4장

4.1 스타일링

- 리액트 네이티브에서는 자바스크립트를 이용해 스타일링 할 수 있습니다. 컴포넌트에는 style속성이 있고, 이 속성에 인라인 스타일을 적용하는 방법과 스타일시트에 정의된 스타일을 사용하는 방법이 있다.

4.1.1. 인라인 스타일링

```
...
const App = () => {
  return (
      <View
        style={{
          flex: 1,
          backgroundColor: '#fff',
          alignItems: 'center',
          justifyContent: 'center',
        }}
  >
      <Text
        style={{
          padding: 10,
          fontSize: 26,
          fontWeight: '600',
          color: 'black',
        }}
      >
        Inline Styling - Text
      </Text>
      <Text
        style={{
          padding: 10,
          fontSize: 26,
          fontWeight: '400',
          color: 'red',
        }}
      >
        Inline Styling - Error
      </Text>
    </View>
  );
};
...
```

- 인라인 스타일링의 장점 - 어떤 스타일이 적용되는지 잘 보인다

- 인라인 스타일링의 단점 - 비슷한 역할을 하는 컴포넌트에 동일한 코드가 반복된다, 어떤 이유로 해당 스타일이 적용되었는지 코드만으로는 명확하게 이해하기 어렵다는 단점이 있다.

4.1.2. 클래스 스타일링

- 클래스 스타일링 방법은 웹 프로그래밍에서 CSS 클래스를 이용하는 방법과 유사하다.

```
import React from 'react';
import { StyleSheet, View, Text } from 'react-native';

const App = () => {
  return (
    <View style == {styles.container}>
      <Text style == {styles.text}>Inline Styling - Text</Text>
      <Text style == {styles.error}>Inline Styling - Error</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
},
  text: {
    padding: 10,
    fontSize: 26,
    fontWeight: '600',
    color: 'black',
  },
  error: {
    padding: 10,
    fontSize: 26,
    fontWeight: '400',
    color: 'red',
  },
});

export default App;
```

→ 간단하게 화면을 확인하는 상황에서는 인라인 스타일을 사용하는 것이 편리할 수 있지만, 장기적으로 생각하면 클래스 스타일을 사용하는 것이 관리 측면에서는 유리합니다.

→ 여러 개의 스타일을 적용할 때 반드시 클래스 스타일만 적용해야 하는 것은 아닙니다. 인라인 스타일과 클래스 스타일 방식을 혼용해서 사용하는 방법도 있다.

