Version: 6.x

useLinkProps

The `useLinkProps` hook let's build our custom link components which let us navigate to a screen using a path instead of a screen name based on the `linking` options. It takes a path and returns an object with some props that you can pass to a component.

Example:

```
import { useLinkProps } from '@react-navigation/native';

// ...

const LinkButton = ({ to, action, children, ...rest }) => {
  const { onPress, ...props } = useLinkProps({ to, action });

  const [isHovered, setIsHovered] = React.useState(false);

  if (Platform.OS === 'web') {
    // It's important to use a `View` or `Text` on web instead of `TouchableX`
    // Otherwise React Native for Web omits the `onClick` prop that's passed
    // You'll also need to pass `onPress` as `onClick` to the `View`
    // You can add hover effects using `onMouseEnter` and `onMouseLeave`
    return (
      <View
        onClick={onPress}
        onMouseEnter={() => setIsHovered(true)}
        onMouseLeave={() => setIsHovered(false)}
        style={{ transitionDuration: '150ms', opacity: isHovered ? 0.5 : 1 }}
        {...props}
        {...rest}
      >
        <Text>{children}</Text>
      </View>
    );
  }

  return (
    <TouchableOpacity onPress={onPress} {...props} {...rest}>
```

```
      <Text>{children}</Text>
    </TouchableOpacity>
  );
};

function Home() {
  return <LinkButton to={{ screen: 'Profile', params: { id: 'jane' } }}>Go to
Jane's profile</LinkButton>;
}
```

Then you can use the `LinkButton` component elsewhere in your app:

```
function Home() {
  return <LinkButton to={{ screen: 'Profile', params: { id: 'jane' } }}>Go to
Jane's profile</LinkButton>;
}
```

The `props` object returned by `useLinkProps` contains the required props for accessible link components. When we use these props on `View`, `Text` etc., the link component responds to user actions such as `Ctrl+Click`/`⌘+Click` to open links in new tab while keeping regular clicks within the same web page.

There are couple of important things to note when using `useLinkProps` with current version of React Native for Web:

1. You must explicitly pass `onPress` as the `onClick` prop, otherwise in-page navigation won't work
2. You can only use `View` or `Text` with `useLinkProps`. The `TouchableX` components don't support a correct `onClick` event which we need

In a future version of React Native for Web, these won't be an issue and you'll be able to have the same code for links on Web, iOS and Android. But until then, you need to write platform specific code for Web and native.

Options

`to`

You can pass an object with a `screen` property:

```
function Home() {
  return (
    <LinkButton
      to={{ screen: 'Profile', params: { id: 'jane' } }}
    >
      Go to Jane's profile
    </LinkButton>
  );
}
```

The syntax of this object is the same as navigating to a screen in a nested navigators. This uses a `navigate` action for navigation by default, unless you specify a different action.

Alternatively, you can also pass an absolute path to the screen, e.g. - `/profile/jane`.

This will be used for the `href` prop as well as for in-page navigation.

`action`

Sometimes we want a different behavior for in-page navigation, such as `replace` instead of `navigate`. We can use the `action` prop to customize it:

Example:

```
import { StackActions } from '@react-navigation/native';

// ...

function Home() {
  return (
    <LinkButton
      to={{ screen: 'Profile', params: { id: 'jane' } }}
      action={StackActions.replace('Profile', { id: 'jane' })}
    >
      Go to Jane's profile
    </LinkButton>
  );
}
```

If the `action` prop is not specified, the path provided to the `to` prop will be used and dispatched as a `navigate` action.

Edit this page