View style={mainView}>
    <Image
      style={flag}
      source={{
        url:

'https://upload.wikimedia.org/wikipedia/commons/
5/5a/Flag_of_Poland.jpg'
      }}
    />
    <Image style={flag} source=
{asset('flag_italy.png')} />
  </View>
```

[01:25] This asset function is going to take a look inside of the static asset directory and get this `flag_italy.png` for us. After I save and refresh that, we're going to see the result over here. We have two images displayed. One is from external source and the other one is from our static assets directory.



Create a Cylinder Surface and attach a component to it in React 360

Instructor: [0:00] So far, we've been creating React components inside of this index.js file, and those components were somehow projected onto this 360 view. How does it happen?

[0:09] The answer is, within the `client.js`, which a second JavaScript file that gets created by default once you create a new React 360 project. Essentially, each Reach 360 application is made out of two parts. There's your React code and the code is going to take your React components and actually project them onto 360 surface.

[0:27] The second piece is what we refer to as the run time. The run time is specified within the `client.js` because this is where your React 360 project gets initialized. Let's focus on this part.

[0:39] Here, we're entering to our surface, our `travelVR` component. The `travelVR` component is this component we have been working on so far. Here, we're using a `DefaultSurface`.

[0:49] In React 360, there are two types of surfaces. There is a cylinder surface and a flat surface. Let's take a look at the cylinder surface because this is what we have been using so far.

[0:58] If you look over here, you will notice that both the top and the bottom of our component is curved because this component is actually projected on a cylinder. The cylinder is actually a sphere of a four-meter radius that is surrounding the entirety of the user view.

[1:13] We can project different components on top of the surface, such as this travel vr component. Let's stop using the default surface and create our own. First, import `Surface` from `react-`

`360-web`.

client.js

```
import { ReactInstance, Location, Surface } from
"react-360-web"
```

Here, we're going to create a new surface. I am going to call it as `mySurface`. It's going to be a `new Surface`.

[1:29] We need to specify a width, which I am going to set to `4,680`. I am going to set the height to `600`. I am going to set the `Surface.SurfaceShape.Cylinder`.

```
const mySurface = new Surface(4680, 600,
Surface.SurfaceShape.Cylinder)
```

The reason for the specific width is that if we set the width to `4680`, we create a surface that's going to wrap the entirety of the user's view.

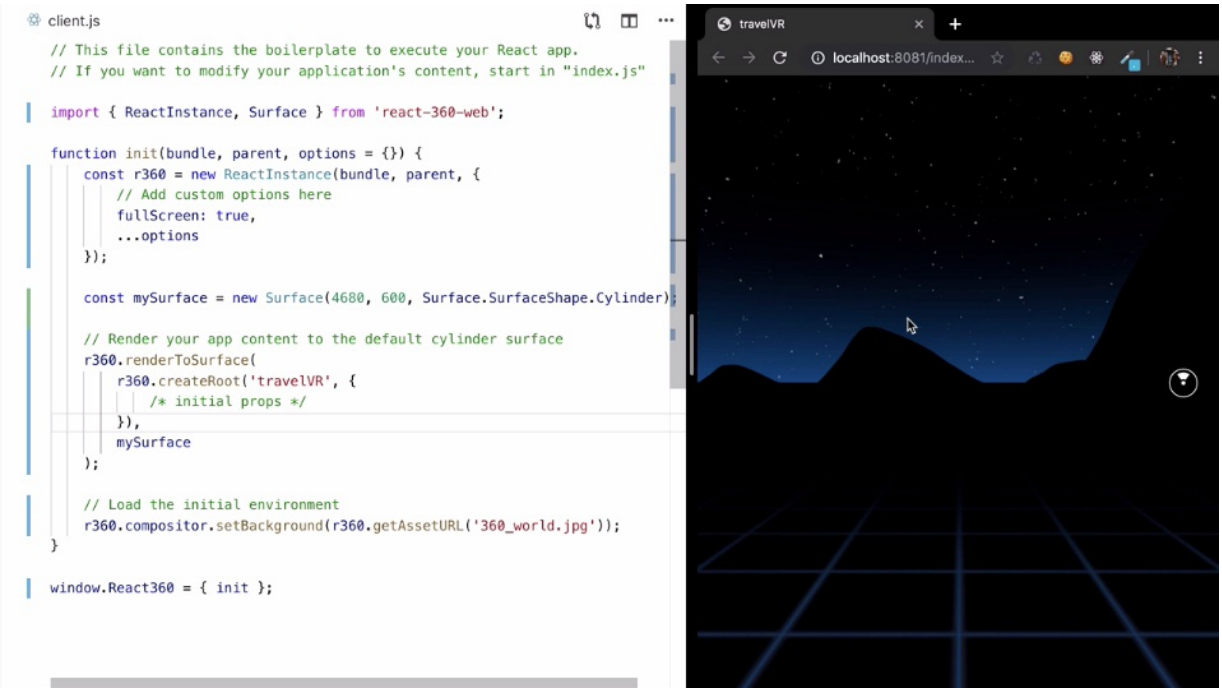
[1:49] What we're going to do is that we're going to have a React component that is going to be displayed everywhere around the users. No matter which way we look, we are always going to see this component.

[1:59] To use this new surface, remove this default surface and use it like this. We are rendering this travel vr component to our newly created `mySurface`. Let me fix it, because I have a typo over here. Save and refresh.

```
const mySurface = new Surface(4680, 600,
Surface.SurfaceShape.Cylinder)

// Render your app content to the default
cylinder surface
r360.renderToSurface(
  r360.createRoot("travelVR", {
    /* initial props */
  }),
  mySurface
)
```

[2:12] At the first glance, it would seem that our component has disappeared.

See 2:13 in the lesson



This is not the case. What happens is that our component will simply move to the beginning of the surface and display over here.

[2:21] Instead of having this view of flags, we're going to create a component that's going to be in front of us no matter which way we look.

[2:28] To do that, go back to `index.js`. Remove both of those flags. We won't need them from now. Remove this flag. Remove those flags as well. We're going to set the `width` of this component to `4,680` so it's exactly the same as the `width` of the cylinder.

index.js

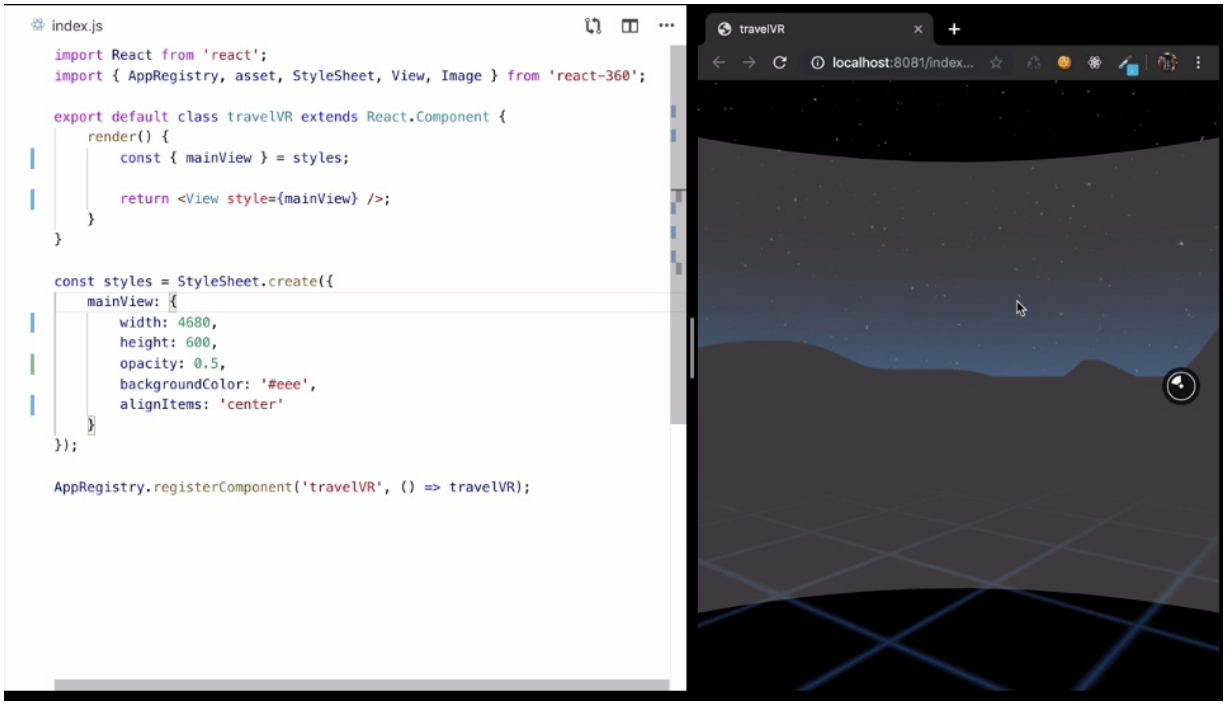
```
export default class travelVR extends
React.Component {
  render() {
    const { mainView } = styles

    return <View style={mainView} />
  }
}

const styles = StyleSheet.create({
  mainView: {
    width: 4680,
    height: 600,
    backgroundColor: "#eee",
    alignItems: "center"
  }
})
```

[2:42] I am going to set `opacity` to `50` percent so that I can actually see what's behind my component after I save. First [inaudible] , we have the desired effect.

