



Занятие №1

# Введение в React Native





Что такое React Native и для чего он  
нужен?

# React Native

**React Native** - расширение библиотеки React для создания мобильных приложений. Имеет общую структуру с React, схожие методы работы. Основная идея заключается в том, чтобы вы создавали свой проект один раз и переносили его на все платформы:

Веб-приложение - **React**

Десктопное приложение - **React + Electron**

Мобильное приложение - **React Native**



Что сделано на React Native?

# Продукты React Native



**Bloomberg**



**Walmart** 



# Преимущества React Native

- Приложение кроссплатформенное. Вы сразу пишете одно готовое приложение под Android и под IOS.
- Код сразу компилируется в реальном времени. В отличие от Java или Swift, где нужно сначала скомпилировать проект, чтобы увидеть результат.
- Код имеет схожий синтаксис с обычным React.js
- Проект гораздо проще написать, так как синтаксис React Native гораздо проще чем Java или Swift.
- Позволяет для приложений делать уведомления.
- Доступ в офлайн режиме и на главном экране телефона.

# Принцип работы

Вы пишете код на JS с использованием React, который затем компилируется в исполняемый код, запускающийся на мобильных устройствах под управлением iOS и Android. То есть нам не нужно учить дополнительный язык Java, Swift или Flutter для разработки мобильных приложений.

Мы можем использовать уже знакомый синтаксис. React Native наследует ту же самую структуру, что и React, компонентный подход, использование props, JSX, имеет схожие (но не одинаковые) хуки жизненного цикла.



# Подготовка приложений

Для работы нам потребуется:

- Установить Node.js. Желательно иметь 16 версию и выше.
- Редактор VS Code или WebStorm
- Плагины для VS Code: React Native Tools, React-Native/React/Redux snippets

Для создания чистого проекта можно использовать [react-native-cli](#) или дополнительного фреймворка [Expo](#). В первом варианте для того чтобы посмотреть готовый результат в виде приложения нам потребуется приложения Android Studio (для Android) или Xcode (только на Mac). На Expo проект разрабатывать немного легче. Можно открывать приложение в браузере или через мобильное приложение Expo Go.



# Установка EXPO-CLI

Сначала нужно глобально установить EXPO: `npm install -g expo-cli`

Для создания базовой версии проекта есть два варианта:

- `expo init name-project` // создание проекта с выбранной папкой
- `npx create-expo-app --template` // для создания базового варианта

Выбрать пустой шаблон (blank)

- `cd name-project` // перейти в папку проекта
- `npm start` // запустить команды для проекта.

После этого нужно выбрать работу в Web версии: `npm run web`.

Но сразу не получится запустить, так как для Web версии нужно установить свои пакеты:

`npx expo install react-native-web react-dom @expo/webpack-config`

# Компоненты React Native

Компоненты в React Native делятся на две группы:  
**Native Components** и **Core Components**.

