React Native

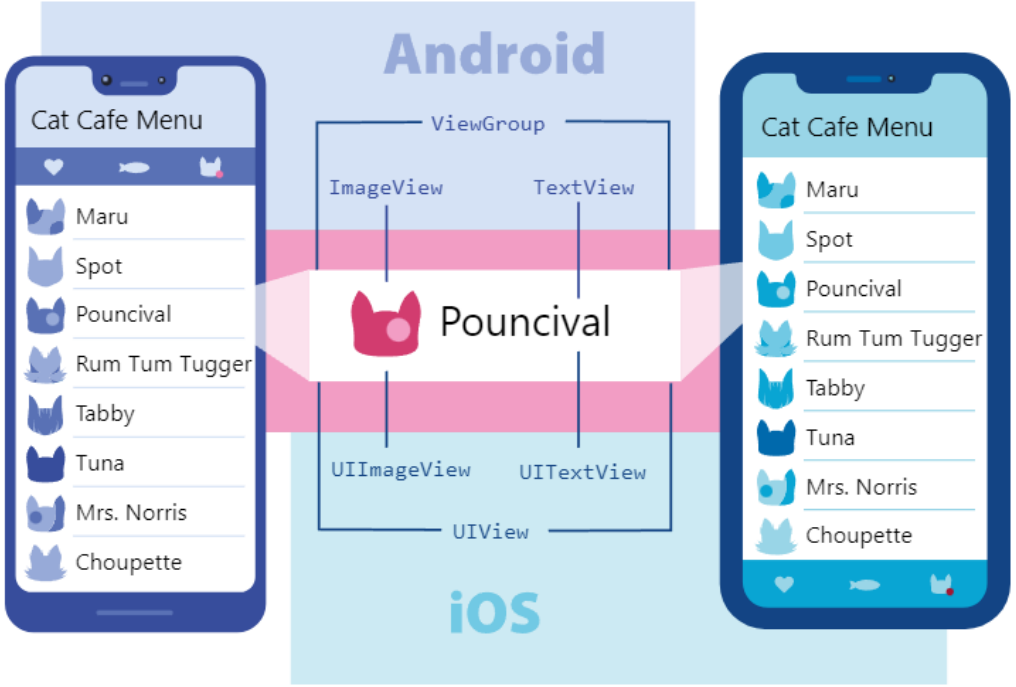
주요 개념

네이티브 컴포넌트 (Native Components)

- Android 및 iOS의 네이티브 UI 요소를 직접 사용하거나 네이티브 코드로 작성된 커스텀 컴포넌트를 포함하며 이를 통해 성능을 향상시키거나 플랫폼별 고유 기능을 사용
- 주요 네이티브 컴포넌트
  - iOS 네이티브 컴포넌트
    - UIScrollView: iOS에서 스크롤 가능한 뷰를 만들 때 사용
    - UITableView: 테이블 형식의 데이터를 표시할 때 사용
    - UICollectionView: 그리드 형식의 데이터를 표시할 때 사용
    - UIPickerView: 선택기를 표시할 때 사용
  - Android 네이티브 컴포넌트
    - RecyclerView: 스크롤 가능한 리스트를 최적화하여 표시할 때 사용
    - ViewPager: 페이지 전환을 구현할 때 사용
    - SurfaceView: OpenGL 콘텐츠를 표시할 때 사용
    - SeekBar: 슬라이더를 구현할 때 사용

네이티브 컴포넌트 (Native Components)

- 네이티브 컴포넌트 동작 방식
  - Android 개발에서는 Kotlin 또는 Java로 뷰를 작성하고, iOS 개발에서는 Swift 또는 Objective-C를 사용
  - React Native를 사용하면 React 컴포넌트를 사용하여 자바스크립트로 이러한 뷰를 호출 가능
  - React Native는 런타임으로 해당 컴포넌트에 해당하는 Android 및 iOS 뷰를 생성
  - Native 컴포넌트는 안드로이드 및 iOS와 동일한 뷰로 지원되기 때문에
    리액트 네이티브 앱은 다른 앱과 같은 모양, 느낌, 성능을 갖는 경우도 있음



코어 컴포넌트 (Core Components)

- React Native에서 제공하는 기본적인 빌딩 블록들로, 대부분의 애플리케이션에서 자주 사용되며 이러한 컴포넌트들은 다양한 모바일 UI 요소들을 JavaScript 코드로 구현할 수 있게 제공

REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
`<View>`	`<ViewGroup>`	`<UIView>`	A non-scrolling `<div>`	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
`<Text>`	`<TextView>`	`<UITextView>`	`<p>`	Displays, styles, and nests strings of text and even handles touch events
`<Image>`	`<ImageView>`	`<UIImageView>`	`<img>`	Displays different types of images
`<ScrollView>`	`<ScrollView>`	`<UIScrollView>`	`<div>`	A generic scrolling container that can contain multiple components and views
`<TextInput>`	`<EditText>`	`<UITextField>`	`<input type="text">`	Allows the user to enter text

코어 컴포넌트 (Core Components)

1. View:
   - 모든 UI 컴포넌트의 기본 컨테이너
   - Flexbox 레이아웃을 사용하여 다른 컴포넌트들을 배치 가능
2. Text:
   - 텍스트를 화면에 표시하는 데 사용
   - 다양한 스타일링 옵션을 지원
3. Image:
   - 이미지를 표시하는 데 사용
   - 원격 URL 또는 로컬 리소스를 통해 이미지를 불러올 수 있음
4. TextInput:
   - 사용자가 텍스트를 입력할 수 있는 입력 필드
   - 다양한 키보드 타입과 입력 옵션을 설정할 수 있음
5. ScrollView:
   - 스크롤 가능한 영역을 만들 수 있음
   - 콘텐츠가 화면을 벗어나는 경우 사용됨
6. FlatList:
   - 최적화된 스크롤 가능한 리스트를 생성
   - 데이터의 가상화를 통해 성능을 향상

코어 컴포넌트 (Core Components)

1. View

- 정의: 레이아웃을 구성하는 가장 기본적인 컴포넌트
- 사용 목적: 컨테이너로서 다른 컴포넌트를 포함하고 레이아웃을 구성하는 목적
- 예제 코드:

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

export default function App() {
  return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
      <Text>Hello, World!</Text>
    </View>
  );
}
```

코어 컴포넌트 (Core Components)

2. Text

- 정의: 텍스트를 화면에 표시하는 컴포넌트
- 사용 목적: 텍스트 콘텐츠를 표시하는 목적
- 예제 코드:

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

export default function App() {
  return (
    <Text style={{ fontSize: 20, color: 'blue' }}>
      Hello, World!
    </Text>
  );
}
```

코어 컴포넌트 (Core Components)

3. Image

- 정의: 이미지를 표시하는 컴포넌트
- 사용 목적: 이미지 콘텐츠를 표시하는 목적
- 예제 코드:

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

export default function App() {
  return (
    <View style={{ alignItems: 'center', justifyContent: 'center', flex: 1 }}>
      <Image
        source={require('./assets/favicon.png')}
        style={{ width: 200, height: 200 }}
        resizeMode="contain"
      />
    </View>
  );
}
```

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

export default function App() {
  return (
    <View style={{ alignItems: 'center', justifyContent: 'center', flex: 1 }}>
      <Image
        source={{ uri:
'https://upload.wikimedia.org/wikipedia/commons/thumb/e/eb/Daedeok_Software_Meister
_High_School.jpg/600px-Daedeok_Software_Meister_High_School.jpg' }}
        style={{ width: 200, height: 200 }}
      />
    </View>
  );
}
```

코어 컴포넌트 (Core Components)

4. TextInput

- 정의: 사용자로부터 텍스트 입력을 받는 컴포넌트
- 사용 목적: 입력 폼을 구성하는 목적
- 예제 코드:

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

export default function App() {
  const [text, setText] = useState('');

  return (
    <View style={{ marginTop: 50, padding: 10 }}>
      <TextInput
        style={{ height: 40, borderColor: 'gray', borderWidth: 1 }}
        placeholder="Type here"
        onChangeText={text => setText(text)}
        value={text}
      />
      <Text style={{ padding: 10, fontSize: 42 }}>
        {text}
      </Text>
    </View>
  );
}
```

코어 컴포넌트 (Core Components)

5. ScrollView

- 정의: 스크롤 가능한 영역을 제공하는 컴포넌트
- 사용 목적: 화면을 벗어나는 콘텐츠를 스크롤로 볼 수 있도록 하는 목적
- 예제 코드:

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

export default function App() {
  return (
    <ScrollView>
      <Text style={{ fontSize: 96 }}>Scroll me plz</Text>
      <Text style={{ fontSize: 96 }}>If you like</Text>
      <Text style={{ fontSize: 96 }}>Scrolling down</Text>
      <Text style={{ fontSize: 96 }}>What's the best</Text>
      <Text style={{ fontSize: 96 }}>Framework around?</Text>
      <Text style={{ fontSize: 80 }}>React Native</Text>
    </ScrollView>
  );
}
```