```
const styles = StyleSheet.create({
  mainView: {
    width: 4680,
    height: 600,
    opacity: 0.3,
    backgroundColor: "#eee",
    alignItems: "center"
  }
})
```

Right now, we have this component displayed on the cylinder. No matter which way I look, I always see this component in front of me.

See 2:53 in lesson



Create a Flat Surface and attach a component to it in React 360

Instructor: [0:00] The second type of a surface available in React 360 is the flat surface. We're going to use it to create a `Flag` component, and we're going to display this component on a flat surface somewhere over here.

[0:09] First up, create a `components` directory, and inside of it, create a `flag.js`. Next, import `React` from `react`, as well as import `asset`, `Stylesheet`, and `Image` from `react-360`. We're going to create a new class, we're going to call it flag, and we're going to extend it from react.component.

[0:27] We're going to `render` a `flag` component, so we're going to destructure `flag` from `styles`, and we're going to `render` an `Image`. We're going to set the `style` to `flag`, and we're going to set the `source` to whatever was provided in the props.

Flag.js

```
import React from "react"
import { asset, StyleSheet, Image } from "react-
360"

export default class Flag extends
React.Component {
  render() {
    const { flag } = styles

    return <Image style={flag} source=
{asset(this.props.image)} />
  }
}
```

[0:41] Let me just create a `StyleSheet` object. We're going to have a `StyleSheet`, and the `flag` is going to have a `height` of `400` and a `width` of `600`. In order to use this component inside of the runtime, we need to first register it.

```
const styles = StyleSheet.create({
  flag: {
    height: 400,
    width: 600
  }
})
```

[0:53] We need to go to `index.js`, and here, we are registering the `travelVR` component. Because this component has been registered, it's available to be used in client.js. This is where our runtime lives. What we need to do is we have to register a `Flag` component, and we need to import it as well.

[1:10] I'm going to import `Flag` from `./componenets/Flag`.

```
import React from "react"
import { AppRegistry, asset, StyleSheet, View,
Image } from "react-360"
import Flag from "./components/Flag"

export default class travelVR extends
React.Component {
  render() {
    const { mainView } = styles

    return <View style={mainView} />
  }
}

const styles = StyleSheet.create({
  mainView: {
    width: 4680,
    height: 600,
    opacity: 0.3,
    backgroundColor: "#eee",
    alignItems: "center"
  }
})

AppRegistry.registerComponent("travelVR", () =>
travelVR)
AppRegistry.registerComponent("Flag", () =>
Flag)
```

Let me save that, and we're going to jump into `client.js`. Over here, create a new flat surface. I'm going to do `const myFlatSurface`, and it's going to be a `new Surface` of `600` by `400`.

[1:29] I'm going to set the `Surface.SurfaceShape.Flat`.

client.js

```
const mySurface = new Surface(4680, 600,
Surface.SurfaceShape.Cylinder)
```

Now, I need to render a component to it. I'm just going to copy and paste this bit. Let me just do it like this. I don't want to render a travel VR component. I want to render the `Flag` component, and I would like to render it to `myFlatSurface`.

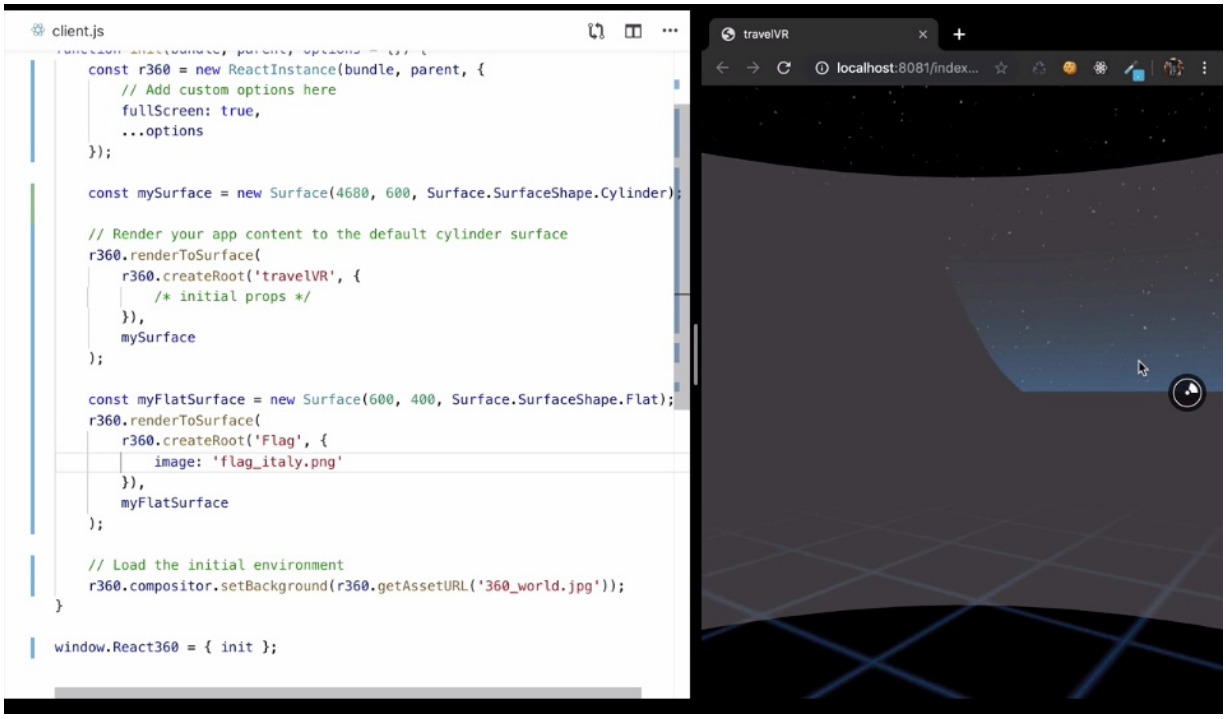
```
const myFlatSurface = new Surface(600, 400,
Surface.SurfaceShape.Flat)
r360.renderToSurface(
  r360.createRoot("Flag", {
    /* initial props */
  }),
  myFlatSurface
)
```

[1:47] Over here, we can provide the props that we would like to pass into the `Flag` component. I'm going to pass in an `image` prop, and I'm going to set it to `flag_italy.png`.

```
const myFlatSurface = new Surface(600, 400,
Surface.SurfaceShape.Flat)
r360.renderToSurface(
  r360.createRoot("Flag", {
    image: "flag_italy.png"
  }),
  myFlatSurface
)
```

Now, and if I save and refresh that, we are not going to see the flag.

See 1:58 in lesson



[1:59] The reason it happens is that we haven't specified where exactly we want to display this flat surface. Both flat and cylinder surfaces are displayed four meters away from the user, but in the case of the flat surface, we need to specify at which angle we want to display this flat surface.

[2:14] What we're going to do is that we're going to do `myFlatSurface.setAngle`.

```
const myFlatSurface = new Surface(600, 400,
Surface.SurfaceShape.Flat)
myFlatSurface.setAngle()
```

This function takes two arguments, how much we want to rotate our flat surface left and right, and how much we want to rotate it up and down.

[2:25] We can specify it in both radians and in degrees. I'm going to start with radians. I would rotate this by `Math` by `four`. The second number, the pitch angle, is going to be equal to `zero`.

```
const myFlatSurface = new Surface(600, 400,
Surface.SurfaceShape.Flat)
myFlatSurface.setAngle(Math.PI / 4, 0)
```

There you go. We can see the flat surface over here.

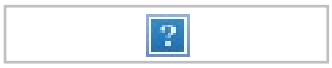
[2:39] If I wanted to, I can get rid of radians, because those are not intuitive. I can change it to `45` degrees, and I have exactly the same effect.

```
const myFlatSurface = new Surface(600, 400,
Surface.SurfaceShape.Flat)
myFlatSurface.setAngle(45 / 0)
```

If I were to modify the pitch angle to be also equal to 45, I would get the result of this flag of Italy displayed over here.

```
const myFlatSurface = new Surface(600, 400,
Surface.SurfaceShape.Flat)
myFlatSurface.setAngle(45 / 45)
```

See 2:54 in the lesson



[2:55] The main difference between a flat surface and a cylinder surface is that a flat surface is not curved. If I were to look at this flat at an angle, what I'm going to see is that there's a slight distortion over here. Whereas with a cylinder surface, no matter which way I look, I am going to see this component in exactly the same way.

Add 3D objects to a React 360 application

Instructor: [00:00] To import our 3D model in React 360 application, go to `components` and create a new file that we're going to call `arrow.js`. Here, `import React` from `react`, and we're going to also `import asset` and `View` from `react-360`.

Earth.js

```
import React from "react"
import { asset, View } from "react-360"
```

[00:12] In order to display 3D models inside of React 360 application, we need to `import Entity` from `Entity`. entity is also part of React 360, but you have to import it in this way. We're going to create a new component. We're going to call it arrow.

[00:25] It's going to extend `React.Component`, and we're going to render a view and the `entity` inside of it.

```
import React from "react"
import { AppRegistry, asset, View } from "react-
360"
import Entity from "Entity"

export default class Earth extends
React.Component {
  render() {
    return (
      <View>
        <Entity></Entity>
      </View>
```

`Entity` allows us to display external 3D models. We can either design something, or we can go to a website, such as Google Poly, and download a free model for our application.

[00:40] We can download this model in a couple of different formats. React 360 supports both object file and GLTF file. GLTF basically like a JPEG for 3D, and we're going to use this one. I'm going to download it and put it inside of our static assets directory.

[00:55] Here, we can see it inside of the static assets directory. To assign this model to the `Entity` component, we need to specify a source. It's going to take an object, `gltf2`. We're going to set it

equal to `asset Earth.gltf`.

