CS47: Cross-Platform Mobile Development

Lecture 3A: User-Interactive Components (TextInput + Lists)

James Landay
Abdallah AbuHashem
Tiffany Manuel
Cisco Vlahakis
Vy Mai

https://cs-47.stanford.edu

cs-47.slack.com

Fall 2019

Live Exercise

To-Do List

1) Run `npm install`
2) Open with Expo

Assignment 2: A Tender Attempt with RN

Due Tuesday, October 8th, at 11:59 PM.

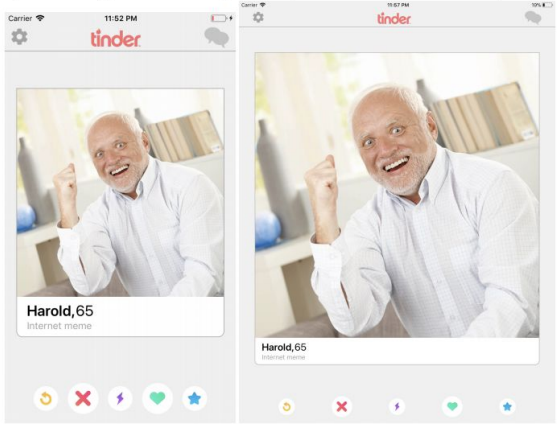
Overview

Imagine working in a team as a mobile developer. You have a team of designers with you. Their task is to design the final version of the application, and your task is to turn the designs into a functional app. Your designers are picky. For the next project, they want you to mimic their specifications exactly. They believe that every pixel matters.

Your job is to take the design, break it up into pieces, and turn it into code. There is no need to do it in one run. We understand that working with React Native will look different than other frameworks / programming languages you might be used to, but we hope that this assignment will help you get familiar with the framework.

Design Details

To get your feet wet, your team of designers have provided these screenshots for you to mimic:



(mobile device interface on the left, and tablet interface on the right)

Due Today, Tuesday
October 8th

Assignment 3: New York Times

Due Thursday, October 17th, at 11:59 PM.

Overview

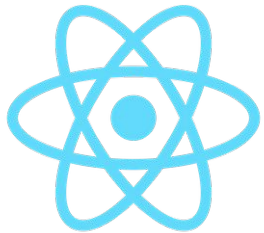
For this assignment you will be building an article browser for New York Times articles. You want something simple that will provide users with a straight-forward browsing experience. You believe that the key to a successful (lightweight) NYT article browser lies on three main principles: users should be able to browse top stories by category, they should be able to search for content across the entire NYT database, and they should be able to see just the relevant snippets of an article at first glance. To accomplish this, you will be tapping into the NYT API and populating articles in a `List`. On top of that you will be building search and category-browsing functionality using a `TextInput` box.

Your final product will look something like this:

Due Thursday, October 17th

Overview for today

- Functional Components (from Thursday's lecture)
- Live demos (throughout lecture)
  - TextInput
  - ScrollView
  - Lists
    - Flatlist
    - SectionList
  - To-Do list Demo
  - Advanced: Infinite Scroll



Functional Component

A `Component` that is declared like a JS function.

Functional Component

A `Component` that is declared like a JS function.

```
const App = () => {
    return (
        <View style={styles.container}>
            <Text>Hello World!</Text>
        </View>
    );
}
```

Functional Component

A `Component` that is declared like a JS function.

