Intro to React (native)
React is a JavaScript library for user interfaces, while React Native is a framework you can use to create native mobile apps using React.
Open-source library that is used for building hybrid mobile applications
Made by Facebook
Released at 2015
Based on React
(Almost) most popular cross platform library
FB ads MVP got too messy, multidirectional data flow caused poor performance

First prototype FaxJS in 2011, React framework in FB stack in 2012
After acquiring Instagram, React was decoupled from FB stack and open-sourced in 2013

React released 2013 by Facebook
2012 FB takes mobile first approach

WebView poor functionality and performance in native container
Native development is expensive, time consuming, required too much support

After a small team managed to generate native UI elements with JS, 48h hackathon was made in summer of 2013, result was initial prototype

Prototyped was used to develop RN which was published at React.js conference in January 2015
- **Declarative**
- **Component-Based**
- **Virtual DOM**
Every declarative approach is built with imperative APIs.

For example, React Native uses the same APIs to render your application on iOS or Android as native developers would use themselves. In React, underlying APIs are considered implementation details, and are not exposed to the developer.

```javascript
function App() {
  return (
    <View>
      <Text>Hello World</Text>
    </View>
  );
}

// UIManager.createView([2,"RCTRawText", 61, {text: "Hello World"}])
Am I going to drive an *imperative* stick shift car or a *declarative* automatic car?”
Java implementation of UIManager - a native module which allows JS to create and update native Views
React popularized a concept known as component-based architecture.

Components let you split the larger user interface into smaller independent, reusable pieces, and think about each piece in isolation.

```javascript
export function Screen() {
  return [
    <Column>
      <BlueBox />
      <YellowBox />
      <GreenBox />
    </Column>
  ];
}
```
const Greeting = (props) => {
  return (
    <View style={{styles.center}}>
      <Text>Hello {props.name}!</Text>
    </View>
  );
}

const LotsOfGreetings = () => {
  return (
    <View style={{styles.center}}>
      <Greeting name='Rexxar' />
      <Greeting name='Jaina' />
      <Greeting name='Valeera' />
    </View>
  );
}
At the end of the component hierarchy, there are so-called primitive components, such as Text, View or TextInput. These components are implemented by React and provided by the platform you are targeting to support most basic user interactions.

```javascript
const blueBoxStyle = {
  backgroundColor: 'blue',
  height: '33%',
  width: '100%',
};

function BlueBox() {
  return <View style={blueBoxStyle} />;
}
```
React Native comes with a set of essential, ready-to-use primitive components you can use to start building your app.

<table>
<thead>
<tr>
<th>REACT NATIVE UI COMPONENT</th>
<th>ANDROID VIEW</th>
<th>IOS VIEW</th>
<th>WEB ANALOG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;View&gt;</td>
<td>&lt;ViewGroup&gt;</td>
<td>&lt;UIView&gt;</td>
<td>A non-scrolling &lt;div&gt;</td>
<td>A container that supports layout with flexbox, style, some touch handling, and accessibility controls</td>
</tr>
<tr>
<td>&lt;Text&gt;</td>
<td>&lt;TextView&gt;</td>
<td>&lt;UITextView&gt;</td>
<td>&lt;p&gt;</td>
<td>Displays, styles, and nests strings of text and even handles touch events</td>
</tr>
<tr>
<td>&lt;Image&gt;</td>
<td>&lt;ImageView&gt;</td>
<td>&lt;UIImageView&gt;</td>
<td>&lt;img&gt;</td>
<td>Displays different types of images</td>
</tr>
<tr>
<td>&lt;ScrollView&gt;</td>
<td>&lt;ScrollView&gt;</td>
<td>&lt;UIScrollView&gt;</td>
<td>&lt;div&gt;</td>
<td>A generic scrolling container that can contain multiple components and views</td>
</tr>
<tr>
<td>&lt;TextInput&gt;</td>
<td>&lt;EditText&gt;</td>
<td>&lt;UITextField&gt;</td>
<td>&lt;input type=&quot;text&quot;&gt;</td>
<td>Allows the user to enter text</td>
</tr>
</tbody>
</table>
React components are typically written using JSX

JSX (JavaScript Syntax Extension) is an extension to the JavaScript language syntax. It is similar in appearance to HTML.

Although JSX is the most convenient way of using React, it’s not a requirement. Each JSX element is just syntactic sugar for calling React.createElement, so, anything you can do with JSX can also be done with just plain JavaScript.