```
export default class Earth extends
React.Component {
  render() {
    return (
      <View>
        <Entity source={{ gltf2:
asset('Earth.gltf') }}/>
      </View>
```

[01:09] Let me save that, and right now, what we need to do is to register this `Earth` component. In order to do that, go to `index.js`, import the arrow component, like we do it with the flag component, and register it over here like this.

index.js

```
import React from "react"
import { AppRegistry, asset, StyleSheet, View,
Image } from "react-360"
import Flag from "./components/Flag"
import Earth from "./components/Earth"

export default class travelVR extends
React.Component {
  render() {
    const { flagContainer } = styles

    return <View style={mainView} />
  }
}

const styles = StyleSheet.create({
  flagContainer: {
    height: 600,
    width: 4680,
    backgroundColor: "rgba(255, 255, 255, 0.3)",
    flexDirection: "row",
    alignItems: "center",
    justifyContent: "center"
  }
})

AppRegistry.registerComponent("travelVR", () =>
travelVR)
AppRegistry.registerComponent("Flag", () =>
Flag)
AppRegistry.registerComponent("Earth", () =>
Earth)
```

[01:24] Right now, we need to jump into `client.js` and render this component somehow. We cannot use a surface, because surfaces are for displaying React components onto flat planes, either on a cylinder or on a flat surface.

[01:36] With 3D models, we have to use something different. We have to use a location. First up, we need to import location from React 360 web, and we're going to create a new location. I'm going to call it `myNewLocation`, and it's going to be a `new Location`.

[01:51] Here, we have to set in an area specifying the position of the location within the x, y, and z-axis. I'm going to put it as `three` meters to the right, `zero` meters in the y-axis, and minus `one` meter in the z-axis. This model is going to be displayed `three` meters to the right and `one` meter in front of me.

[02:08] We're going to render the arrow component to this new location. We're going to do `r360.RenderToLocation`. Then we're going to create a new root, with the `Earth` component. I'm not going to set any props, and I'm going to use `myNewLocation`, like this.

client.js

```
const myNewLocation = new Location([3, 0, -1])
r360.renderToLocation(r360.createRoot("Earth"),
myNewLocation)
```

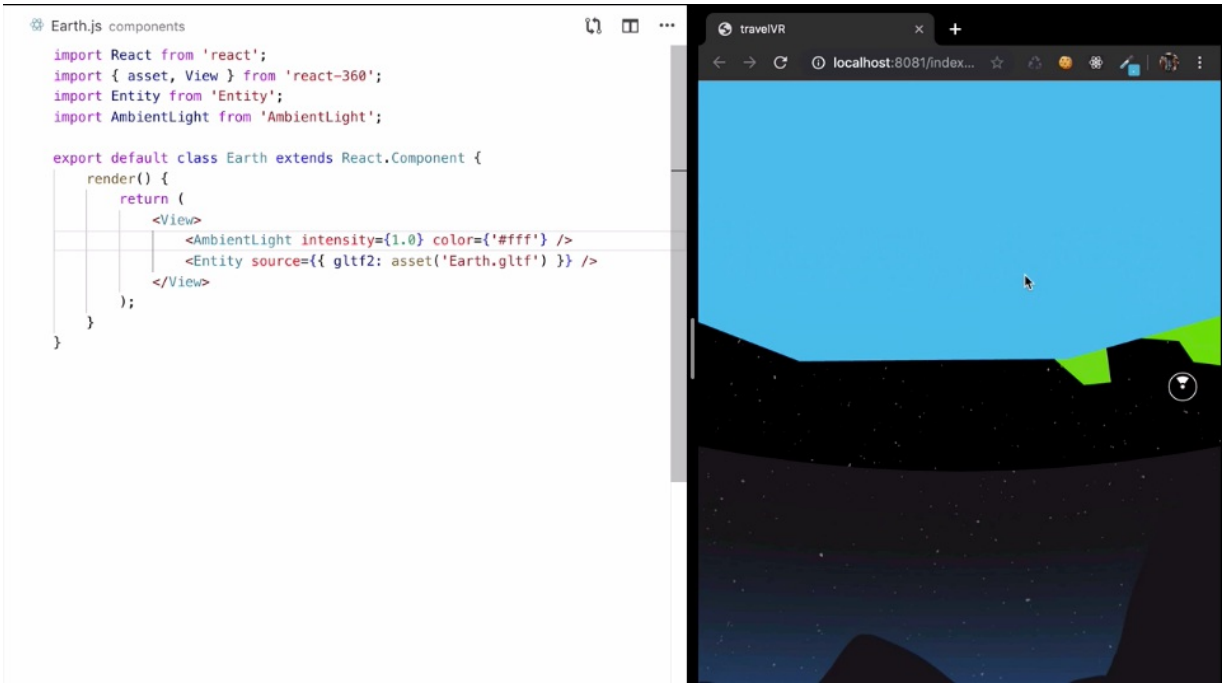
[02:24] I'm going to comment out this flag comment, because we are not going to need that. After we save and refresh that, we are not going to be able to see the arrow. We need to shed some light on this situation. There are a couple of different types of lighting we can use in React 360 applications.

[02:38] In this case, I'm going to use an ambient light. We're going to import it from `AmbientLight`, and I'm going to set the `intensity` of the light to `1.0` We're going to set the `color` of the light to `white`. After I save and refresh that, we can see our problem.

Earth.js

```
import React from "react"
import { AppRegistry, asset, View } from "react-
360"
import Entity from "Entity"
import AmbientLight from "AmbientLight"

export default class Earth extends
React.Component {
  render() {
    return (
      <View>
        <AmbientLight intensity={1.0} color=
{'#fff'} />
        <Entity source={{ gltf2:
asset('Earth.gltf') }} />
      </View>
```

See 2:48 in lesson

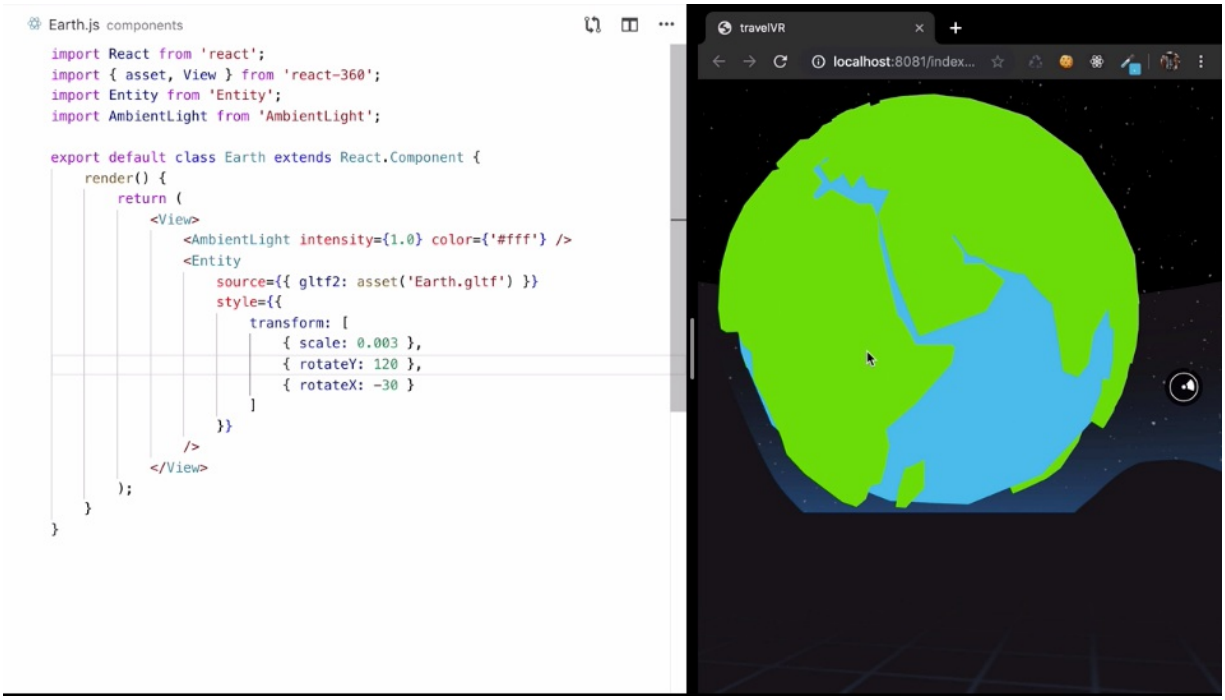


[02:50] We were displaying the earth, but it's absolutely massive. We need to scale it down a bit. To change that, we need to use this transform property. I'm going to specify a style. Inside of the style, we're going to have a `transform`.

[03:00] `Transform` property takes an array. We're going to have `scale` of `0.03`, because our arrow is really huge. We're going to rotate it in the y direction by `120` degrees, and we're going to rotate it in the x direction by minus `-30` degrees.