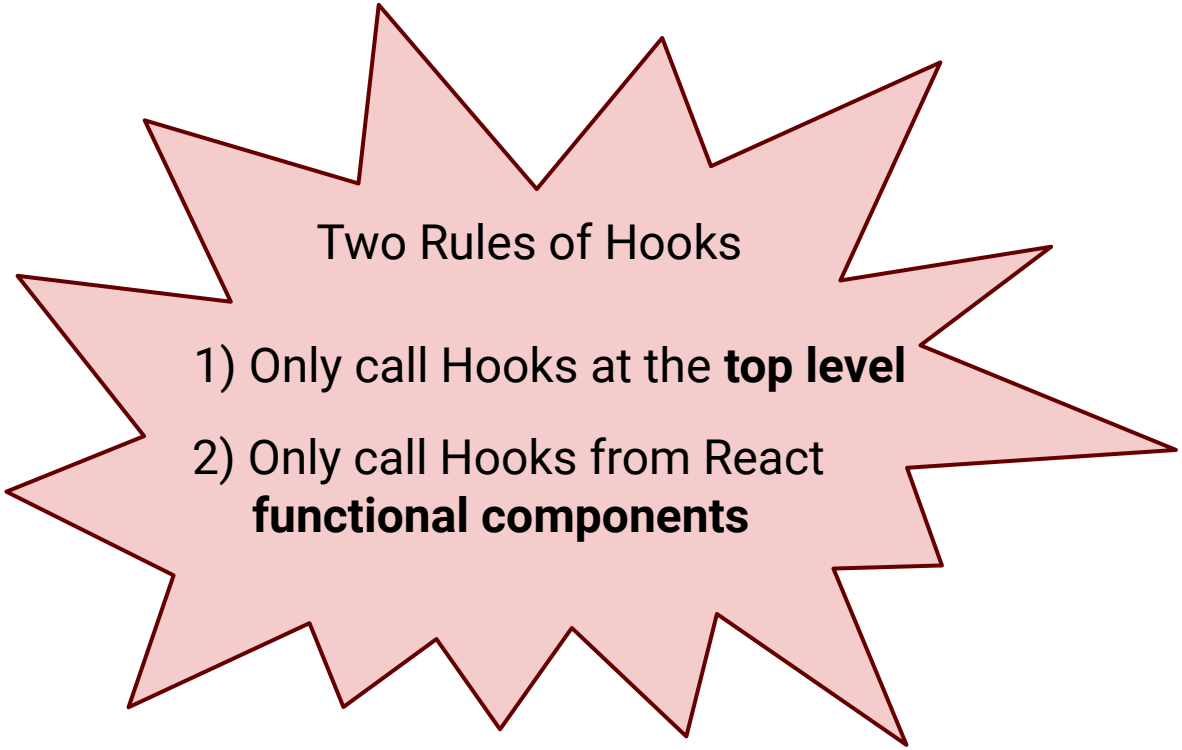
```
const App = (greeting) => {
    return (
        <View style={styles.container}>
            <Text>{greeting}</Text>
        </View>
    );
}
```

Hooks

Functions that allow you to “hook into” React state and life cycle features from function components.

Hooks

Functions that allow you to “hook into” React state and life cycle features from function components.

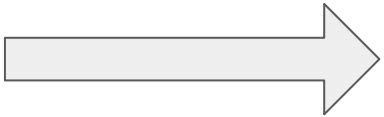


`state` object (Class Components)

`useState` Hook (Functional Components)

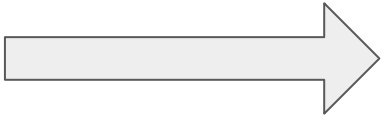
```
import { useState } from 'react';
```

```
state = { x: z }
```



```
const [x, setX] = useState(z)
```

```
this.setState({ x: y })
```



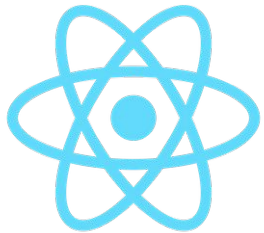
```
setX(y)
```

How would you change the code for our Jedi ID Card to use ***functional components and the `useState` Hook*** (instead of class components and props/state)?

(We'll leave this as a ***highly recommended*** exercise,
but the solution can be found here.)

Overview for today

- Functional Components (from Thursday's lecture)
- Live demos (throughout lecture)
  - TextInput
  - ScrollView
  - Lists
    - Flatlist
    - SectionList
  - To-Do list Demo
  - Advanced: Infinite Scroll



TextInput

- Allows for the intake of text by the user
- Two important props
  - *onChangeText* listens to keystrokes
  - *value* displays the text itself
- Additional props
  - *placeholder* is text shown when nothing is typed
- Snack: https://snack.expo.io/BJNX1__xZ

```
import React, { Component } from 'react';
import { TextInput } from 'react-native';

export default function UselessTextInput() {
  const [value, onChangeText] = React.useState('Useless
Placeholder');

  return (
    <TextInput
      style={{ height: 40, borderColor: 'gray',
borderWidth: 1 }}
      onChangeText={text => onChangeText(text)}
      value={value}
    />
  );
}
```

