Project-06-Score-keeper-app

SCSS and Atmosphere Packages

- first we are going to be downloading atmospherejs.com to get the scss
- this packages are made for Meteor
- go to atmospherejs.com and search for scss
- we are going to be using fourseven:scss
- go to the terminal and type in a separate tab

```
meteor --help
```

- notice the add command,that will add a package to this project
- we also have remove which removes the package and list, which lists all of the packages you are currently using
- check out list

```
meteor list --release 1.4.2.1
```

- you should be able to see at the bottom how to add another package
- dont for get to add the equal sign = after the @ sign

```
meteor add fourseven:scss@=3.10.1 --release 1.4.2.1
```

- for demo lets just remove this package

```
meteor remove fourseven:scss --release 1.4.2.1
```

- now that we know how to remove it lets add it again

```
meteor add fourseven:scss@=3.10.1 --release 1.4.2.1
```

- next go to client main.css and change the name to main.scss
- now we are going to test it by changing all the text in the app to red
- go to client/main.scss
- we select all of the document by using the *

```
$red : red;


* {
```

```
    color: $red;
}
```

CSS Reset And Header Styling

- first delete the above scss demo
- then reset all elements

```
*{
    margin: 0;
    padding: 0;
}
```

- then lets start with global elements

```
/* CSS declarations go here */
*{
    margin: 0;
    padding: 0;
}

body{
    color : #555555 ;
    font-family: Helvetica, Arial, sans-serif ;
}
```

- now lets do the title bar

```
*{
    margin: 0;
    padding: 0;
}

body{
    color : #555555 ;
    font-family: Helvetica, Arial, sans-serif ;
```

```
}


.title-bar{
    background: #e35557;
}
```

- now lets add .title-bar class to TitleBar.js

```
    render(){
      return(
        <div className='title-bar'>
          <h1>{this.props.title}</h1>
          {this.renderSubtitle()}
        </div>
      );
    };
  }
```

- lets head back to main.scss
- let keep styling the .title-bar by changing the color and also by using nesting elements

```
.title-bar{
    background: #e35557;
    color: #ffffff;
    h1{
        font-weight: 300;
        font-size: 2.4rem;
    }
}
```

- since we used relative font size, we have to give our html doc a starting font-size

```
*{
    margin: 0;
    padding: 0;
}
```

```
html{
    font-size: 62.5% ;
}
```

- let's now take h1 tag from .title-bar and put it above it

```
h1{
    font-weight: 300;
    font-size: 2.4rem;
}

.title-bar{
    background: #e35557;
    color: #ffffff;
}
```

- let's create a wrapper class

```
.title-bar{
    background: #e35557;
    color: #ffffff;
}

.wrapper{
    margin: 0 auto;
    max-width: 50rem;
}
```

- now go to TitleBar.js and add the wrapper class

```
      return(
        <div className='title-bar'>
            <div className = 'wrapper'>
                <h1>{this.props.title}</h1>
                {this.renderSubtitle()}
            </div>
        </div>
```

```
      );
    };
  }
```

- go back to your scss file
- and give the wrapper more sytling

```
.wrapper{
    margin: 0 auto;
    max-width: 50rem;
    padding: 1.3rem;
}
```

SCSS Imports

- over in the imports folder create another folder called client, inside client create another folder called styles
- create a file in styles called _main.scss, this is called a partial
- cut out the code from main.scss and paste on to _main.scss
- now we are going to import our partial to our main.scss

```
@import '../imports/client/styles/main';
```

- notice that we didn't add _ or the .scss, it is not needed when importing a partial
- lets create a folder called components in the styles folder, and create a file called _title-bar.scss
- cut and paste the .title-bar class rule from _main.scss and paste it in _title-bar.scss

```
.title-bar{
    background: #e35557;
    color: #ffffff;
}
```

- go back to _main.scss
- and we are going to be putting all of our imports at the bottom

```
.wrapper{
    margin: 0 auto;
    max-width: 50rem;
    padding: 1.3rem;
```

```
}


@import './components/title-bar';
```