```
export default class Earth extends
React.Component {
  render() {
    return (
      <View>
        <AmbientLight intensity={1.0} color=
{'#fff'} />
        <Entity
        source={{ gltf2: asset('Earth.gltf') }}
        style={{
          transform: [
            { scale: 0.003 },
            { rotateY: 120 },
            { roateX: -30 }
          ]
          }}
        />
      </View>
```

[03:15] After I save and refresh that, we have the desired effect.



We have this 3D model of the arrow displayed a couple of meters to our right. The thing is, is that this model is flat right now. The reason it happens is that right now, we are using this `AmbientLight`.

[03:28] This ambient light has the same intensity in all kinds of direction. In order to have shadows displayed on this arrow model, we need to use some figures. First, `import PointLight` from `PointLight`.

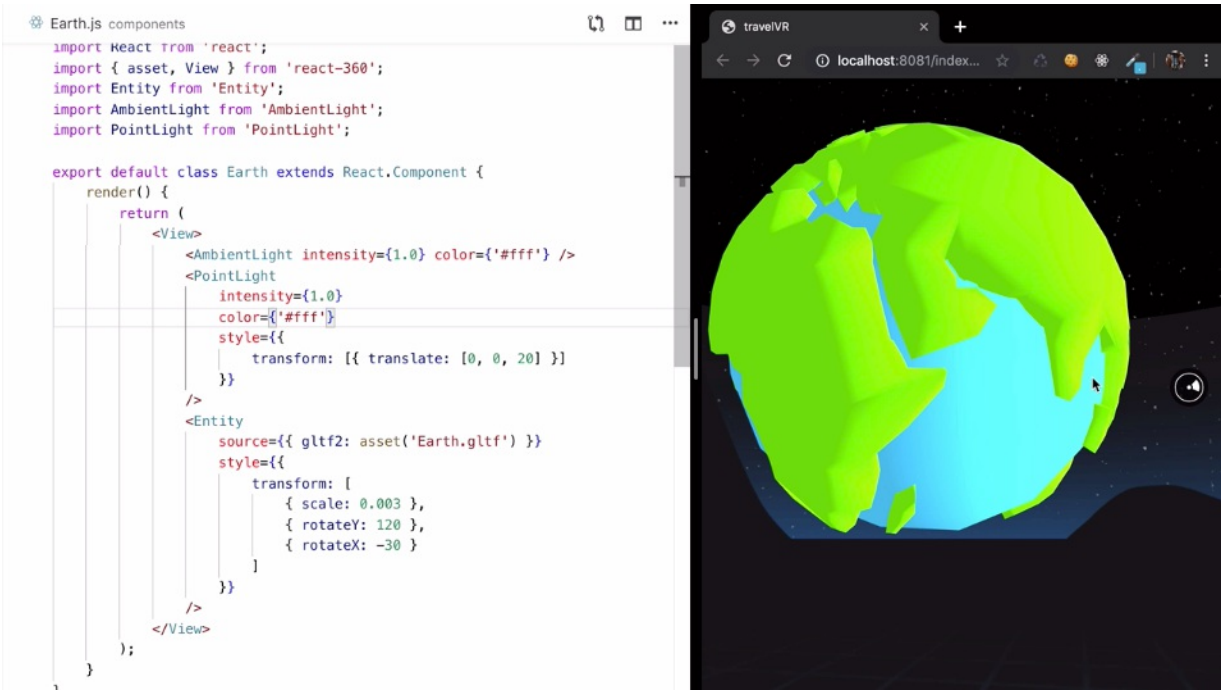
```
import React from "react"
import { AppRegistry, asset, View } from "react-
360"
import Entity from "Entity"
import AmbientLight from "AmbientLight"
import PointLight from "PointLight"
```

I'm going to copy this ambient light, because point light is similar to ambient light.

[03:43] It also has an intensity, also has a color, but I'm going to transform it. I'm going to use `style`, and I'm going to supply a `transform` property. It's going to `translate`, as in move, our light source to `0,0,20`. It's going to appear 20 meters behind our back.

```
<View>
  <AmbientLight intensity={1.0} color={"#fff"}
/>
  <PointLight
    intensity={1.0}
    color={"#fff"}
    style={{
      transform: [{ translate: [0, 0, 20] }]
    }}
  />
  <Entity
    source={{ gltf2: asset("Earth.gltf") }}
    style={{
      transform: [{ scale: 0.003 }, { rotateY:
120 }, { roateX: -30 }]
    }}
  />
</View>
```

[03:59] After I save and refresh that, we're going to see that our model looks much better, much more alive, because we are using the point size, so we can see the shadows on this model.



Capture user interaction in React 360 with VrButton component

Instructor: [0:00] We have an updated version of our app, so we have a single view component. Inside of this view component, we are entering some flags, which are displayed over here over, it's in the service. So far, our app has been entirely static, as in I was not able to actually do anything with any of those components.

[0:15] It's time we changed that and add some interactivity to our React 360 application. To do that, first `import VrButton` from React 360.

index.js

```
import React from "react"
import {
  AppRegistry,
  asset,
  StyleSheet,
  View,
  Image,
  VrButton
} from "react-360"
import Flag from "./components/Flag"
import Earth from "./components/Earth"
```

Next, wrap this flag component inside of a `VrButton` like this. Let me move the key to `VrButton` component, and then we're going to attach some events to this `VrButton`.

```
export default class travelVR extends
React.Component {
  renderFlags() {
    return FLAGS_IMAGES.map(image => (
      <VrButton key={image}>
        <Flag image={image} />
      </VrButton>
    ))
```

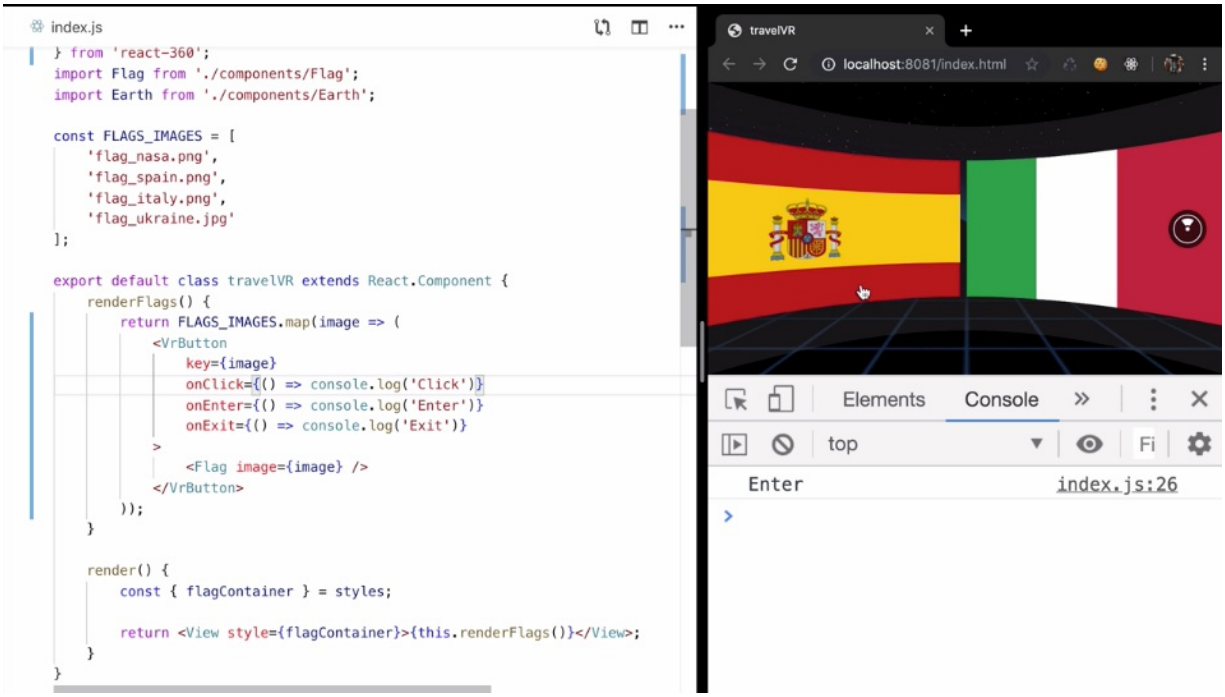
[0:33] First, we can add an `onClick`. It's going to take a function, and it's going to `console.log Click`. Let me copy that, because we are also going to add an `onEnter`, as well as `onExit`, like this.

```
export default class travelVR extends
React.Component {
  renderFlags() {
    return FLAGS_IMAGES.map(image => (
      <VrButton
        key={image}>
        onClick={() => console.log('Click')}
        onEnter={() => console.log('Enter')}
        onExit={() => console.log('Exit')}
      >
        <Flag image={image} />
      </VrButton>
    ))
```

After I save and I refresh that, we can open dev tools to check the result.

[0:51] If I hover my mouse over the flag of Spain, I'm going to get "enter" in the console.

See 0:55 in lesson



If I move my mouse away from the flag of Spain, I'm going to get an exit. If I click on it, I'm going to get the click event triggered.

[1:03] Right now, we have our events connected to this `VrButton`. Important thing to note to this when it comes to `VrButton` is that `onEnter` and `onExit` behave differently based on the device. If I were to run this app inside of a VR headset, if I were to literally look at the flag of Spain, then I would get the `onEnter` handle triggered.

[1:20] If I looked away from the flag, then I would get the `onExit`. We would like to add some state to our app. Basically, if I hover over some of those flags, I would like to make it more visible to indicate that it's currently active.

[1:31] To do that, jump to `Flag.js` component. Over here, by default, we're going to make the opposite of the flag to `0.7`.

Flag.js

```
const styles = StyleSheet.create({
  flag: {
    height: 400,
    width: 600,
    marginRight: 20,
    opacity: 0.7
  },
```

By default, all our flags are inactive. Next, we're going to add `active`, and I'm going to make the opacity `one`.

[1:44] Let me destructure that from styles, and we are going to also destructure both `image` and `activeFlag` from these props. Next, we need to modify this type prop. I'm going to pass in the `flag` prop, and whenever the `activeFlag` is equal to current image, we're going to apply the `active` styles as well.

```
export default class Flag extends
React.Component {
  render() {
    const { flag, active } = styles
    const { image, activeFlag } = this.props

