Implementing Android-specific components and APIs

This chapter covers

* How to implement and use Android-specific APIs
* Using DrawerLayoutAndroid to create a side menu
* Creating a native toolbar with ToolbarAndroid
* Create paging views using ViewPagerAndroid
* Using DatePickerAndroid to create and manage dates in your application
* Managing time with TimePickerAndroid
* Creating toasts using ToastAndroid

In this chapter, we will implement the most used Android specific APIs & components, discuss their props and methods, and create examples that will mimic functionality and logic that will get you up to speed quickly.

We will do so by creating a demo app that will show off these Android specific APIs and components.

* 1. Creating the menu using DrawerLayoutAndroid

To get started, we will first create a slide out menu. This menu will link to each of these pieces of functionality. This slide out menu will basically serve as a way for us to navigate between the components we create. We will create this menu using the DrawerLayoutAndroid component.

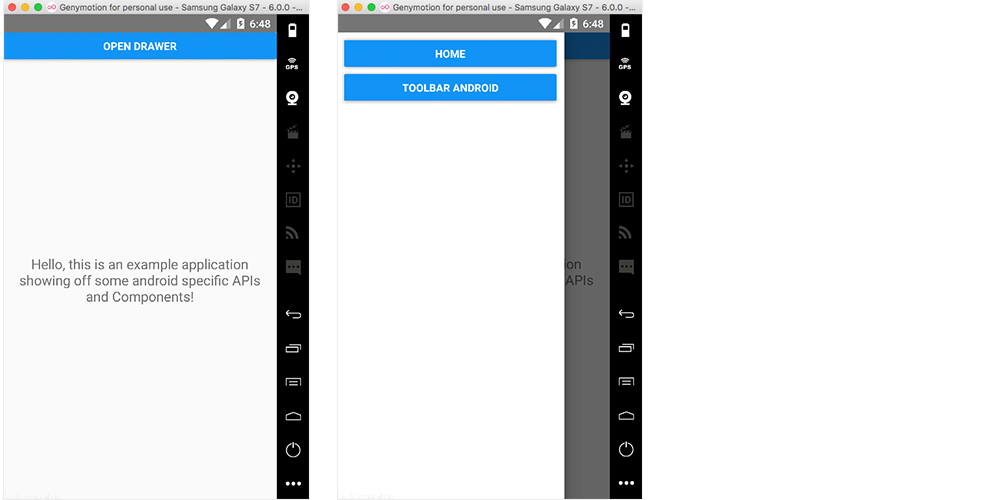


Figure 10.1 Initial layout of our application using DrawerLayoutAndroid. The button at the top in the first screen, open drawer, will call a method that opens the drawer. The second screen will be the opened drawer.

The first thing we need to do is create a new android application. From your command line in the folder that you will be working in, create a new application with YourApplication being the name of the application you are choosing.

react-native init YourApplication

The next thing we need to do is create the files we will be using to create all this functionality. In the root of the application, create a folder named app and three files in this folder named App.js, Home.js, Menu.js, and Toolbar.js.

Now we need to update index.android.js to use our first Android specific API, DrawerLayoutAndroid, which is the sliding toolbar from the left of the screen (figure 10.1).

To get started, let’s edit index.android.js to include and implement this component.

Listing 10.1 Implementing the DrawerLayoutAndroid API index.android.js

import React from 'react'

import {

AppRegistry,

DrawerLayoutAndroid, // A

Button,

View

} from 'react-native'

import Menu from './app/Menu' // C

import App from './app/App' // B

class chapter10 extends React.Component {

constructor () {

super()

this.state = {

scene: 'Home' // C

}

this.jump = this.jump.bind(this)

this.openDrawer = this.openDrawer.bind(this)

}

openDrawer () {

this.drawer.openDrawer() // D

}

jump (scene) { // E

this.setState({

scene

})

this.drawer.closeDrawer()