코어 컴포넌트 (Core Components)

6. FlatList

- 정의:

  최적화된 스크롤 가능한

  리스트를 제공하는 컴포넌트
- 사용 목적:

  큰 데이터셋을 효율적으로

  렌더링하는 목적
- 예제 코드:

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

const DATA = [
  { id: '1', title: 'Item 1' },
  { id: '2', title: 'Item 2' },
  { id: '3', title: 'Item 3' },
];

const Item = ({ title }) => (
  <View style={{ padding: 20 }}>
    <Text style={{ fontSize: 24 }}>{title}</Text>
  </View>
);

export default function App() {
  return (
    <FlatList
      data={DATA}
      renderItem={({ item }) => <Item title={item.title} />}
      keyExtractor={item => item.id}
    />
  );
}
```

코어 컴포넌트 (Core Components)

6. FlatList

- 주요 프로퍼티
  - data:
    - 정의: 렌더링할 데이터 배열
    - 설명: 리스트에 표시할 항목들의 배열을 지정하는 프로퍼티.

      각 항목은 고유의 키를 가져야 하며, 이 키는 keyExtractor 프로퍼티로 지정함
  - renderItem:
    - 정의: 각 항목을 렌더링하는 함수
    - 설명: 리스트의 각 항목을 어떻게 렌더링할지 정의하는 함수

      함수는 item 객체를 매개변수로 받아 JSX 요소를 반환함
  - keyExtractor:
    - 정의: 고유 키를 추출하는 함수
    - 설명: 각 항목의 고유 키를 반환하는 함수이며, 리스트의 각 항목이 고유한 식별자를 가지도록 보장

코어 컴포넌트 (Core Components)

7. SectionList

- 정의:

  섹션별로 나뉜 리스트를

  제공하는 컴포넌트

- 사용 목적:

  섹션으로 구분된 리스트를

  효율적으로 렌더링하는 목적

- 예제 코드:

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

const DATA = [
  {
    title: 'Main dishes',
    data: ['Pizza', 'Burger', 'Risotto'],
  },
  {
    title: 'Sides',
    data: ['French Fries', 'Onion Rings', 'Fried Shrimps'],
  },
];

export default function App() {
  return (
    <SectionList
      sections={DATA}
      keyExtractor={(item, index) => item + index}
      renderItem={({ item }) => <Text style={{ padding: 20 }}>{item}</Text>}
      renderSectionHeader={({ section: { title } }) => (
        <View style={{ backgroundColor: '#f0f0f0', padding: 10 }}>
          <Text style={{ fontWeight: 'bold', fontSize: 18 }}>{title}</Text>
        </View>
      )}
    />
  );
}
```

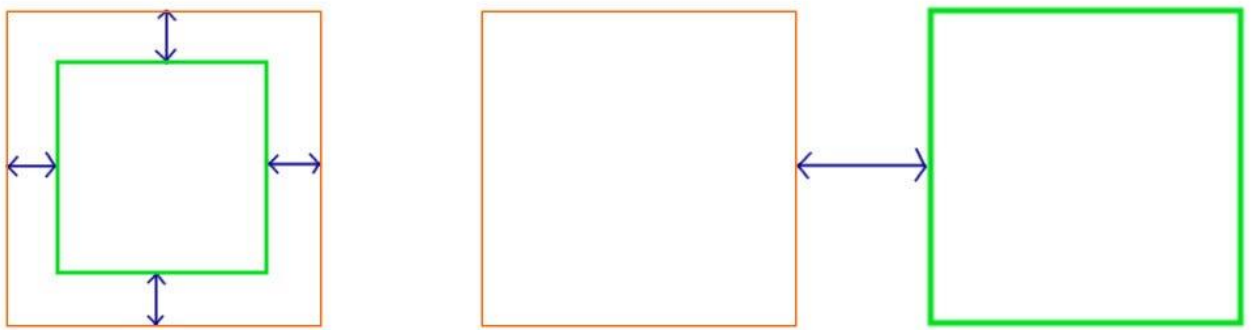
코어 컴포넌트 (Core Components)