    return (
      <Image
        style={[flag, activeFlag === image &&
active]}
        source={asset(this.props.image)}
      />
    )
  }
}
```

[2:02] Let me save that, and let's go back to the `index.js`. Let's add some state. By default, we're going to have a `state` that `activeFlag` is an empty string. We're going to change this `onEnter` handler. Instead of `console.log`, I'm going to do `this.setState`, and we're going to set the `activeFlag` to current `image`.

index.js

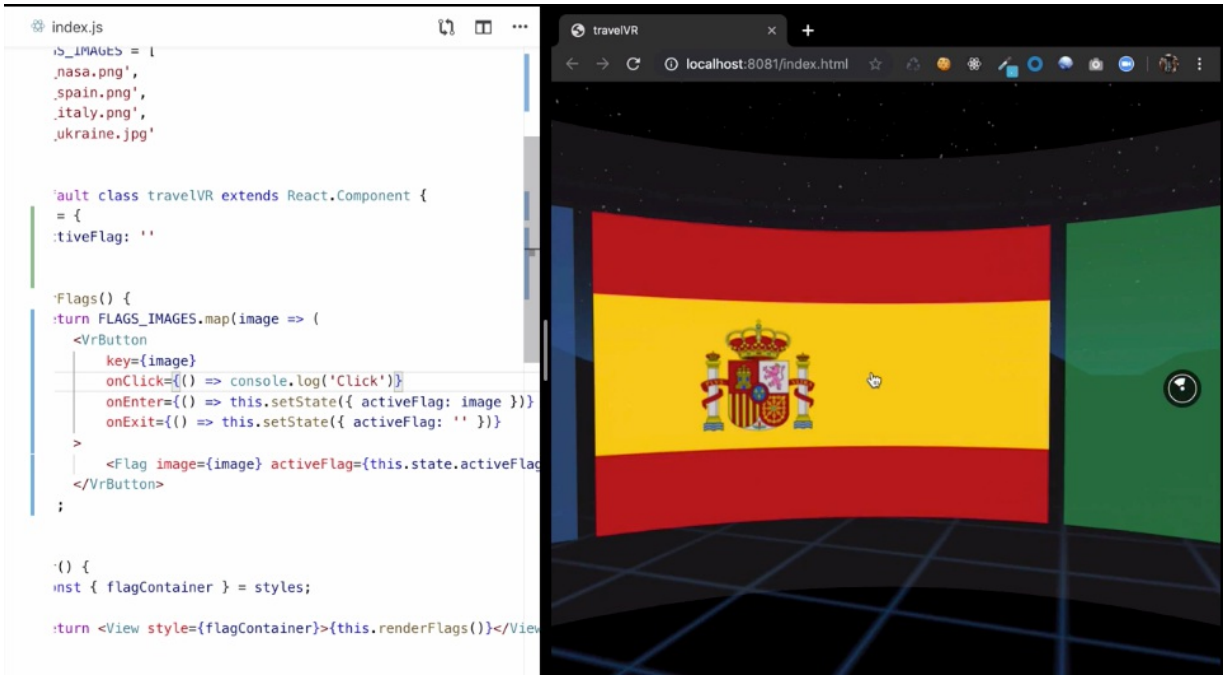
```
renderFlags() {
  return FLAGS_IMAGES.map(image => (
    <VrButton
      key={image}
      onClick={() => console.log('Click')}
      onEnter={() => this.setState({ activeFlag:
image})}
      onExit={() => console.log({'Exit')}
    >
      <Flag image={image}  />
    </VrButton>
  ));
}
```

[2:20] I'm going to do something similar to the `onExit` handler, but I'm going to reset the `activeFlag` to an empty string. We need to pass in this active flag state to flag component as well. I'm going to do this state, `activeFlag`.

```
renderFlags() {
  return FLAGS_IMAGES.map(image => (
    <VrButton
      key={image}
      onClick={() => console.log('Click')}
      onEnter={() => this.setState({ activeFlag:
image})}
      onExit={() => this.setState({ activeFlag:
'' })}
    >
      <Flag image={image} activeFlag=
{this.state.activeFlag}/>
    </VrButton>
  ));
}
```

[2:35] After I save and refresh that, we have the desired effect. Right now, if I hover over any of those flags, it's clear which one is currently being active.

See 2:35 in lesson



Change 360 panorama background in React 360 app

Instructor: [00:00] There are a couple of places in which we can change the background that gets displayed around the user. The first place is inside of the current JS. Here, we can specify the default background that will get displayed.

[00:09] Right now, it's 360 work. If I were to change it to `spain.jpeg`, I am going to change the background to this picture of Spain that I took myself a couple of years ago.

client.js

```
// Load the initial environment
r360.compositor.setBackground(r360.getAssetURL('
spain.jpg'))
}

window.React360 = { init }
```

See 0:15 in lesson

[00:18] This picture is inside of the `static_assets` folder. Apart from changing the default background, we can also change the background dynamic inside of our React application.

[00:27] Let me go ahead and revert this change, and we're going to jump into `index.js`. Over here, we have the updated places. Each place has a flag and also a panorama associated with it. First, Spain has a flag of Spain and also this panorama that we can see over here.

[00:41] Here in the flag methods, we are mapping over all those places and displaying a flag for each country. Our goal would be whenever a user is going to click on one of those flags, we would like to travel to this place.

[00:52] To achieve this effect, first `import environment` from `react-360`.

index.js

```
import React, { Fragment } from "react"
import {
  AppRegistry,
  asset,
  StyleSheet,
  View,
  Image,
  Environment,
  VrButton
} from "react-360"
import Flag from "./components/Flag"
import Earth from "./components/Earth"
```

Next, we're going to create a `changeBackground` method. It's going to take the `panorama` as argument. Here, we're going to [inaudible] `Environment.setBackgroundImage`. I am going to use the `asset` function to get the image associated with this `panorama`.

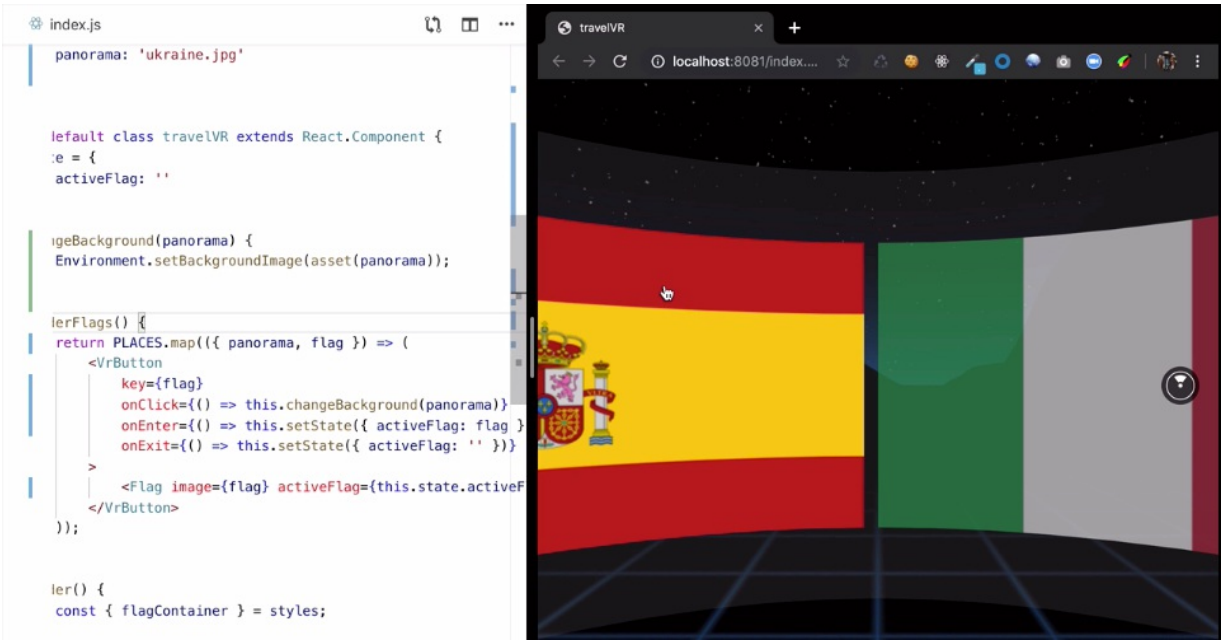
```
export default class travelVR extends
React.Component {
  state = {
    activeFlag: ''
  };

  changeBackground(panorama) {

Environment.setBackgroundImage(asset(panorama));
  }
```

[01:10] Over here on this `onClick` handler, I am going to call this function `changeBackground` and I am going to pass in the `panorama` of this place.

```
renderFlags() {
  return PLACES.map(({panorama, flag }) => (
    <VrButton
      onClick={() =>
this.changeBackground(panorama)}
      onEnter={() => this.setState({ activeFlag:
flag })}
      onExit={() => this.setState({ activeFlag:
'' })}
    >
```

Now, we can use our app to travel. If I were to click on the flag of Spain, I am going to go to Spain. If I click on the flag of Pakistan, I am going to go to this place.

See 1:20 in lesson



Use Prefetch component to fetch images before they are needed in React 360

Instructor: [0:00] We have a bit of a performance problem in our app because whenever we load the app we're going to get this 360 world image, which is around a megabyte. We're going to get all those images for their flags. If I were to click on the flag of Spain, only then I'm going to send the request to fetch this 360 panorama, which is around 2.3 megabytes. This can be tricky, especially on mobile.

[0:21] Ideally, what we would like to do is to be able to prefetch all those images in the background. Whenever I click on a flag, we are not going to send a request because this image is already going to be loaded.

[0:31] In order to do that, first `import Prefetch` from `react-360`. We're going to `import Fragment` from React as well.

index.js

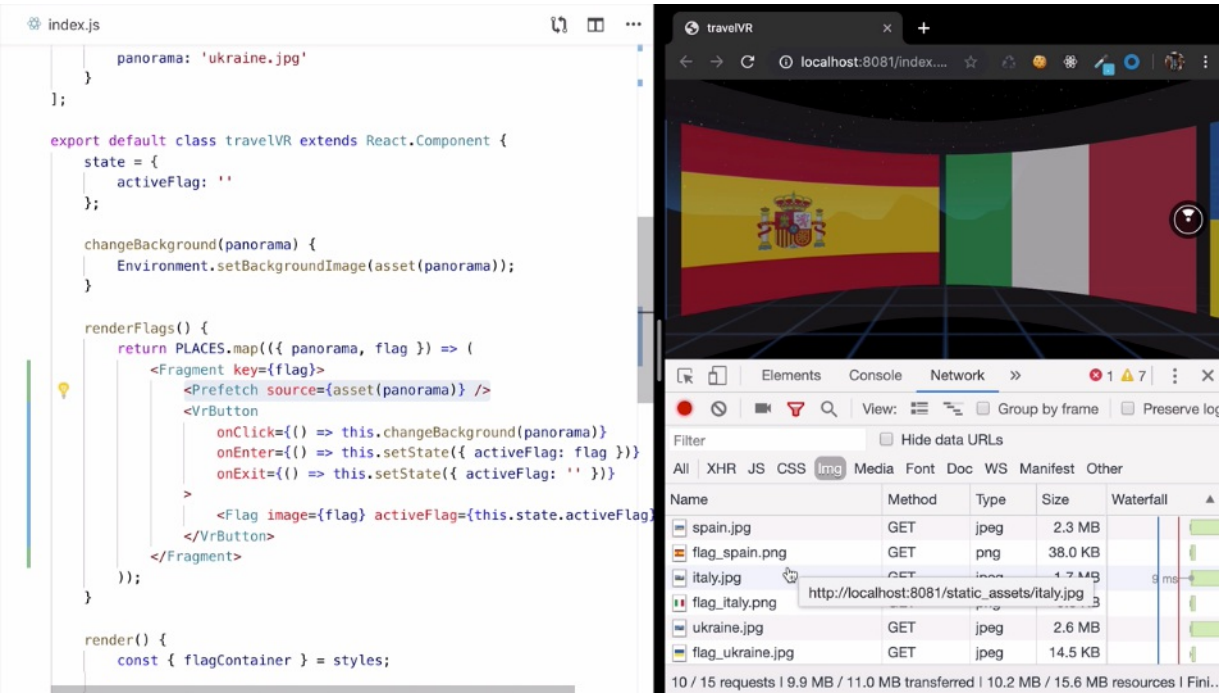
```
import React, { Fragment } from "react"
import {
  AppRegistry,
  asset,
  StyleSheet,
  View,
  Prefetch,
  Image,
  Environment,
  VrButton
} from "react-360"
import Flag from "./components/Flag"
import Earth from "./components/Earth"
```

Next, in the `renderFlags` method, we're going to wrap this `VrButton` inside of a `Fragment`. We're going to copy this key to the `Fragment` as well. I'm going to render the `Prefetch` component.

[0:49] `Prefetch` component takes a source, so I'm going to set the `source` to be `asset panorama`. When all those places are being rendered, I'm going to also prefetch the appropriate panorama per country. Right now, we have the desired effect.

