

# İçindekiler

Introduction	1.1
Yararlandığım Kaynaklar	1.2
Babel, ES2015, ES6, ES7 Nedir ?	1.3
ES6 ve ES7 Örnekler	1.3.1
Fonksiyonel Programlama	1.4
Composition	1.4.1
Composition ile React Native Örneği(Yazılacak)	1.4.2
React JS Nedir ?	1.5
JSX Nedir ?	1.5.1
Lifecycle Methodlar	1.5.2
Props Mantığı	1.5.3
State Mantığı	1.5.4
React Native Nedir ?	1.6
Installation	1.7
expo	1.7.1
Windows Installation	1.7.2
macOS Installation	1.7.3
Gnu/Linux Installation	1.7.4
React Native Başlangıç	1.8
Style ve FlexBox	1.9
Temel Component'ler	1.10
View	1.10.1
Text	1.10.2
Image	1.10.3
ScrollView	1.10.4
ListView	1.10.5
FlatList	1.10.6
Component API'lar	1.11
Animasyon	1.11.1
LayoutAnimation	1.11.1.1

Animated	1.11.1.2
Animated.Event( )	1.11.1.2.1
PanResponder	1.11.2
Navigation	1.12
wix/react-native-navigation	1.12.1
IOS kurulumu	1.12.1.1
Android Kurulumu	1.12.1.2
Temel Kullanım	1.12.1.3
Icon Ekleme-TabBar	1.12.1.4
react-navigation	1.12.2
İç İçe Navigator Kullanımı	1.12.2.1
Header Seçenekleri( Yazılacak )	1.12.2.2
State Yönetimi	1.13
mobx	1.13.1
Redux(Yazılacak)	1.13.2
Yılan Oyunu Tutorial	1.14
Yılan Oyunu Part 1	1.14.1
Yılan Oyunu Part 2	1.14.2
Yılan Oyunu Part 3	1.14.3
Android Apk Oluşturma	1.15

# React Native Türkçe

Hızlı, cross-platform, mobile uygulama yazmak için React Native

Eğer web projesi geliştirmek anlamında tecrübeiniz varsa, bu çok aşina olduğunuz geliştirme ortamıyla mobile uygulamalar geliştirebilmek React Native ile çok kolay.

Android ve IOS uygulamaları (yavaş yavaş windows telefonlar için de geliyor) %90 aynı kodlarla yazabilirsiniz.

Ve unutmayın, React Native gerçek anlamda bir mobil uygulama yazma imkânı sunuyor; bu ne demek derseniz, isterseniz Android veya IOS telefonunuzun ya da tabletinizin donanımsal özelliklerini mobile uygulamanızda kullanabilirsiniz.

Bu dökümantasyonu hem kendi öğrendiklerimi daha da pekiştirmek, hem de başka yazılımcı ya da yazılımcı adayı arkadaşlara faydası olabilir diye tutmayı, güncellemeyi amaçlıyorum. Her türlü görüş, öneri ve eleştirilerinizi lütfen esirgemeyin.

Katkıda bulunmak, ben de şu konuyu anlatmak istiyorum derseniz şuraya mail atabilirsiniz [ysfzrn@gmail.com](mailto:ysfzrn@gmail.com)

veya

şu repo'ya <https://github.com/ysfzrn/react-native-turkce> pull request gönderebilirisiniz.

## Emeği geçenler

- Yusuf Zeren
- Haydar Şahin ( Gnu/Linux Kurulumu )
- pleycpl ( Kod, kelime hataları düzeltildi )

# React Native Öğrenirken Yararlandığım Kaynaklar

- [React Native Resmi Dökümantasyonu](#)
- [React Native Express](#)
- [StackOverFlow](#)
- [JS Coach](#)
- [React Discuss](#)
- [Facebook React Native Community](#)
- [React Rocks](#)
- [Twitter List](#)

## React Native Resmi Dökümantasyonu

Kendi dökümantasyonu için söyleyecek çok bir şey yok. React Native yazıyorsanız en çok girdiğiniz sayfa burası olacak muhtemelen.

## React Native Express

React Native öğrenmeye hızlı bir başlangıç yapmak istiyorsanız, oldukça faydalı bir kaynak.

## StackOverFlow

Bir yazılımcı için tabi ki olmazsa olmaz. “react-native” tag’i ile arama yapmanız yeterli.

## JS Coach

React ve React Native ile ilgili yazılmış, github üzerindeki açık kaynak kodlu componentler'i bulabilir, star sayısına, güncelleme zamanına göre sıralayabilirsiniz. Dünya'da insanlar react konusunda neler yapıyor sorusuna cevap niteliğinde bir sayfa.

## React Discuss

Stackoverflow'dan farkı; hata çözmekle ilgili değil de insanların daha çok doğru bir yaklaşım yakalamak için fikir alışverişi yaptığı bir platform.

## Facebook React Native Community

Her türlü seviyeden insanın olduğu, react native ile ilgili her türlü şeyin tartışılp konuşulduğu bir grup.

## React Rocks

Bir showroom veya showcase olarak düşünebilirsiniz bu sayfayı. Hazır çalışır açık kaynak kodlu uygulamalar.

## Twitter List

Twitter üzerinde kendime yaptığım, javascript konusunda isim yapmış insanlardan oluşan bir liste. İster bu listeyi takip edebilir, isterseniz kendinize böyle bir liste yapabilirsiniz. Son gelişmeleri takip etmek daha kolay hale geliyor.

# Modern JavaScript

## Babel

React Native yazarken biz JavaScript yazıyoruz ama eski tarzda değil. Bizim yazdığınıza bazı özel fonksiyonlarımızın (Map(), async, vb) çalışabilmesi için **Babel** derleyicisine ihtiyacımız var. Babel, JavaScript kodunu derleyen, onu Native platformun anlayacağı dile çeviren temel araç.

## Babel Ayarları

Babel ayarlarını, **.babelrc** konfigurasyon dosyamıza yazdığınıza obje tanımlamasıyla yapıyoruz. Bu dosyaya ihtiyacımız olan plugini ekleyip, bu pluginin sunduğu sentaksi, platformun anlayacağı dile çevir diye Babel'e komut veriyoruz.

Örneğin, React Native yazarken biz **babel-preset-react-native** pluginini kullanıyoruz. Bu bizim React Native kodumuzu, IOS ve Android simulatörlerimizin anlayacağı dile çeviriyor. Ekstra plugin kullanmak istiyorsak .babelrc dosyamıza eklemeler yapabiliriz.

Gerçi react native projesine başlarken .babelrc dosyasını otomatik olarak bize veriyor. Ve çoğu durumda bu dosyayı açıp da değiştirmeye ihtiyacımız olmayacak.

React Native'in default olarak kullandığı, yukarıda bahsi geçen, babel-preset plugini bizim **es2015, es2016, es2017 ve react** pluginlerini kullanabilmemize imkân sağlıyor.

Eski tarz JavaScript yazmıyoruz demiştim. Kullandığımız yeni tarzımızın adı da ECMAScript (ES6, ES7)

(Spoiler: Zamanla ECMAScript'in sentaksını görüp nasıl ya, bu fonksiyon mu şimdiden gibisinden tepkiler vereceğinize eminim ve bu çok normal ama alışıkça seviyorsunuz. Eğer sevmezseniz, kasmayın ve siz eski tarzda yazmaya devam edin. Zaten mevcut uygulamaların production hallerinin, hep eski tarzda yazıldığı söyleniyor. Babel kullanmalısınız ama ES6 ve ES7 ile yazmak zorunda değilsiniz. )

Gelecek bölümde, **ES6, ES7 ve JSX** in daha da içine girelim.

Babel'in kendi plugin dökümantasyonu: <https://babeljs.io/docs/plugins/#transform-plugins>

# ES6 ve ES7 Örnekler

## Block Scoped Declaration

```
const a = 1
let b = 'foo'

// Not allowed!
// a = 2

// Ok!
b = 'bar'

if (true) {
  const a = 3
}
```

**const** ve **let** ifadelerine bakalım. Normal JavaScript'te bu ifadeler yerine değişkenleri tanımlamak için **var** kullanılır. Aralarındaki temel fark **const** ve **let** ifadeleri yalnızca bulundukları blok için geçerlidir.

**const** bildiğimiz "CONSTANT" kelimesinin kısaltması, sabit yani tanımlandığı bloktaki kendisine yalnızca bir kere atama yapılabilir

**let** ifadesi değişkenler için uygundur. Yukarıdaki örnekte üst blokta tanımlanmış `const "a"` ifadesine bir kere atama yapılmış daha sonra başka bir değişiklik yapılınca buna izin verilmemiş. ama farklı bir bloktaki tekrar atama yapılmıştır.

## Arrow Function ( => )

Arrow (`=>`) ya da fat arrow function () denilen bu fonksiyon başta kafa karıştırıcı olabilir. Normal bir fonksiyondan temel anlamda bir kaç tane farkı vardır.

Birincisi; ES5 ile yazılmış bir JavaScript kodunda, bir fonksiyonu bind etmek için kullandığımız, `this` kodu eğer biz arrow function kullanıyorsak gerekli değildir. Çünkü arrow function için fonksiyonun kendi bloğu ya da dış blok aynı kapsadır. Bir örnekle bakalım;

```
//ES5:
function merhaba(){
  this.msj = 'Merhaba';
  this.selamver = function(adi){
    return this.msj + adi
  }.bind(this)
}

//ES6:
function merhaba(){
  this.msj = 'Merhaba';
  this.selamver = (adi) =>{
    return this.msj + adi
  }
}
```

ES5 ile yazılan örnekte `selamver` fonksiyonu kendi üst bloğundaki `msj` 'a ulaşmak için `bind` edilmesi gereklidir; ES6 ile yazılan halinde, arrow function kullanılmış ve `bind` edilmesine gerek kalmadan `msj` değişkenine ulaşılmıştır

İkincisi; Eğer fonksiyonunuz bir parametre alıyorsa, süslü parantez `{}` ve `return` kullanmanız gereklidir. Kullanırsanız da bir şey değişmez. Örneğin;

```
var half = function(x){
  return x/2
}

var halfES6 = (x)=>{
  return x/2
}

var halfES6_2 = ( x => x/2 )

half(6);      //3
halfES6(8);   //4
halfES6_2(10); //5
```

<http://jsbin.com/docuhuy/2/embed?live>

## Modüller

```
//Modülleri import etme işlemi

//ES5;
var ReactNative = require('react-native');

//ES6;
import ReactNative from 'react-native'
```

```
//Modülleri export etme işlemi

//ES5;
module.exports=(App)

//ES6;
export default App;
export {View, Text, Image } ;
```

## Default Parametre

```
const printAnimal = (animal = 'cat') => {
  console.log(animal)
}
printAnimal() // cat
printAnimal('dog') // dog
```

## Class

ES5 'de, `class` yaratmak için sadece fonksiyon kullanabiliyoruz ve `MyFunction.prototype` ile method atıyalıyız. ES6, `class` için bize daha kolay bir sentaks veriyor.

ES6 bize javascript ile nesneye yönelik programlama imkânı veriyor. `inheritance` yapmak, `static` ve `instance` fonksiyonlar oluşturmak mümkün. Bide bir tane özel bir fonksiyonumuz var; `constructor` . `constructor`, bir class yaratıldığında otomatik olarak çağrılır. Ayrıca `static` kodu ile de static class fonksiyonları tanımlayabiliriz. `inheritance` için de kullandığımız keyword, `extends`

```

class Animal {
  constructor(name) {
    this.name = name
  }

  static beProud() {
    console.log('I AM AN ANIMAL')
  }

  printName() {
    console.log(this.name)
  }
}

const animal = new Animal('Cat')
animal.printName() // Cat
Animal.beProud() // I AM AN ANIMAL

```

```

class Cat extends Animal {
  printName() {
    super.printName()
    console.log(`My name is ${this.name}`) //ES6'nın güzelliklerinden concat işlemine dikkat ediniz `${}`
  }
}

```

## Array Spread

```

const arabalar = ['mercedes', 'fiat', 'bmw']
const yeniArabalar = [...arabalar] //arabalar dizisinin elemlarıyla, yeni bir dizi oluşturduk
//arabalar = yeniarabalar

const birSuruAraba = [...arabalar, 'kartal', 'tempra', 'lada'] //arabalar dizisiyle birlikte, yeni arabalar ekleyip birSuruAraba dizisini oluşturduk

const meyveler = [
  {adi: 'muz', renk: 'sarı'},
  {adi: 'elma', renk: 'kırmızı'},
]
const yeniMeyveler = [...meyveler]
console.log(yenimeyveler[0].adi) //muz

yeniMeyveler[0].adi = 'ayva'
console.log(yenimeyveler[0].adi) //ayva

```

## Static Variables Class

```
class Cat {
  static legCount = 4
}
console.log(Cat.legCount) // 4
```

## Object Spread

ES6 daki Array Spread gibi, Object Spread `...` objenin elemanlarını dağıtır. ES7 ile gelen bir özelliktir.

`{...originalObj}` bu bizim ilk objemiz olsun. Biz bunu Object Spread ile kopyasını alıp, istediğimiz bir özelliğini değiştirek yeni objemizi yaratalım.

```
const cat = {
  name: 'Luna',
  friends: {best: 'Ellie'},
  legs: 4,
}
const strangeCat = {...cat, legs: 6}
//cat objesinin kopyası şeklinde yeni bir obje yaratacakken, legs özelliğini yeni obje
mizde değiştirdik
```

## Async ve Await

Asenkron iş planına sahip uygulamamızı hem daha mantıklı hem de daha okunur kılar `async`, ES7 ile gelmiştir. `await` de onunla beraber kullanılır. `async` fonksiyonu, asenkron operasyon tamamlanana veya hata alana kadar bir sonraki kod bloğunun çalışmasını engeller.

```
const taskRunner = async () => {
  try {
    const firstValue = await asyncTask1()
    const secondValue = await asyncTask2(firstValue)
  } catch(e) {
    console.error("Bir şeyler ters gitti sanki", e)
  }
}
```

## ES6 Generator

Aşamalı, iterative biçimde ilerlemesi gereken bir fonksiyona ihtiyaç duyuyorsanız ES6 generator ler tam size göre.

```
function* greet(){
  console.log('You called greet');
  yield "hello";
  yield "world";
}

console.log('start')
let greeter = greet();
let next = greeter.next();
console.log(next)
let done = greeter.next();
console.log(done)

// console' a yazan değerler
/*
"start"
"You called greet"
[object Object] {
  done: false,
  value: "hello"
}
[object Object] {
  done: false,
  value: "world"
}
[object Object] {
  done: true,
  value: undefined
}
*/
```

Tüm generator işlemlerini sırayla yapılmasını istiyorsanız

```
for (let word of greeter){
  console.log(word);
}

// console' a yazılıan
"start"
"You called greet"
"hello"
"world"
```

# FONKSİYONEL PROGRAMLAMA NEDİR

React Native ile ilgili bir içerikte bu konunun ne işi var demeyin sakın. Çünkü yeni javascript dünyasında son 3-4 senedir yazılan kodlar buradaki prensiplere dayanarak yazılmakta. Yazılan kodları anlayabilmek adına, bu prensipler nedir hem okurken, hem de yazarken göz ardı etmememiz gereken bilgiler. Ayrıca Redux ile ilgili bir yazı yazmak istiyordum fakat fonksiyonel programlamaya degenmeden Redux'ı öğrenmeye kalkmak biraz bir şeyler çalışıyor ama nasıl çalışıyor anlamıyorum moduna itiyor insanı ve Redux kullanmanın avantajlarını ve dezavantajlarını tam olarak göremiyoruz. Bu yüzden Redux öğrenmekten çok daha kıymetli olacağını düşündüğüm fonksiyonel programlama ile ilgili bir şeyler karalamak istedim.

Bu seride yazılacak bilgiler [şuradaki](#) makalenin birebir değil ama kısmen çevirisi可以说吧。

**Fonksiyonel Programlama**, side effect'lerden, mutable data kullanmaktan, ortak state paylaşımı yapmaktan kaçınarak, pure fonksiyonları birleştirerek ( composition ), uygulama yapma silsilesidir. Fonksiyonel programlama, imperative değil, declarative'dır ve uygulamanın state'i pure fonksiyonlar boyunca akar. Object oriented programmanın tersine, uygulamanın state'i objelerin içinde olur ve fonksiyonlar yardımıyla erişilir. (*Burada kullanılan, İngilizce terimleri yazı boyunca açıklamaya çalışacağım*)

Fonksiyonel programlama, object oriented programlama ve procedural programlama gibi bir yaklaşım, paradigmadır.

## Fonksiyonel programlamanın temel prensipleri

- Pure fonksiyonlar
- Composition
- Ortak state kullanımından kaçınma
- State mutate etmekten kaçınma
- Side effect'lerden kaçınma

## Pure Fonksiyonlar

Pure fonksiyonların tanımına bakarsanız şöyle bir şey görürsünüz, \_verilen bir inputa göre, her zaman aynı output'u dönen fonksiyonlar, pure fonksiyonlardır. \_Bu tanım ilk bakışta çok anlamsız gelebilir ki bana öyle geliyordu.

Mesela şöyle bir fonksiyonumuz olsun;

```
//es5
function double(x){
    return x * 2
}

//es6
const double = x => x * 2
```

`double` fonksiyonu nereden çağrırlırsa çağrılsın, herhangi bir şeye bağımlı olmadan, aynı input'ta aynı sonucu verecektir. `double(5)` her yerde 10 sonucunu verecektir. 10 gördüğünüz yere `double(5)` yazmanız hiçbir şey farkettirmeyecektir. Çünkü `double(5)` bir pure fonksiyondur. Karşı örnek olarak en basitinden `math.random()` fonksiyonu çağrıldığı her yerde, zaten input almıyor, farklı sonuç verecektir. Dolayısıyla `math.random()` 'a pure fonksiyon diyemeyiz.

