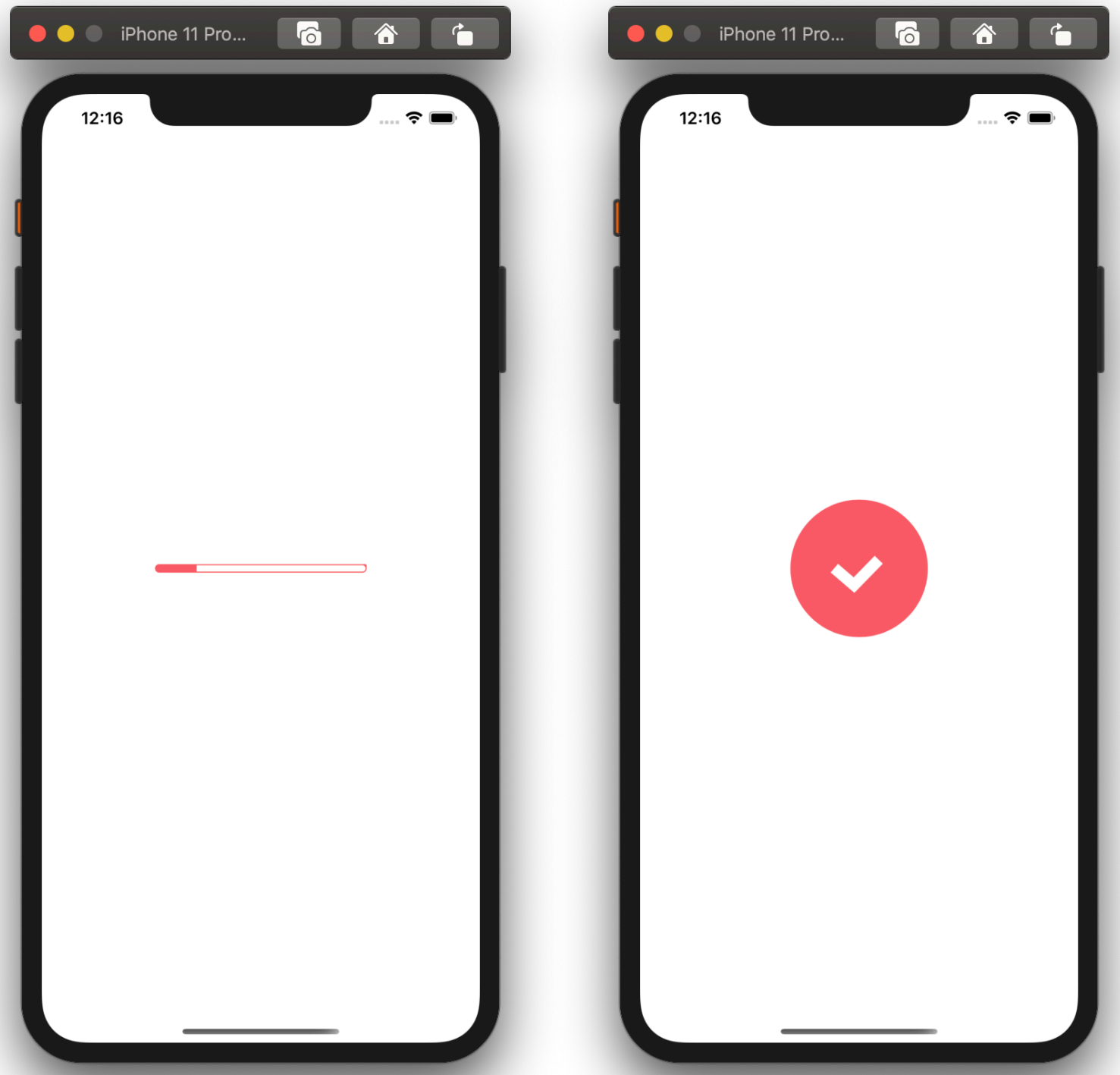
# Networking Exercises

**Exercise time: 1 hour**

## Saving the Listing

When the user submits the listing form, send the listing to the server and show a progress bar. When the upload is complete, show an animation and reset the form. Before getting started, read my notes on the next page.



