Version: 6.x

Passing parameters to routes

Remember when I said "more on that later when we talk about `params`!"? Well, the time has come.

Now that we know how to create a stack navigator with some routes and navigate between those routes, let's look at how we can pass data to routes when we navigate to them.

There are two pieces to this:

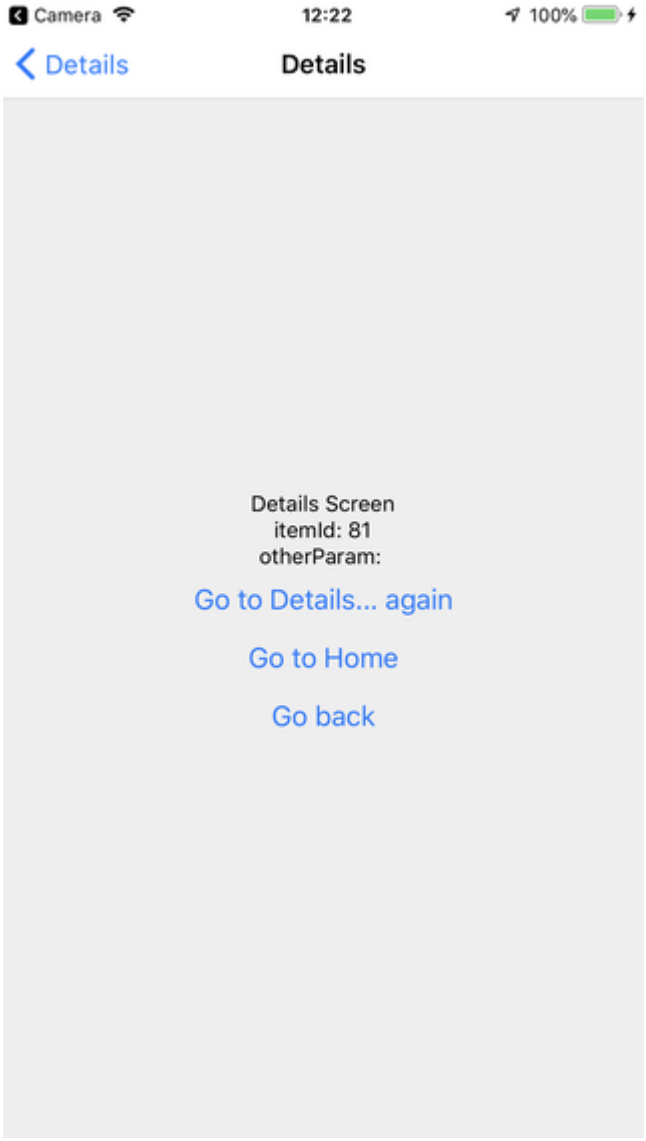
1. Pass params to a route by putting them in an object as a second parameter to the `navigation.navigate` function: `navigation.navigate('RouteName', { /* params go here */ })`
2. Read the params in your screen component: `route.params`.

> We recommend that the params you pass are JSON-serializable. That way, you'll be able to use state persistence and your screen components will have the right contract for implementing deep linking.

```
function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
      <Button
        title="Go to Details"
        onPress={() => {
          /* 1. Navigate to the Details route with params */
          navigation.navigate('Details', {
            itemId: 86,
            otherParam: 'anything you want here',
          });
        }}
      />
    </View>
  );
}
```

```
function DetailsScreen({ route, navigation }) {
  /* 2. Get the param */
  const { itemId, otherParam } = route.params;
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Details Screen</Text>
      <Text>itemId: {JSON.stringify(itemId)}</Text>
      <Text>otherParam: {JSON.stringify(otherParam)}</Text>
      <Button
        title="Go to Details... again"
        onPress={() =>
          navigation.push('Details', {
            itemId: Math.floor(Math.random() * 100),
          })
        }
      />
      <Button title="Go to Home" onPress={() => navigation.navigate('Home')} />
      <Button title="Go back" onPress={() => navigation.goBack()} />
    </View>
  );
}
```

Try this example on Snack



Initial params

You can pass some initial params to a screen. If you didn't specify any params when navigating to this screen, the initial params will be used. They are also shallow merged with any params that you pass. Initial params can be specified with an `initialParams` prop:

```
<Stack.Screen
  name="Details"
  component={DetailsScreen}
  initialParams={{ itemId: 42 }}
/>
```

Try this example on Snack

Updating params

Screens can also update their params, like they can update their state. The `navigation.setParams` method lets you update the params of a screen. Refer to the API reference for `setParams` for more details.

Basic usage:

```
navigation.setParams({
  query: 'someText',
});
```

Try this example on Snack

> Note: Avoid using `setParams` to update screen options such as `title` etc. If you need to update options, use `setOptions` instead.