Pure fonksiyonların bir diğer özelliği, **hicbir side effect( yan etkileri )'leri yoktur**. Yani dışarıdan bir müdahale ile değiştirilemezler

Javascript'te bir objenin argümanları, özellikleri, o objenin referanslarıdır. Eğer bir fonksiyon dışarıdan o objenin referanslarını değiştirirse bu objeyi mutasyona uğratmak yani mutate etmek yani değiştirmek artık ne derseniz deyin, bu durum pure fonksiyonlarda asla olmaması gereken bir durumdur. ( Redux'la beraber çalışırsanız göreceğiniz en keskin kural Redux'ta kullanacağınız reducer'ları pure fonksiyonlardan oluşturmanız gerekmektedir ).

Örneğin; aşağıda kullanılan push methodu mutate bir fonksiyondur. Ve objenin referanslarını değiştirir. Bu istenmeyen bir durum yaratabilir. Kodun tahmin edilebilirliğini ve güvenilirliğini kötü etkileyebilir.

```
const addToCart=(cart, item)=>{
    cart.push( item ); // push mutate bir fonksiyon
    return cart;
}
```

Bu fonksiyonu immutable yapıp pure fonksiyona dönüştürmek için, push yerine Object Assign veya array spread operator kullanılabilir.

```
//es5
const addToCart=(cart, item)=>{
    Object.assign(cart, { item }) //Object Assign, immutable bir fonksiyondur.push yeri
    ne kullanılabilir
    return cart;
}
```

veya

```
//es6
const addToCart=(cart, item)=>{
  cart = [...cart, item ] //array spread operator ile de pure fonksiyon yaratılabilir.
  return cart;
}
```

Örnek olarak [şuradaki](#) impure fonksiyonu, pure yapmaya çalışabilirsiniz.

Bir sonraki yazıda React'ın temellerini oluşturan, composition mantığından yani fonksiyon birleştirmekten bahsedeceğiz.

# FONKSİYONEL PROGRAMLAMA VE COMPOSITION

Composition, iki ya da daha fazla fonksiyonu birleştirerek yeni bir fonksiyon yaratma işlemi.

Matematikten,  $f()$  ve  $g()$  fonksiyonlarını düşünelim. $f(g(x))$  , bir composition işlemidir. Ve işlem sırası aşağıdaki gibidir

- $x$
- $g$
- $f$

Esasında, ortaokul matematiğinde gördüğümüz fog yani bileşke fonksiyonlar.

$$c(x) = f(g(x))$$

Fonksiyonları birleştirirken kullanılan diğer temel fonksiyon türlerine göz atalım.

## Curry, Partial ve Pipe Fonksiyonlar

### Curry Fonksiyon

Bir fonksiyon, input olarak bir fonksiyon ve çoklu parametre alıp, output olarak **tek parametreli bir fonksiyon** dönüyorsa, bu bir **curry fonksiyondur**.

```
const curry = fn => (...args) => fn.bind(null, ...args);
```

### Partial Fonksiyon

Bir fonksiyon, input olarak, bir fonksiyon ve çoklu parametreler alır, output olarak **daha az parametreyle bir fonksiyon** dönüyorsa, bu bir **partial fonksiyondur**.

```
const partial=(fn, ...args)=>{ fn.bind(null, ...args) }
```

### Pipe Fonksiyon

İki ya da daha fazla fonksiyonu input olarak alıp, parametre olan bu fonksiyonları, bir önceki fonksiyonunun outputu ile sırayla çalıştırabilecek şekilde ayarlayan ve output olarak bize tek fonksiyon dönen fonksiyonlar, **pipe fonksiyonlardır**.

```
const _pipe = (f, g) => (...args) => g(f(...args))
const pipe = (...fns) => fns.reduce(_pipe);
```

Şimdi terimsel tanımlamaları geçip, bu fonksiyonları nasıl kullanacağımız ona bakalım.

İlk önce kullanacağımız standart, fonksiyonlarımıza yazalım.

```
const _pipe = (f, g) => (...args) => g(f(...args))
const pipe = (...fns) => fns.reduce(_pipe);
const partial = (fn, ...args) => fn.bind(null, ...args);
```

Şimdi de kullanacağımız pure fonksiyonlarımıza bakalım.

```
const add = (a, b, c = 0) => a + b + c;
const multiply = (a, b) => a * b;
const dec = (a) => a - 1;
```

Bunları hep beraber, comeTogether adlı fonksiyonda pipe yardımını ile birleştirelim yani compose edelim.

```
const comeTogether = pipe(multiply, dec, partial(add, 10, 3));
```

Artık bir sürü iş yapan fonksiyonumuzu aşağıdaki şekilde çağrılabiliriz.

```
const result = comeTogether(2, 4); // 20
```

### Sahne arkasında neler olup bitti ?

- İlk önce `multiply` fonksiyonu, 2 ve 4 ü çarptı ve 8 buldu.
- `multiply` 'dan dönen 8 değerini `dec` aldı ve 1 eksiltti, 7 buldu.
- `dec` 'den dönen 7 değerini `add` fonksiyonu direkt alamadı. Çünkü o çoklu parametreyle çalışan bir fonksiyon olduğu için `partial` fonksiyonunun yardımına ihtiyaç duydu. `partial` fonksiyonu yardımını ile `add` fonksiyonu 7 yi aldı, 10 ve 3 ile toplayıp 20 değerini döndü.

[source code](#)

*Not: Burada bahsedilen, curry, partial ve pipe fonksiyonlarının içeriğini daha da ayrıntılı biçimde analiz edeceğimiz bir yazı yazmayı planlıyorum. Fakat bu kısıtlı zamanda şimdilik hem kendim için hem de ufak önbilgi edinmek isteyenler için bu kadarını paylaşmak istedim.*

*Yoksa bu adı geçen fonksiyonlar yoktan var olmadı. Bilindik kısaltılmış es6 ile yazılan javascript fonksiyonları ama ilk bakışta ne olduğu ve anlaşılması zor fakat kullanımı itibarı ile standart fonksiyonlar.*

Kaynak: [Master the JavaScript Interview: What is Function Composition](#)



# React JS Nedir ?

Öncelikle bir framework değildir. Sadece uygulamanızın ön yüz kısmını ilgilendiren bir View, UI Kütüphanesidir.

React 'ın bize sunduğu şey bir kod taslağı ve bu taslağını kullanarak parça parça componentler oluşturup, lego oynar gibi uygulamamızı tasarlamak. Ve bu tasarımın sonunda bize output olarak verdiği şey, sadece HTML kodu. Evet yanlış okumadınız, sadece HTML üretmek, React'in görevi. O yüzden tam teşekküllü bir uygulama geliştirmek için sadece React öğrenmek yetmeyecektir. Ama bu gözünüzü korkutmasın React'ın bu sunduğu declarative kod taslağı bir çok önyüzde yaşayacağınız problemlere derman olacaktır. Ve yazması bağımlılık yaratacaktır, emin olun.

React Js ekibinin, en büyük sloganı, **bir kere öğren , her yerde yaz.**İşte tam bu sırada React Native devreye giriyor. Siz React öğrenerek aslında React Native ile de mobile uygulamalar geliştirmek için de önemli bir yol katetmiş oluyorsunuz. O yüzden ilk başta sadece React Js ile nasıl kod yazıyoruz ona bir bakalım.

# JSX Nedir ?

(Bu sayfada jsbin den gömülü kod kullanılmaktadır. Eğer kodu göremiyorsanız, browserinizda adres çubuğundaki **load unsafe script** seçeneğini seçmelisiniz)

[source code](#)

Yukarıdaki basit bir React componenti. React ve ReactDOM kütüphanelerini script tag'i ile HTML sayfamızın başına yazdık.

Sonra Javascript derleyicimizi, JSX seçtik ( ES6/Babel de seçebilirdik )

Burada dikkatinizi çeken bir şey oldu mu ? Biz javascript kodumuza, HTML kodu yazdık :)

Önceden bahsettiğimiz, React in bize sunduğu lego tasarılar gibi uygulama geliştirmek tam olarak böyle, parça parça HTML kodlarını component olarak yazıp hepsini belli bir hiyerarşîyle bir araya getirmek.

[source code](#)

Aslında en yukarıda yaptığımız JavaScript içine HTLM kod yazarak yaptığımız şey tam olarak React.createElement yerine HTML tagleri kullanmak. Bana sorarsanız HTML yazmak daha kolay.

# Lifecycle Methodlar

`React.Component` bizim en temel class'ımız. Her componentin kendine özel lifecycle methodları var. Bu methodları 3 grupta topluyoruz.

## Bir componentin mount edilmesi ne demek ?

Browserların bir DOM (Document Object Model ) yapısı var. Bu document <html> tag i başlayıp sırayla, <head>, <body> diğer html taglarıyle devam ediyor. Bir component mount edildi demek de componentin DOM'a yerleştirilmesi, browser da görünmesi demek.

## Mounting Methodlar

Bir component DOM'da yaratılınca, insert edilince çağrılan methodlardır

- `constructor()`
- `componentWillMount()`
- `render()`
- `componentDidMount()`

## Updating Methodlar

Bir componentin state ya da propsunun değişmesiyle, tekrardan render edilmesi sonucu çağrılan methodlardır.

- `componentWillReceiveProps()`
- `shouldComponentUpdate()`
- `componentWillUpdate()`
- `render()`
- `componentDidUpdate()`

## Unmounting Methodlar

Bir component DOM'dan çıkarılmak üzereyken çağrılan fonksiyondur.

- `componentWillMount`

---

### `constructor()`

```
constructor(props) {
  super(props);
  this.state = {
    color: props.initialColor
  };
}
```

constructor methodu içinde componentin propslarını kullanmak isterseniz, `super(props)` 'u çağrırmalısınız. Aksi takdirde props'ları görmeyecektir. Diğer methodlardaki gibi `this.props` 'u bu method için kullanamazsınız. Onun yerinde yukarıda örnekte olduğu gibi `props.initialColor` gibi kullanabilirsiniz.

Başlangıç state'i tanımlamak ve bir fonksiyonu componente bind etmek için en uygun yer constructor. Bu iki tanımlamaya ihtiyacınız yoksa constructor methodunu tanımlamayabilirsiniz.

### **componentWillMount( )**

Component mount edilmeden hemen önce çağrılır. Dolayısıyla `render()` methodundan da önce çağrılır. Bu yüzden bu method içinde state değiştirip, `render()` methodunu tekrardan çalıştırılmamalıdır. Önerileni, bu methodu sadece server rendering yaparken kullanmamız. `render()` 'dan önce çalışmasını istediğiniz bir fonksiyon varsa `constructor()` buradan daha uygun bir yer

### **render( )**

`React.Component` class'ı için olmazsa olmaz, çağrılmazsa sorun çıkaracak tek method `render()`

Bu method çağrılinca karşılığında size, ona verdiğiniz props ve state'e göre `<div />`, `<img />` gibi sizin tanımladığınız bir DOM componentinin görünümü verecektir.

`render( )` methodunda, eğer siz hiçbir şey mount etmek istemiyorsanız dönüş değerini olarak, **null**, **false** değerlerini de verebilirisiniz. Higher order component'lerde veya koşullu bir componenti render ederken çok işinize yarayacaktır.

`render( )` methodu **pure** olmalı. React ile uğraşırken bu cümleyi çok görürsünüz. state' e göre size dönüş üreten bir fonksiyon içinde state değiştirmek, sizi ve browser'ınızı sonsuz döngüye sokacaktır.

`render( )` methodu DOM elementlerinin bir görüntüsünü, yani asıl tabiriyle virtual, sanal DOM size üretir. Eğer DOM yapısına direkt müdahale etmek istiyorsanız, `render()` methodu yerine **componentDidMount()** methodu kullanmanız önerilir.

### **componentDidMount( )**

Component mount edildiğinde apar topar çağrılan methodumuz. DOM'a etki eden bir şeyin tanımlanması gereken yer burası. Atıyorum **D3.js** gibi third party bir library kullanacaksanız, başlangıç değerlerini burada atayın. Bir API çağıracaksanız, bir şeyi fetch edecekseniz en müsait yer bu method.

### **componentWillReceiveProps( nextProps )**

Component yeni, değişmiş props'unu alıp mount edilmeden önce çağrılr. Eğer props'a göre state değiştirmeye ihtiyaç duyarsanız, aşağıda örnekteki gibi kullanabilirsiniz;

```
componentWillReceiveProps( nextProps ){
  if(this.props.open !== nextProps.open){
    this.setState({ reset : true })
  }
}
```

React, bu methodu component yeni props almasa da çağrıbilir, bunun bir garantis yok. O yüzden nextProps ve this.props karşılaştırmasına göre işlem yapmanız avantajınıza olacaktır.

React bu methodu başlangıcta çağrırmaz. Sadece component mount edildikten sonraki süreçte update olursa bu method çalışır.

### **shouldComponentUpdate( nextProps, nextState )**

Yeni props ve state alınıp, render edilmeden önce bu method çağrılr. React ' da herhangi bir state değişince otomatik olarak render methodu çalışır. Bu da aslında kendisiyle alakası olmayan state değişiminden, başka bir componentin etkilenmesi ve boşuna render edilip, zaten hali hazırda olan aynı görseli oluşturma demek. Bu yüzden props veya state değişikliği, ilgili componentinizi etkilemiyorsa, bu methodu kullanıp, ciddi bir performans kazanabilirsiniz.

```
shouldComponentUpdate( nextProps, nextState ){
  if(nextState.open === this.state.open){
    return false    // "open" state i değişmediyse componenti tekrar render etme
  }
}
```

shouldComponentUpdate false dönerse, componentWillUpdate(), render() ve componentDidUpdate() methodları çalışmaz

### **componentWillUpdate( nextProps, nextState )**

Bu method, yeni props ve state alındığında, `render()` dan önce çalışır. React bu methodu başlangıçta çağrırmaz. Bu method içinde state set edilmez. Bunun yerine önceden bahsettiğimiz `componentWillReceiveProps()` kullanılabilir.

### **componentDidUpdate( prevProps, prevState )**

`shouldComponent` false dönerse bu method çağrılmaz.

`render` methodundan sonra hemen çağrılan methoddur. Hali hazırdaprops değeri ile bir önceki propsu karşılaştırmak için idealdir. Props değişikliği varsa network istekleri de buradan yapılabilir.

### **componentWillUnmount( )**

Şahsen benim abi bu method çok iyi ya dediğim ama nadiren kullandığım bir fonksiyon. Componentimiz artık yok edilecekken, yok edilmeden hemen önce çağrılan method. Componentin başında yaptığımız bir listener'ı kaldırmak için, önceden yapılmış bir network requestini iptal etmek için, özetle bir şeyleri sıfırlamak için çok ideal bir yer. Component'e gitmeden önce hesap sorduğumuz yer diyelim.

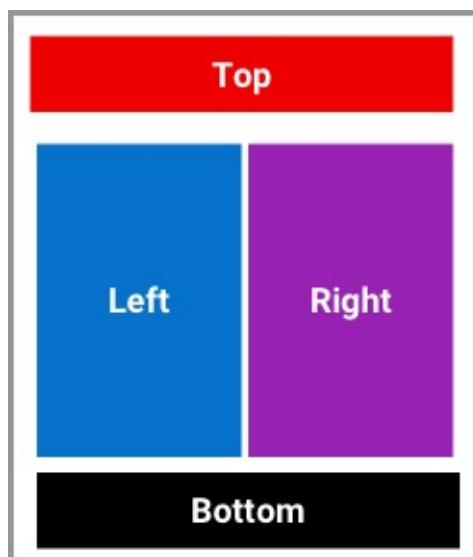
# Props Mantığı

React Js ' de her bir component aslında bir fonksiyon. Nasıl görünceklerini, ekranda nasıl bir yer kaplayacaklarını da bu fonksiyonlara verdiğimiz, input değerleriyle belirliyoruz. İşte bu input değerlere **props** diyoruz.

```
const MyButton = (props)=>{
  return(
    <button> { props.text } </button>
  )
}
```

Üzerindeki yazıyı dışarıdan props aracılığıyla alan basit bir button componenti. Bu componenti tekrar tekrar farklı props'lar ile kullanabiliriz. Aşağıdaki örneği inceleyebilirsiniz.

[source code](#)



Yukarıda gördüğünüz bir önyüz. Ve bu UI **Top**, **Left**, **Right** ve **Bottom** componentleriyle 4 e bölünmüş durumda. Bu componentler aldığıları propslar ile kendi içlerinde bağımsız şekilde alabiliyorlar. Peki farklı componentler arasındaki olası ilişkiyi nasıl sağlayacağımız ? İşte bu anda **state mantığı** devreye giriyor

# State Mantığı

State tanımı, componentimizin her sıfırdan çağrıldığında çalışacak olan constructor fonksiyonunda yapılır.