## About this Exercise

This exercise is a bit more challenging than the previous exercises and I don't expect you to complete it successfully on your own. If you can, bravo! But if not, it's perfectly fine! Don't let that discourage you.

It took me quite some time to solve this problem too. My first solution wasn't that neat. My second solution, which you're going to see, is a lot better. Is it the best solution? I doubt it! If I try solving this problem again in the future, I'll probably come up with a better solution.

So now, I want you to challenge yourself. Read my hints below and implement this new feature on your own. The thought process you go through as part of this is super important to me. The final solution is not.

If the hints below are not sufficient and you really need more help, come back and read the rest of this PDF. I've broken down this problem into a series of small problems with more explanations. Don't read them right from the get-go!

**Hints:**

- To post the listing, use a **FormData** object.

- To track the upload progress, pass an Axios config object when posting the listing with ApiSauce. With this config object, you can track the upload progress using the **onUploadProgress** property.

- To show a progress bar, use **react-native-progress.**

- To show the animation at the end, use **done.json** (included in the supplementary materials of this lesson.)

## Problem 1: Extending the API Layer

Extend the listings API and add the ability to send a listing to the server.

Since we're going to upload images, instead of a plain JavaScript object, we should send a **FormData** object using ApiSauce.

```
const data = new FormData();
data.append('key', value);
…
client.post('/listings', data);
```

If we post a **FormData** object, ApiSauce automatically sets the **content-type** header of the request to **multipart/form-data.** This header tells the server that we're going to send a large request and the body of the request is going to be divided into multiple parts.

Here are the keys that you should include in the **FormData**:

- title
- price
- categoryId
- description
- images (one or more instances)
- location

For each image, you should add a key/value pair to the **FormData**:

```
data.append('images', {
    name: 'unique name',
    type: 'image/jpeg',
    uri: 'uri of the image on the device'
});
```

To include the location, you need to serialize it as a string:

```
data.append('location', JSON.stringify(location));
```

Once the API is ready, test it when submitting the form. If the call to the server fails, show an alert and log the response object to the console so you can identify and solve the problem.

## Problem 2: Tracking the Upload Progress

To track the upload progress, pass an Axios config object as the third argument to **client.post** method:

```
client.post('/listings', data, {});
```

This Axios config object has a property called **onUploadProgress**. You can set it to a function for tracking the upload progress. In this function, just log the progress to the console to make sure you can track the upload progress.

## Problem 3: Notifying the UI Layer

At this point, you can track the upload progress in your API layer, but you need to communicate it back to the UI layer to show a progress bar.

So, your API should raise an event to which the UI layer subscribes. This is the same technique we use in React components. If a child component wants to notify a parent component, it raises an event. The parent handles the event by passing a function to the child:

<Child onChange={handleChange} />

We can use the same technique between our UI and API layers.

## Problem 4: Building the Upload Screen

Now that you can track the upload progress in your UI layer, you should show it in a modal. Show the modal when the user submits the form and hide it when the API call is done. For now, just render the upload percentage inside a Text component.

## Problem 5: Adding a Progress Bar

Now, replace the Text component with a progress bar.

https://github.com/oblador/react-native-progress

## Problem 6: Showing the Done Animation

To show the done animation, use **done.json** (included in the supplementary materials of this lesson).

On upload screen, render a progress bar if progress is less than 1; otherwise, render a **LottieView**.

Because the upload screen disappears when the upload is done, you'll not see the animation. To solve this problem, **UploadScreen** should raise an event (**onDone**) to which **ListingEditScreen** subscribes. You should raise this event when the animation is done. Read about **onAnimationFinish** event of **LottieView**.

## Problem 7: Resetting the Form

Research how to reset a Formik form. Once you find the solution, you'll realize that two of the fields on the form are not reset. Investigate and resolve the issue.