ScrollView

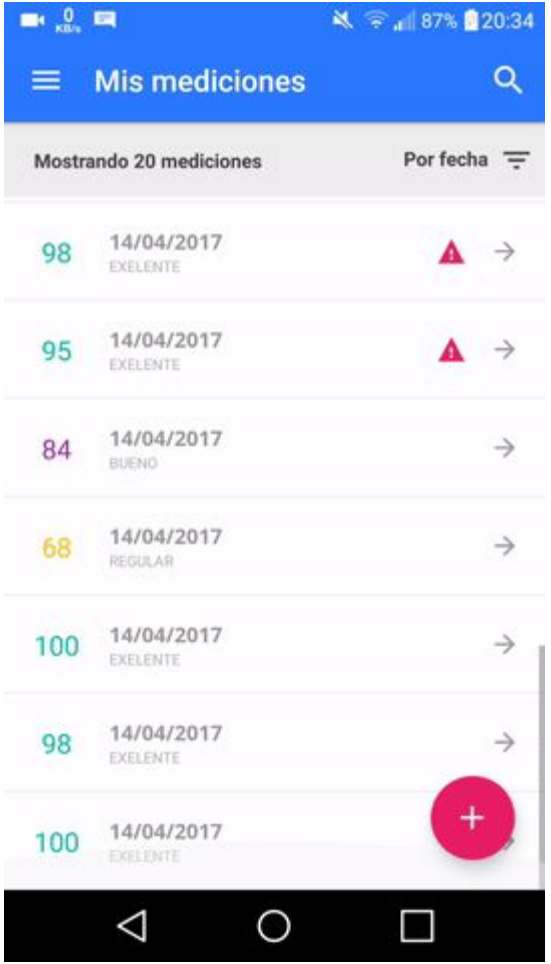
- Basic View that allows for scrolling
- Renders all components at once
  - Slower performance and larger memory usage
  - Must bind height to render correctly
- ScrollView “works best to present a small amount of things of a limited size”.
- Snack: https://snack.expo.io/SySerKNp-