( Sıfırdan çağrılmamasından kastım daha mount, render edilmemiş bir componentin çağrılması. Yoksa hali hazırda ekranımızda olan mount edilmiş bir componentte constructor tekrar çağrılmaz )

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      open:false
    };
  }
  ...
}
```

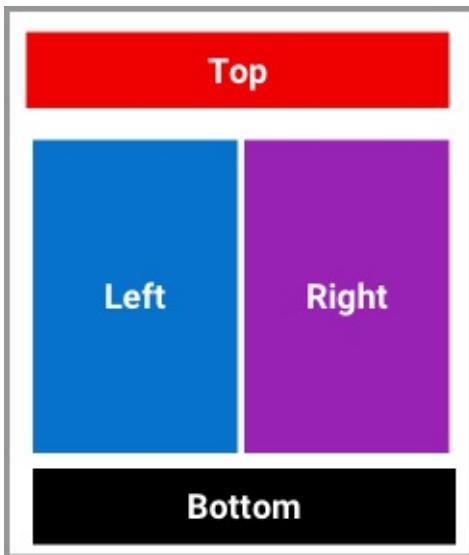
State ne yapar; statefull component denilen, kendi içinde state tanımı mevcut olan componentin her state'i değiştiğinde componentin **render** fonksiyonunun tekrardan çalışmasını sağlar.

## Stateless Component Örneği

```
const MyComponent = (props)=>{
  return(
    <div>...</div> //state'in değiştireceği bir render methodu yok
  )
}
```

## Statefull Component Örneği

```
class LinkButton extends React.Component {
  render() { //her state değiştiğinde render() tekrar tekrar çalışacak
    return (
      <div>...</div>
    );
  }
}
```



Bir önceki sayfadaki ekranımızı, propslar yardımı ile jsbin de composition yaparak oluşturalım.

[source code](#)

Burada üç tane stateless (bazıları **dumb component** diyor) component, **FlexItem**, **FlexColumn**, **FlexRow** var.

Bir tane de state almaya müsait statefull ( bazıları **container component** diyor ) component, **MyComponent** var.

Burada herhangi bir asenkron state değişimi sorunu ile karşılaşmamanız adına en dıştaki, hiyerarşide en üstte bulunan componentte state i belirleyip değiştirip, alttaki componentlere dağıtmamanız. Aksi takdirde aşağıdan yukarı bir state yönetimi Redux gibi state yönetimini kullanmadan neredeyse imkansız.

Şimdi bir tane state tanımlayalım ve bir timer yardımıyla saniyede bir arttırıralım. Altta componentlere dağıtalım her componentte kendi içinde state'i nasıl değerlendirir, nasıl şekil alır o componentin davranışına kalsın.

[source code](#)

Burada Container componentimizdeki state timer yardımıyla count state'ini her bir saniyede bir artırıp state'i altındaki componentlere props olarak veriyor. **FlexItem** componenti de çift sayıarda kendi count propsunu , tekli sayıarda da children propsunu göstererek bir davranış sergiliyor.

Sonuç olarak burada bizim yaptığımız yukarıdan aşağıya doğru senkron bir state yönetimini göstermekti. Çok karmaşık işlemlerde state yönetimini daha önceden de söylediğimiz gibi Redux ile React Native yazarak yapmaya çalışacağız. Ama eğer siz state yönetiminde herhangi bir sorun yaşamıyorsanız Redux ya da Mobx gibi state yönetim sistemine ihtiyacınız yok.



# REACT NATIVE NEDİR ?

Kitabın öncesinde React JS için bahsettiğimiz ne varsa, Lifecycle methodlar, state , props mantığı, statefull veya stateless componentler ve daha bir sürü şey React Native için de geçerli. Tabi React Native'de bir DOM yapımız yok. Mobil işletim sistemlerinde bunu çözmek için React Native takımı, şöyle bir yol geliştirmiştir.

Android ve IOS da hali hazırda olan native componentler için araya bir köprü koymuş. Bu köprü sayesinde, bu componentleri Javascriptin algılayabileceği hale getirmiştir. Şimdi o bazı componentlerin html tagleri karşılıklarına bakalım.

(React Native 0.42 için geçerlidir)

HTML	React Native
div	View
button	Button
img	Image
input	TextInput
label	Text

React JS yazarken, kullandığımız temel html taglerini nerdeyse aynı özellikleriyle React Native'de de kullanabiliyoruz.

( button ve label ile ilgili bazı söylemesi gereken şeyler var fakat Temel Componentler kısmında oraya değinelim, ayrıca sunulan componentler de bu kadar değil daha fazlası var )

# Installation

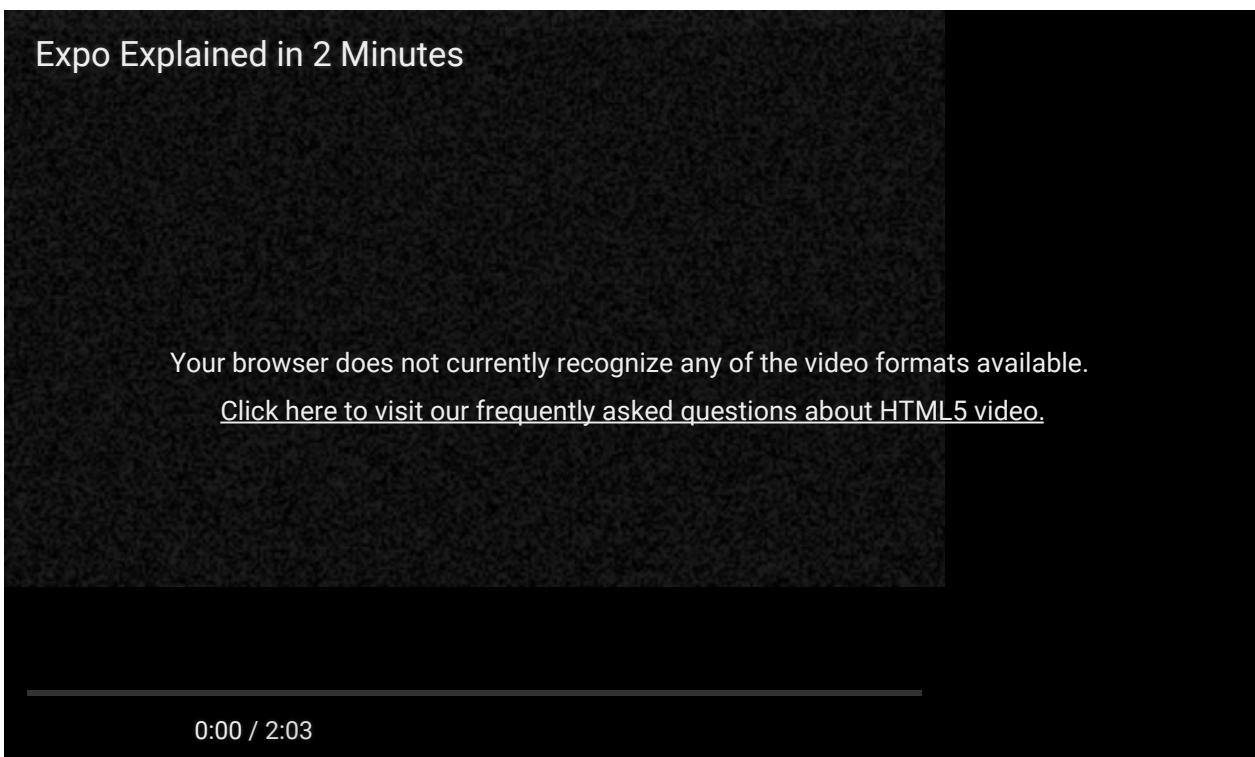
- [Exponent Client](#)
- [Windows Installation](#)
- [macOS Installation](#)
- [Gnu/Linux Installation](#)

# EXPO

Exponent ekibinin yaptığı react-native ile online kod yazmayı mümkün kıلان [snack.expo projesi](https://snack.expo.io/), react-native yazmak için en hızlı yol.

- <https://snack.expo.io/> adresine girip hemen kodlamaya başlayabilirsiniz. Online emulator'de kodunuzun çıktısını görebilirsiniz.
- Proje oluşturup, lokal makinenize indirip, normal react-native projesi gibi projenize devam edebilirsiniz.
- Yazdığınız kodu telefonunuzda derlemek için;
  - IOS kullanıyorsanız, [AppStore](#)'dan expo client'ı telefonunuza indirmelisiniz.
  - Android kullanıyorsanız, [GooglePlay](#)'den expo client'ı telefonunuza indirmelisiniz.
  - <https://snack.expo.io/> adresindeki projenizin QR code'unu expo client uygulamasında okutup, projenizi telefonunuzda derleyebilirsiniz.

Expo Projesi, ilk başlayanların react-native'e göz atması, hızlı bir şey denemek adına ya da kod paylaşmak için çok kullanışlı. Ancak internet hızınız düşükse development süreci çok çetrefilli olabilir. Performans sorunlarından dolayı da, şimdilik bu platform üzerinde gerçek bir proje çıkarmayı düşüneniz iyi edersiniz.



Not: Sunum videosu gözükmüyorsa, adres çubuğundan **load unsafe script**'i aktif etmelisiniz.



# Windows Installation

## 1-Node Js kurulumu

[Node JS](#) kurulumunu kendi sayfası üzerinden klasik olarak next next diyerek kurabilirsiniz.



### Downloads

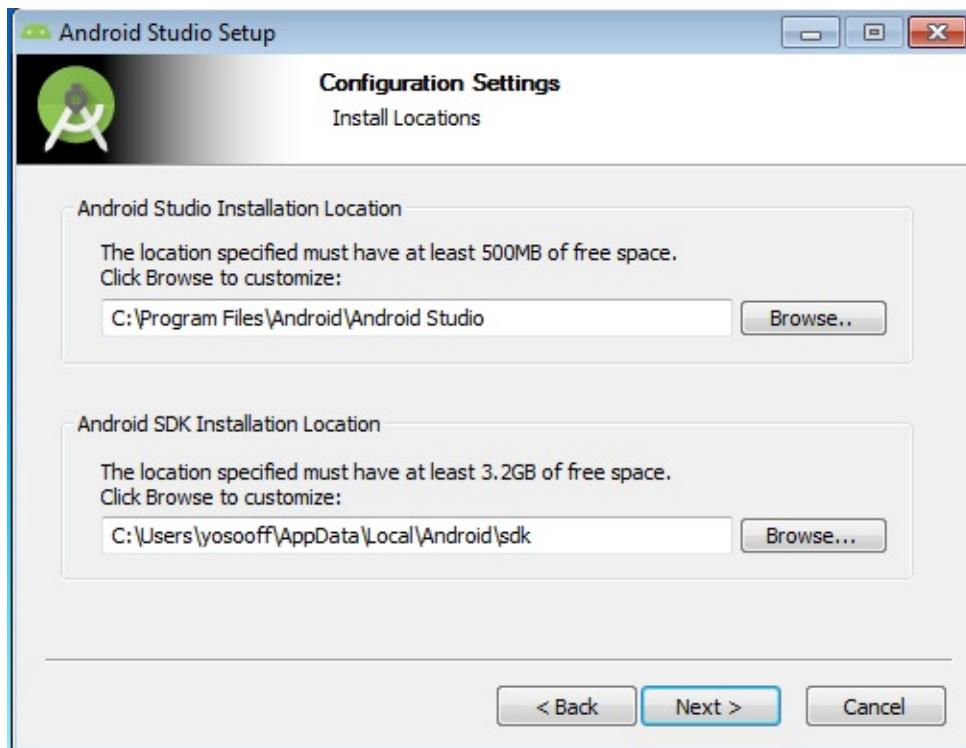
Latest LTS Version: v6.9.2 (includes npm 3.10.9)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

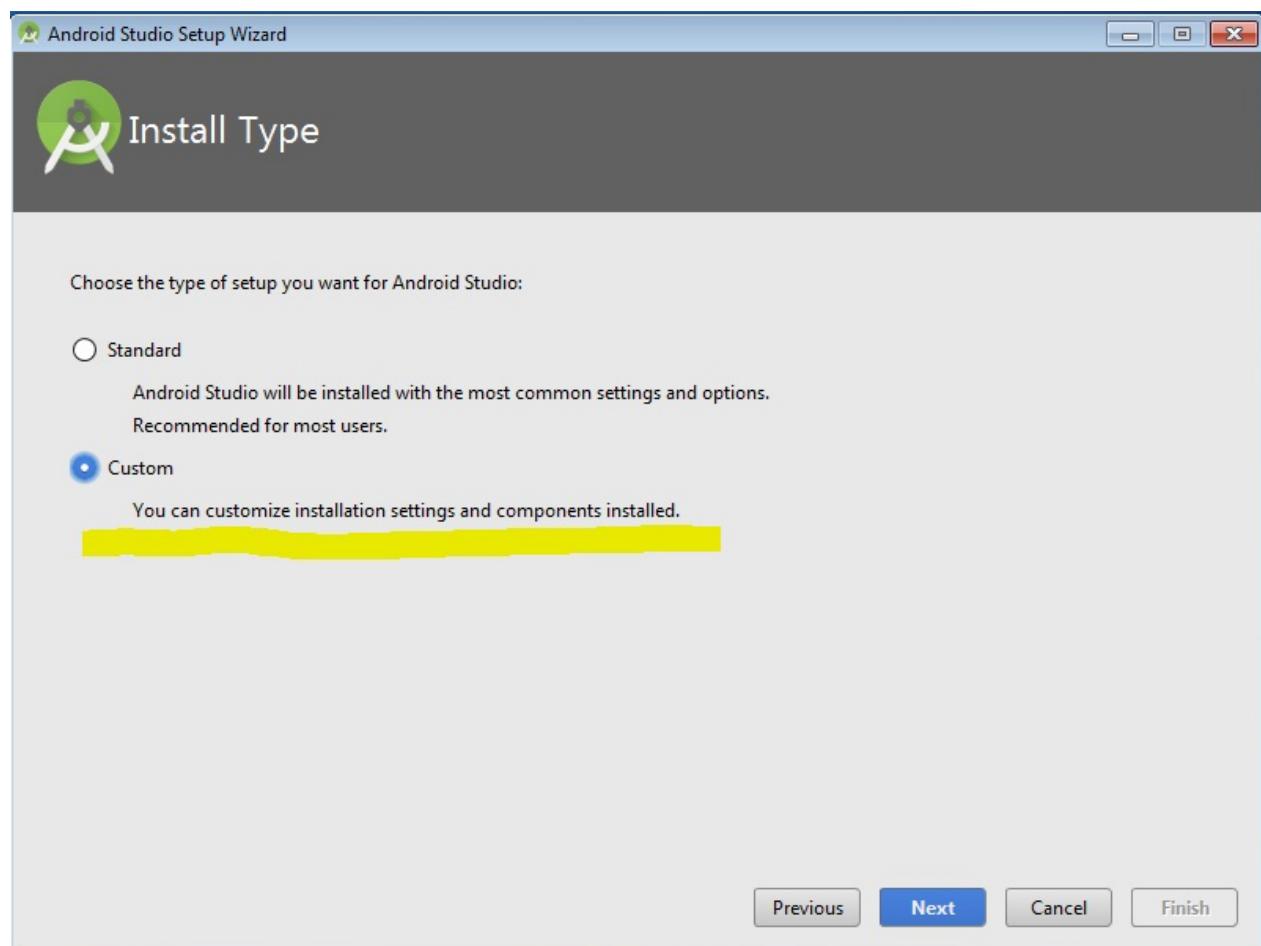
LTS Recommended For Most Users	Windows Installer	Macintosh Installer	Source Code
Current Latest Features	node-v6.9.2-x64.msi	node-v6.9.2.pkg	node-v6.9.2.tar.gz
<a href="#">Windows Installer (.msi)</a>	32-bit	64-bit	
<a href="#">Windows Binary (.exe)</a>	32-bit	64-bit	

## 2-Android Studio Kurulumu

[Android Studio](#) kurulum dosyasını indirip, kuralım. Android SDK nin nereye yükleneceği bilgisi önemli, aklımızda bulunsun.

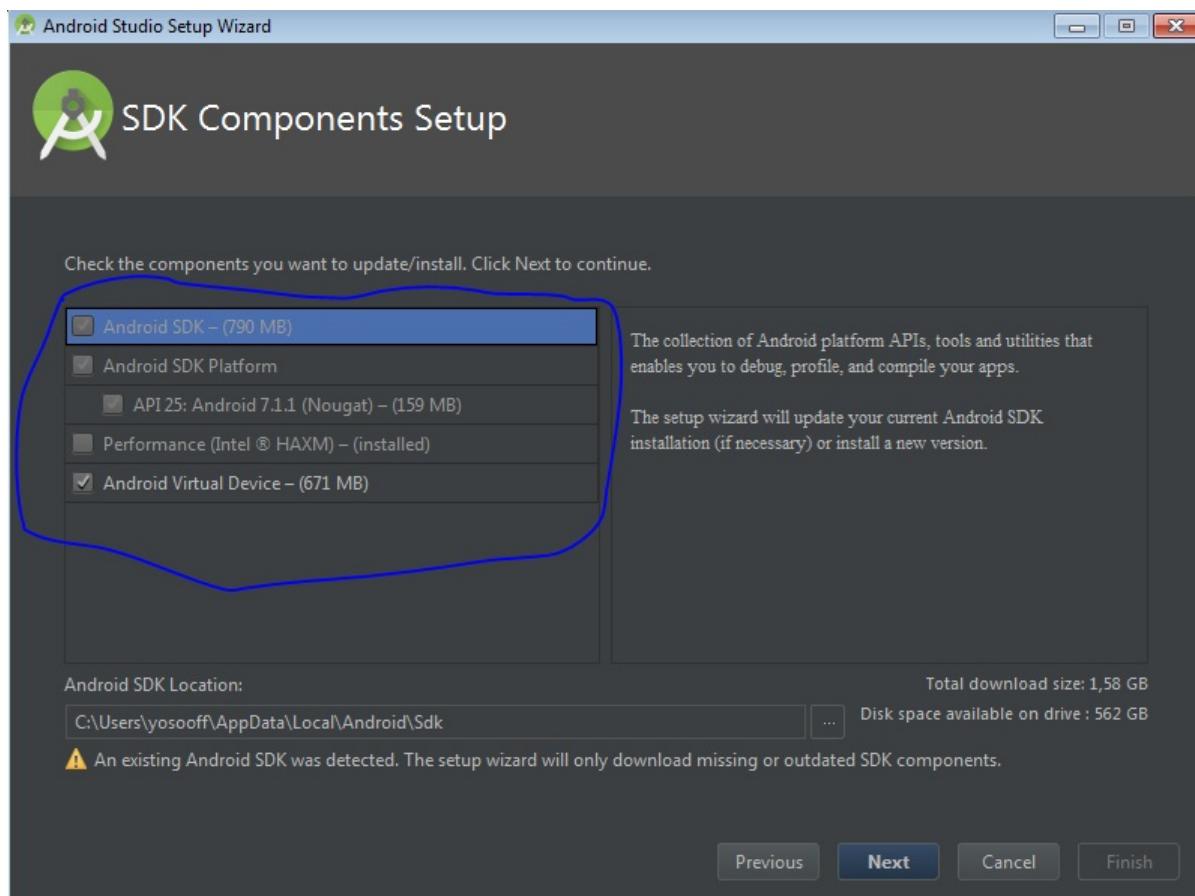


Kurulumu yaparken bir seçim ekranı gelecek, burada “Custom Installation” seçeneğini seçmelisiniz.

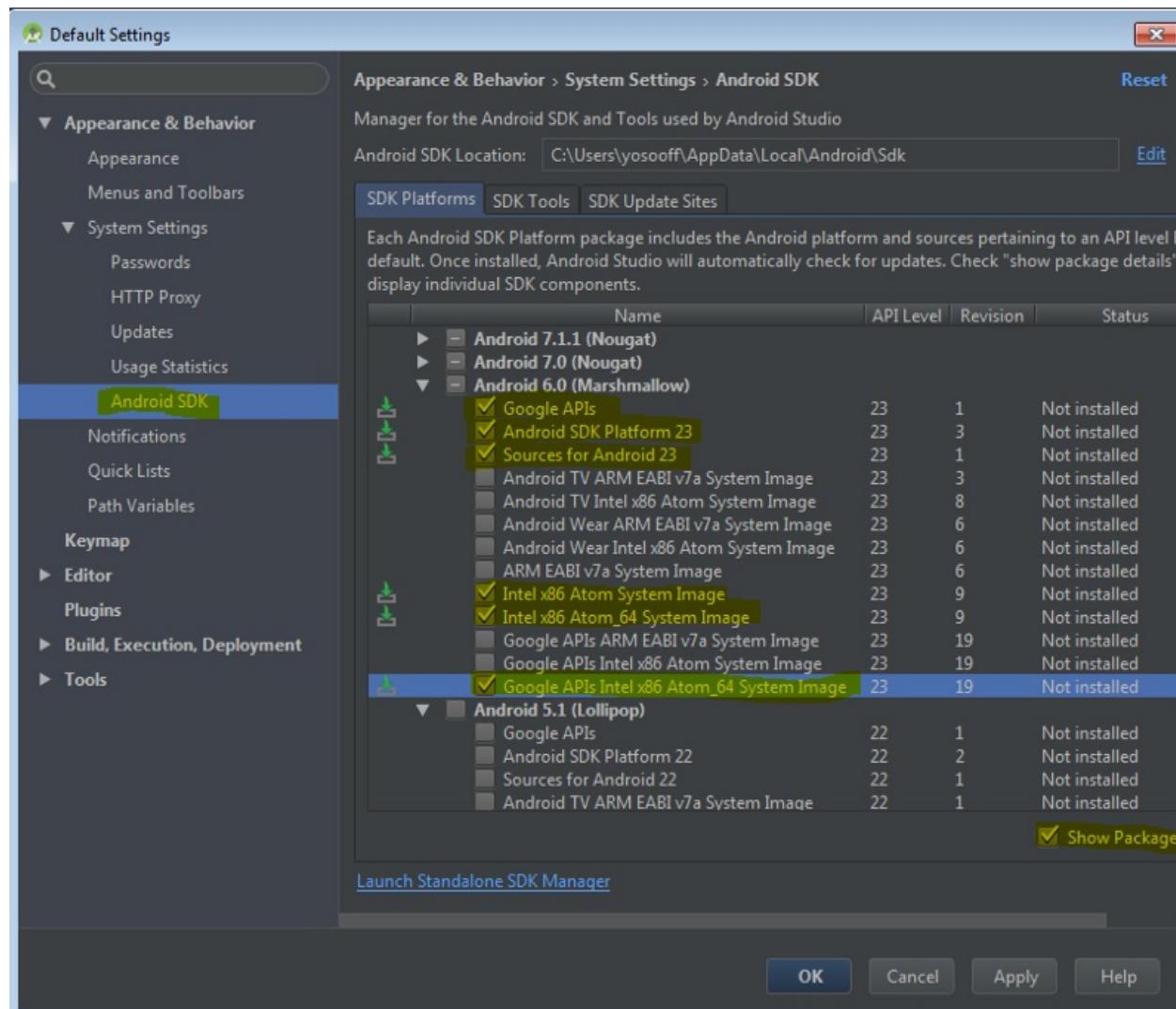


Yükleme sırasında ya da zaten bilgisayarınızda yüklü bir Android Studio var ise aşağıdaki listedeki araçların yükleniğinden emin olun. Yoksa “SDK Components Setup” ekranından yükleyin.

- - Android SDK
  - Android SDK Platform
  - Performance (Intel HAXM)
  - Android Virtual Device

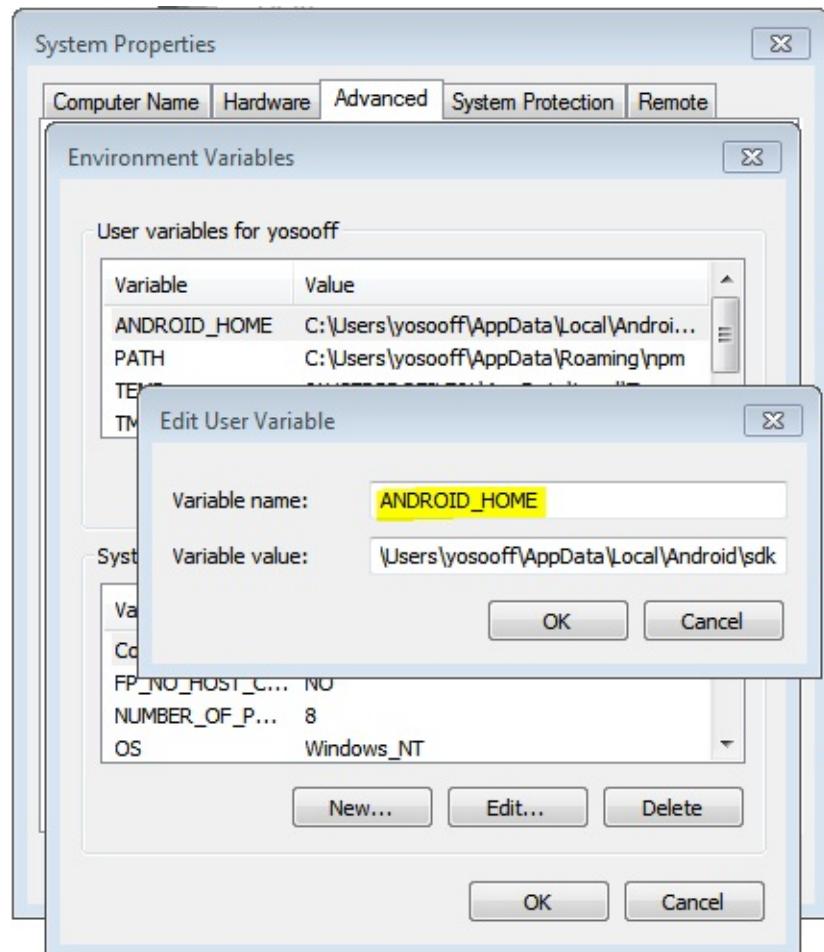


Yukarıda ki resimde görüldüğü gibi “Android Studio Installation” otomatik olarak bilgisayarınıza Android’ın en son sürümünü(şu an için Android 7.1.1 Nougat) kuracaktır. Ancak React-Native, Android 6.0 (Marshmallow) sürümünü istemektedir. Bunun için Android Studio’nun açılış sayfasında “Configure” butonuna tıklayıp, Ayarlar bölümünde “Android SDK” yi işaretleyip, buradan Marshmallow sürümünü kuralım.



Marshmallow sürümünü kurarken yukarıda işaretli araçları seçmeyi unutmayın. Eğer bu araçları göremiyorsanız, ekranın sağ alt köşesindeki "Show Package Details" seçeneğini işaretleyerek paket detaylarında görebilirsiniz. Apply butonuna basıp, tekrar next next diye devam ediyoruz.

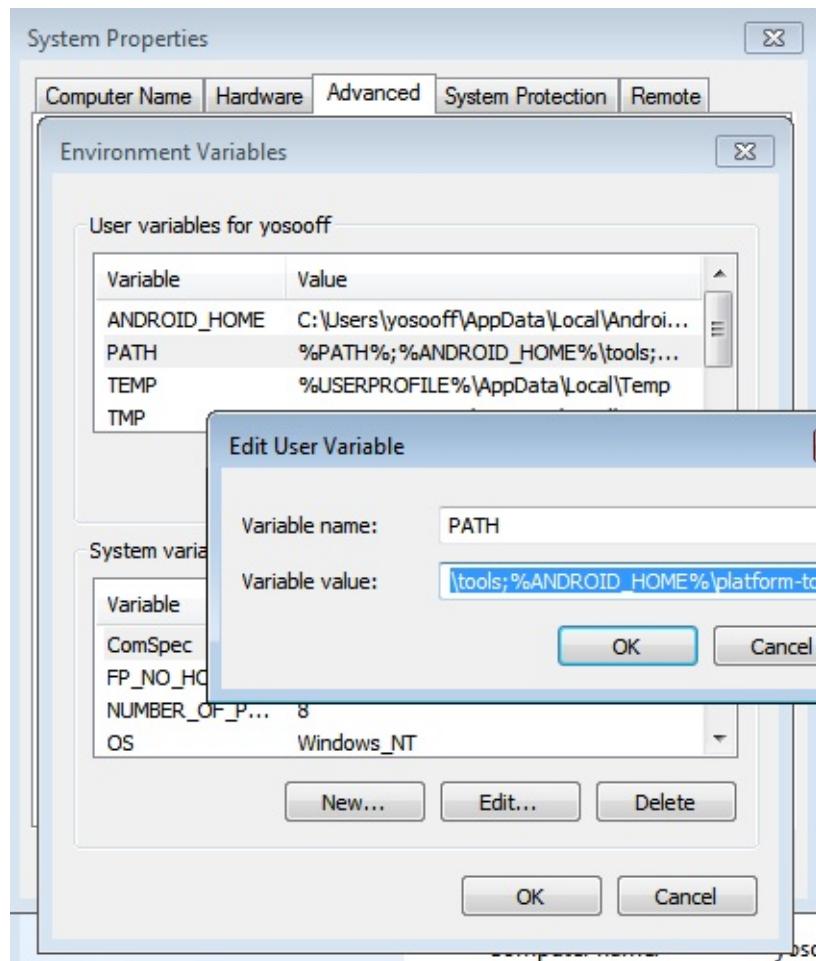
Şimdi Sistem Değişkenlerine **"ANDROID\_HOME"** değişkenimizi ekleyelim.



ANDROID\_HOME değişkenini ekledikten sonra PATH değişkenine aşağıdaki satırı kopyalayıp yapıştırın. (

Tahminim en çok insanların gözünden kaçıldığı nokta burası oluyor, ortam değişkenlerini eklemeye, harf, noktalı virgül atlamamaya çok dikkat edin)