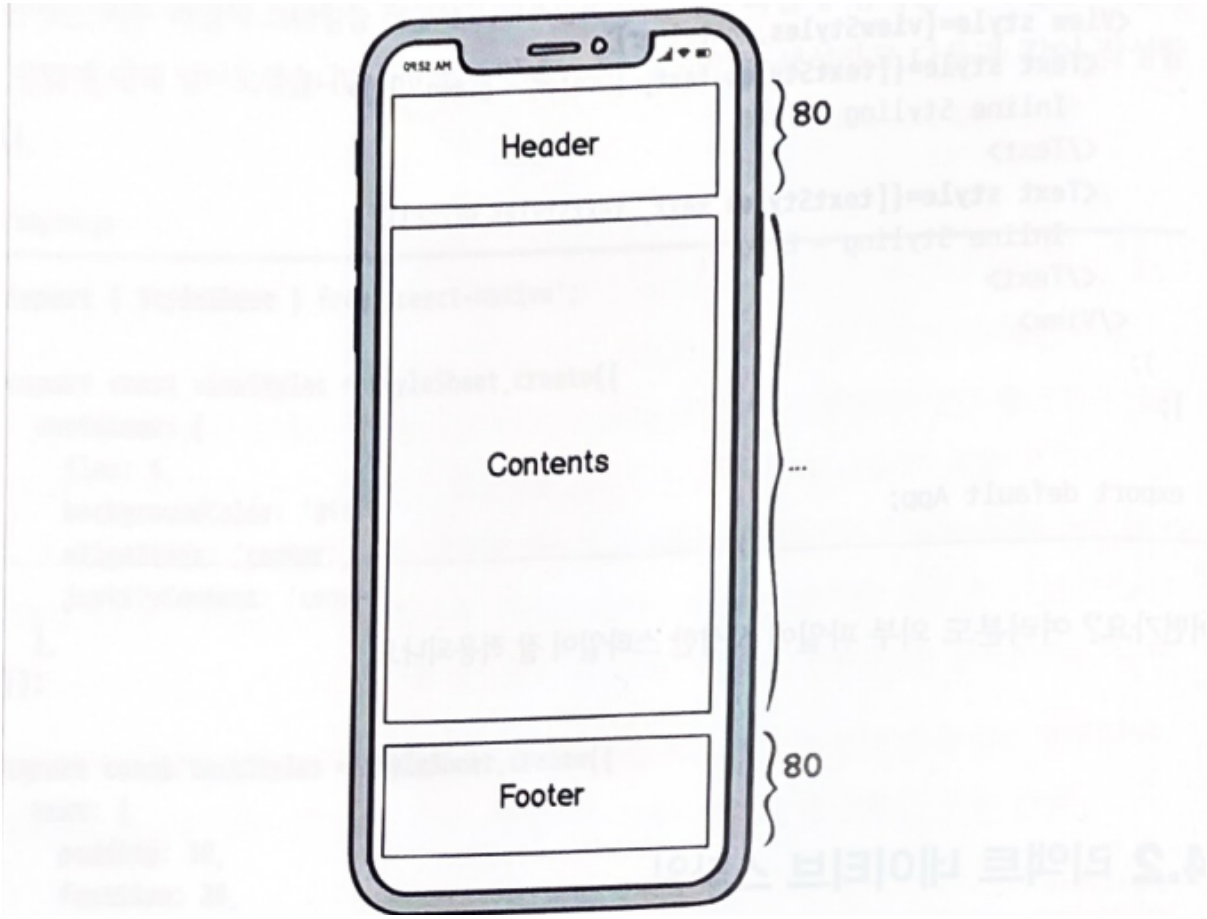
4.1.4 외부 스타일 이용하기

```
import { StyleSheet } from 'react-native';

export const viewStyles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});

export const textStyles = StyleSheet.create({
  text: {
    padding: 10,
    fontSize: 26,
    fontWeight: '600',
    color: 'black',
  },
  error: {
    fontWeight: '400',
    color: 'red',
  },
});
```

```
import React from 'react';
import { View, Text } from 'react-native';
import { viewStyles, textStyles } from './styles';
const App = () => {
  return (
    <View style={viewStyles.container}>
      <Text style={[textStyles.text, { color: 'green' }]}>
        Inline Styling - Text
      </Text>
      <Text style={[textStyles.text, textStyles.error]}>
        Inline Styling - Error
      </Text>
    </View>
  );
};
```

4.2 리액트 네이티브 스타일

4.2.1 flex와 범위




```
import React from 'react';
import { StyleSheet, View, Text } from 'react-native';

export const Header = () => {
  return (
    <View style={[styles.container, styles.header]}>
      <Text style={styles.text}>Header</Text>
    </View>
  );
};

export const Contents = () => {
  return (
    <View style={[styles.container, styles.contents]}>
      <Text style={styles.text}>Contents</Text>
    </View>
  );
};

export const Footer = () => {
```

```
  return (
    <View style={[styles.container, styles.footer]}>
      <Text style={styles.text}>Footer</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    width: '100%',
    alignItems: 'center',
    justifyContent: 'center',
    height: 80,
  },
  header: {
    backgroundColor: '#f1c40f',
  },
  contents: {
    flex: 1,
    backgroundColor: '#1abc9c',
    height: 640,
  },
  footer: {
    backgroundColor: '#3498db',
  },
  text: {
    fontSize: 26,
  },
});
```