7. SectionList

- 주요 프로퍼티
  - sections:
    - 정의: 섹션 데이터 배열
    - 설명: 섹션과 항목들로 구성된 데이터를 지정하는 프로퍼티이며, 각 섹션은 title과 data 배열을 포함
  - renderItem:
    - 정의: 각 항목을 렌더링하는 함수
    - 설명: 리스트의 각 항목을 어떻게 렌더링할지 정의하는 함수
      함수는 item 객체를 매개변수로 받아 JSX 요소를 반환함
  - renderSectionHeader:
    - 정의: 섹션 헤더를 렌더링하는 함수
    - 설명: 각 섹션의 헤더를 어떻게 렌더링할지 정의하는 함수
      함수는 섹션 객체를 매개변수로 받아 JSX 요소를 반환함
  - keyExtractor:
    - 정의: 고유 키를 추출하는 함수
    - 설명: 각 항목의 고유 키를 반환하는 함수이며, 리스트의 각 항목이 고유한 식별자를 가지도록 보장

UI: margin & padding

- margin: 컴포넌트의 외부 여백을 설정, 컴포넌트와 주변 요소 사이의 간격을 제어
  부모 컴포넌트나 같은 부모 컴포넌트 안에 있는 자식들과의 여백을 지정하는데 사용
- padding: 컴포넌트의 내부 여백을 설정, 컴포넌트의 콘텐츠와 테두리 사이의 간격을 제어
  margin과는 반대로 자신의 자식 컴포넌트와의 여백을 주는 속성



https://medium.com/frontendshortcut/margin-vs-padding-c1fc8ea8bfaf

UI: margin & padding

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

const App = () => {
  return (
    <View style={styles.container}>
      <View style={styles.box}>
        <Text style={styles.text}>Box with margin</Text>
      </View>
      <View style={styles.box}>
        <Text style={styles.textWithPadding}>Box with padding</Text>
      </View>
    </View>
  );
};