```
;%ANDROID_HOME%\tools;%ANDROID_HOME%\platform-tools
```



## 3-GenyMotion Kurulumu

GenyMotion bir Android Emulator. Android Studio'nun da sunduğu bir emulator var ama ben GenyMotion kullanmanızı öneririm.

GenyMotion 'ı kendi sitesine üye olup ücretsiz olarak [şuradan](#) indirebilirsiniz. Tabi bu arada GenyMotion 'ın Virtual Box üzerinde çalışan sanal bir telefon olduğunu düşünürsek, bilgisayarınızda Virtual Box da kurulu değilse, Virtual Box la beraber sunulan yükleme dosyasını indirmelisiniz.

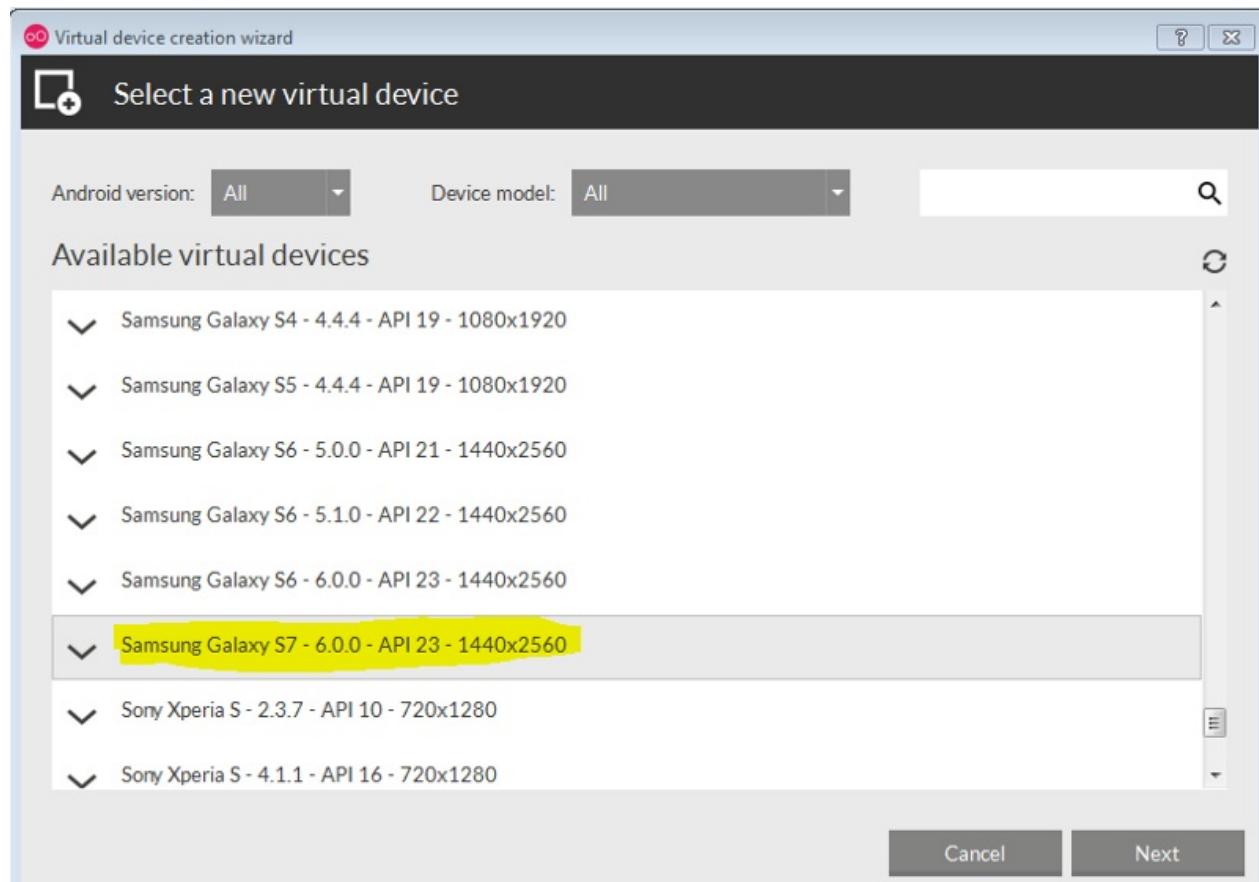
The header of the Genymotion website features the Genymotion logo with a green play button icon, followed by navigation links for Solutions, Download, Pricing, Help, Resources, Other Products, and a Contact Us button.

## Download Genymotion 2.8.1

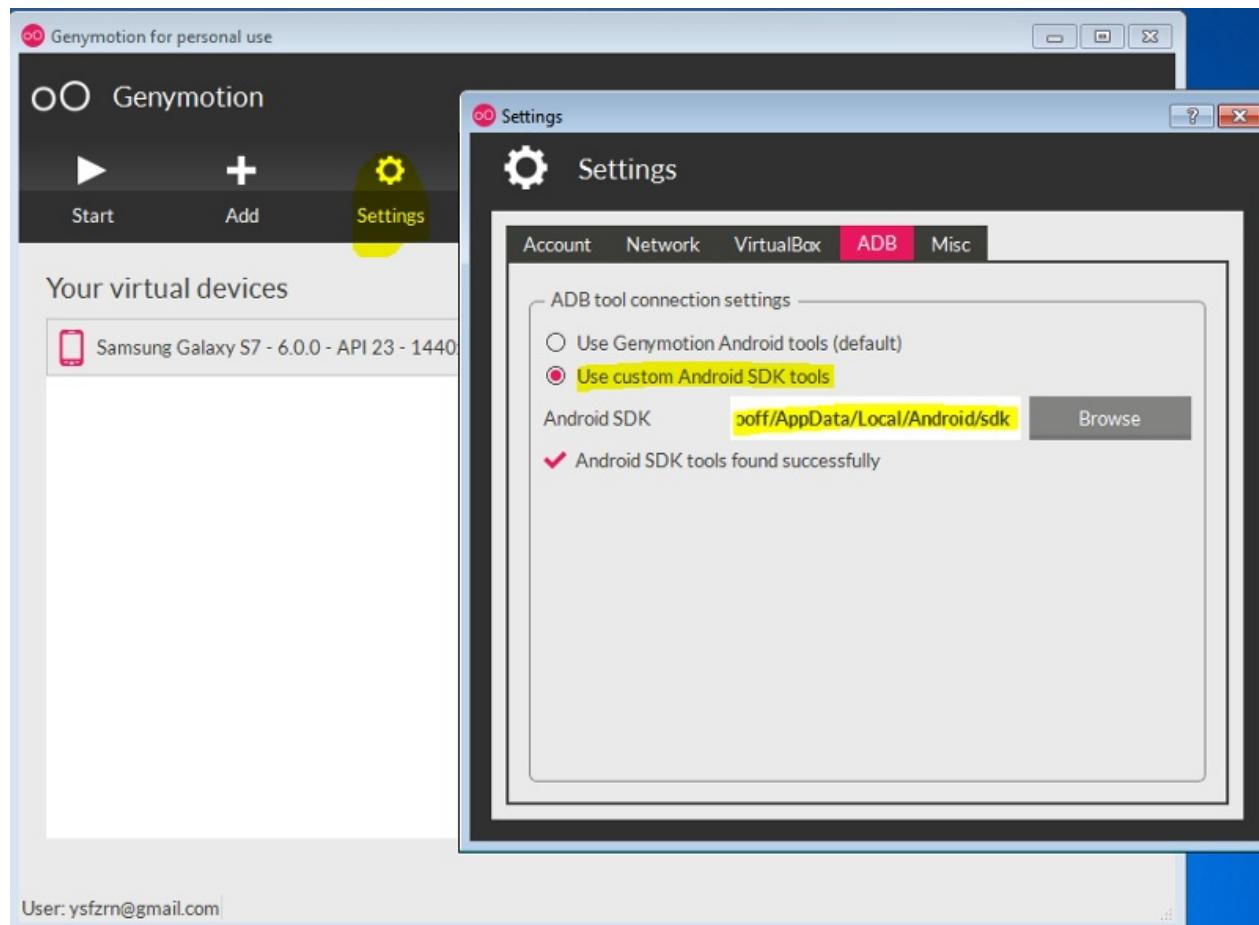
This screenshot shows the download section of the Genymotion website. It includes download links for Windows (152MB with VirtualBox, 46MB without) and a link to register a license. To the right, system requirements for Microsoft Windows 7, 8/8.1, 10 (32/64 bit), 64-bit CPU, recent GPU, and 400 MB disk space are listed. Below the requirements are two download links for checksums: one for Windows with VirtualBox and one for Windows without VirtualBox.

We are using cookies to provide statistics that help us give you the best experience of our site. By continuing to use the site you are agreeing to our use of cookies. You can find our [Privacy Statement](#).

GenyMotion kurduktan sonra GenyMotion hesabımızla giriş yapıp yeni bir sanal telefon ekleyelim. Ben kurduğumuz SDK ya uygun olarak, Samsung Galaxy S7 'yi tercih ettim.



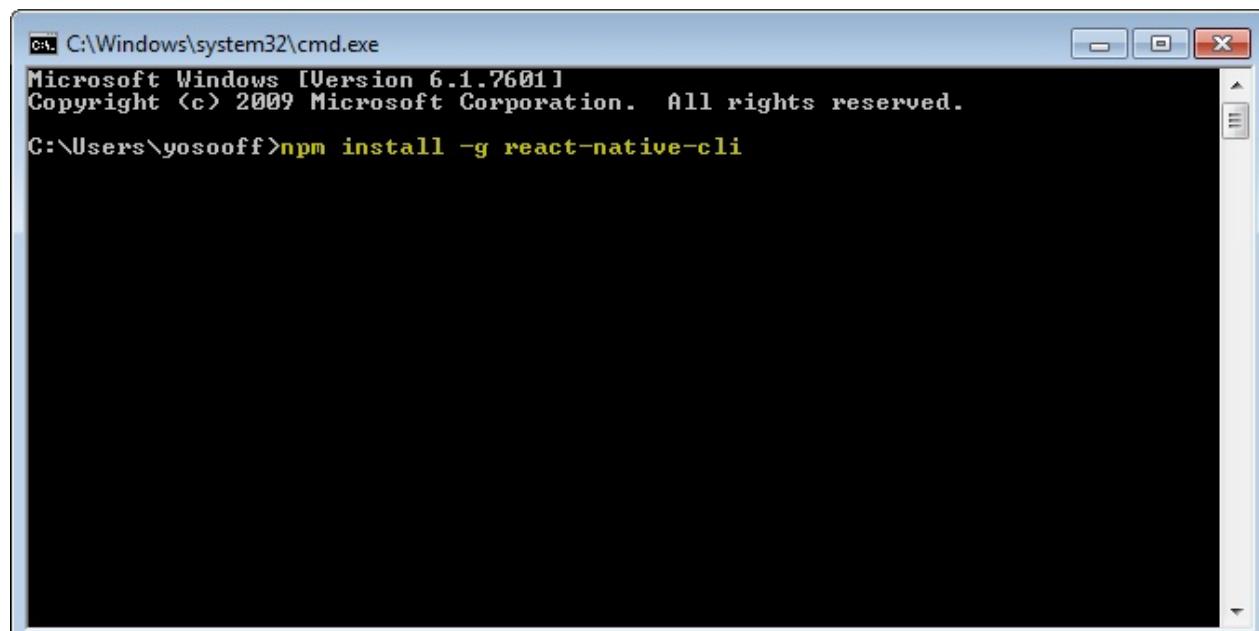
Genymotion muhtemelen kendi SDK'sı ile çalışmayacaktır. Bu yüzden az önce yüklediğimiz Android SDK'yı aşağıdaki gibi Genymotion'ın ayarlarına ekleyelim. Bundan sonra Start komutu ile Android cihazımızı çalıştırabiliriz.



## 4-React-Native CLI Kurulumu

İşin zor kısmını atlattık. Şimdi Javascript dünyasına nihayet girebiliriz. İlk olarak komut satırına aşağıdaki kodu yazarak “react-native-cli” ‘ı global olarak yükleyelim.