```
renderFlags() {
  return PLACES.map(({ panorama, flag }) => (
    <Fragment key={flag}>
      <Prefetch source={asset(panorama)} />
      <VrButton
        onEnter={() => this.setState({
activeFlag: flag })}
        onExit={() => this.setState({
activeFlag: '' })}
        onClick={() =>
this.changeBackground(panorama)}
      >
        <Flag image={flag} activeFlag=
{this.state.activeFlag} />
      </VrButton>
    </Fragment>
  ))
}
```

If I save and refresh that, we're going to see all those background images fetched in the background.



[1:09] Right now, we won't have to send an additional request whenever a user is going to click on a flag. Those images are going to be preloaded. Our app will have a better perceived performance.

[1:18] With that being said, sometimes we might not want to prefetch the images in the background because user might not decide to visit all the countries. It's up to you as a developer to decide whether you want to prefetch images or videos or not.

Add animations to React 360 components

Instructor: [0:00] We have an updated version of our app with the app component over here. It would be great to be able to animate that. We would like to make the Earth spin and bounce up and down. In order to do that, `import animated` from `react-360`.

Earth.js

```
import React from "react"
import { Animated, asset, View } from "react-
360"
import Entity from "Entity"
import AmbientLight from "AmbientLight"
import PointLight from "PointLight"
```

[0:13] Next, we're going to create a new `AnimatedEntity` component. Basically, we're going to wrap the entity component inside of an animated library imported from React 360.

```
import React from "react"
import { Animated, asset, View } from "react-
360"
import Entity from "Entity"
import AmbientLight from "AmbientLight"
import PointLight from "PointLight"

const AnimatedEntity =
Animated.createAnimatedComponent(Entity)
```

We're going to replace the entity with the animated entity.

```
<AnimatedEntity
  source={{ gltf2: asset("Earth.gltf") }}
  style={{
    transform: [
      { translateY: this.jumpValue },
      { scale: 0.001 },
      { rotateY: this.rotation }
    ]
  }}
/>
```

[0:27] Next up, we're going to create a new animated value. I'm going to create a rotation value, and it's going to be a `new Animated.Value`. By default, I'm going to set it to `zero`.

```
export default class Earth extends
React.Component {
  rotation = new Animated.Value(0);
```

Next up, we're going to create a `spin` function.

[0:38] Inside of this function, first, we're going to reset the rotation value to `zero`. Next, we're going to use the `Animated.timing` function to modify this rotation value from to 360. We're going to do `this.rotation`, and we're going to provide an options object.

[0:55] First, we would like to set the rotation value `toValue 360` over the course of `four` seconds. We're going to set the start function. Start function takes a callback, so what should happen after the rotation is completed.

```
spin() {
  this.rotation.setValue(0);
  Animated.timing(this.rotation, {
    toValue: 360,
    duration: 4000,
  }).start(() =>

  )
}
```

[1:08] We should start the rotation again, because we would like to make an infinite animation. Next, I'm going to create a `componentDidMount` function, and I'm going to run this spin.

```
spin() {
  this.rotation.setValue(0);
  Animated.timing(this.rotation, {
    toValue: 360,
    duration: 4000,
  }).start(() => this.spin())
}

componentDidMount() {
  this.spin()
```

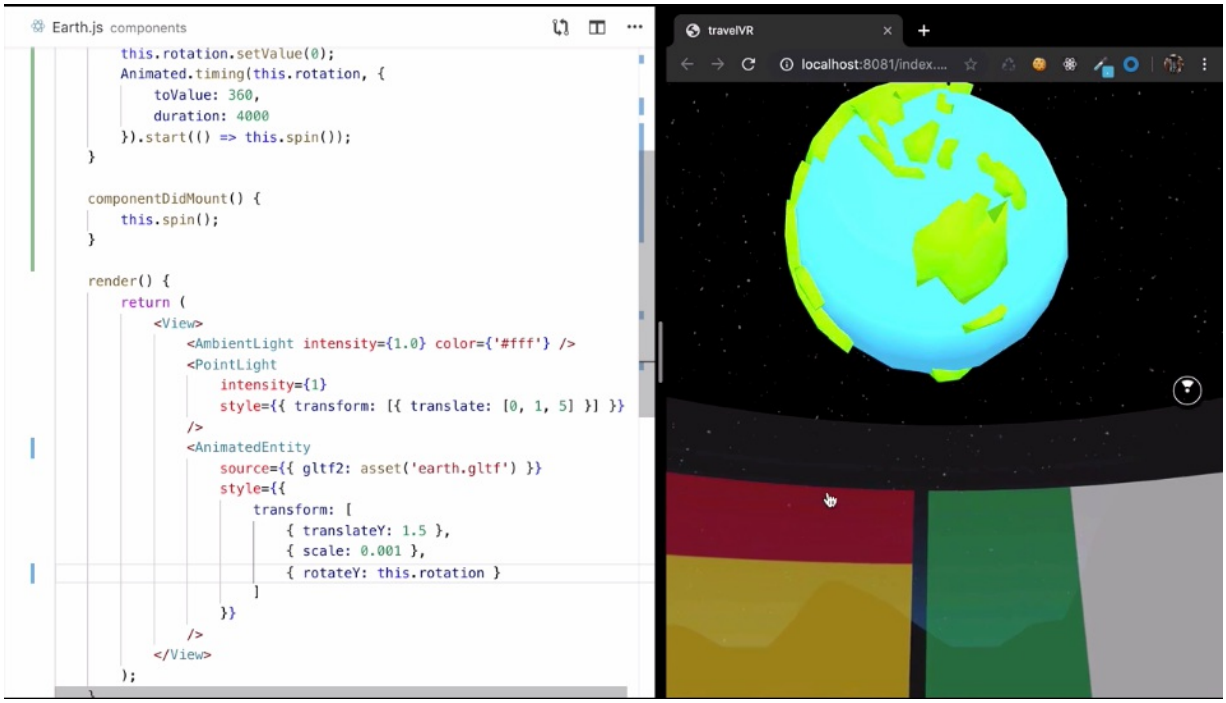
In order to use the spin value, we're going to use it here in the rotate Y transform value.

[1:23] I'm going to do `this.rotation`.

```
<AnimatedEntity
  source={{ gltf2: asset("Earth.gltf") }}
  style={{
    transform: [
      { translateY: this.jumpValue },
      { scale: 0.001 },
      { rotateY: this.rotation }
    ]
  }}
/>
```

After I save and refresh that, we're going to see the updated results. Right now, we have this spinning model of the Earth.

See 1:29 in lesson



The problem is that you will notice that this animation is going to slow down at the end.

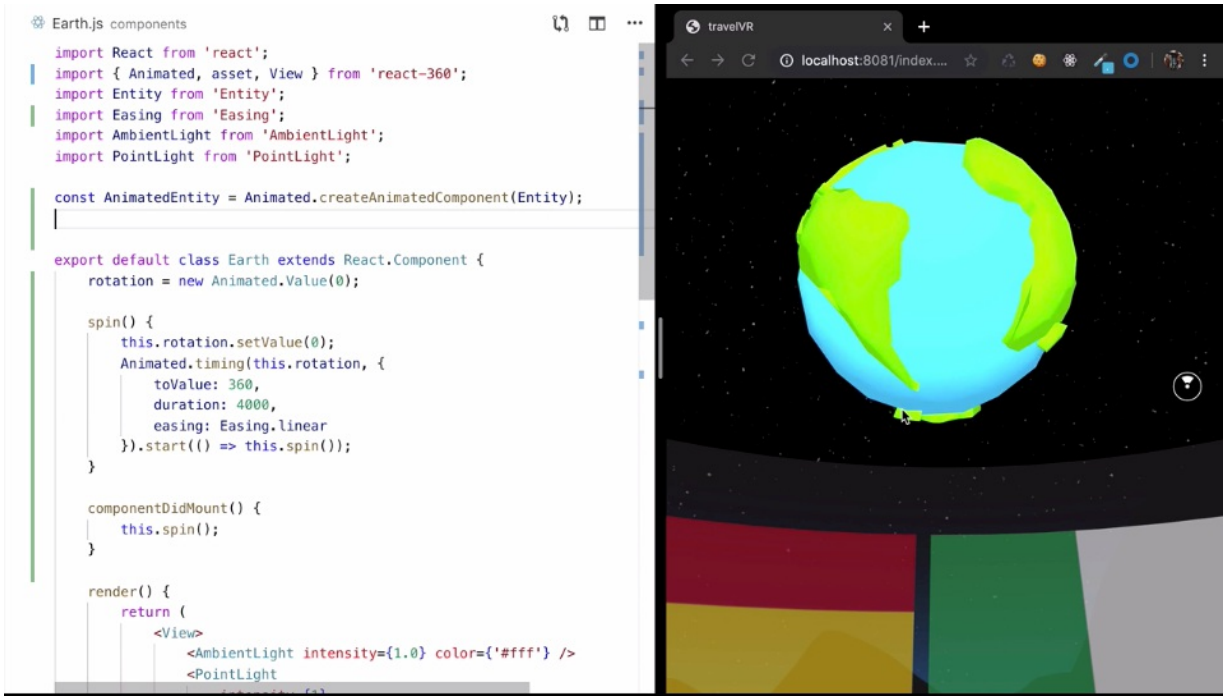
[1:37] The reason this happens is that by default, the animated timing function is using an in and out easing function. Basically, what that means is that both the beginning and the end of the animation is a bit slower.

[1:49] In order to fix that, we're going to `import Easing` from `Easing`. I'm going to set the `easing` to `Easing.linear`.