Lists: FlatList and SectionList

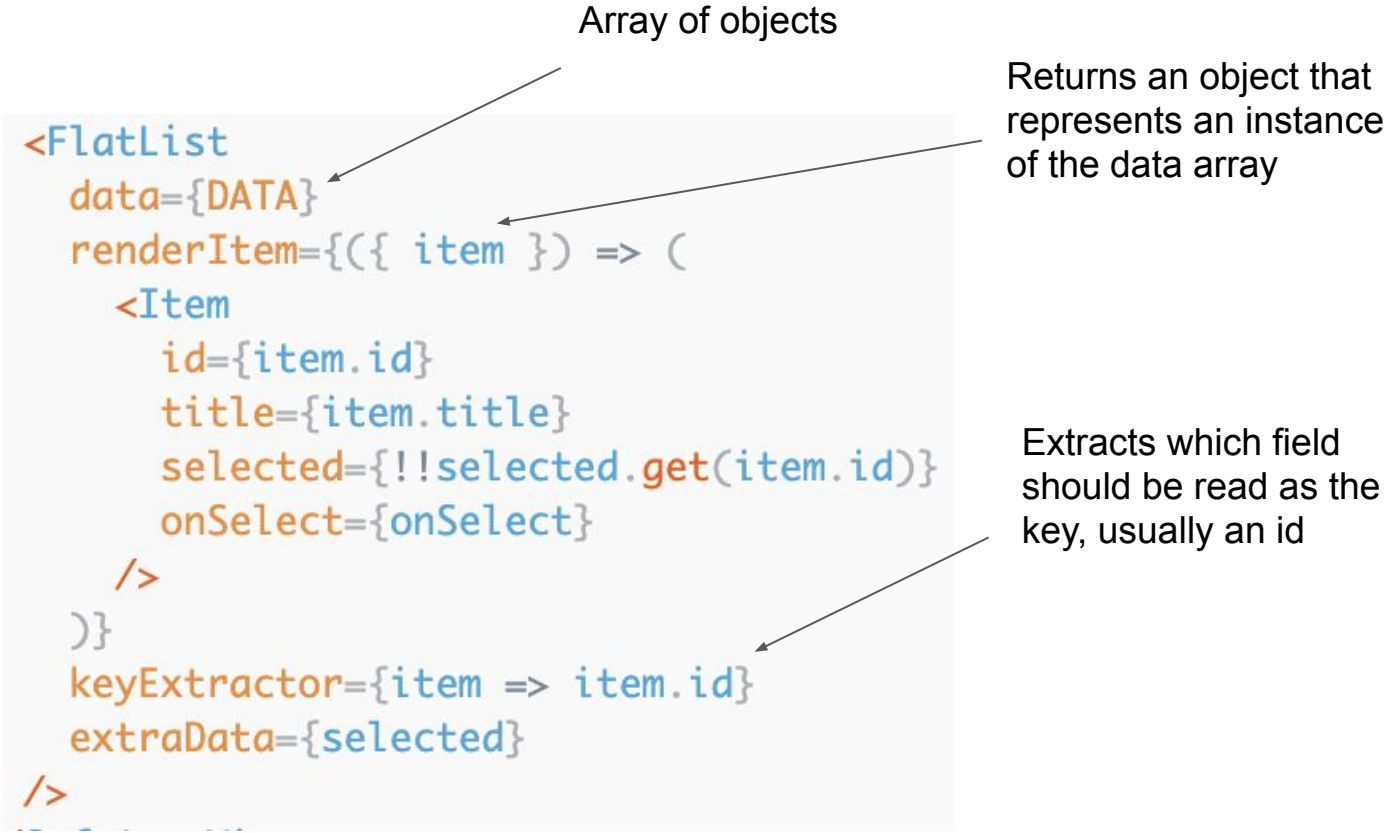
- Fully Cross-Platform
- Header/Footer Support
- Pull to Refresh
- Horizontal Scrolling
- Column Support
- https://facebook.github.io/react-native/blog/2017/03/13/better-list-views

FlatList

- Allows for the display of a list with scrolling built in
- Three important props
  - *data* accepts JSON
  - *renderItem* returns the particular piece of the Flatlist
    - This is where you specify the UI layout of each JSON object
  - *keyExtractor* returns a unique key for every item
    - Common cause of YellowBox warnings is to incorrectly assign keys
- Snack: https://snack.expo.io/@bacon/flatlist

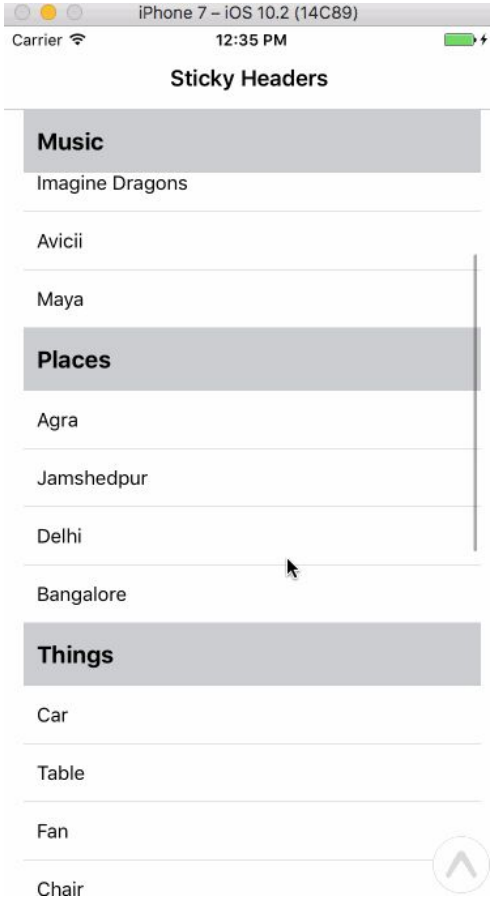


FlatList


Array of objects
Returns an object that represents an instance of the data array
Extracts which field should be read as the key, usually an id
<FlatList
data={DATA}
renderItem={({ item }) => (
<Item
id={item.id}
title={item.title}
selected={!!selected.get(item.id)}
onSelect={onSelect}
/>
)}
keyExtractor={item => item.id}
extraData={selected}
/>