```
npm install -g react-native-cli
```



Sonra ilk react-native projemizi kendi oluşturduğumuz bir workspace klasöründe aşağıdaki npm komutu ile yaratalım.

```
react-native init helloNative
```

```
C:\Windows\system32\cmd.exe
+-- fs.realpath@1.0.0
+-- inflight@1.0.6
| +-- wrappy@1.0.2
+-- inherits@2.0.3
+-- minimatch@3.0.3
| +-- brace-expansion@1.1.6
| | +-- balanced-match@0.4.2
| | +-- concat-map@0.0.1
| +-- once@1.4.0
| +-- path-is-absolute@1.0.1
+-- winston@0.8.3
+-- colors@0.6.2
+-- cycle@1.0.3
+-- eyes@0.1.8
+-- isstream@0.1.2
+-- pkginfo@0.3.1
+-- stack-trace@0.0.9
-- semver@5.3.0

C:\Users\yosooff>mkdir workspace
C:\Users\yosooff>cd workspace
C:\Users\yosooff\workspace>react-native init helloNative
```

GenyMotion'a yüklediğimiz sanal cihazımızı çalıştıralım ve proje klasörümüzüne komut satırıyla girip aşağıdaki kod ile projemizi sanal cihazımızda çalıştırıralım.

```
react-native run-android
```

```
C:\Users\yosooff\workspace\helloNative>react-native run-android
Starting JS server...
'adb' is not recognized as an internal or external command,
operable program or batch file.
Building and installing the app on the device <cd android && gradlew.bat installDebug>...
ERROR: JAVA_HOME is not set and no 'java' command could be found in your PATH.

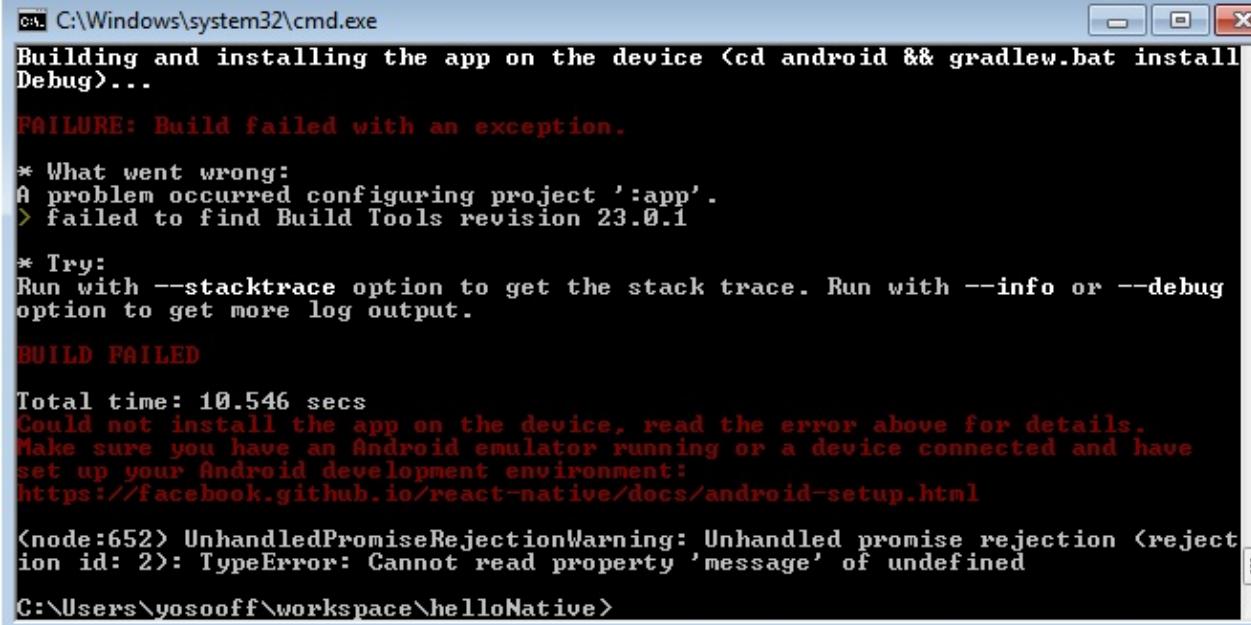
Please set the JAVA_HOME variable in your environment to match the
location of your Java installation.
Could not install the app on the device, read the error above for details.
Make sure you have an Android emulator running or a device connected and have
set up your Android development environment!
https://facebook.github.io/react-native/docs/android-setup.html

<node:4076> UnhandledPromiseRejectionWarning: Unhandled promise rejection <rejec
tion id: 2>: TypeError: Cannot read property 'message' of undefined
C:\Users\yosooff\workspace\helloNative>
```

**Hata :** Bilgisayarımızda JAVA ve JDK yüklü değil.

**Çözüm:** [Şuradaki](#) adresinden son sürüm JDK'yi indirebilirsiniz.

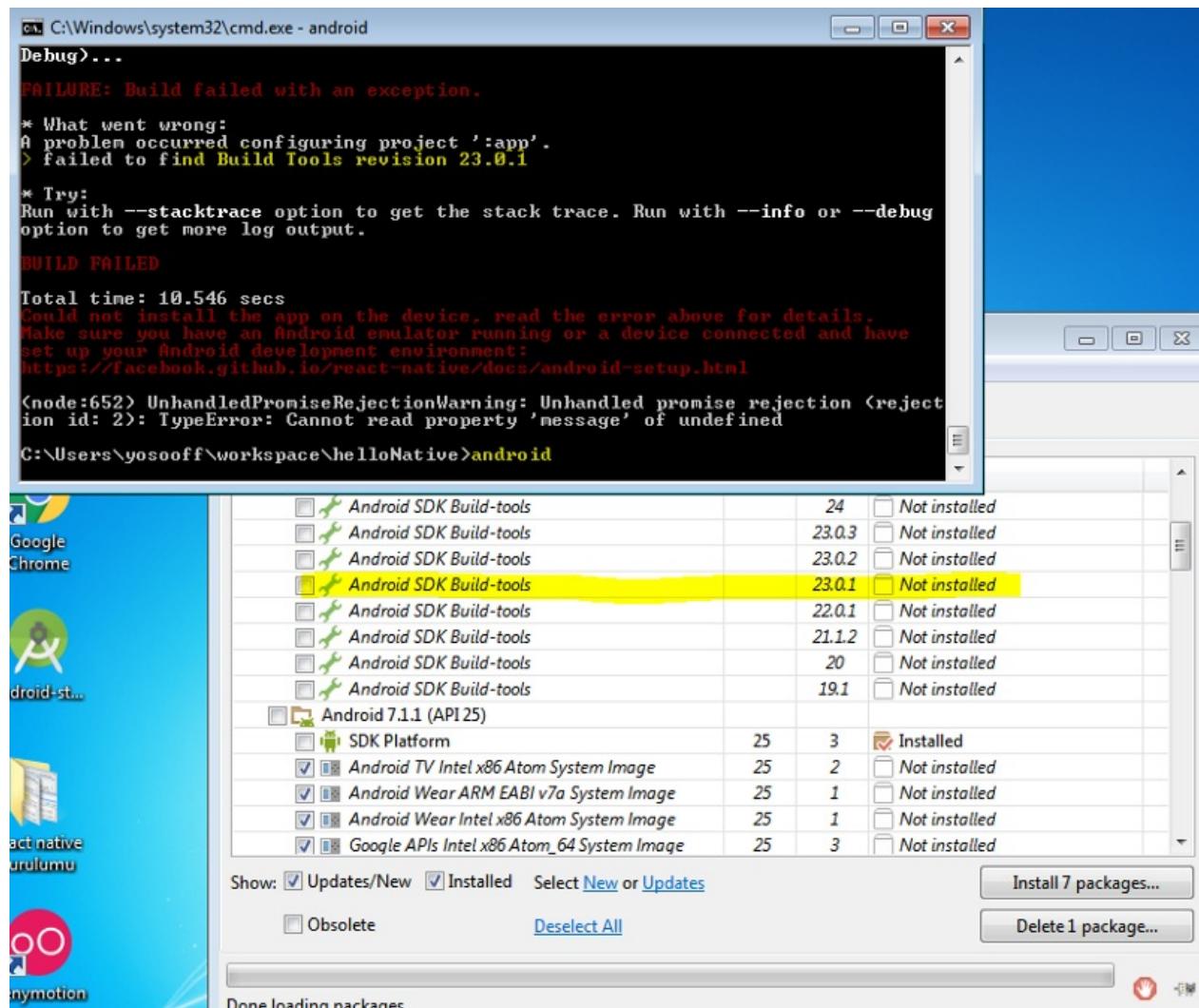
JDK'yi kurduktan sonra, tekrar react-native run-android komutu ile projemizi çalıştırmayı deneyelim.



The screenshot shows a Windows Command Prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The command entered was 'Building and installing the app on the device <cd android && gradlew.bat installDebug>...'. The output indicates a failure: 'FAILURE: Build failed with an exception.' It details the error: 'A problem occurred configuring project ':app'. > failed to find Build Tools revision 23.0.1'. It suggests trying with '--stacktrace' for the stack trace or '--info' or '--debug' for more log output. The build failed with a total time of 10.546 seconds. It also states that the app could not be installed on the device due to missing tools and provides a link for setup: <https://facebook.github.io/react-native/docs/android-setup.html>. A warning message from Node.js is shown: '(node:652) UnhandledPromiseRejectionWarning: Unhandled promise rejection [rejection id: 2]: TypeError: Cannot read property 'message' of undefined'. The path 'C:\Users\yosooff\workspace\helloNative>' is visible at the bottom.

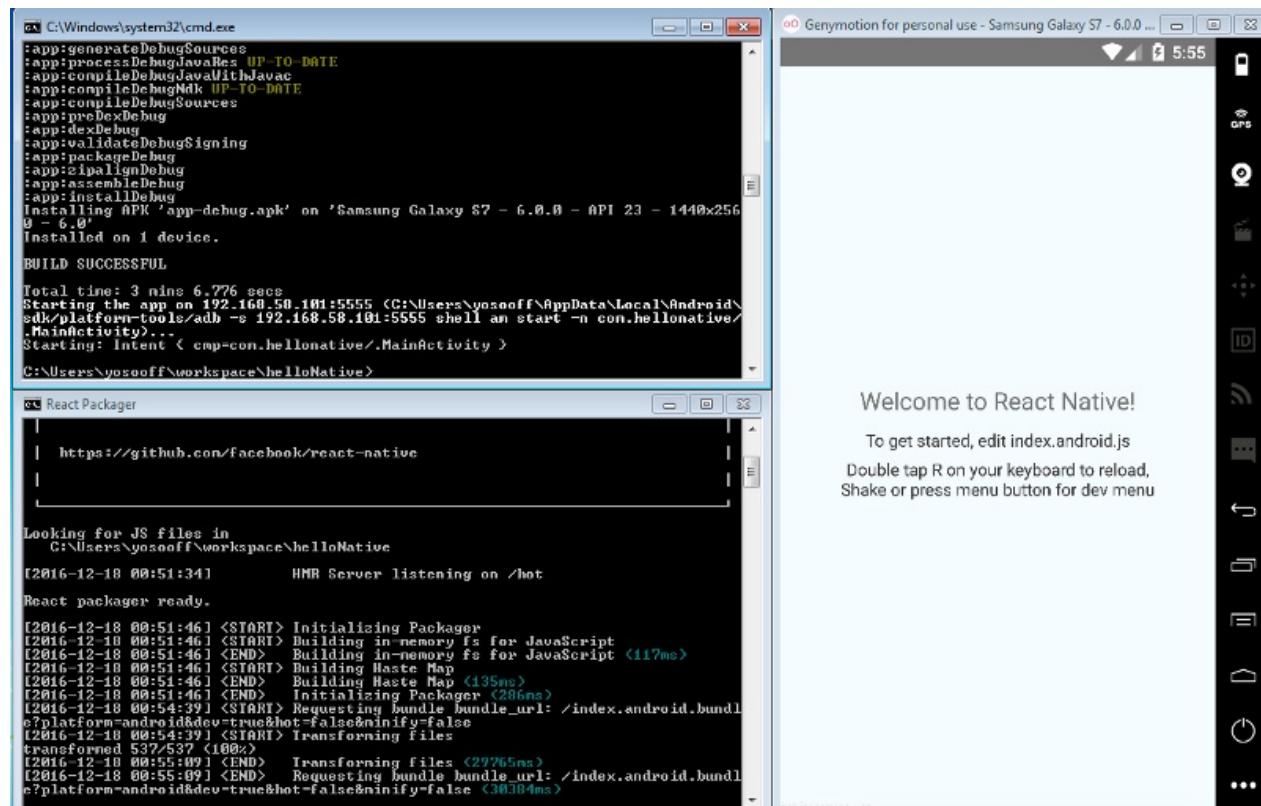
**Hata:** Build Tools 23.0.1'i yüklememişiz.

**Çözüm:** Komut satırına "android" komutu yazarak SDK manager'i açıp, eksik olan tool'u (Build Tools 23.0.1) kuralım.

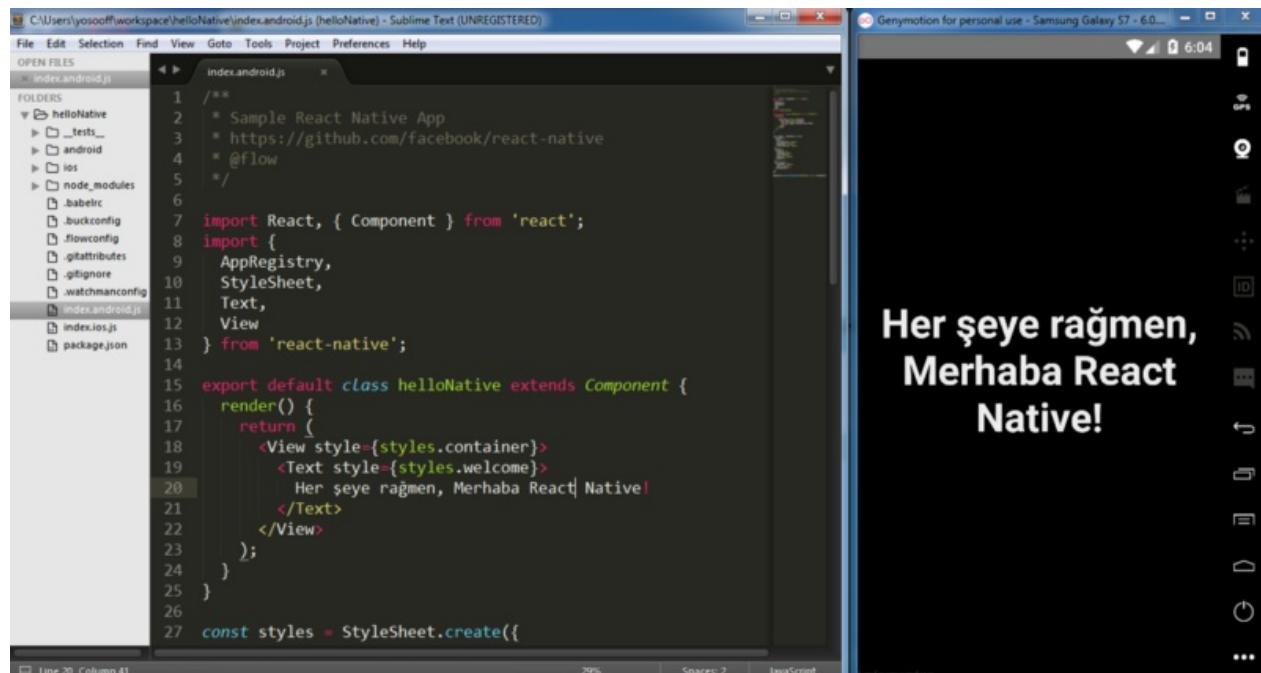


Evet son sorunumuzu da çözdükten sonra tekrar **react-native run-android** komutunu çalıştırıralım.

Ve nihayet çalıştık.



Ve şimdi aşağıda olduğu gibi, uygulamamızı yazabiliriz.



Kendi kurulumunuzda sorun yaşarsanız, yorumlarda yardımcı olmaya çalışırım.

*Not: React Native versiyonu olarak bu yazının yazıldığı an itibariye son sürüm olarak, 0.39.2 yi kurduk.*

# macOS Installation

MacOS yüklü bir bilgisayarda react-native uygulaması yazmak için yapmamız gerekenler;

- Bilgisayarınızda güncel bir [Homebrew](#) yüklü mü kontrol edin. Güncel değil ise [buradan](#) yazılan komutları takip ederek yükleyin.
- Daha sonra Homebrew kullanarak, [Node](#) ve [Watchman](#) yükleyin.