}

render () {

return (

<DrawerLayoutAndroid // F

ref={drawer => this.drawer = drawer} // a

drawerWidth={300} // b

drawerPosition={DrawerLayoutAndroid.positions.Left} // c

renderNavigationView={() => <Menu onPress={this.jump} />}> // d

<View style={{ margin: 15 }}> // e

<Button onPress={() => this.openDrawer()} title='Open Drawer' />

</View>

<App // e

openDrawer={this.openDrawer}

jump={this.jump

scene={this.state.scene} />

</DrawerLayoutAndroid>

)

}

}

AppRegistry.registerComponent('chapter10', () => chapter10)

1. import DrawerLayoutAndroid from React Native
2. import the yet to be created App component
3. create a component state setting scene to ‘Home’
4. create a method to open the Drawer
5. create a method to update the scene state, and then call closeDrawer()
6. implement the DrawerLayoutAndroid component
   1. create a reference to the drawer to call methods on the component
   2. give the drawer a width of 300
   3. position the drawer to the left
   4. render the navigation view, which is a Menu component we have yet to create
   5. pass in a button as a child and attach the jump method to it. We will use this across the application to open the drawer. We also pass in the App component as a child, giving the openDrawer, jump, and scene as props.

Next, we will need to create the menu we will be using in the drawer. In app/Menu.js, create Menu.s(listing 10.2).

Listing 10.2 Menu.js - Creating the DrawerLayoutAndroid menu

import React from 'react'

import { View, StyleSheet, Button } from 'react-native'

let styles

const Menu = ({onPress }) => {

const {

button

} = styles

return (

<View style={{ flex: 1 }}>

<View style={button} >

<Button onPress={() => onPress('Home')} title='Home' />

</View>

<View style={button} >

<Button onPress={() => onPress('Toolbar')} title='Toolbar Android' />

</View>

</View>

)

}

styles = StyleSheet.create({

button: {

margin: 10,

marginBottom: 0

}

})

export default Menu

Next, let’s create the App component that will render based on the scene prop that is passed to it.

<App

openDrawer={this.openDrawer}

jump={this.jump

scene={this.state.scene} />

In app/App.js, create the following component which basically takes in a scene as a prop, and returns a component based on the prop.(listing 10.3).

Listing 10.3 app/App.js - Creating the DrawerLayoutAndroid menu

import React from 'react'

import Home from './Home' // A

import Toolbar from './Toolbar' // B

function getScene (scene) { // C

switch (scene) {

case 'Home':

return Home

case 'Toolbar':

return Toolbar

default:

return Home

}

}

const App = (props) => {

const Scene = getScene(props.scene) // D

return (

<Scene openDrawer={props.openDrawer} jump={props.jump} /> // E

)

}

export default App

1. import the Home component that we have yet to create
2. import the Toolbar component that we have yet to create
3. create a getScene method that will check the scene and return the correct component
4. create a component based on the current scene prop
5. render the component, passing in openDrawer and jump as props

Now we can start creating components to interact with the menu. For our current setup to work, we need to create a Home and a Toolbar component because we have already imported them and they have yet to be created.

In app/Home.js, create the following component which will be a basic introduction page(listing 10.4).

Listing 10.4 app/Home.js - Creating the DrawerLayoutAndroid menu

import React, { Component } from 'react'

import {

View,

Text,

StyleSheet

} from 'react-native'

let styles

class Home extends Component {

render () {

return (

<View style={styles.container}>

<Text

style={styles.text}>

Hello, this is an example application showing off some android specific APIs and Components!

</Text>

</View>

)

}

}

styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center'

},

text: {

margin: 20,

textAlign: 'center',

fontSize: 18

}

})

export default Home

And in app/Toolbar.js, create the following component which will show that we are in the toolbar by showing a message that says “Hello from Toolbar”(listing 10.5).