```javascript
function Hello(props) {
    return <div>Hello {props.toWhat}</div>;
}

function Hello(props) {
    return React.createElement('div', null, 'Hello ${props.toWhat}');
}
```
Props are arguments passed into React components.

Conceptually, components are literally JavaScript functions, and as JavaScript functions, components can accept arguments. In React function arguments are typically called “props” which stands for properties.

React is focused more on functional programming.

```jsx
<Hello toWhat="World" />
```

```javascript
// React.createElement(Hello, {toWhat: 'World'}, null)
```
Props are used to pass data between React components and allow to write more reusable components.

```jsx
function App() {
  return (
    <View style={style}>
      <Hello toWhat="World" />
      <Hello toWhat="Mooncascade" />
    </View>
  )
}

function Hello(props) {
  return <Text>Hello {props.toWhat}!</Text>;
}
```
Imperative approach (jQuery)

- no rules, multi-directional data flow
- When to rerender?
  - you need to control it yourself
- Which part of the UI re-render?
  - you need to control it yourself
- With great power, comes great responsibility
  - Usually a real mess
Declarative approach (React)

- you don’t have to worry about exactly what changes on every update
- 1-way data flow, UI = fn(state)
- When to rerender?
  - new state → new UI
  - composition model: If a parent re-renders, children are also re-rendered
- Which part of the UI re-render?
  - In worst case scenario - the part when the state has changed and everything below (can be less when optimized)
  - Never upward or sideways
The virtual DOM is “virtual”, representation of a UI which is kept in memory. You can think of it as descriptions of what you want to see on the screen.

React creates a virtual DOM by recursively calling components function.

```javascript
// Note: this structure is simplified
...
{
    type: 'h1',
    props: {
        className: 'greeting',
        children: 'Hello, world!'  
    }
}
...
```
Reconciliation - updating UI

Change of the component state → A new Virtual DOM object is created. Then the “diffing” algorithm traverse both old and new versions of virtual DOM object looking for differences → When the list of all those changes is ready, React will go ahead and surgically apply those changes to the actual DOM

VIRTUAL DOM

DIFFERENCE

PATCH

DOM

COMMIT PHASE

RENDER PHASE
Optimizing re-renders

- General idea is to change variables, references as little as necessary.
- Also avoid micro-optimizations.
- Object and arrays are compared by reference.
- In Object or array cases it is useful to use `useMemo` hooks (React built-in function) to compute out a value that is used to render so it does not rerender when not needed.
- For Redux use Reselect library to create selectors that only update when computed value changes.
- Avoid single huge components. Splitting to smaller components helps.
Challenges of React

- Very loosely structured - up to developer to design file system
- No strict code rules out of the box
  - Everything can be everywhere - 1 file with 1000000 lines or 1000000 files with 1 line
  - Additional code style libraries are available such as E(T)Slint, Prettier etc.
- A lot of third party dependencies are available
  - How to distinguish between good and bad?
- Completely different code styles
Application state management
Application state management

**Built-in ways:**

- Class component state
- Function component hooks
- Context Class provides properties

**Popular libraries:**

- Redux (+ Redux-Saga, Reselect, etc) - data store library
- MobX - observable pattern library
Redux is a predictable state container for JavaScript apps. It is a pattern and library for managing and updating application state, using events called "actions".
MobX

State management library that uses observable pattern. It allows you to manage your application state outside of any UI framework.

- **Events**: Invoked actions. Actions are the only thing that modify state and may have other side effects.
- **State**: Is observable and minimally defined. Should not contain redundant or derivable data. Can be a graph, contain classes, arrays, refs, etc.
- **Computed values**: Are values that can be derived from the state using a pure function. Will be updated automatically by MobX and optimized away if not in use.
- **Reactions**: Are like computed values and react to state changes. But they produce a side effect instead of a value, like updating the UI.