```
brew install node  
brew install watchman
```

- Eğer bilgisayarınızda zaten node yüklü ise, node versiyonunuzun 4 veya daha üstü olması gerekmektedir. Bunun için komut ekranına `node` yazarak girin ve çıkan node ekranında `process.version` kodunu çalıştırarak node'un versyonunu kontrol edin.
- React Native projelerini baştan yaratmadan kullanacağımız, react-native-cli aracını global olarak bilgisayarınıza, aşağıdaki komutu kullanarak indirin.

```
npm install -g react-native-cli  
  
//eğer yarn kullanıyorsanız  
yarn global add react-native-cli
```

- Daha sonra [Mac App Store](#)'a girerek Xcode'u bilgisayarınıza indirin ve kurun. Xcode'u kurduğunuzda, otomatik olarak IOS simulator'u ve ihtiyacınız olan native tüm araçları zaten kurmuş olacaksınız.
- Eğer bilgisayarınızda zaten Xcode yüklü ise versiyonunun 8 veya daha üzerinde olduğuna emin olun.

## Hepsi bu kadar :)

Şimdi komut satırına, `react-native init merhabaReactNative` yazarak ilk projenizi yaratabilirsiniz.

Daha sonra `cd merhabaReactNative` yazarak projenize girip

`react-native run-ios` komutu ile projenizi simulatorde çalıştırabilirsiniz.

# Gnu/Linux Installation

Bu kurulum anlatılımı, popüler Linux dağıtıımı olan Ubuntu işletim sistemine göre uyarlanmıştır. Sudo komutu ile başlayan komutlardan sonra şifrenizi girmenizi isterse giriniz.

## 1 - Nodejs ve NPM kurulumu

Linux için kurulum yapmak için öncelikle terminal'i (Türkçe sistemlerde ucbirim) açıyoruz. Ardından Ubuntu repositorylerinin (Yazılımların adreslerinin bulunduğu adres depoları) güncellemek için şu komutu terminale giriyoruz.

```
sudo apt-get update
```

Şimdi repolarımızı güncellediğimize göre nodejs ve npm (nodejs package manager) kurulumuna başlayabiliriz. Aşağıdaki komuta birde resimde bulunmayan yüklemeyi otomatik onaylamak için -y komutu ekledim. Kurulum için şu komutu terminalde çalıştırın.

```
sudo apt-get install -y nodejs npm
```

```
haydar@haydar-Inspiron-3542: /usr/share/applications
haydar@haydar-Inspiron-3542:/usr/share/applications$ sudo apt-get install nodejs npm
[sudo] password for haydar:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  gyp javascript-common libjs-inherits libjs-jquery libjs-node-uuid libjs-underscore
  libssl-dev libssl-doc libssl1.0-dev libssl1.0.0 libuv libuv-dev node-abbrev
  node-ansi node-ansi-color-table node-archy node-async node-balanced-match
  node-block-stream node-brace-expansion node-builtins node-combined-stream
  node-concat-map node-cookie-jar node-delayed-stream node-forever-agent
  node-form-data node-fs.realpath node-fstream node-fstream-ignore
  node-github-url-from-git node-glob node-graceful-fs node-gyp node-hosted-git-info
  node-inflight node-inherits node-ini node-is-builtins node-isexe
  node-json-stringify-safe node-lockfile node-lru-cache node-mime node-minimatch
  node-mkdirp node-mute-stream node-node-uuid node-nopt node-normalize-package-data
  node-npmlog node-once node-osenv node-path-is-absolute node-pseudomap node-qs
  node-read node-read-package-json node-request node-retry node-rimraf node-semver
  node-sha node-slide node-spdx-correct node-spdx-expression-parse
```

Daha sonra npm paketlerimizi yüklerken nodejs'in çağrılrken sorun çıkarmaması için şu komutu terminalde çalıştıralım. Bu komut node ve nodejs ile hard link adı verilen bir ilişki (Windows ortamındaki kısa yol gibi düşünün) oluşturarak iki komut çağrıldığında aynı şekilde çalışmasını sağlar.

```
ln -s /usr/bin/nodejs /usr/bin/node
```

## - React Native CLI Kurulumu

React Native komutlarınımızı çalıştığımız paketi yüklemek için aşağıdaki komutu terminalden çalıştırın.

```
sudo npm install -g react-native-cli
```

```
haydar@haydar-Inspiron-3542:/usr/share/applications$ sudo npm install -g react-native-cli
[sudo] password for haydar:
/usr/local/bin/react-native -> /usr/local/lib/node_modules/react-native-cli/index.js
/usr/local/lib
└── react-native-cli@2.0.1
    ├── chalk@1.1.3
    ├── ansi-styles@2.2.1
    ├── escape-string-regexp@1.0.5
    ├── has-ansi@2.0.0
    └── ansi-regex@2.1.1
        ├── strip-ansi@3.0.1
        └── supports-color@2.0.0
    ├── minimist@1.2.0
    ├── prompt@0.2.14
    ├── pkginfo@0.4.1
    ├── read@1.0.7
    └── mute-stream@0.0.7
        └── revalidator@0.1.8
```

## 2 - JDK Kurulumu

Android emülatörümüzün çalışabilmesi için JDK'nın (java developer kit) bilgisayarımızda kurulu olması gerekiyor. Yalnız öncelikle uyarmak isterim ki React Native JDK olarak OpenJDK değil OracleJDK kurulmasını istiyor. Bu yazının yazıldığı tarihte en güncel Java sürümünün Java 9 ve React Native'in resmi sitesinde Java 8 ve üzeri JDKları desteklediğini söylemesine karşın, Java 9 kurulumundan sonra gradle(ilerde nedir öğrenceksiniz) tabanlı sorunlar olması nedeniyle burada Java 8 JDK anlatılacaktır.

Aşağıdaki kodu terminelden çalıştığımızda oracle java reposunu sistemimize eklemiş oluyoruz.

```
sudo add-apt-repository ppa:webupd8team/java -y
```

Yeni bir repo eklediğimizden ötürü sistemimizi şu komutla güncellememiz gerekiyor. Bu işlem biraz zaman alabilir.

```
sudo apt-get update
```

Sonunda Java kurulumu yapacağımız komutu çalıştırıyoruz. Bu işlem biraz zaman alabilir.

```
sudo apt-get install oracle-java8-installer
```

Şimdi yüklemiş olduğumuz java sürünum sistemin ortam değişkenlerine kaydedeceğiz. Bunun için şu komutu çalıştırın.

```
sudo apt-get install oracle-java8-set-default
```

## 3 - Android Studio Kurulumu

Android SDK'leri (Software development kit / Yazılım geliştirme kiti) ve AVD (Android virtual machine) içeriğinden dolayı Android Studio'yı kurmamız gerekmektedir. İşletim sisteminiz 64bit ise şu paketleri terminalden yüklemeniz gerekmekte.

```
sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386 lib32z1 libbz2-1.0:i386
```

Öncelikle [Buradan](#) güncel Android studio sürümü masaüstümüze indirelim. Sonra masaüstümğe gelip sağ --> Open Terminal sonra aşağıdaki komutu çalıştıralım.