- create a file in components called _wrapper.scss
- now we are going to do the same thing with the wrapper rule
- cut and past the wrapper rule from _main.scss to _wrapper.scss

```
.wrapper{
    margin: 0 auto;
    max-width: 50rem;
    padding: 1.3rem;
}
```

- and now import it in _main.scss

```
h1{
    font-weight: 300;
    font-size: 2.4rem;
}


@import './components/title-bar';
@import './components/wrapper'
```

Styling the App List

- lets also center our App.js component
- go to App.js and create a div with class wrapper

```
export default class App extends React.Component{
    render(){
        return(
            <div>
                <TitleBar title={this.props.title}/>
                <div className = 'wrapper'>
                   <PlayerList players={this.props.players}/>
                   <AddPlayer />
                </div>
```

```
            </div>
        );
    }
}
```

- go to _main.scss and change the body background and change the font size

```
body{
    background: #f2f2f2;
    color : #555555 ;
    font-family: Helvetica, Arial, sans-serif ;
    font-size: 1.6rem;
}

h1{
    font-weight: 300;
    font-size: 2.4rem;
}
```

- lets create another partial in the component folder called _item.scss
- lets import it first in our _main.scss

```
h1{
    font-weight: 300;
    font-size: 2.4rem;
}

@import './components/title-bar';
@import './components/wrapper';
@import './components/item';
```

- head over to the _item.scss and create a class called .item and start adding styles

```
.item{
    background: #ffffff;
    border: 1px solid #e8e8e8;
```

```
    margin-bottom: 1.3rem;
    padding: 1.3rem;
}
```

- now lets add the .item class to some components
- go to AddPlayerjs and this class

```
    render(){
        return(
            <div className='item'>
                <form onSubmit = {this.handleSubmit.bind(this)}>
                    <input type="text" name="playerName" placeholder="Player name"/>
                    <button>Add Player</button>
                </form>
            </div>
        );
    }
}
```

- now lets add this class in the Player.js
- first of all change the p tag that's wrapping everything into a div
- then add a p tag to the first line inside the div

```
export default class Player extends React.Component{
    render(){
        return(
     <div key={this.props.player._id} className='item'>
      <p>{this.props.player.name} has {this.props.player.score} point(s).</p>
      <button onClick={()=>Players.update(this.props.player._id, {$inc:{score:1}})}>+1</button>
      <button onClick={()=>Players.update(this.props.player._id, {$inc:{score:-1}})}>-1</button>
      <button onClick={()=> Players.remove(this.props.player._id)}>X</button>
    </div>
        )
    }
}
```

- now head over to PlayerList
- and change the first return statement by adding a div

```
export default class PlayerList extends  React.Component{
    renderPlayers(){
        if(this.props.players.length === 0){
            return (
                <div className="item">
                    <p>There are no players at the moment.  Please add your first player to get
started!</p>
                 </div>
            );
        }else{
```

BEM(Block Element Modifier) Naming

- go to _item.scss

```
.item{
    background: #ffffff;
    border: 1px solid #e8e8e8;
    margin-bottom: 1.3rem;
    padding: 1.3rem;
}

.item__message{
    font-size: 1.3rem;
    font-style: italic;
    font-weight: 300;
    text-align: center;
}
```

- and now implement that class, go to PlayerList.js

```
export default class PlayerList extends  React.Component{
    renderPlayers(){
        if(this.props.players.length === 0){
            return (
                <div className="item">
                    <p className ="item_message">There are no players at the moment.  Please add
```

```
 your first player to get started!</p>
                </div>
            );
```

- lets make another partial in our components, name it _button.scss
- and now import it on the _main.scss

```
h1{
    font-weight: 300;
    font-size: 2.4rem;
}

@import './components/title-bar';
@import './components/wrapper';
@import './components/item';
@import './components/button';
```

- now go back _button.scss and start adding some styles
- we are going to have a .button class and a .button-round modifier

```
.button{
    color: #555555;
}

.button--round{

}
```