Listing 10.5 app/Toolbar.js - Creating the DrawerLayoutAndroid menu

import React from 'react'

import {

View,

Text

} from 'react-native'

class ToolBar extends React.Component {

render () {

return (

<View style={{ flex: 1 }}>

<Text>Hello from Toolbar</Text>

</View>

)

}

}

export default ToolBar

Now, we should be able to start the application and see the toolbar as seen in figure 10.1.

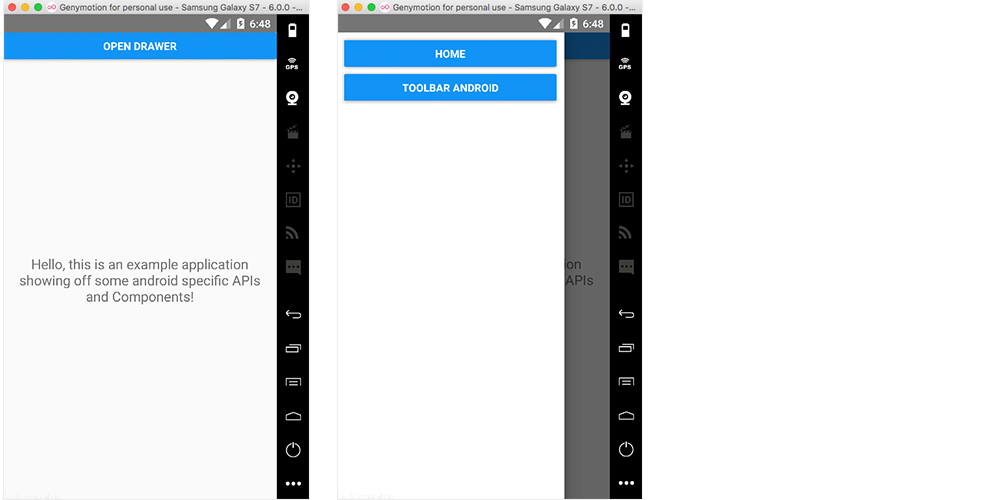


Figure 10.1 Initial layout of our application using DrawerLayoutAndroid. The button at the top in the first screen, open drawer, will call a method that opens the drawer. The second screen will be the opened drawer.

* 1. Creating a toolbar with ToolbarAndroid

Now that everything is set up, let’s add a new component, ToolbarAndroid. ToolbarAndroid is a React Native component that wraps the native Android Toolbar. This component can display a variety of things, including a title, subtitle, log, and navigation icon.

In our example, we will implement ToolbarAndroid with a title, subtitle, and two actions (Options and Menu). When Menu is clicked, we will trigger the openDrawer method that we have available as a prop which will open the menu.

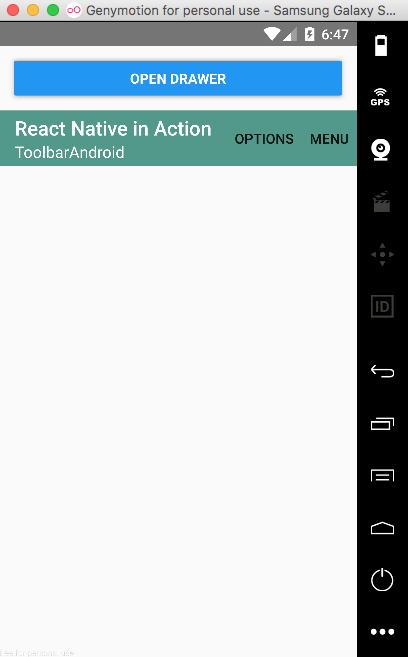


Figure 10.2 ToolbarAndroid with title, subtitle, and two actions. This menu is configurable, but we are only working with the default settings in this example.

In app/Toolbar.js, update our code to the following to implement the actual toolbar.(listing 10.6).