```
unzip indirdiğimiz dosyanın adı -d $HOME
```

Örnek olarak,

```
unzip android-studio-ide-171.4443003-linux.zip -d $HOME
```

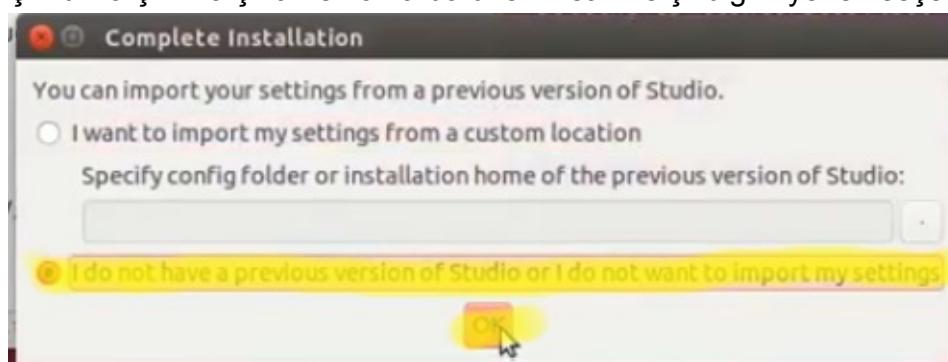
Bu komut indirdiğimiz zip paketinin içesindekileri Home klasörüne çıkartıyor. Şimdi kurulum scriptinin olduğu klasöre gidelim. Bunun için şu kodu çalıştırıralım.

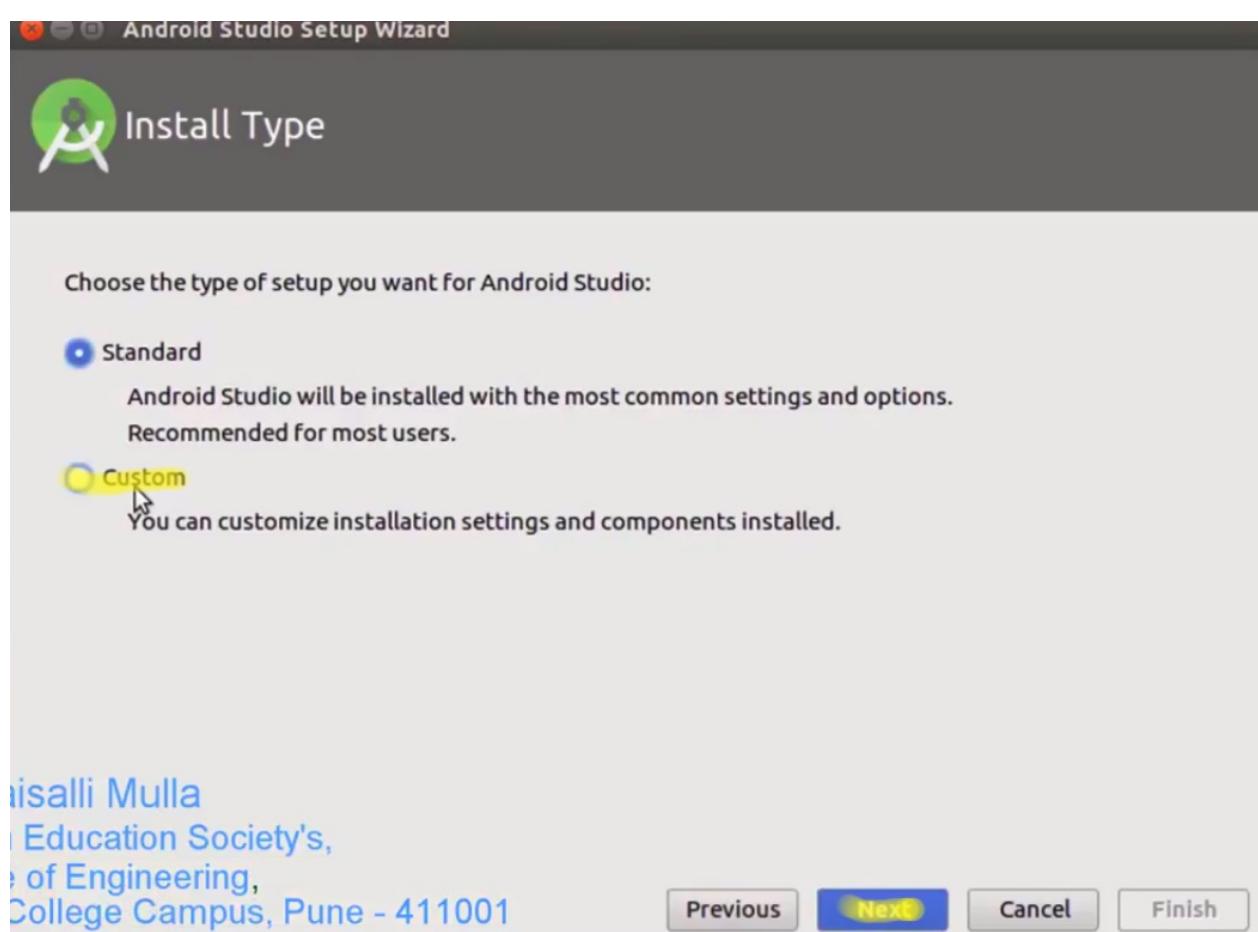
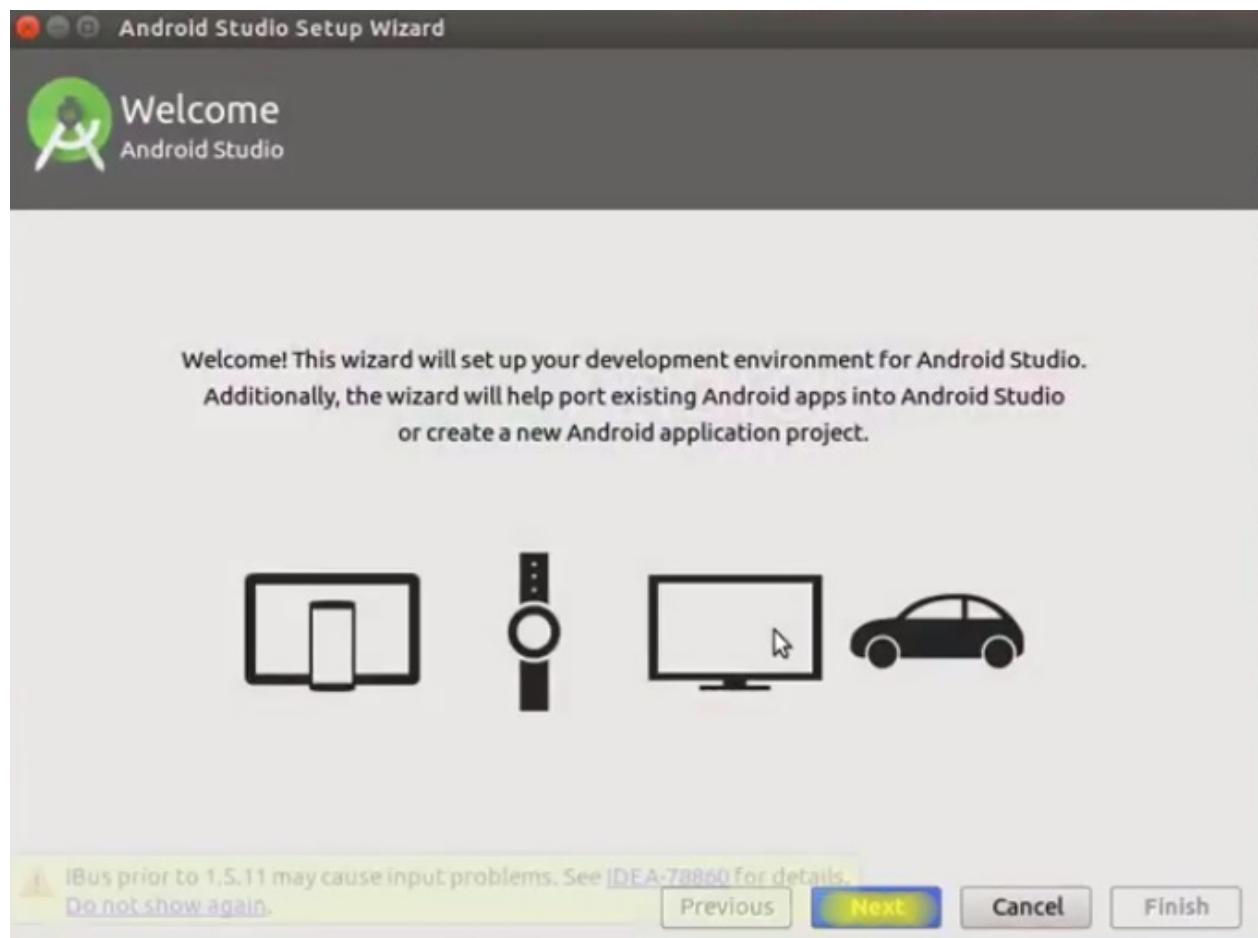
```
cd $HOME/android-studio/bin
```

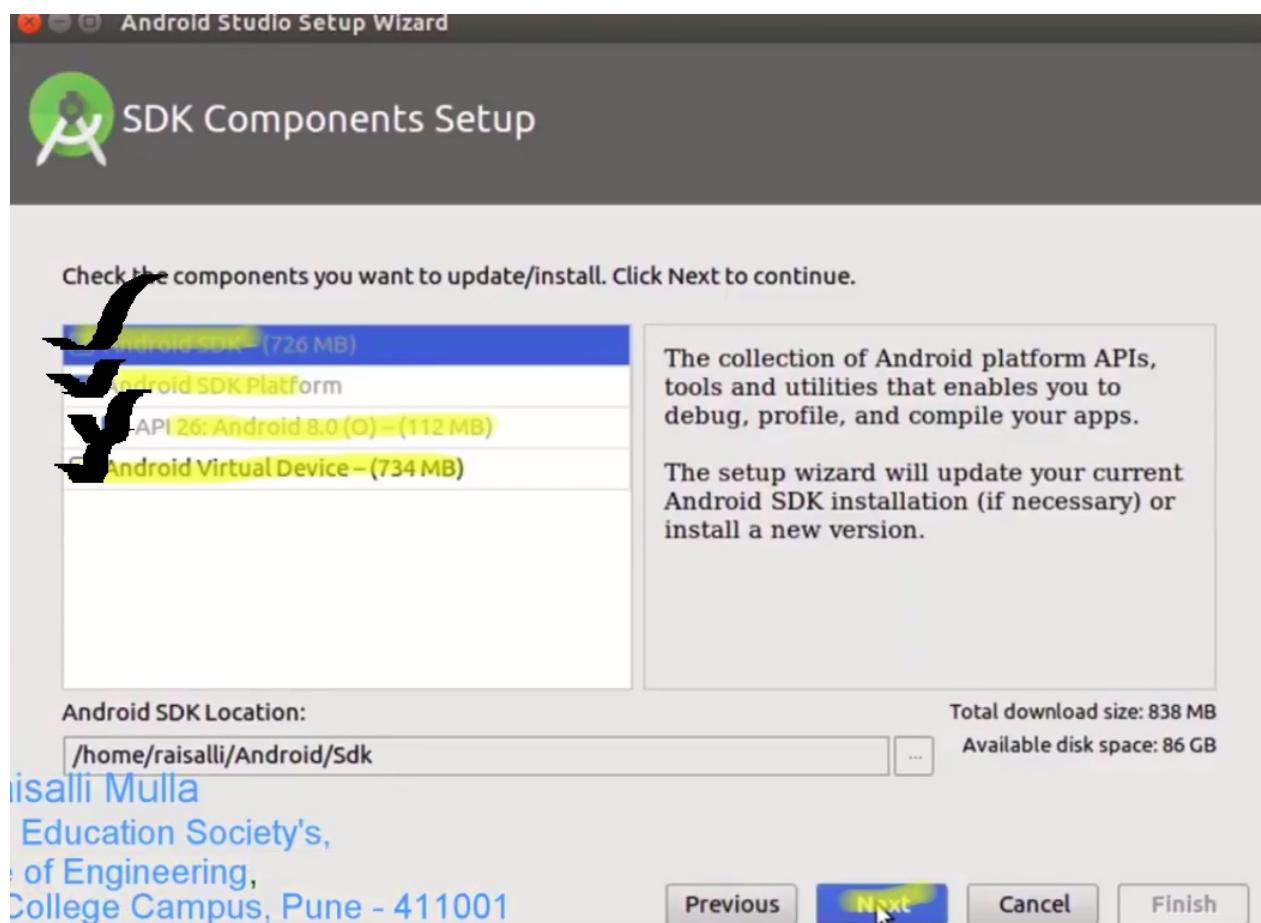
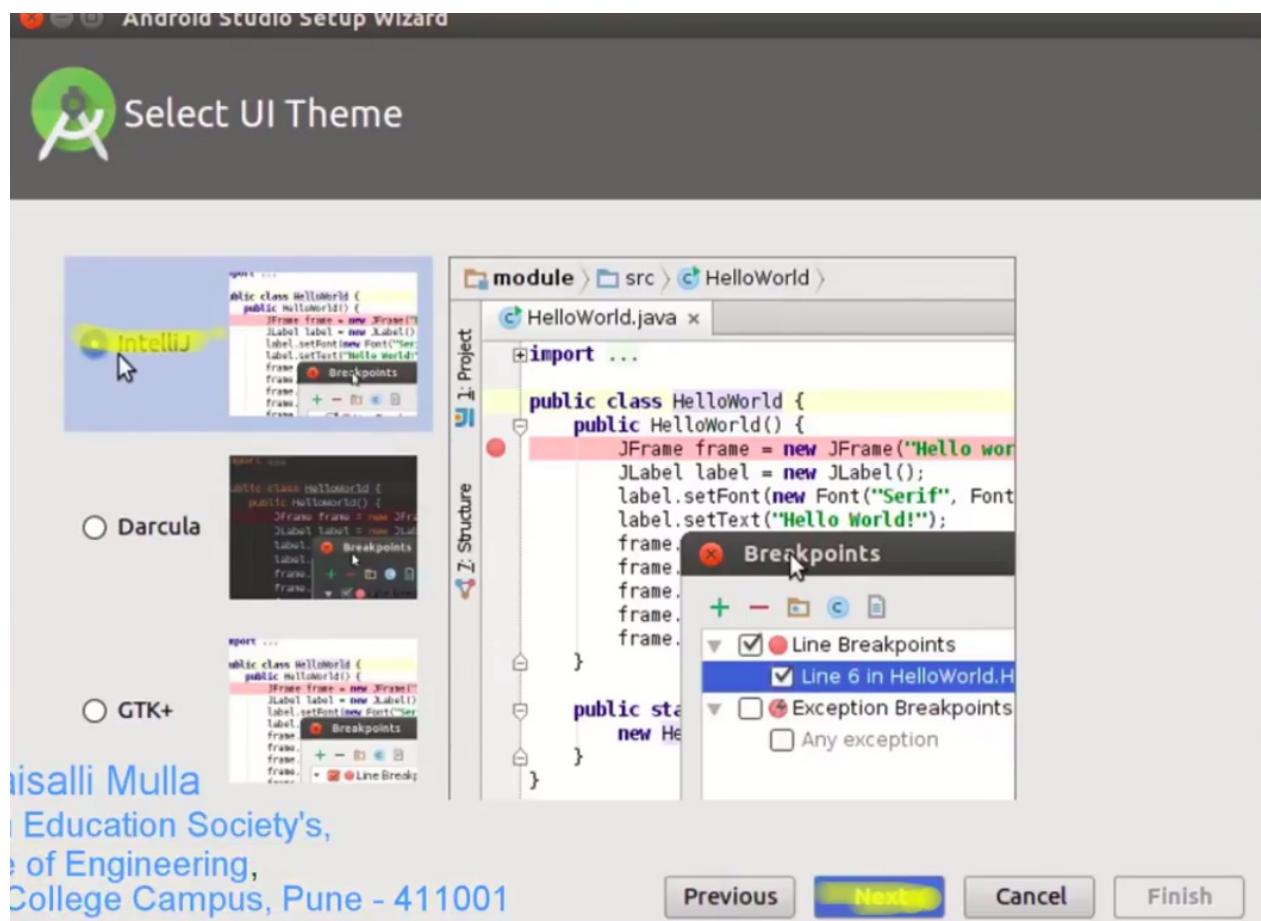
Sıra geldi Scriptimizi çalıştırmaya... Script çalışırken söyleyene kadar terminal penceresini kapatmayın.

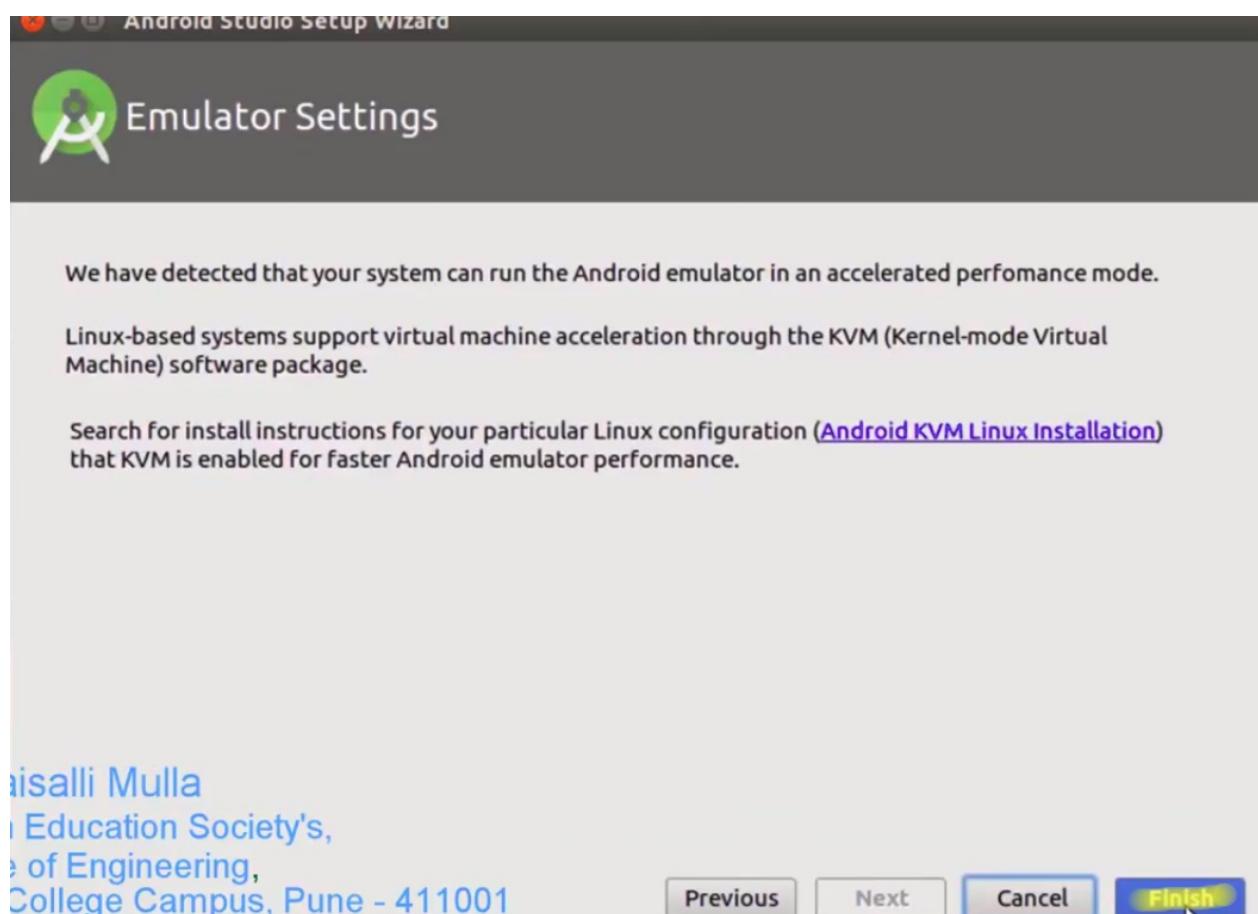
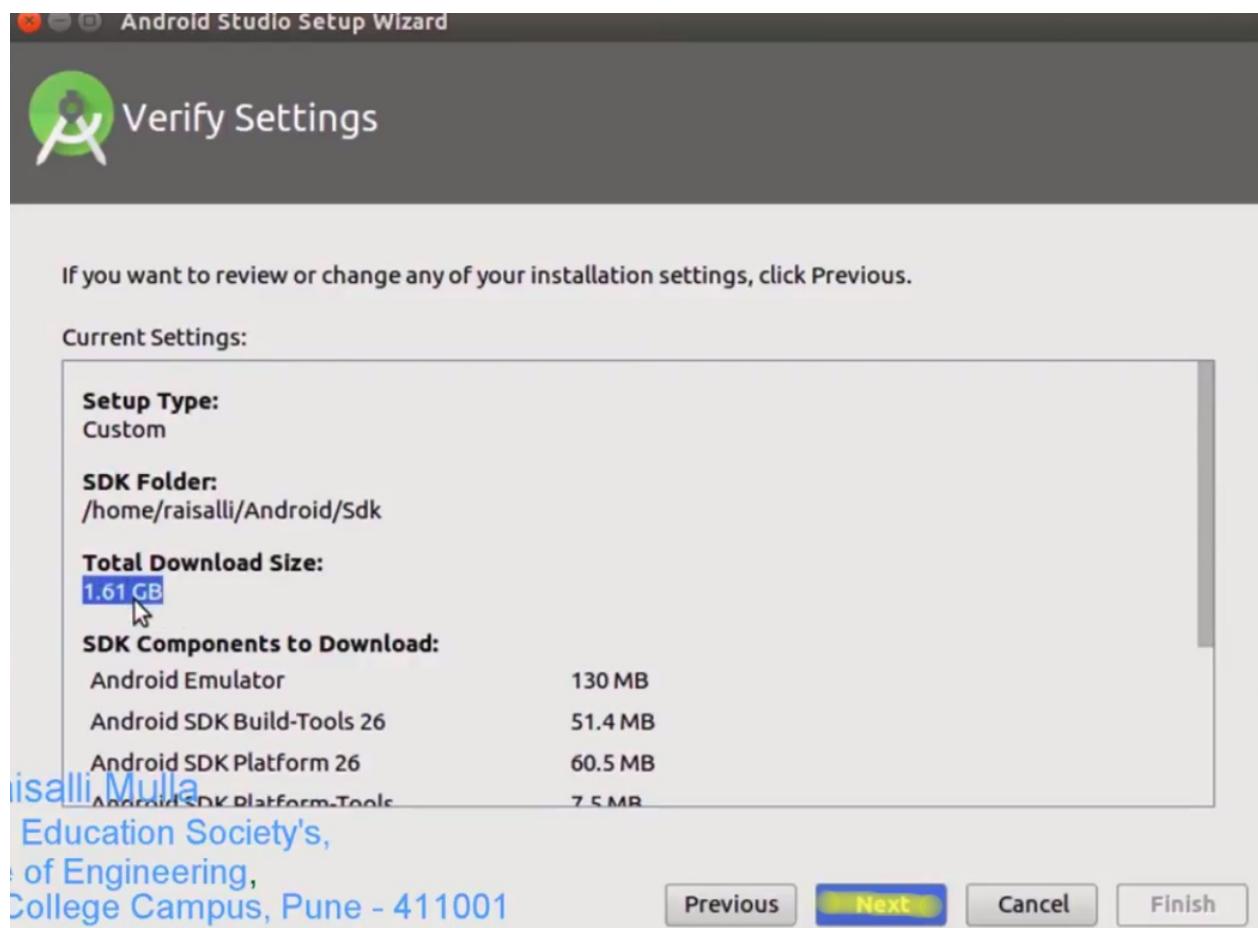
```
./studio.sh
```

Şimdi karşımıza çıkan ekranlarda üzerini sarı ile çizdiğim yerleri seçerek devam ediyoruz.

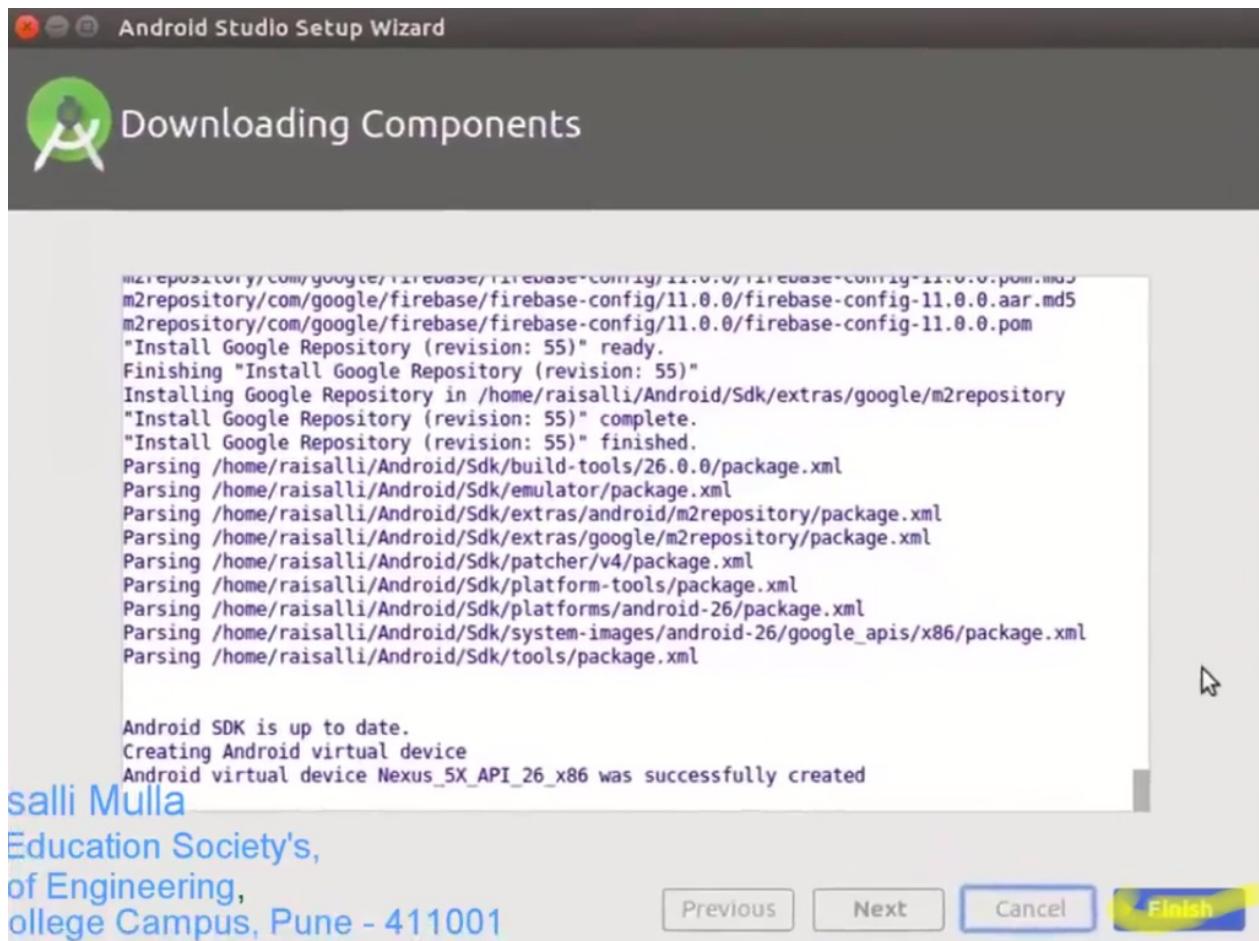




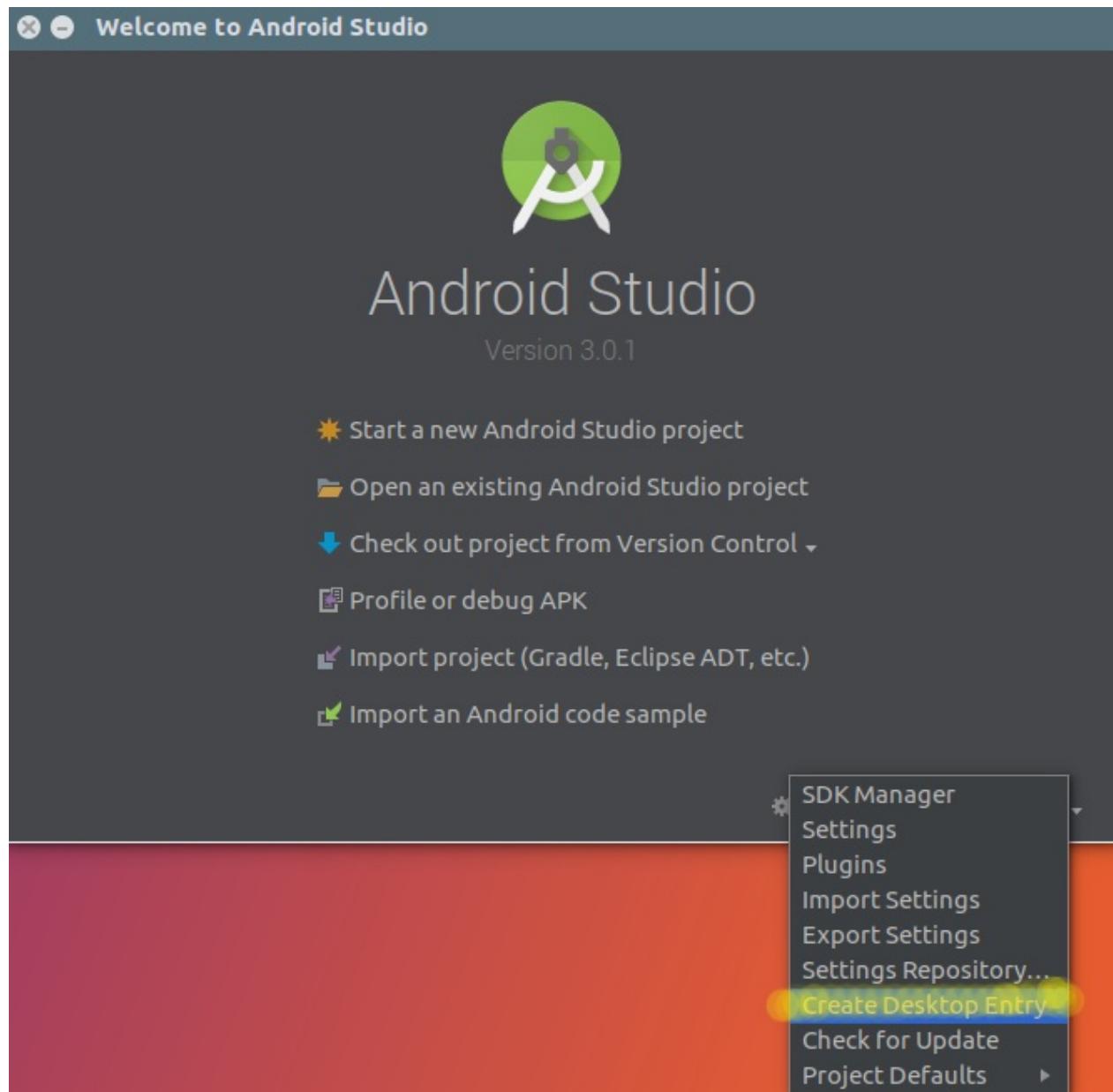




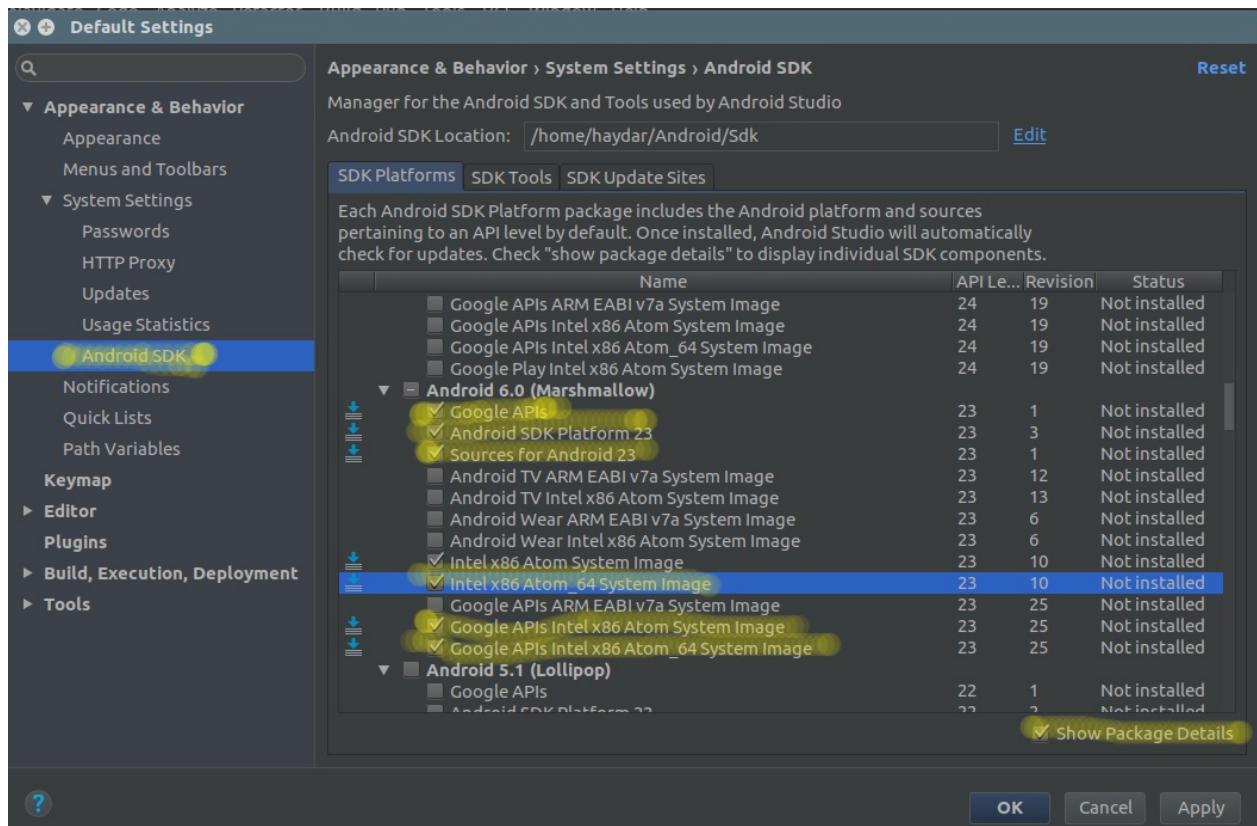
Bu adım internetten veri indirdiği için uzun sürebilir lütfen sabırı olun.



Şimdi terminal penceresini kapatabilirsiniz. Çıkan ekranda Configure --> Create Desktop Entry yoluna tıklayınız. Daha sonra şifrenizi soran pencereye şifrenizi girip ok butonuna basınız.



Artık Android Studio'yı kullanmaya başlayabiliriz. Uygulamalar arasında "Android Studio" diye aratın ve Android Studio logosuna çift tıklayın. Daha sonra Configure --> SDK Manager yoluna tıklayın. Çıkan ekranda önce resimdeki "Show Pakage Details", daha sonra diğer üzeri fosforlanmış yerlere ve sonra ok butonuna tıklayın.

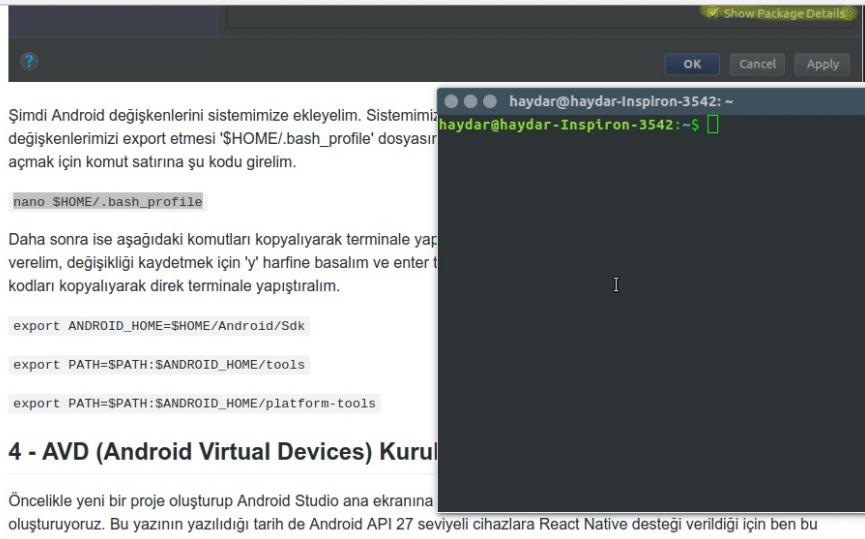


Şimdi Android değişkenlerini sistemimize ekleyelim. Sistemimize her açtığımızda bunu yapmamıza gerek kalmasın diye tüm değişkenlerimizi export etmesi '\$HOME/.bash\_profile' dosyasına yazmamız gereklidir. Öncelikle \$HOME/.bash\_profile dosyamızı açmak için komut satırına şu kodu girelim.