4.2.2 정렬

- **flexDirection**

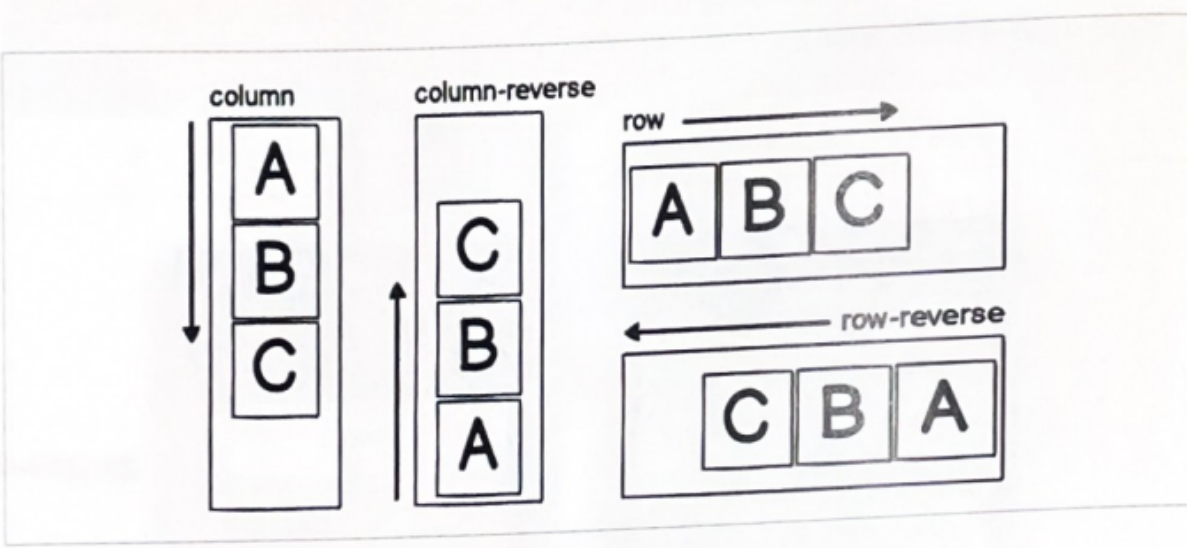


그림 4-6 flexDirection

flexDirection에 설정할 수 있는 값으로 네 가지가 있습니다.

- column: 세로 방향으로 정렬(기본값)
- column-reverse: 세로 방향 역순 정렬
- row: 가로 방향으로 정렬
- row-reverse: 가로 방향 역순 정렬

- **justifyContent**

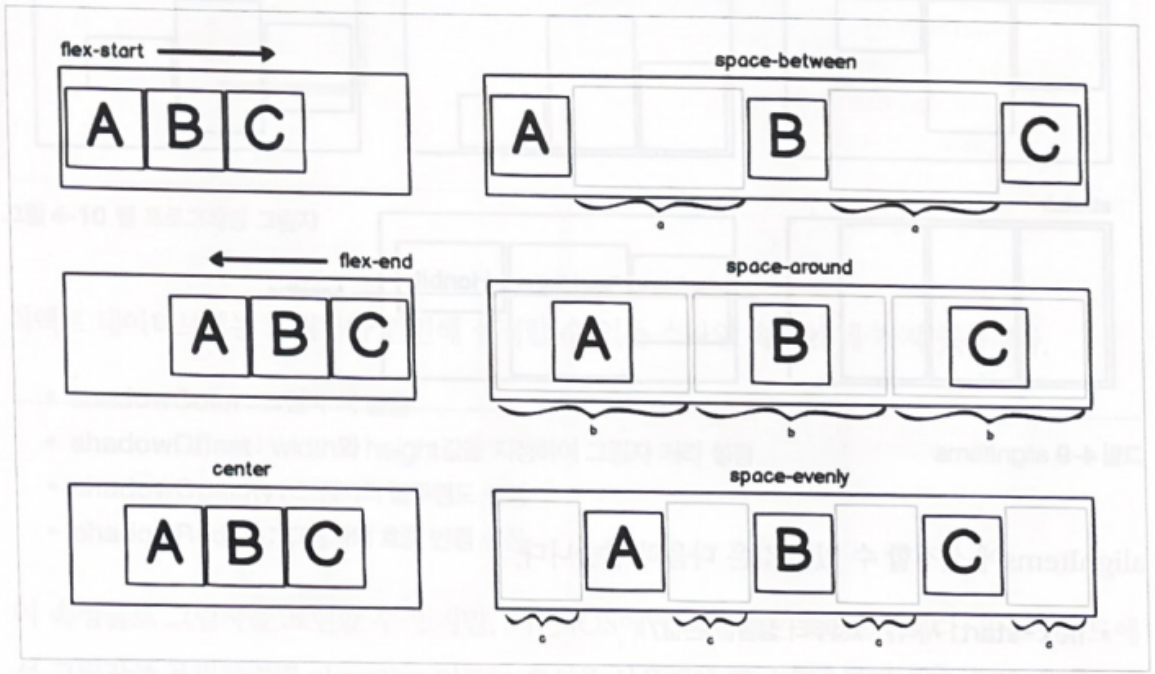


그림 4-8 justifyContent

- flex-start: 시작점에서부터 정렬(기본값)
- flex-end: 끝에서부터 정렬
- center: 중앙 정렬
- space-between: 컴포넌트 사이의 공간을 동일하게 만들어서 정렬
- space-around: 컴포넌트 각각의 주변 공간을 동일하게 만들어서 정렬
- space-evenly: 컴포넌트 사이와 양 끝에 동일한 공간을 만들어서 정렬