Listing 10.6 app/Toolbar.js – ToolbarAndroid implementation.

import React from 'react'

import {

ToolbarAndroid, // A

View

} from 'react-native'

class Toolbar extends React.Component {

render () {

const onActionSelected = (index) => { // B

if (index === 1) {

this.props.openDrawer()

}

}

return (

<View style={{ flex: 1 }}>

<ToolbarAndroid // C

subtitleColor='white' // a

titleColor='white' // b

style={{ height: 56, backgroundColor: '#52998c' }} // c

title='React Native in Action' //d

subtitle='ToolbarAndroid' // e

actions={[ { title: 'Options', show: 'always' }, { title: 'Menu', show: 'always' } ]} // f

onActionSelected={onActionSelected} // g

/>

</View>

)

}

}

export default Toolbar

1. import the ToolbarAndroid component
2. create an onActionSelected method. This method takes in an index, and will call this.props.openDrawer if the index is one. We will later have an array of actions, each action will call this method when clicked, passing in its own index.
3. return ToolbarAndroid
   1. pass in white as the subtitleColor prop
   2. pass in white as the titleColor prop
   3. give the component a height and backgroundColor
   4. pass in a title prop of ‘React Native in Action’
   5. pass in a subtitle prop of ‘ToolbarAndroid’
   6. pass in an array of actions. When these actions are clicked, they will be called with their index in the array as an argument
   7. pass in onActionSelected as the onActionSelected method

Now, we should not only see the ToolbarAndroid when we refresh our device, but we should also be able to open the DrawerLayoutAndroid menu by clicking on the button labeled menu.

* 1. Implementing scrollable paging with ViewPagerAndroid

Next, let’s create a new example page and component using ViewPagerAndroid. ViewPagerAndroid is a component that easily allows you to swipe left and right between views. Every child of ViewPagerAndroid will be treated as its own separate swipeable view (figure 10.3).

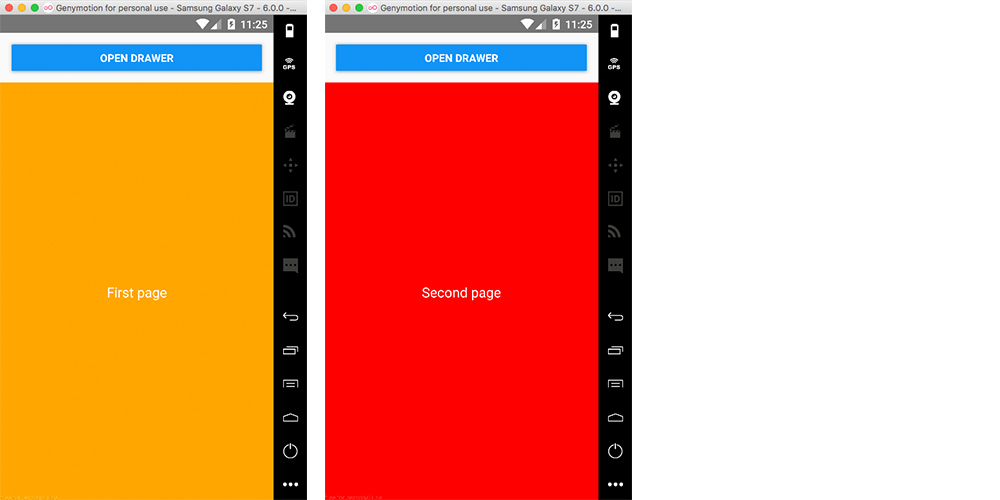


Figure 10.3 ViewPagerAndroid with two child views. When you swipe the pages, they will scroll left and right to show the next page.

In app/ViewPager.js, create the following component which will implement the actual ViePagerAndroid Component. We will discuss all of the functionality after the following listing(listing 10.7).