Passing params to a previous screen

Params aren't only useful for passing some data to a new screen, but they can also be useful to pass data to a previous screen too. For example, let's say you have a screen with a create post button, and the create post button opens a new screen to create a post. After creating the post, you want to pass the data for the post back to previous screen.

To achieve this, you can use the `navigate` method, which acts like `goBack` if the screen already exists. You can pass the `params` with `navigate` to pass the data back:

```
function HomeScreen({ navigation, route }) {
  React.useEffect(() => {
    if (route.params?.post) {
      // Post updated, do something with `route.params.post`
      // For example, send the post to the server
    }
  }, [route.params?.post]);

  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Button
```

```
        title="Create post"
        onPress={() => navigation.navigate('CreatePost')}
      />
      <Text style={{ margin: 10 }}>Post: {route.params?.post}</Text>
    </View>
  );
}

function CreatePostScreen({ navigation, route }) {
  const [postText, setPostText] = React.useState('');

  return (
    <>
      <TextInput
        multiline
        placeholder="What's on your mind?"
        style={{ height: 200, padding: 10, backgroundColor: 'white' }}
        value={postText}
        onChangeText={setPostText}
      />
      <Button
        title="Done"
        onPress={() => {
          // Pass and merge params back to home screen
          navigation.navigate({
            name: 'Home',
            params: { post: postText },
            merge: true,
          });
        }}
      />
    </>
  );
}
```

Try this example on Snack

Here, after you press "Done", the home screen's `route.params` will be updated to reflect the post text that you passed in `navigate`.

Passing params to nested navigators

If you have nested navigators, you need to pass params a bit differently. For example, say you have a navigator inside the `Account` screen, and want to pass params to the `Settings` screen inside that navigator. Then you can pass params as following:

```
navigation.navigate('Account', {
  screen: 'Settings',
  params: { user: 'jane' },
});
```

Try this example on Snack

See Nesting navigators for more details on nesting.

What should be in params

It's important to understand what kind of data should be in params. Params are like options for a screen. They should only contain information to configure what's displayed in the screen. Avoid passing the full data which will be displayed on the screen itself (e.g. pass a user id instead of user object). Also avoid passing data which is used by multiple screens, such data should be in a global store.

You can also think of the route object like a URL. If your screen had a URL, what should be in the URL? Params shouldn't contain data that you think should not be in the URL. This often means that you should keep as little data as possible needed to determine what the screen is. Think of visiting a shopping website, when you are seeing product listings, the URL usually contains category name, type of sort, any filters etc., it doesn't contain the actual list of products displayed on the screen.

For example, say if you have a `Profile` screen. When navigating to it, you might be tempted to pass the user object in the params:

```
// Don't do this
navigation.navigate('Profile', {
  user: {
    id: 'jane',
    firstName: 'Jane',
    lastName: 'Done',
    age: 25,
```

```
  },
});
```

This looks convenient, and lets you access the user objects with `route.params.user` without any extra work.

However, this is an anti-pattern. Data such as user objects should be in your global store instead of the navigation state. Otherwise you have the same data duplicated in multiple places. This can lead to bugs such as the profile screen showing outdated data even if the user object has changed after navigation.

It also becomes problematic to link to the screen via deep linking or on the Web, since:

1. The URL is a representation of the screen, so it also needs to contain the params, i.e. full user object, which can make the URL very long and unreadable
2. Since the user object is in the URL, it's possible to pass a random user object representing a user which doesn't exist, or has incorrect data in the profile
3. If the user object isn't passed, or improperly formatted, this could result in crashes as the screen won't know how to handle it

A better way is to pass only the ID of the user in params:

```
navigation.navigate('Profile', { userId: 'jane' });
```

Now, you can use the passed `userId` to grab the user from your global store. This eliminates a host of issues such as outdated data, or problematic URLs.

Some examples of what should be in params are:

1. IDs like user id, item id etc., e.g. `navigation.navigate('Profile', { userId: 'Jane' })`
2. Params for sorting, filtering data etc. when you have a list of items, e.g. `navigation.navigate('Feeds', { sortBy: 'latest' })`
3. Timestamps, page numbers or cursors for pagination, e.g. `navigation.navigate('Chat', { beforeTime: 1603897152675 })`
4. Data to fill inputs on a screen to compose something, e.g. `navigation.navigate('ComposeTweet', { title: 'Hello world!' })`

In essence, pass the least amount of data required to identify a screen in params, for a lot of cases, this simply means passing the ID of an object instead of passing a full object. Keep your application data separate from the navigation state.

Summary

- `navigate` and `push` accept an optional second argument to let you pass parameters to the route you are navigating to. For example: `navigation.navigate('RouteName', { paramName: 'value' })`.
- You can read the params through `route.params` inside a screen
- You can update the screen's params with `navigation.setParams`
- Initial params can be passed via the `initialParams` prop on `Screen`
- Params should contain the minimal data required to show a screen, nothing more

Edit this page