- now lets implement those classes
- go to AddPlayer.js

```
    render(){
        return(
            <div className='item'>
                <form onSubmit = {this.handleSubmit.bind(this)}>
                    <input type="text" name="playerName" placeholder="Player name"/>
                    <button className="button">Add Player</button>
                </form>
            </div>
```

```
        );
    }
}
```

- also go Player.js

```
export default class Player extends React.Component{
    render(){
        return(
     <div key={this.props.player._id} className='item'>
      <p>{this.props.player.name} has {this.props.player.score} point(s).</p>
      <button className="button" onClick={()=>Players.update(this.props.player._id, {$inc:{scor
e:1}})}>+1</button>
      <button className="button"  onClick={()=>Players.update(this.props.player._id, {$inc:{scor
e:-1}})}>-1</button>
      <button className="button"  onClick={()=> Players.remove(this.props.player._id)}>X</button
>
    </div>
        )
    }
}
```

- go back to _button.scss and keep styling

```
.button{
    border: 1px solid #555555;
    color: #555555;
    cursor: pointer;
    font-weight: 600;
    margin-left: 1.3rem;
    outline: none;
    padding: 1.3rem;

}

.button--round{

}
```

- now lets sytle the .button-round

```
.button{
    border: 1px solid #555555;
    color: #555555;
    cursor: pointer;
    font-weight: 600;
    margin-left: 1.3rem;
    outline: none;
    padding: 1.3rem;

}

.button--round{
    border-radius: 50%;
    height: 4rem;
    width: 4rem;
}
```

- go to Player.js and implement that class

```
export default class Player extends React.Component{
    render(){
        return(
     <div key={this.props.player._id} className='item'>
      <p>{this.props.player.name} has {this.props.player.score} point(s).</p>
      <button className="button button--round" onClick={()=>Players.update(this.props.player._i
d, {$inc:{score:1}})}>+1</button>
      <button className="button button--round"  onClick={()=>Players.update(this.props.player._i
d, {$inc:{score:-1}})}>-1</button>
      <button className="button button--round"  onClick={()=> Players.remove(this.props.player._
id)}>X</button>
    </div>
        )
    }
}
```

- lets keep styling the _botton.scss

```
..button{
    background: white;
    border: 1px solid #555555;
    color: #555555;
    cursor: pointer;
    font-weight: 600;
    margin-left: 1.3rem;
    outline: none;
    padding: 1.3rem;
    transition: background .2s ease transform .2s ease;
    &:hover, &:focus{
        background: #e8e8e8;
    }
    &:active{
        background: #cccccc;
        transform: scale(1.1);
    }
}

.button--round{
    border-radius: 50%;
    height: 4rem;
    width: 4rem;
}
```

- now lets style the subtitle so first go to _title-bar.scss and lets create that element

```
.title-bar{
    background: #e35557;
    color: #ffffff;
}

.titlebar__subtitle{
    font-size: 1.3rem;
```

```
    font-weight: 400;
}
```

- now lets implement that class to the TitleBar.js

```
import React from 'react';
import PropTypes from 'prop-types';

export default class TitleBar extends React.Component{
    renderSubtitle(){
        if(this.props.subtitle){
           return  <h2 className='titlebar__subtitle'>{this.props.subtitle}</h2>
        }
    }
```

- lets create a partial under components _player.sss
- and create 4 new classes

```
.player{

}

.player__name{

}

.player__stats{

}

.player__actions{

}
```

Styling Player with Flexbox

- make another partial called _form.scss
- now lets go over to _main.scss and import player and form

```
body{
    background: #f2f2f2;
    color : #555555 ;
    font-family: Helvetica, Arial, sans-serif ;
    font-size: 1.6rem;
}

h1{
    font-weight: 300;
    font-size: 2.4rem;
}

@import './components/title-bar';
@import './components/wrapper';
@import './components/item';
@import './components/button';
@import './components/form';
@import './components/player';
```