Listing 10.7 ViewPagerAndroid

import React, { Component } from 'react'

import {

ViewPagerAndroid, // A

View,

Text

} from 'react-native'

let styles

class ViewPager extends Component {

render () {

const {

pageStyle,

page1Style,

page2Style,

textStyle

} = styles

return (

<ViewPagerAndroid // B

style={{ flex: 1 }}

initialPage={0}>

<View style={[ pageStyle, page1Style ]}>

<Text style={textStyle}>First page</Text>

</View>

<View style={[ pageStyle, page2Style ]}>

<Text style={textStyle}>Second page</Text>

</View>

</ViewPagerAndroid>

)

}

}

styles = {

pageStyle: {

justifyContent: 'center',

alignItems: 'center',

padding: 20,

flex: 1,

},

page1Style: {

backgroundColor: 'orange'

},

page2Style: {

backgroundColor: 'red'

},

textStyle: {

fontSize: 18,

color: 'white'

}

}

export default ViewPager

1. import ViewPagerAndroid from React Native
2. return ViewPagerAndroid with two child views, one of them with an orange background and one with a red background

Next we need to update Menu.js to add the button to view the new component. In Menu.js, add this button below the Toolbar Android button.

<View style={button} >

<Button onPress={() => onPress('ViewPager')} title='ViewPager Android' />

</View>

Finally, we need to import the new component and update the switch statement in App.js to render the new component (listing 10.8).

Listing 10.8 App.js with new ViewPager component

import React from 'react'

import Home from './Home'

import Toolbar from './Toolbar'

import ViewPager from './ViewPager'

function getScene (scene) {

switch (scene) {

case 'Home':

return Home

case 'Toolbar':

return Toolbar

case 'ViewPager':

return ViewPager

default:

return Home

}

}

const App = (props) => {

const Scene = getScene(props.scene)

return (

<Scene openDrawer={props.openDrawer} jump={props.jump} />

)

}

export default App

Now, we should be able to run the app and see the new ViewPager Android Button in the side menu, and can view and interact with the new component.

* 1. DatePickerAndroid

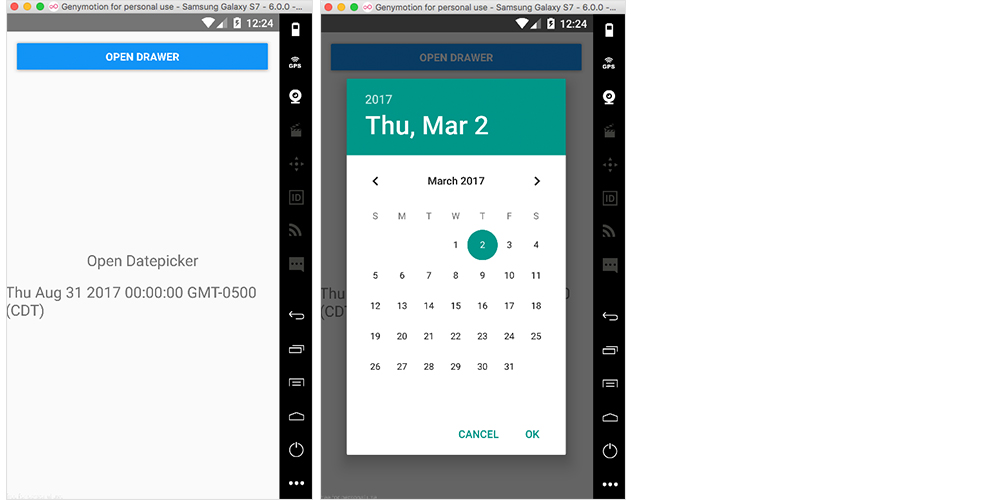
DatePickerAndroid lets us open and interact with the native Android date picker dialog (figure 10.4).

Figure 10.4 DatePickerAndroid with a button that opens the datePicker, and then shows the selected date in the view

To open and use the DatePickerAndroid component, we import DatePickerAndroid and call DatePickerAndroid.open().

To get started, create app/DatePicker.js and create DatePicker.js (listing 10.9).

