10 Implementowanie komponentów i API specyficznych dla iOS

Tematyka rozdziału:

- strategia skutecznego tworzenia kodu specyficznego dla danej platformy,
- korzystanie z pól wyboru `DatePickerIOS` i `PickerIOS`,
- pokazywanie postępu ładowania za pomocą `ProgressViewIOS`,
- wybieranie widoków za pomocą `SegmentedControlIOS` i `TabBarIOS`,
- wywoływanie i wybieranie elementów z arkusza akcji za pomocą `ActionSheetIOS`.

Jednym z celów projektu React Native jest minimalizacja ilości logiki i kodu specyficznego dla danej platformy. Dzięki istniejącym API kod specyficzny dla danej platformy można oddzielić od frameworka, co powoduje, że takie aplikacje są niezależne od platformy oraz pozwalają na łatwe tworzenie uniwersalnych funkcjonalności.

Niestety zawsze będą istnieć API specyficzne dla danej platformy, których nie da się w pełni uogólnić za pomocą podejścia, które miałoby sens na różnych platformach. Dlatego często musimy użyć nawet kilku interfejsów API i komponentów specyficznych

dla danej platformy. W tym rozdziale omówimy API i komponenty specyficzne dla iOS, przedyskutujemy ich właściwości i metody oraz zbudujemy przykłady zawierające funkcjonalności i logikę umożliwiające szybsze wykonywanie zadań.

10.1. Tworzenie kodu specyficznego dla platformy

Główną ideą kodu specyficznego dla platformy jest pisanie komponentów i plików w taki sposób, który powoduje wygenerowanie kodu specyficznego dla iOS lub Androida na podstawie platformy, na której aktualnie się znajdujemy. Istnieje kilka technik, które można wdrożyć, aby wyświetlać komponenty, opierając się na platformie, na której działa aplikacja. Omówimy tutaj dwie najbardziej przydatne z tych technik: użycie właściwego rozszerzenia pliku i użycie Platform API.

10.1.1. Rozszerzenia plików iOS i Androida

Pierwszym sposobem tworzenia kodu specyficznego dla platformy jest nadanie plikowi poprawnego rozszerzenia, w zależności od platformy, na którą jest on przeznaczony. Na przykład jednym z komponentów, który różni się nieco między iOS i Androidem, jest `DatePicker`. Jeśli chcemy zastosować określone style dla `DatePicker`, tworząc cały kod w głównym komponencie, to rezultat może być rozwlekły i trudny do utrzymania. Zamiast tego utworzymy dwa pliki – `DatePicker.ios.js` i `DatePicker.android.js` – i zaimportujemy je do głównego komponentu. Po uruchomieniu projektu React Native automatycznie wybierze odpowiedni plik na podstawie używanej platformy. Spójrzmy na prosty przykład z listingów 10.1, 10.2 i 10.3. (Zauważmy, że ten przykład w obecnej postaci wygeneruje błąd – do poprawnego działania `DatePicker` wymaga zarówno właściwości, jak i metod).

Listing 10.1. Kod specyficzny dla iOS

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

export default () => (
  <View>
    <Text>This is an iOS specific component</Text>
    <DatePickerIOS />
  </View>
)
```

Listing 10.2. Kod specyficzny dla Androida

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

export default () => (
  <View>
    <Text>This is an Android specific component</Text>
    <DatePickerAndroid />
  </View>
)
```