SectionList

- Allows for the display of a list + scrolling + header and separators
- Four important props
  - *renderItem* lets you set the layout of a single item
  - *renderSectionHeader* lets you set the layout of the section headers
  - *sections* is a JSON that maps each section to the data underneath it
  - *keyExtractor* returns a unique key for every item
- Snack: https://snack.expo.io/@bakabon86/section-list



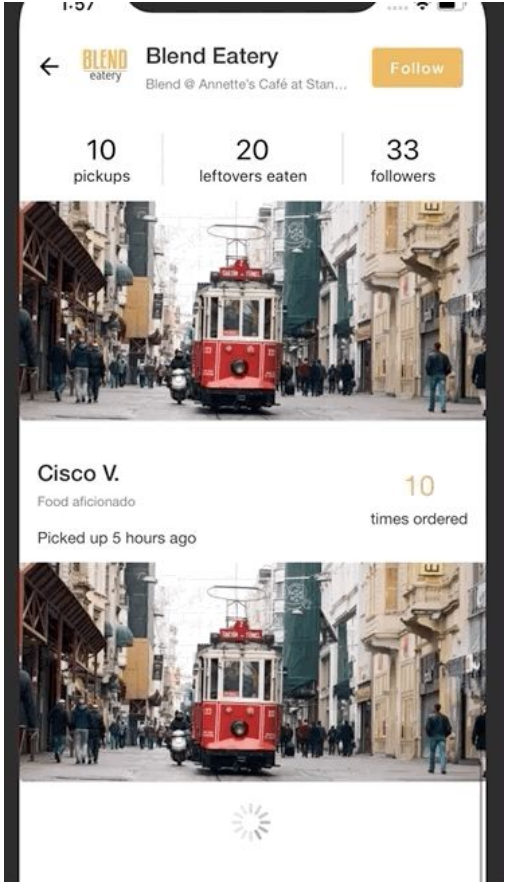
SectionList

Returns header
of a section

```
<SectionList
  sections={DATA}
  keyExtractor={(item, index) => item + index}
  renderItem={({ item }) => <Item title={item} />}
  renderSectionHeader={({ section: { title } }) => (
    <Text style={styles.header}>{title}</Text>
  )}
/>
```

Infinite Scroll

- What do you do when you need to render many items fetched from a database or endpoint?
- If a database stores 10,000 images and they are to be displayed as a feed, it is unwise to fetch all 10,000 images at once
- Instead, fetch a few items at a time, then load more when the user has reached the end of the List
- This concept is called Infinite Scroll
- FlatList and SectionList have a prop called *onEndReached* that accepts a callback.
    - This is where you should write the logic to load more items



Live Demo

SectionList + Infinite Scroll

Let’s look at an application that has a list with:

- Customized items
- Sticky headers
- Infinite Scrolling
- Pull down to refresh
- Fetches info from a server
- Handles Concurrent Calls

Live Demo

SectionList + Infinite Scroll

STARTER CODE

1) Run `npm install`
2) Open with Expo

Live Exercise

To-Do List

Create an application that appends notes to a to-do list.

Live Exercise

To-Do List

1) Run `npm install`
2) Open with Expo

Assignment 3: New York Times

Due Thursday, October 17th, at 11:59 PM.

Overview

For this assignment you will be building an article browser for New York Times articles. You want something simple that will provide users with a straight-forward browsing experience. You believe that the key to a successful (lightweight) NYT article browser lies on three main principles: users should be able to browse top stories by category, they should be able to search for content across the entire NYT database, and they should be able to see just the relevant snippets of an article at first glance. To accomplish this, you will be tapping into the NYT API and populating articles in a `List`. On top of that you will be building search and category-browsing functionality using a `TextInput` box.

Your final product will look something like this:

Due Thursday, October 17th

CS47: Cross-Platform Mobile Development

Lecture 3A: User-Interactive Components (TextInput + Lists)

James Landay
Abdallah AbuHashem
Tiffany Manuel
Cisco Vlahakis
Vy Mai

https://cs-47.stanford.edu

cs-47.slack.com

Fall 2019