Listing 10.9 Implementing a DatePicker component - DatePicker.js

import React, { Component } from 'react'

import { DatePickerAndroid, View, Text } from 'react-native' // A

let styles

class DatePicker extends Component {

constructor() {

super()

this.state = { // B

date: new Date()

}

this.openDatePicker = this.openDatePicker.bind(this)

}

openDatePicker () { // C

DatePickerAndroid.open({

date: this.state.date

})

.then((date) => {

const { year, month, day, action } = date

if (action === 'dateSetAction') {

this.setState({ date: new Date(year, month, day) })

}

}) }

render() {

const {

container,

text

} = styles

return (

<View style={container}> // D

<Text onPress={this.openDatePicker} style={text}>Open Datepicker</Text>

<Text style={text}>{this.state.date.toString()}</Text>

</View>

)

}

}

styles = {

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center'

},

text: {

marginBottom: 15,

fontSize: 20

}

}

export default DatePicker

1. import DatePickerAndroid from React Native
2. create the state, setting the date as a new Date()
3. create openDatePicker method, passing in the current date as the date to show on when the datepicker opens. The open method returns a promise, giving us an object with the chosen day, month, year, and the action that was chosen. If you choose a date, then the action is dateSetAction. If the modal is dismissed, then the action is dismissedAction.
4. We create a button that will call the openDatePicker method, and display the date in our View.

Now that we have the component created, let’s update app/App.js to include the new component (listing 10.10).

Listing 10.10 app/App.js with new DatePicker component

import React from 'react'

import Home from './Home'

import Toolbar from './Toolbar'

import ViewPager from './ViewPager'

import DatePicker from './DatePicker'

function getScene (scene) {

switch (scene) {

case 'Home':

return Home

case 'Toolbar':

return Toolbar

case 'ViewPager':

return ViewPager

case 'DatePicker':

return DatePicker

default:

return Home

}

}

const App = (props) => {

const Scene = getScene(props.scene)

return (

<Scene openDrawer={props.openDrawer} jump={props.jump} />

)

}

export default App

Finally, we can update the Menu to add the new button that will open our new DatePicker component. In app/Menu.js, add the following button below the ViewPager Android button.

<View style={button} >

<Button onPress={() => onPress('DatePicker')} title='DatePicker Android' />

</View>

* 1. TimePickerAndroid

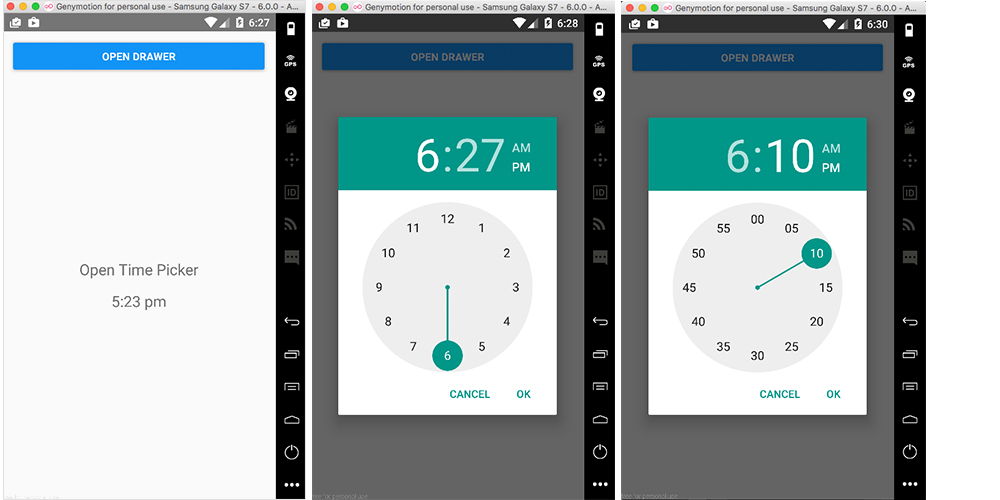
Next up is TimePickerAndroid. TimePickerAndroid is like DatePickerAndroid in that you import it and call the open method to interact with it. This component brings up a TimePicker dialog that allows you to choose a time and use it in your application (figure 10.5).