```javascript
@action onClick = () => {
    this.props.todo.done = true;
}

@observable todos = [[
    title: "learn MobX",
    done: false
]]

@computed get completedTodos() {
    return this.todos.filter(
        todo => todo.done
    )
}

const Todos = observer({
    todos: () => {
        <ul>
            {todos.map(todo => <TodoView ... />)
        </ul>
    }
})
```
How RN works?

- Business and app logic written in JS
- UI is native
- JS - Native bridge

React Native APP

Native Modules
- Android - Java
- iOS - Obj C / Swift

RN Bridge
(Java/C++)

JS Virtual Machine
(JavaScriptCore)
The NativeModule system exposes instances of native classes to JS as JS objects, thereby allowing you to execute arbitrary native code from within JS.

- connect to native API-s
- integrate native SDK-s
With React Native, you style your application using JavaScript by setting a `style` prop to the component.

- inline
- style sheet
- tools
const FixedDimensionsBasics = () => {
  return (
    <View>
      <View style={{
        width: 50, height: 50, backgroundColor: 'powderblue'
      }} />
      <View style={{
        width: 100, height: 100, backgroundColor: 'skyblue'
      }} />
      <View style={{
        width: 150, height: 150, backgroundColor: 'steelblue'
      }} />
    </View>
  );
};
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';

const LotsOfStyles = () => {
  return (  
    <View style={{styles.container}}>
      <Text style={{styles.red}}>just red</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    marginTop: 50,
  },
  red: {
    color: 'red',
  },
});

export default LotsOfStyles;
export default class App extends React.Component {
    render() {
        return (
            <Container>
                <Title>React Native with 🛠 Styled Components</Title>
            </Container>
        );
    }
}

const Container = styled.View`
    flex: 1;
    background-color: papayawhip;
    justify-content: center;
    align-items: center;
`;

const Title = styled.Text`
    font-size: 24px;
    font-weight: 500;
    color: palevioletred;
`;
UI libraries helps to focus on features, rather than basic styling and functionalities

- Ready to use UI solutions
- Cross platform consistency
- Out of the Box Accessibility

NativeBase

paper

react-native-elements
• Android and IOS applications are built separately as you would build native application
• Both applications must be deployed separately
  ○ Separate pipelines for automation
  ○ Builds are submitted according to Google play, App store rules
• Android is simpler, IOS ecosystem more demanding
• For testing purposes - Google Firebase
Pros of React Native

- Elegant declarative ui
- Component system
  - Works really well with design system
- Very concise code (func. components, jsx, styles)
- Hot reload (when it works)
- Two apps for the effort of 1.5 ???
Cons of React Native

- Fragility, Update hell:
  - native + react + react native
  - state management
  - 3rd party libraries (often ow quality, not maintained)
- Performance issues Bridge limitations
- Inconsistencies between Android and iOS
- Obscure crash logs
React Native New Architecture

- JSI (Interoperability between threads)
- Fabric (JS and NativeUI thread in sync)
- Turbo Modules (load modules on demand)
Where RN is used?
Alternatives of React Native

- Native platforms
- Flutter
- Xamarin
- NativeScript
- Ionic
- Cordova
Useful links

React native setup + guidelines:  
https://reactnative.dev/docs

Thinking in React:  
https://reactjs.org/docs/thinking-in-react.html

Online sandbox:  
https://snack.expo.dev/