```
import React from "react"
import { Animated, asset, View } from "react-
360"
import Entity from "Entity"
import Easing from "Easing"
import AmbientLight from "AmbientLight"
import PointLight from "PointLight"

const AnimatedEntity =
Animated.createAnimatedComponent(Entity);

export default class Earth extends
React.Component {
  rotation = new Animated.Value(0)

  spin() {
  this.rotation.setValue(0);
  Animated.timing(this.rotation, {
    toValue: 360,
    duration: 4000,
    easing: Easing.linear
  }).start(() => this.spin())
}

componentDidMount() {
  this.spin()
```

If I save and refresh that, we're also going to have a spinning animation, but the speed of the animation is consistent, so it doesn't slow down.

See 2:01 in lesson



[2:06] Next, we're going to make our Earth model jump up and down. First, I'm going to create two variables for low and `TOP_Jump_Value`. I'm going to create a new `jumpValue`, and it's going to be a `new Animate.Value`.

[2:18] The default value is going to be `LOW_JUMP_VALUE`.

```
const AnimatedEntity =
Animated.createAnimatedComponent(Entity)
const LOW_JUMP_VALUE = 1.5
const TOP_JUMP_VALUE = 1.75

export default class Earth extends
React.Component {
  rotation = new Animated.Value(0)
  jumpValue = new Animated.Value(LOW_JUMP_VALUE)

  spin() {
    this.rotation.setValue(0)
    Animated.timing(this.rotation, {
      toValue: 360,
      duration: 4000,
      easing: Easing.linear
    }).start(() => this.spin())
    }
```

Next, I'm going to create a `jump` function. It's going to take the current value as an argument, and we're going to create a new variable for the next value. `nextValue` is going to be whatever the `currentValue` is equal to, `TOP_JUMP_VALUE`.

[2:38] We're going to replace that with the `LOW_JUMP_VALUE`. In the other case, we're going to use the `TOP_JUMP_VALUE`.

```
jump(currentvalue) {
  let nextValue = currentValue ===
TOP_JUMP_VALUE ? LOW_JUMP_VALUE : TOP_JUMP_VALUE
```

We're going to use the `Animated.timing` function to modify `this.jumpValue`. We're going to provide options.

[2:53] We would like to modify this jump value `toValue` of `nextValue` over the course of a half a second. Here, we're going to `.start`, provide a callback, in which we're going to call the `jump` function with the `nextValue`.

```
jump(currentvalue) {
  let nextValue = currentValue ===
TOP_JUMP_VALUE ? LOW_JUMP_VALUE : TOP_JUMP_VALUE

  Animated.timing(this.jumpValue, {
    toValue: nextValue,
    duration: 500
  }).start(() => this.jump(nextValue))
}
```

[3:07] Then we're going to call it in the `componentDidMount`. I'm going to do `this.jump`, and I'm going to provide `LOW_JUMP_VALUE` argument by default.

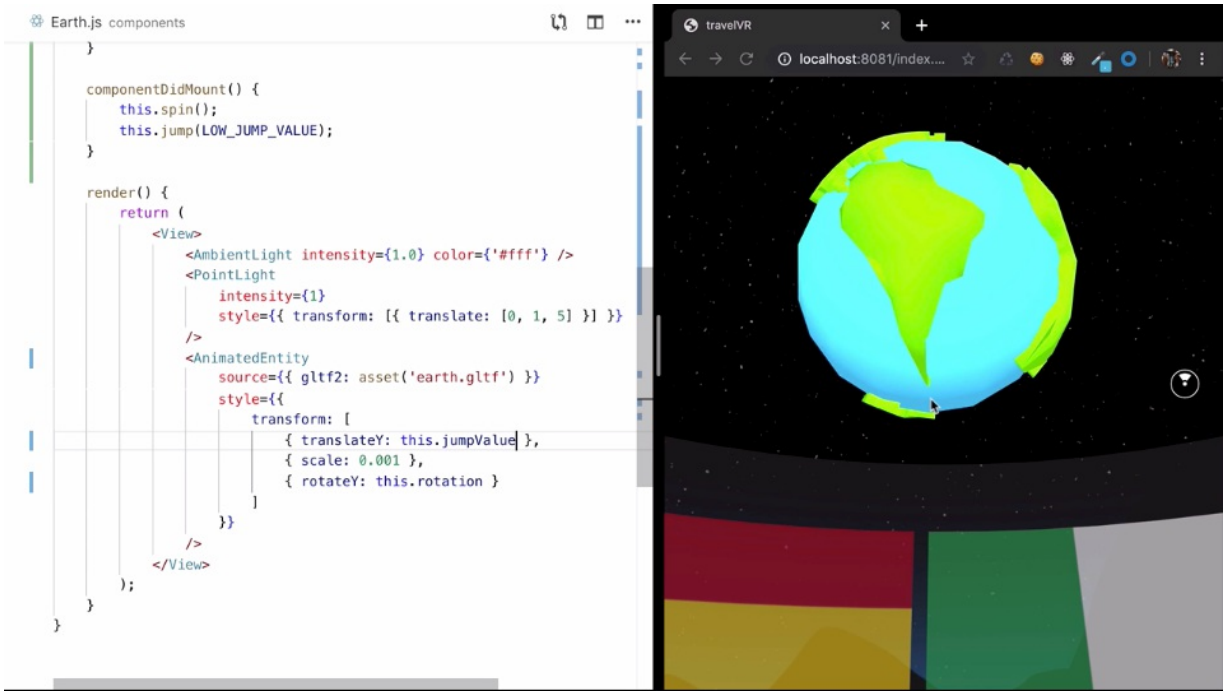
```
componentDidMount() {
  this.spin()
  this.jump(LOW_JUMP_VALUE)
}
```

I'm going to use this value here in the translate Y. I'm going to replace that 1.5 with `this.jumpValue`.

```
<AnimatedEntity
  source={{ gltf2: asset('Earth.gltf') }}
  style={{
    transform: [
      { translateY: this.jumpValue },
      { scale: 0.001 },
      { rotateY: this.rotation }
      ]
```

[3:22] After I save and refresh that, we can see both infinite animations running at the same time. The arrow is both spinning and jumping.

See 3:22 in lesson



Create Native Modules to extend React 360 app functionality

Instructor: [00:00] You would like to have a feature that whenever I travel to a different country, I would like to change the title of the place to say, "Welcome to Spain," "Welcome to Italy," and so on.

[00:08] Over here, we have the updated places array and each object inside of this array has a name, flag, and a panorama property. We're getting those over here in the flag method.

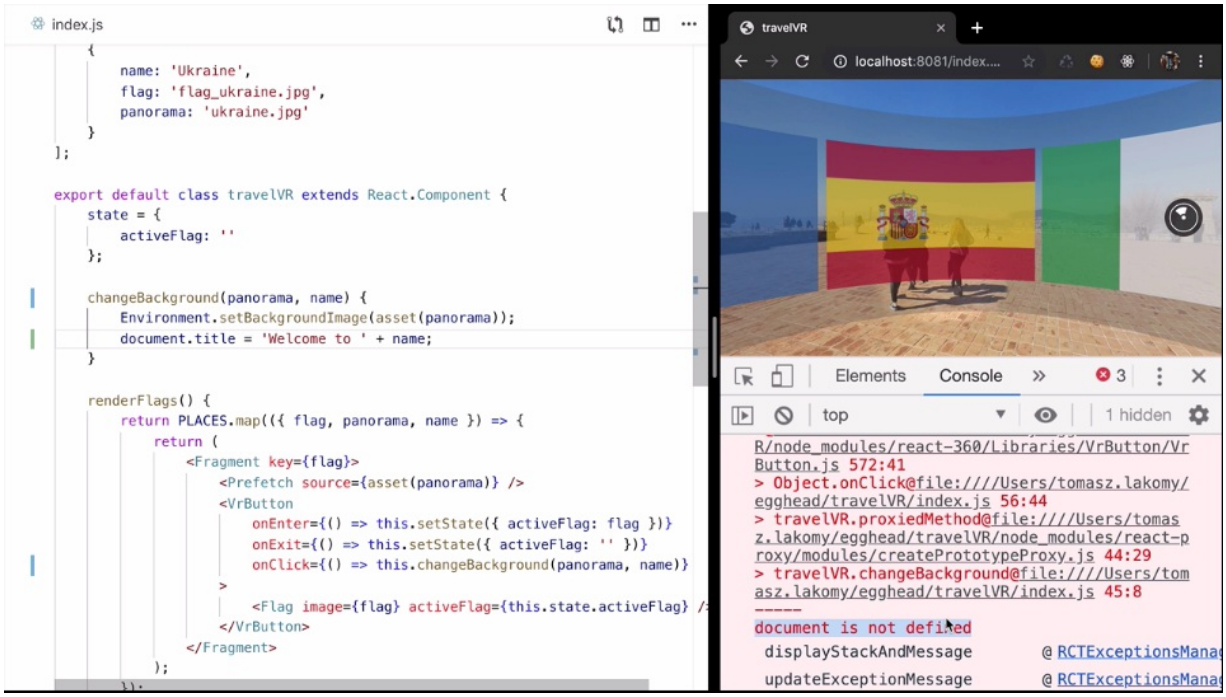
[00:18] I am going to pass in the name to `changeBackground` method, get it over here as well. In order to change the title of the page, I am going to do `document.title` equals `Welcome to + name`.

index.js