Figure 10.5 TimePickerAndroid with both hour and minute views

To standardize our time formats, we will be using a library called momentjs. To get started with this library, let’s install moment. In the root directory of the project, install moment using npm or yarn.

npm install moment –save

Next, let’s create the TimePicker component. In app/TimePicker.js, create the following component (figure 10.11).

Figure 10.11 app/TimePicker.js – TimePickerAndroid using moment.js

import React, { Component } from 'react'

import { TimePickerAndroid, View, Text } from 'react-native' // A

import moment from 'moment' // B

let styles

class TimePicker extends Component {

constructor () {

super()

this.state = {

time: moment().format('h:mm a') // C

}

this.openTimePicker = this.openTimePicker.bind(this)

}

openTimePicker () { // D

TimePickerAndroid.open({

time: this.state.time

})

.then((time) => {

const { hour, minute, action } = time

if (action === 'timeSetAction') {

const time = moment().minute(minute).hour(hour).format('h:mm a')

this.setState({ time })

}

})

}

render () {

const {

container,

text

} = styles

return (

<View style={container}> // E

<Text onPress={this.openTimePicker} style={text}>Open Time Picker</Text>

<Text style={text}>{this.state.time.toString()}</Text>

</View>

)

}

}

styles = {

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center'

},

text: {

marginBottom: 15,

fontSize: 20

}

}

export default TimePicker

1. import TimePickerAndroid from React Native
2. import moment from moment
3. create an initial time and store it in the state. We call moment().format('h:mm a') to format the date. The h:mm a that was passed in tells moment that we only want the hour, minute and whether the time is am or pm.
4. create openTimePicker method. Again, like DatePickerAndroid, the open method returns a promise, with a time object that contains hour, minute, and action. We check to see if the action is timeSetAction, and if so we update the state to reflect the new time.
5. create a button in the view to call the openTimePicker method and display the time in the view.

Now that we have the component created, let’s update app/App.js to include the new component (listing 10.12).

Listing 10.12 app/App.js with added TimePicker component

import React from 'react'

import Home from './Home'

import Toolbar from './Toolbar'

import ViewPager from './ViewPager'

import DatePicker from './DatePicker'

import TimePicker from './TimePicker'

function getScene (scene) {

switch (scene) {

case 'Home':

return Home

case 'Toolbar':

return Toolbar

case 'ViewPager':

return ViewPager

case 'DatePicker':

return DatePicker

case 'TimePicker':

return TimePicker

default:

return Home

}

}

const App = (props) => {

const Scene = getScene(props.scene)

return (

<Scene openDrawer={props.openDrawer} jump={props.jump} />

)

}

export default App

Finally, we can update the Menu to add the new button that will open our new TimePicker component. In app/Menu.js, add the following button below the DatePicker Android button.