- **alingItems**

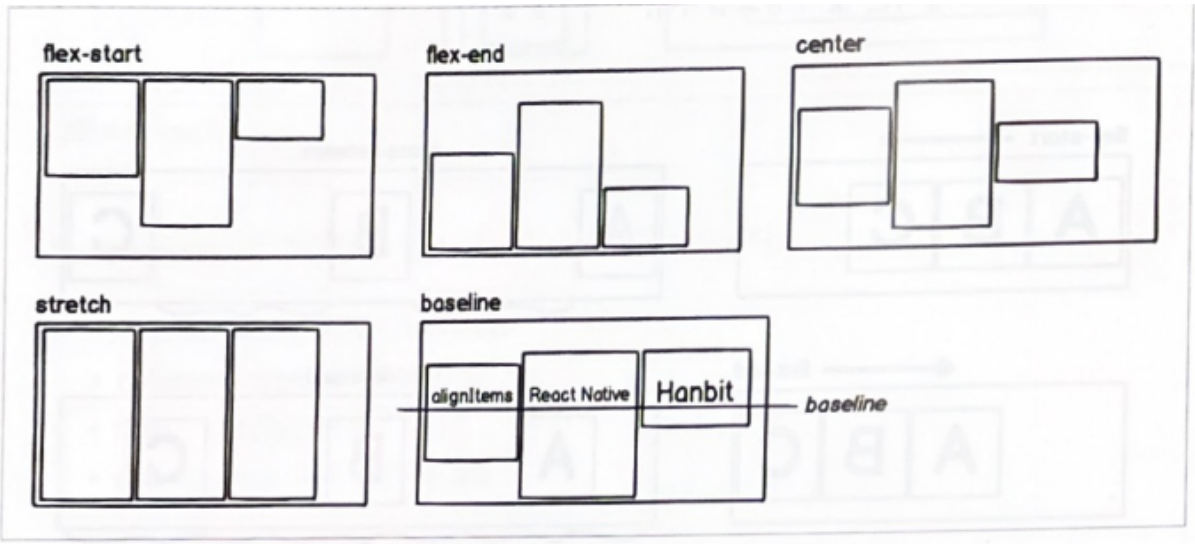


그림 4-9 alignItems

- flex-start: 시작점에서 정렬(기본값)
- flex-end: 끝에서부터 정렬
- center: 중앙 정렬
- stretch: alignItems의 방향으로 컴포넌트 확장
- baseLine: 컴포넌트 내부의 텍스트 베이스라인을 기준으로 정렬

4.2.3 그림자

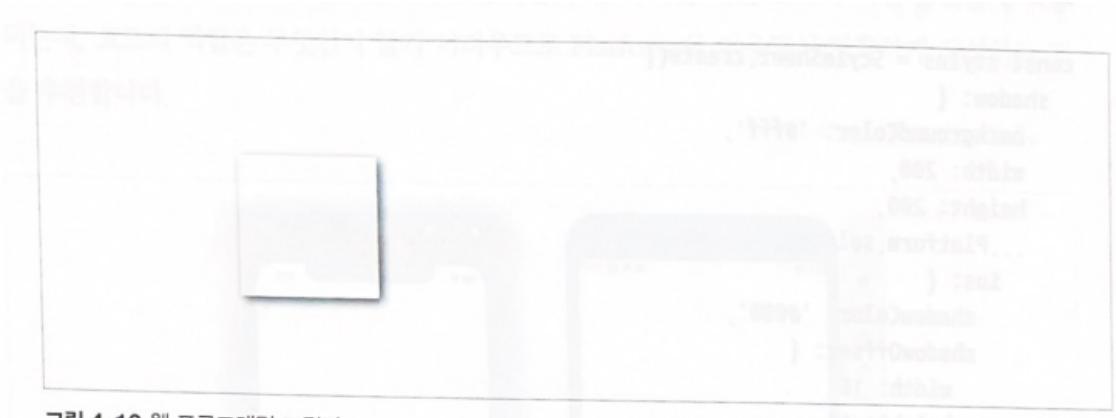


그림 4-10 웹 프로그래밍 그림자

리액트 네이티브에는 그림자와 관련해 설정할 수 있는 스타일 속성이 네 가지 있습니다.

- **shadowColor**: 그림자 색 설정
- **shadowOffset**: width와 height값을 지정하여 그림자 거리 설정
- **shadowOpacity**: 그림자의 불투명도 설정
- **shadowRadius**: 그림자의 흐림 반경 설정

```
import React from 'react';
import { StyleSheet, View, Platform } from 'react-native';

export default () => {
  return <View style={styles.shadow}></View>;
};

const styles = StyleSheet.create({
  shadow: {
    backgroundColor: '#fff',
    width: 200,
    height: 200,
    ...Platform.select({
      ios: {
        shadowColor: '#000',
        shadowOffset: {
          width: 10,
          height: 10,
        },
        shadowOpacity: 0.5,
        shadowRadius: 10,
      },
      android: {
        elevation: 20,
      },
    }),
  },
});
```

4.3 스타일드 컴포넌트

- 스타일드 컴포넌트는 자바스크립트 파일 안에 스타일을 작성하는 CSS-in-JS 라이브러리며, 스타일이 적용된 컴포넌트라고 생각하면 된다.
- 스타일드 컴포넌트를 이용하는 방법 “styled.{컴포넌트 이름}” 형태 뒤에 백틱(`)을 사용하여 만든 문자열 붙이고 그 안에 스타일을 지정하면 됩니다.