```
changeBackground(panorama, name) {

Environment.setBackgroundImage(asset(panorama))
  document.title = 'Welcome to ' + name
}
```

After I save, in their first [inaudible] , we're going to have a problem.

[00:34] The problem is that as soon as I click on the flag, we're going to get a message, "Document is not defined."



It is not defined because by default, the React 360 runs inside of web worker, and web workers do not have access to the DOM.

[00:46] In order to enhance our React 360 application with something that requires the DOM, we need to implement our value on native module. In order to do that, first, go to `client.js`.

[00:55] Next, `import module` from `react-360-web`. Next, we need to create a new `class`. I am going to call it a `TitleChanger`. The title changer needs to extent the `Module` that we'll import it from React 360.

client.js

```
import { ReactInstance, Location, Surface,
Module } from "react-360-web"

class TitleChanger extends Module {}
```

[01:06] We need to specify constructor. This `constructor` is basically going to around the super method with the name of the class. I am going to do `super TitleChanger`. Whatever is specified over here is going to be exposed to React 360 application.

```
import { ReactInstance, Location, Surface,
Module } from "react-360-web"

class TitleChanger extends Module {
  constructor() {
    super('TitleChanger')
  }
```

[01:18] I am going to create a new method called `changeTitle`. It's going to take `title` and argument. I am going to do `document.title = title`.

```
changeTitle(title) {
  document.title = 'Welcome to ' + title
}
```

[01:26] Next, we need to expose its native module to our React code. In order to do that, whenever we're creating a new React 360 instance, we can also pass in some custom options. Here, what we need to do is to pass in `nativeModules`. It takes an array. We're going to create a `newTitleChanger` instance over here.

```
function init(bundle, parent, options = {}) {
  const r360 = new ReactInstance(bundle, parent,
{
    // Add custom options here
    fullScreen: true,
    nativeModules: [new TitleChanger()],
    ...options
  })
```

[01:43] I'll save that and go to `index.js`. Over here, import `NativeModules` from `react-360`.

index.js

```
import React, { Fragment } from "react"
import {
  AppRegistry,
  asset,
  StyleSheet,
  Environment,
  Prefetch,
  View,
  Image,
  NativeModules,
  VrButton
} from "react-360"
import Flag from "./components/Flag"
import Earth from "./components/Earth"
```

Next, we need to get our title changer module from all native modules. I am going to do `TitleChanger` equals `NativeModules` to get the title changer module.

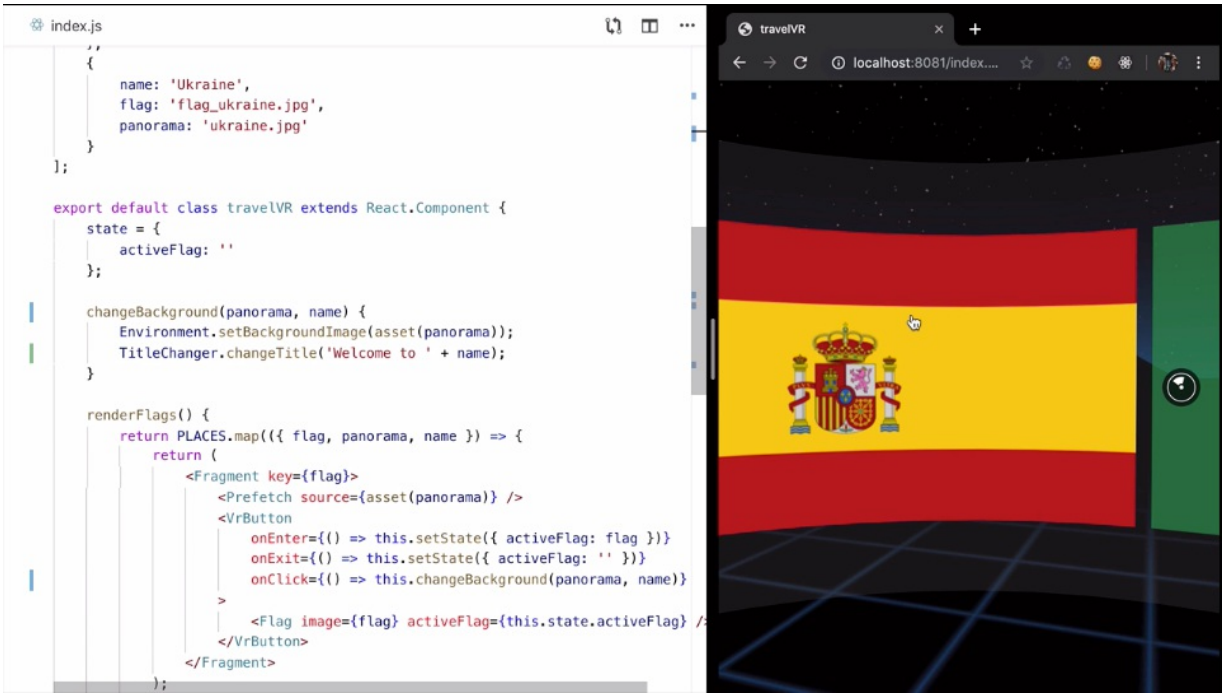
```
const { TitleChanger } = NativeModules
```

[01:59] Next, we need to remove this document title caller, and we're going to do `TitleChanger`, `changeTitle`, and then I am going to set the title to `Welcome to ' + name`. Let's see if it works.

```
changeBackground(panorama, name) {

Environment.setBackgroundImage(asset(panorama))
  TitleChanger.changeTitle(Welcome to ' + name)
}
```

[02:10] I am going to click on the flag of Spain. We see the title change to "Welcome to Spain." If I click on the flag of NASA, I am going to see the title change, "Welcome to space."

See 2:10 in lesson



[02:18] There're a plenty of native modules already implemented for us. If I decide to log all of those, we're going to see our title changer module as well as useful stuff such as networking, location, or history.

Build a React 360 app for production

Instructor: [00:00] We have finished our React physics application. We would like to build it and enter this into production. In order to do that, I add `npm` on the bundle inside of your product directory. This is going to take a while.

```
npm run bundle
```

[00:10] Once this is done, we're going to get a new build folder inside of our product directory. If you have static assets inside of our application, and we do, we need to copy `static_assets` to the build directory.

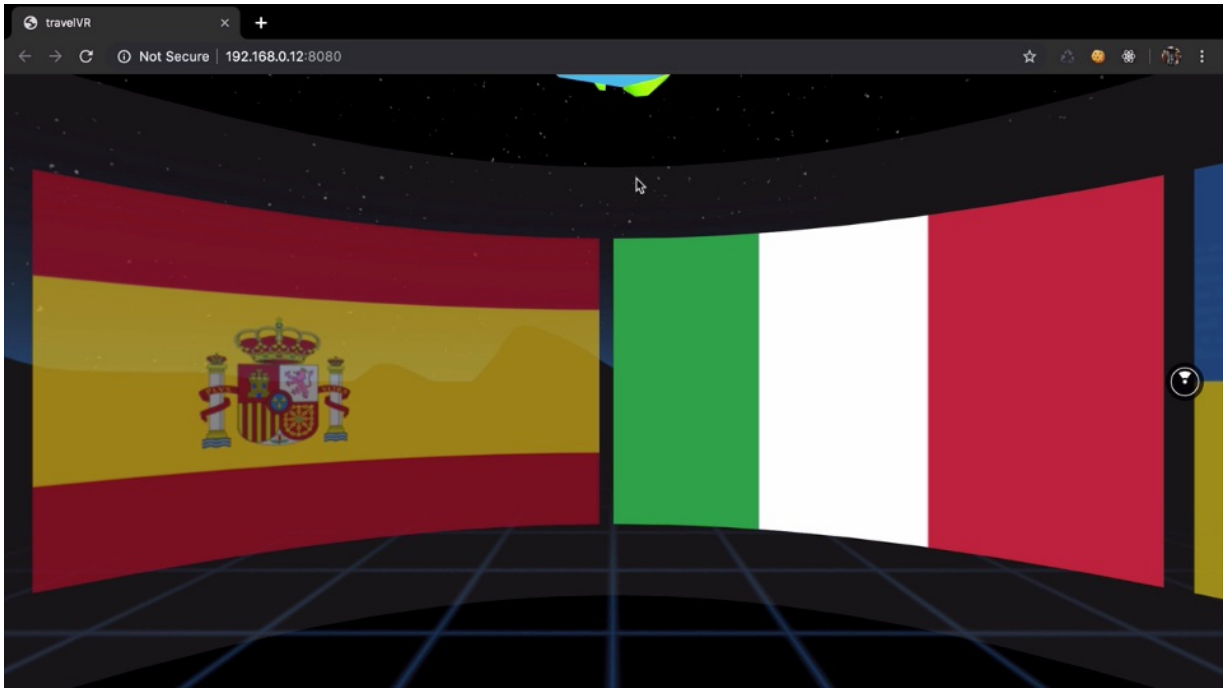
```
cp -rf static_assets build
```

[00:22] Once this is done, we can run our production build. I'm going to create a new http server inside of the build directory.

```
http-serve build
```

I'm just going to copy this URL to the browser. We can see the result over here. We have a production version of our application.

See 0:30 in the lesson



[00:34] If you take a look inside of the console, we're going to see that it is a production build because development of the warning are off, and performance optimizations are on. We can safely go ahead and deploy this build to a production environment.