- now head over to _form.scss

```
.form{
    display: flex;

}

.form__input{

}
```

- now head one over to AddPlayer.js and implement that class

```
    render(){
        return(
            <div className='item'>
                <form className='form' onSubmit = {this.handleSubmit.bind(this)}>
                    <input type="text" name="playerName" placeholder="Player name"/>
```

```
                    <button className="button">Add Player</button>
                </form>
            </div>
        );
    }
}
```

- now go back to _form.scss and continue styling with flex

```
.form{
    display: flex;


}


.form__input{
    flex-grow: 1;
}
```

- now lets go implement that class
- go back to AddPlayer.js

```
    render(){
        return(
            <div className='item'>
                <form className='form' onSubmit = {this.handleSubmit.bind(this)}>
                    <input className='form__input' type="text" name="playerName" placeholder="Pl
ayer name"/>
                    <button className="button">Add Player</button>
                </form>
            </div>
        );
    }
}
```

- go back to styling the form

```
.form{
    display: flex;
```

```
}

.form__input{
    border: 1px solid #e8e8e8;
    flex-grow: 1;
    padding: 0 1.3rem;
}
```

- lets go to _main.scss to add more sytling

```
body{
    background: #f2f2f2;
    color : #555555 ;
    font-family: Helvetica, Arial, sans-serif ;
    font-size: 1.6rem;
}

input, button , select{
    font-size: 1.3rem;
}

h1{
    font-weight: 300;
    font-size: 2.4rem;
}

@import './components/title-bar';
@import './components/wrapper';
@import './components/item';
@import './components/button';
@import './components/form';
@import './components/player';
```

- lets go over to Player.js and we are going to be changing mark up language
- and wrapp a div around the selected area below and gave it className

```
export default class Player extends React.Component{
    render(){
        return(
     <div key={this.props.player._id} className='item'>
        <div className='player'>
            <p>{this.props.player.name} has {this.props.player.score} point(s).</p>
            <button className="button button--round" onClick={()=>Players.update(this.props.play
er._id, {$inc:{score:1}})}>+1</button>
            <button className="button button--round"  onClick={()=>Players.update(this.props.pla
yer._id, {$inc:{score:-1}})}>-1</button>
            <button className="button button--round"  onClick={()=> Players.remove(this.props.pl
ayer._id)}>X</button>
        </div>
    </div>
        )
    }
}
```

- we are also going to be adding an h3 and copying and pasting this.props.player.name inside of it
- and make the additional changes

```
export default class Player extends React.Component{
    render(){
        return(
     <div key={this.props.player._id} className='item'>
        <div className='player'>
            <h3 className='player__name'>{this.props.player.name}</h3>
            <p className='player__stats'> {this.props.player.score} point(s).</p>
            <button className="button button--round" onClick={()=>Players.update(this.props.play
er._id, {$inc:{score:1}})}>+1</button>
            <button className="button button--round"  onClick={()=>Players.update(this.props.pla
yer._id, {$inc:{score:-1}})}>-1</button>
            <button className="button button--round"  onClick={()=> Players.remove(this.props.pl
ayer._id)}>X</button>
        </div>
    </div>
        )
```

```
    }
}
```

- lets make another div and keep modifying

```
export default class Player extends React.Component{
    render(){
        return(
     <div key={this.props.player._id} className='item'>
        <div className='player'>
            <div>
                <h3 className='player__name'>{this.props.player.name}</h3>
                <p className='player__stats'> {this.props.player.score} point(s).</p>
            </div>
            <div className = 'player__actions'>
                <button className="button button--round" onClick=
{()=>Players.update(this.props.player._id, {$inc:{score:1}})}>+1</button>
                <button className="button button--round"  onClick=
{()=>Players.update(this.props.player._id, {$inc:{score:-1}})}>-1</button>
                <button className="button button--round"  onClick={()=> Players.remove(this.prop
s.player._id)}>X</button>
            </div>
        </div>
        )
    }
}
```