→ 이 문법을 태그드 템플릿 리터럴

```
import React from 'react';
import styled from 'styled-components/native';

const ButtonContainer = styled.TouchableOpacity`
  background-color: ${props =>
    props.title === 'Hanbit' ? props.theme.blue : props.theme.purple};
  border-radius: 15px;
  padding: 15px 40px;
  margin: 10px 0px;
  justify-content: center;
`;
const Title = styled.Text`
  font-size: 20px;
  font-weight: 600;
  color: ${props => props.theme.text};
`;

const Button = props => {
  return (
    <ButtonContainer title={props.title}>
      <Title>{props.title}</Title>
    </ButtonContainer>
  );
};

export default Button;

/*
import React from 'react';
import { StyleSheet, TouchableOpacity, Text } from 'react-native';

const styles = StyleSheet.create({
  container: {
    backgroundColor: '#9b59b6',
    borderRadius: 15,
    paddingVertical: 15,
    paddingHorizontal: 40,
    marginVertical: 10,
    justifyContent: 'center',
```

```
  },
  title: {
    fontSize: 20,
    fontWeight: '600',
    color: '#fff',
  },
});

const Button = props => {
  return (
    <TouchableOpacity
      style={[
        styles.container,
        { backgroundColor: props.title === 'Hanbit' ? '#3498db' : '#9b59b6' },
      ]}
    >
      <Text style={styles.title}>{props.title}</Text>
    </TouchableOpacity>
  );
};

export default Button;
*/
```

- 스타일드 컴포넌트에서는 스타일을 작성하는 백틱 안에서 props에 접근할 수 있다는 장점이 있다.

4.3.4 attrs 사용하기

- 스타일드 컴포넌트를 이용하면 스타일을 작성하는 곳에서 컴포넌트의 속성도 설정할 수 있다.
- 속성을 설정할 때도 전달된 props를 이용할 수 있으므로 props의 값에 따라 속성을 변경할 수 있다.

```
import React from 'react';
import styled from 'styled-components/native';

const StyledInput = styled.TextInput.attrs(props => ({
  placeholder: 'Enter a text...',
  placeholderTextColor: props.borderColor,
}))`
  width: 200px;
  height: 60px;
  margin: 5px;
  padding: 10px;
  border-radius: 10px;
  border: 2px;
  border-color: ${props => props.borderColor};
  font-size: 24px;
```

```
`;

const Input = props => {
  return <StyledInput borderColor={props.borderColor} />;
};

export default Input;
```

4.3.5 ThemeProvider

- ThemeProvider는 Context API를 활용해 애플리케이션 전체에서 스타일드 컴포넌트를 이용할 때 미리 정의한 값들을 사용할 수 있도록 props를 전달한다.

```
export const lightTheme = {
  background: '#ffffff',
  text: '#ffffff',
  purple: '#9b59b6',
  blue: '#3498db',
};

export const darkTheme = {
  background: '#34495e',
  text: '#34495e',
  purple: '#9b59b6',
  blue: '#3498db',
};

/*
export const theme = {
  purple: '#9b59b6',
  blue: '#3498db',
};
*/
```

```
import React, { useState } from 'react';
import { Switch } from 'react-native';
import styled, { ThemeProvider } from 'styled-components/native';
import Button from './components/Button';
import Input from './components/Input';
import { lightTheme, darkTheme } from './theme';

const Container = styled.View`
  flex: 1;
  background-color: ${props => props.theme.background};
  align-items: center;
  justify-content: center;
`;

const App = () => {
```

```
  const [isDark, setIsDark] = useState(false);
  const _toggleSwitch = () => setIsDark(!isDark);

  return (
    <ThemeProvider theme={isDark ? darkTheme : lightTheme}>
      <Container>
        <Switch value={isDark} onValueChange={_toggleSwitch} />
        <Button title="Hanbit" />
        <Button title="React Native" />
        <Input borderColor="#3498db" />
        <Input borderColor="#9b59b6" />
      </Container>
    </ThemeProvider>
  );
};

export default App;
```