<View style={button} >

<Button onPress={() => onPress('TimePicker')} title='TimePicker Android' />

</View>

* 1. Implementing Android Toasts using ToastAndroid

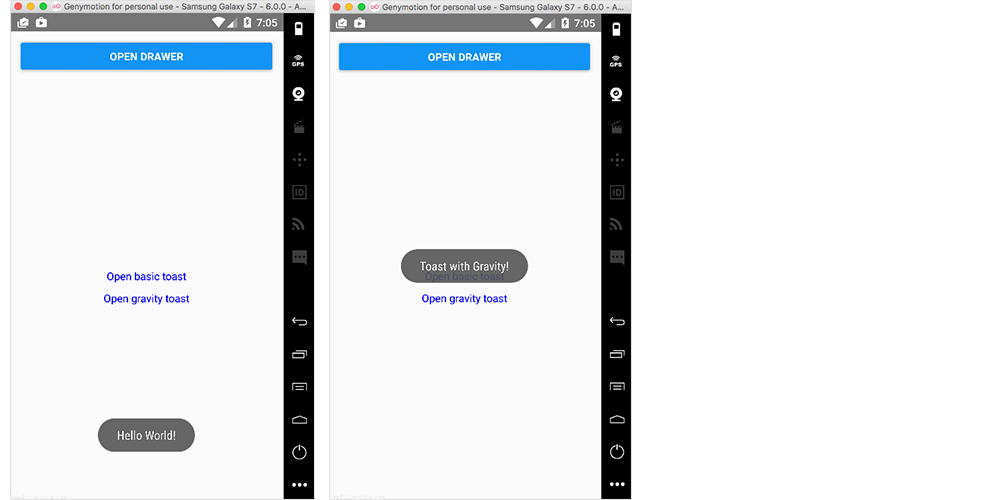
ToastAndroid allows us to easily call native Android toasts from within our React Native application. An android toast is just a popup with a message that goes away after a given period (figure 10.6).

Figure 10.6 ToastAndroid with default and middle positioned toasts.

To get started building out this component, create app/Toast.js, explanation given below the following listing (listing 10.13).

Listing 10.13 app/Toast.js - ToastAndroid

import React from 'react'

import { View, Text, ToastAndroid } from 'react-native' // A

let styles

const Toast = () => {

let {

container,

button

} = styles

const basicToast = () => { // B

ToastAndroid.show('Hello World!', ToastAndroid.LONG)

}

const gravityToast = () => { // C

ToastAndroid.showWithGravity('Toast with Gravity!', ToastAndroid.LONG, ToastAndroid.CENTER)

}

return (

<View style={container}> // D

<Text style={button} onPress={basicToast}>Open basic toast</Text>

<Text style={button} onPress={gravityToast}>Open gravity toast</Text>

</View>

)

}

styles = {

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center'

},

button: {

marginBottom: 10,

color: 'blue'

}

}

export default Toast

1. import ToastAndroid from React Native
2. create a basicToast method that will call ToastAndroid.show(), passing in two arguments: 1. A message and 2. A length of time to show the toast. Can be either SHORT (about 2 seconds) or LONG (about 4 seconds)
3. create gravityToast method that will call ToastAndroid.showWithGravity(). This method is like ToastAndroid.show(), but it allows for a third argument to be passed, allowing us to position the toast either at the top, bottom, or center of the view. We pass in ToastAndroid.CENTER as the third argument, centering the toast in the middle of the screen.
4. create two buttons in the view, attaching our methods to these buttons

Now that we have the component created, let’s update app/App.js to include the new component (listing 10.14).

Listing 10.14 app/Menu.js – Adding Toast component to app

import React from 'react'

import Home from './Home'

import Toolbar from './Toolbar'

import ViewPager from './ViewPager'

import DatePicker from './DatePicker'

import TimePicker from './TimePicker'

import Toast from './Toast'

function getScene (scene) {

switch (scene) {

case 'Home':

return Home

case 'Toolbar':

return Toolbar

case 'ViewPager':

return ViewPager

case 'DatePicker':

return DatePicker

case 'TimePicker':

return TimePicker

case 'Toast':

return Toast

default:

return Home

}

}

const App = (props) => {

const Scene = getScene(props.scene)

return (

<Scene openDrawer={props.openDrawer} jump={props.jump} />

)

}

export default App

Finally, we can update the Menu to add the new button that will open our new Toast component. In app/Menu.js, add the following button below the TimePicker Android button.

<View style={button} >

<Button onPress={() => onPress('Toast')} title='Toast Android' />

</View>

* 1. Summary
* We implemented DrawerLayoutAndroid to create the main menu of the application
* Use ToolbarAndroid to create an interactive app Toolbar
* Use ViewPagerAndroid to create swipeable views
* Use DatePickerAndroid to create and manipulate dates in your application
* Use TimePickerAndroid to create and manipulate time in your application
* Use ToastAndroid to create native android Toast notifications