- go back to _player.scss and lets styling

```
.player{
    display: flex;
    justify-content: space-between;
}

.player__name{
    font-weight: 300;
```

```
    font-size: 1.8rem;
}

.player__stats{
    font-size: 1.3rem;
    font-style: italic;
    font-weight: 300;
}

.player__actions{

}
```

- now we are going to fix the problem where if the user inputs a very long name the buttons will will have a weird effect

```
.player{
    align-items: center;
    display: flex;
    justify-content: space-between;
}

.player__name{
    font-weight: 300;
    font-size: 1.8rem;
}

.player__stats{
    font-size: 1.3rem;
    font-style: italic;
    font-weight: 300;
}

.player__actions{
    flex-shrink: 0;
}
```

- go to your button partial _button.scss

```
.button{
    background: white;
    border: 1px solid #555555;
    color: #555555;
    cursor: pointer;
    font-weight: 600;
    line-height: 1;
    margin-left: 1.3rem;
    outline: none;
    padding: 1.3rem;
    transition: background .2s ease transform .2s ease;
    &:hover, &:focus{
        background: #e8e8e8;
    }
    &:active{
        background: #cccccc;
        transform: scale(1.1);
    }
}

.button--round{
    border-radius: 50%;
    height: 4rem;
    width: 4rem;
}
```

Using 3rd Party React Components (flipMove)

- first we are going to fix the responsiveness of our app, so head over to main.html and lets add a meta tag

```
<head>
  <title>Score Keep</title>
  <meta name="viewport" content="width=device-width, initial-scale=1"/>
</head>
```

```
<body>
<div id='app'></div>


</body>
```

- google react-flip-move
- go to your terminal to install this module

```
meteor npm install react-flip-move@2.7.2 --save
```

- now go to PlayerList.js and import this module and implement it to the renderPlayers()

```
import React from 'react';
import Player from './Player';
import FlipMove from 'react-flip-move';



export default class PlayerList extends  React.Component{
    renderPlayers(){
        if(this.props.players.length === 0){
            return (
                <div className="item">
                    <p className ="item_message">There are no players at the moment.  Please add
 your first player to get started!</p>
                 </div>
            );
        }else{
            return this.props.players.map((player) => {
                return (
                  <Player key={player._id} player={player}/>
                );
              });
        }
    }
    render(){
        return (
```

```
            <div>
                <FlipMove>
                    {this.renderPlayers()}
                </FlipMove>
            </div>
        );
    }
};

PlayerList.propTypes = {
    players : React.PropTypes.array.isRequired
}
```

- we are going to be adding a prop to the FlipMove component

```
    render(){
        return (
            <div>
                <FlipMove maintainContainerHeight={true}>
                    {this.renderPlayers()}
                </FlipMove>
            </div>
        );
    }
};
```

Calculating Player Standings with ES6

- lets explore object spread operator
- go to main.js

```
let user = {
    name: 'jose',
    location: 'San Diego'
}

let person={
```

```
    ...user,
    age: 25
}

console.log(person);
```

- one thing to note is that if we try to do the following

```
let user = {
    name: 'jose',
    location: 'San Diego',
    age: 0
}

let person={
    ...user,
    age: 25
}

console.log(person);
```

- the age below will overwrite the one above, so order does matter
- but if we do this instead

```
let user = {
    name: 'jose',
    location: 'San Diego',
    age: 0
}

let person={
    age: 25,
    ...user,

}

console.log(person);
```

- the answer would be age: 0, it would overwrite age: 25 because ...user its right below it
- now lets inspect Object Property Shorthand

```
//Object Property Shorthand


let bike = 'Scot';
let stuff = {
    bike: bike
}


console.log(stuff);
```

- in ES6 there is a shorthand

```
let bike = 'Scot';
let stuff = {
    bike
}


console.log(stuff);
```

- to do the positioning we are going to create a brand new function that is going to go through the array of players and its going to figure out who's in first, whos in second, and whos in third. in our application we get our players information in the client main.js. this is where we query for all players

```
 let players = Players.find({}, {
      sort: {
        score: -1
      }
    }).fetch();
```