export default App;
```

```jsx
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#f0f0f0',
  },
  box: {
    width: 200,
    height: 100,
    backgroundColor: '#4CAF50',
    margin: 20, // 외부 여백
  },
  text: {
    color: 'white',
    textAlign: 'center',
    lineHeight: 100, // 텍스트를 수직 중앙 정렬하기 위해
  },
  textWithPadding: {
    color: 'white',
    textAlign: 'center',
    padding: 20, // 내부 여백
    backgroundColor: '#8BC34A',
  },
});
```

UI: Flexbox

- Flexbox는 React Native에서 레이아웃을 구성하는 주요 방법
- React Native는 기본적으로 CSS의 Flexbox 레이아웃 모델을 따르며,
  이를 통해 화면 크기나 방향에 따라 자동으로 적응하는 UI를 만들 수 있음

- Flexbox의 주요 개념

1. 컨테이너(Container):
   - Flexbox 레이아웃을 적용하는 요소
   - flexDirection, justifyContent, alignItems 등의 속성을 설정
2. 아이템(Item):
   - Flexbox 컨테이너 안에 배치되는 요소
   - flex, alignSelf, flexBasis 등의 속성을 설정

Flexbox 주요 속성

- flex
  - 주 축을 따라 사용 가능한 공간에 항목을 "채우는" 방법을 정의
  - 공간은 각 요소의 플렉스 속성에 따라 분할
  - 기본값: 0
- flexDirection
  - 주 축 방향을 설정하는 속성
  - 옵션: row (기본값), column, row-reverse, column-reverse
- justifyContent
  - 주 축 방향으로 아이템을 정렬하는 속성
  - 옵션: flex-start (기본값), center, flex-end, space-between, space-around, space-evenly
- alignItems
  - 교차 축 방향으로 아이템을 정렬하는 속성
  - 옵션: stretch (기본값), flex-start, center, flex-end, baseline

Flexbox 예제

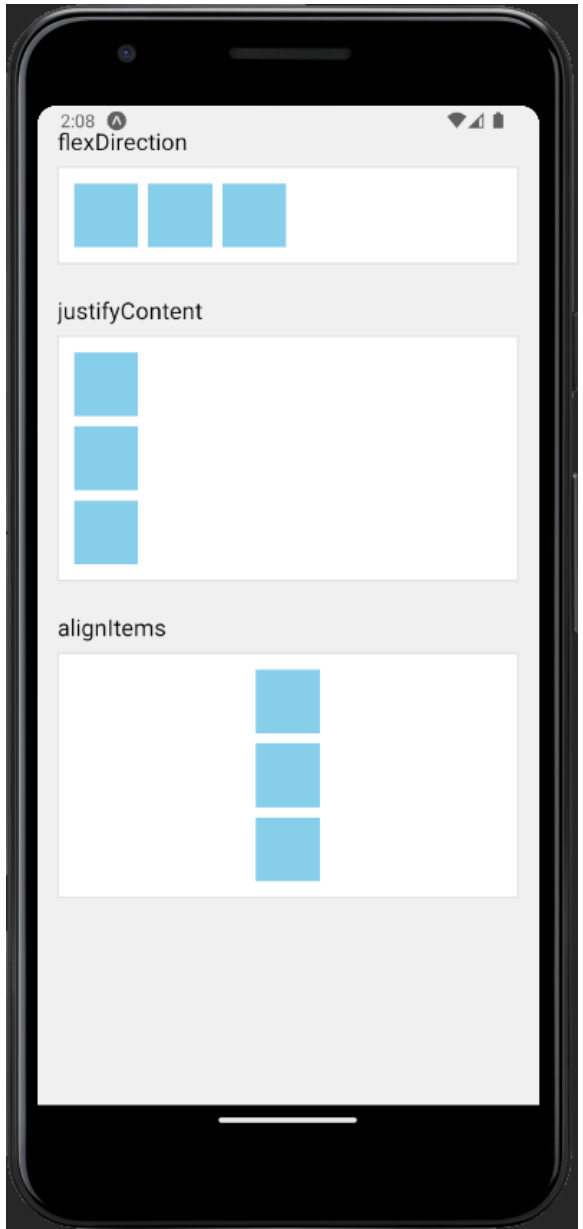
```jsx
import React from 'react';
import {StyleSheet, View} from 'react-native';

const Flex = () => {
  return (
    <View
      style={[
        styles.container,
        {
          // Try setting `flexDirection` to `"row"`.
          flexDirection: 'column',
        },
      ]}>
      <View style={{flex: 1, backgroundColor: 'red'}} />
      <View style={{flex: 2, backgroundColor: 'darkorange'}} />
      <View style={{flex: 3, backgroundColor: 'green'}} />
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 20,
  },
});

export default Flex;
```

Flexbox 예제




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

const FlexboxExamples = () => {
  return (
    <ScrollView style={styles.container}>
      {/* flexDirection 예제 */}
      <View style={styles.section}>
        <Text style={styles.sectionTitle}>flexDirection</Text>
        <View style={[styles.boxContainer, { flexDirection: 'row' }]}>
          <View style={styles.box} />
          <View style={styles.box} />
          <View style={styles.box} />
        </View>
      </View>

      {/* justifyContent 예제 */}
      <View style={styles.section}>
        <Text style={styles.sectionTitle}>justifyContent</Text>
        <View style={[styles.boxContainer, { justifyContent: 'space-between'
}]}>
          <View style={styles.box} />
          <View style={styles.box} />
          <View style={styles.box} />
        </View>
      </View>

      {/* alignItems 예제 */}
      <View style={styles.section}>
        <Text style={styles.sectionTitle}>alignItems</Text>
        <View style={[styles.boxContainer, { alignItems: 'center' }]}>
          <View style={styles.box} />
          <View style={styles.box} />
          <View style={styles.box} />
        </View>
      </View>
    </ScrollView>
  );
};
```

```jsx
const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 16,
    backgroundColor: '#f0f0f0',
  },
  section: {
    marginBottom: 24,
  },
  sectionTitle: {
    fontSize: 18,
    marginBottom: 8,
  },
  boxContainer: {
    flexDirection: 'column',
    backgroundColor: '#fff',
    padding: 8,
    borderWidth: 1,
    borderColor: '#ddd',
  },
  box: {
    width: 50,
    height: 50,
    backgroundColor: 'skyblue',
    margin: 4,
  },
});

export default FlexboxExamples;
```