**Core Components** - это те, компоненты, которые есть в языках разметки для приложений iOS и Android.

**Native Components** - это дополнительные компоненты, которые вы можете создавать самостоятельно.

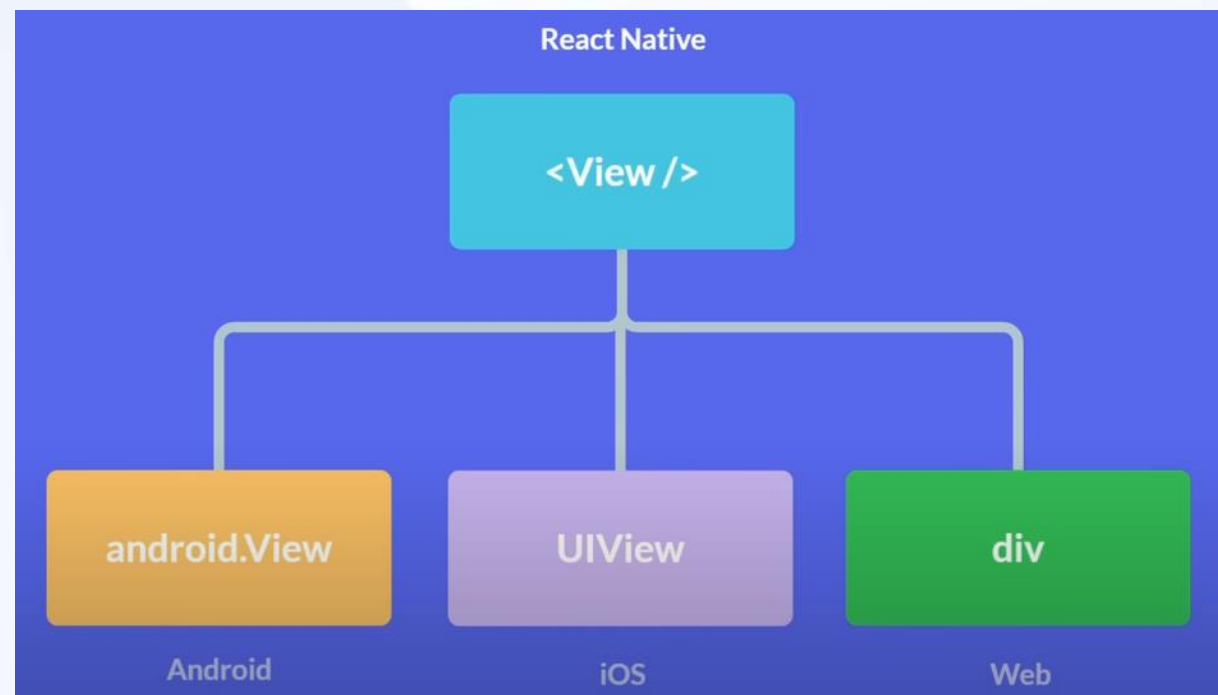
# Базовые компоненты

**View** - основной блок для группировки других компонентов. Аналог тега `div`

**Text** - блок для отображения текста.  
Аналог тега `p`, `span`

Импорт базовых компонентов выполняется из библиотеки `react-native`

```
import {Text, View} from "react-native";
```






Если ты знаешь React, то уже на 50%  
знаешь React Native

# React и React Native


## ReactJS



```
function App(){  
  return (  
    <div>  
      <p>Archakov Blog</Text>  
    </div>  
  )  
}
```



## React Native



```
function App(){  
  return (  
    <View>  
      <Text>Archakov Blog</Text>  
    </View>  
  )  
}
```

# Функция StyleSheet

Для создания таблицы стилей используется функция `StyleSheet.create`

Данная функция должна принимать объект с описаниями стилей. Обычно такие стили расписываются в разные объекты, чтобы их можно было назначать на разные компоненты.

Такой принцип очень похож на то, как мы использовать CSS Модули. Но можно стили также писать напрямую к элементу с помощью атрибута `style` и двойных фигурных скобках. Стили в React Native схожи с тем, что используют в CSS, кроме того, что они записываются в `camelCase`.

```
<View style={{backgroundColor: 'red'}}></View>
```

# Стили в React Native и CSS

ВАЖНО! Не все стили схожи с CSS, поэтому не всё привычное вам будет работать

Для размеров вы можете задавать только пиксели, без приписки px, или проценты (тогда значение будет строковым, то есть с кавычками)

```
<View style={{height: '50%', width: 200}}></View>
```

# Flex в React Native

По умолчанию все элементы в React Native имеют свойство `display flex`. И поменять его нельзя. Это свойство даже не работает. Но работает он схожим образом, как и в CSS, за исключением некоторых моментов:

- `flexDirection` по умолчанию имеет значение `column`
- `flexShrink` по умолчанию имеет значение 0
- `flex` принимает только число и по сути означает, насколько много пространства будет занимать компонент (`flex-grow`)



# Изображения в React Native

Для отрисовки изображений в React Native используется тег `Image`.

Но он работает не так, как тег `img`. В `Image` можно подключить картинку напрямую:

```
import logo from './react.png'
```

```
<Image source={logo} style={{flex: 1}}/>
```

Либо можно указать ссылку на картинку, но тогда `source` должен быть объектом со свойством `uri`

```
<Image source={{uri: 'https://reactnative.dev/docs/assets/p_cat2.png'}} style={{flex: 1}}/>
```

# События в React Native

Так как мы работаем с другим окружением, события тоже поменялись в React Native. Для работы с нажатиями мы будем использовать компонент Button, а само нажатие обрабатывается событием `onPress`

ВНИМАНИЕ! Текст в кнопке записывается в props `title`

```
<Button title="Press" />
```

# Поле для ввода текста в React Native

В React Native вместо обычного `input` используется `TextInput`. Считать данные из `TextInput` можно при помощи события `onChangeText`. Функция-обработчик данного события принимает в качестве параметра исходный текст. Для работы с данными мы будем пользоваться хуком `useState`.

```
let [text, setText] = useState("")
```

```
<TextInput defaultValue="" onChangeText={(text)=>setText(text)} />
```

```
<Text>{text}</Text>
```



Давайте подведем итоги!  
Чему мы научились?  
Что мы использовали?