- what we want to do is modify that array that comes back, and we are going to do that a function call that is going to get defined over inside players.js. lets call it calculatePlayerPositions, this function will take an argument, and is going to return a brand version of that argument, we are going to take the players array and is going to return the players but with properties, like rank and position
- lets create a ran variable which is 1, the first person is going to be always first

```
export const calculatePlayerPositions = (players) =>{
    let rank = 1;
}
```

- next we are going to return players.map that will let us modify with a function

```
import{Mongo} from 'meteor/mongo';

export const Players = new Mongo.Collection('players');

export const calculatePlayerPositions = (players) =>{
    let rank = 1;

    return players.map(()=>{

    });
```

- then we are going to add our argument, the individual player, which a second argument, which is the index that is going to start at 0

```
import{Mongo} from 'meteor/mongo';

export const Players = new Mongo.Collection('players');

export const calculatePlayerPositions = (players) =>{
    let rank = 1;

    return players.map((player, index)=>{

    });
}
```

- we then are going to have an if statemetn to determine whether or not we should cahnge the rank
- if the the condition meets the requirement then the rank will increase by 1

```
import{Mongo} from 'meteor/mongo';

export const Players = new Mongo.Collection('players');

export const calculatePlayerPositions = (players) =>{
    let rank = 1;
```

```
    return players.map((player, index)=>{
        if(){
            rank++;
        }
    });
}
```

- now the first person is alway going to be 1, no matter if its a tied
- so in the index is not equal to 0 and then we are going to check if the person who came in previous has a score greater than my own score

```
export const calculatePlayerPositions = (players) =>{
    let rank = 1;

    return players.map((player, index)=>{
        if(index !== 0 && players[index -1].score > player.score){
            rank++;
        }
    });
}
```

- now that we have that in place we are going to compose the correct object down below
- 1st we are going to use the spread object
- next we are going to add on the rank, and for this we will be using the ES6 shorthand
- and the last thing is calculate the position, for this we are going to be using a library called numeral
- which you can find in numeraljs.com

```
import{Mongo} from 'meteor/mongo';

export const Players = new Mongo.Collection('players');

export const calculatePlayerPositions = (players) =>{
    let rank = 1;

    return players.map((player, index)=>{
        if(index !== 0 && players[index -1].score > player.score){
            rank++;
        }
```

```
        return {
            ...player,
            rank,

        }
    });
}
```

- so first we are going to be installing numerals
- go to your terminal and type

```
meteor npm install numeral@2.0.1  --save
```

- now lets import this in players.js
- and we can now implement that in our object

```
import{Mongo} from 'meteor/mongo';
import numeral from 'numeral';

export const Players = new Mongo.Collection('players');

export const calculatePlayerPositions = (players) =>{
    let rank = 1;

    return players.map((player, index)=>{
        if(index !== 0 && players[index -1].score > player.score){
            rank++;
        }

        return {
            ...player,
            rank,
             position: numeral(rank).format('0o')
        }
    });
}
```

- now we are going to call this function over at main.js client
- first lets import it

```
import React from 'react';
import ReactDOM from 'react-dom';
import {Meteor} from 'meteor/meteor';
import {Tracker} from 'meteor/tracker';
import {Players, calculatePlayerPositions} from './../imports/api/players';
import App from './../imports/ui/App'
```

- first let create a variable postionedPlayers equal to our method and passing in players

```
Meteor.startup( () => {

  Tracker.autorun(() => {
    let players = Players.find({}, {
      sort: {
        score: -1
      }
    }).fetch();
    let positionedPlayers =  calculatePlayerPositions(players);
    let title = 'Score Keeper';
    let subtitle = 'Created by Jose Mendoza';
    ReactDOM.render(<App title={title}  players={players}/>, document.getElementById('app'))
  });
});
```

- and now we are going to injected this new variable instead of players down below

```
Meteor.startup( () => {

  Tracker.autorun(() => {
    let players = Players.find({}, {
      sort: {
        score: -1
      }
    }).fetch();
    let positionedPlayers =  calculatePlayerPositions(players);
    let title = 'Score Keeper';
    let subtitle = 'Created by Jose Mendoza';
```

```
    ReactDOM.render(<App title={title}  players={positionedPlayers}/>,
document.getElementById('app'))
  });
});
```

- the last thing is for this to show up on the screen
- we are going to show the rank and the position
- so lets go to Player.js

```
export default class Player extends React.Component{
    render(){
        return(
     <div key={this.props.player._id} className='item'>
        <div className='player'>
            <div>
                <h3 className='player__name'>{this.props.player.name}</h3>
                <p className='player__stats'>
                {this.props.player.rank} {this.props.player.position} {this.props.player.score}
point(s).</p>
            </div>
            <div className = 'player__actions'>
                <button className="button button--round" onClick=
{()=>Players.update(this.props.player._id, {$inc:{score:1}})}>+1</button>
                <button className="button button--round"  onClick=
{()=>Players.update(this.props.player._id, {$inc:{score:-1}})}>-1</button>
                <button className="button button--round"  onClick={()=> Players.remove(this.prop
s.player._id)}>X</button>
            </div>
        </div>
    </div>
```

Dynamic Classes and SCSS Functions

- go to Player.js
- instead of passing a static className='item' we are going to create a variable and that is going to use the rank and that will let us create a dynamic class
- it will will be an template string and leter we will be injecting js later
- remove {this.props.player.rank}
- and add your new variable to the className below

```
 export default class Player extends React.Component{
    render(){
        let itemClassName = `item item--position-${this.props.player.rank}`;
        return(
     <div key={this.props.player._id} className={itemClassName}>
        <div className='player'>
            <div>
                <h3 className='player__name'>{this.props.player.name}</h3>
                <p className='player__stats'>
               {this.props.player.position} place - {this.props.player.score} point(s).</p>
            </div>
```

- now lets style ranks 1 to 3
- go to the patial _item.scss and give them colors

```
.item{
    background: #ffffff;
    border: 1px solid #e8e8e8;
    margin-bottom: 1.3rem;
    padding: 1.3rem;
}

.item--position-1{
    background: green;
}
.item--position-2{
    background: blue;
}
.item--position-3{
    background: red;
}

.item__message{
    font-size: 1.3rem;
    font-style: italic;
    font-weight: 300;
    text-align: center;
```

```
}
```

- now we aer going to _main.css and create variable for three color and give them hex color instead

```
$green : #e4ede0;
```

- and now we are goign to darken that green, to explore we can google *scss script function*
- lets create another variable

```
    /* CSS declarations go here */
    $green : #e4ede0;
    $green-alt: darken($green, 15%);
*{
    margin: 0;
    padding: 0;
}
```

- go back to _item.scss and lets implement that new variable

```
.item{
    background: #ffffff;
    border: 1px solid #e8e8e8;
    margin-bottom: 1.3rem;
    padding: 1.3rem;
}

.item--position-1{
    background: $green;
    border-color: $green-alt;
```

- we are going to be making some changes to the buttons background
- go to _button.scss

```
.button{
    background: transparent;
    border: 1px solid #555555;
    color: #555555;
```

```
    cursor: pointer;
    font-weight: 600;
    line-height: 1;
    margin-left: 1.3rem;
```

- now lets finish the other 2 colors
- in _main.scss

```
    $green : #e4ede0;
    $green-alt: darken($green, 15%);
    $blue : #dfe7ed;
    $blue-alt: darken($blue, 15%);
    $red : #eddfdf;
    $red-alt: darken($red, 15%);
*{
    margin: 0;
    padding: 0;
}
```

- and now use those variable over in _itemm.scss

```
.item--position-1{
    background: $green;
    border-color: $green-alt;
}
.item--position-2{
    background: $blue;
    border-color: $blue-alt;
}
.item--position-3{
    background: $red;
    border-color: $red-alt;
```

-