```
nano $HOME/.bash_profile
```

Daha sonra ise aşağıdaki komutları kopyalıyalım ve terminalde yapıştırıralım. Ctrl + X kombinasyonu ile dosyamızı kapatma emri verelim, değişikliği kaydetmek için 'y' harfine basalım ve enter tuşıyla onaylayalım. En sonda tek seferliye mahsus aşağıdaki kodları kopyalıyalım ve direkt terminalde yapıştırıralım.

```
export ANDROID_HOME=$HOME/Android/Sdk
export PATH=$PATH:$ANDROID_HOME/tools
export PATH=$PATH:$ANDROID_HOME/platform-tools
```



Şimdi Android değişkenlerini sistemimize ekleyelim. Sistemimizi değiikenlerimi export etmesi '\$HOME/.bash\_profile' dosyasını açmak için komut satırına şu kodu girelim.

```
nano $HOME/.bash_profile
```

Daha sonra ise aşağıdaki komutları kopyalyarak terminale yapıştırın, değişikliği kaydetmek için Y' harfine basalım ve enter tuşuna basıp komutları kopyalyarak direk terminale yapıştırılsın.

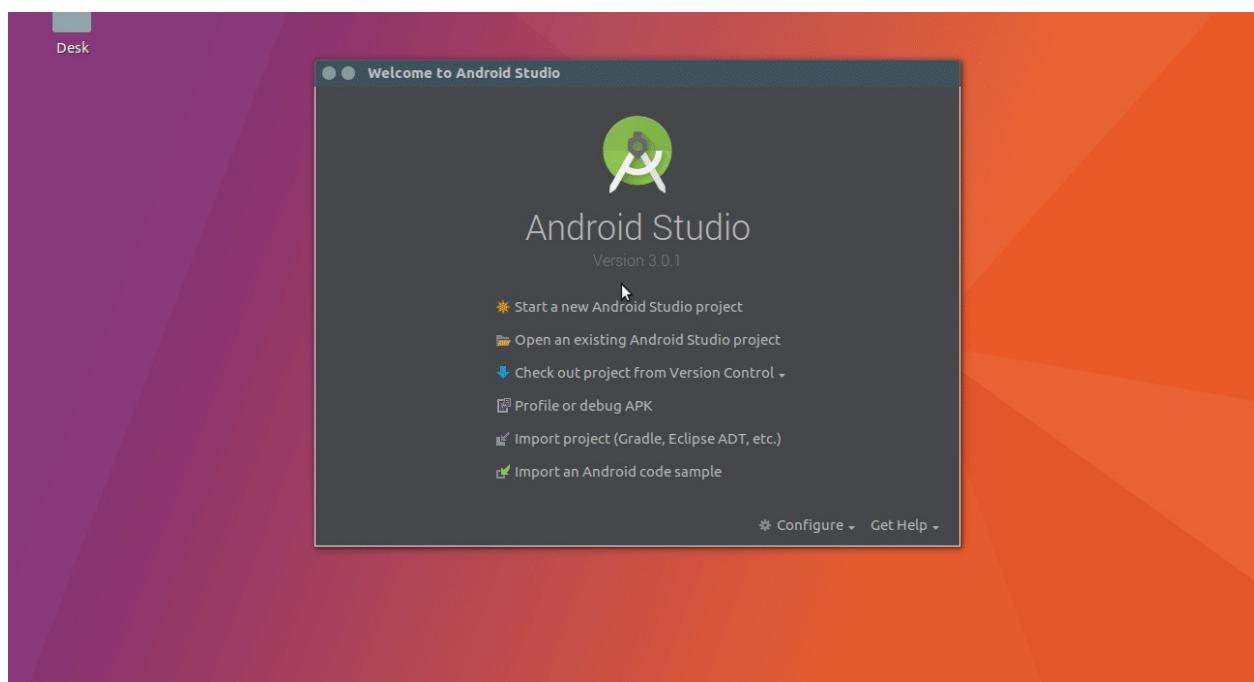
```
export ANDROID_HOME=$HOME/Android/Sdk
export PATH=$PATH:$ANDROID_HOME/tools
export PATH=$PATH:$ANDROID_HOME/platform-tools
```

**4 - AVD (Android Virtual Devices) Kurulumu**

Öncelikle yeni bir proje oluşturup Android Studio ana ekranına ulaşıyoruz. Bu yazının yazılıdığı tarih de Android API 27 seviyeli cihazlara React Native desteği verildiği için ben bu seviyede bir AVD oluşturuyorum. Eğer bir sürüm bazlı bir sorun yaşarsanız Android 6.0 (API 23) bir AVD ile React Native uygulamanızı ayağa kaldırmayı deneyin. Ben kurulum esnasında API 27'yi bilgisayarımı indirdiğim için burada indirme yapmadan direkt geçebildim. Eğer yaratmak istediğiniz sanal cihaz seviyesinin üzerinde 'Download' yazısı varsa onun üzerine tıklayıp indirmelisiniz.

## 4 - AVD (Android Virtual Devices) Kurulumu

Öncelikle yeni bir proje oluşturup Android Studio ana ekranına ulaşıyoruz. Daha sonra AVD simgesine tıklayıp yeni bir avd oluşturuyoruz. Bu yazının yazılıdiği tarih de Android API 27 seviyeli cihazlara React Native desteği verildiği için ben bu seviyede bir AVD oluşturuyorum. Eğer bir sürüm bazlı bir sorun yaşarsanız Android 6.0 (API 23) bir AVD ile React Native uygulamanızı ayağa kaldırmayı deneyin. Ben kurulum esnasında API 27'yi bilgisayarımı indirdiğim için burada indirme yapmadan direkt geçebildim. Eğer yaratmak istediğiniz sanal cihaz seviyesinin üzerinde 'Download' yazısı varsa onun üzerine tıklayıp indirmelisiniz.



## 5 - İlk Projemizi Oluşturalım

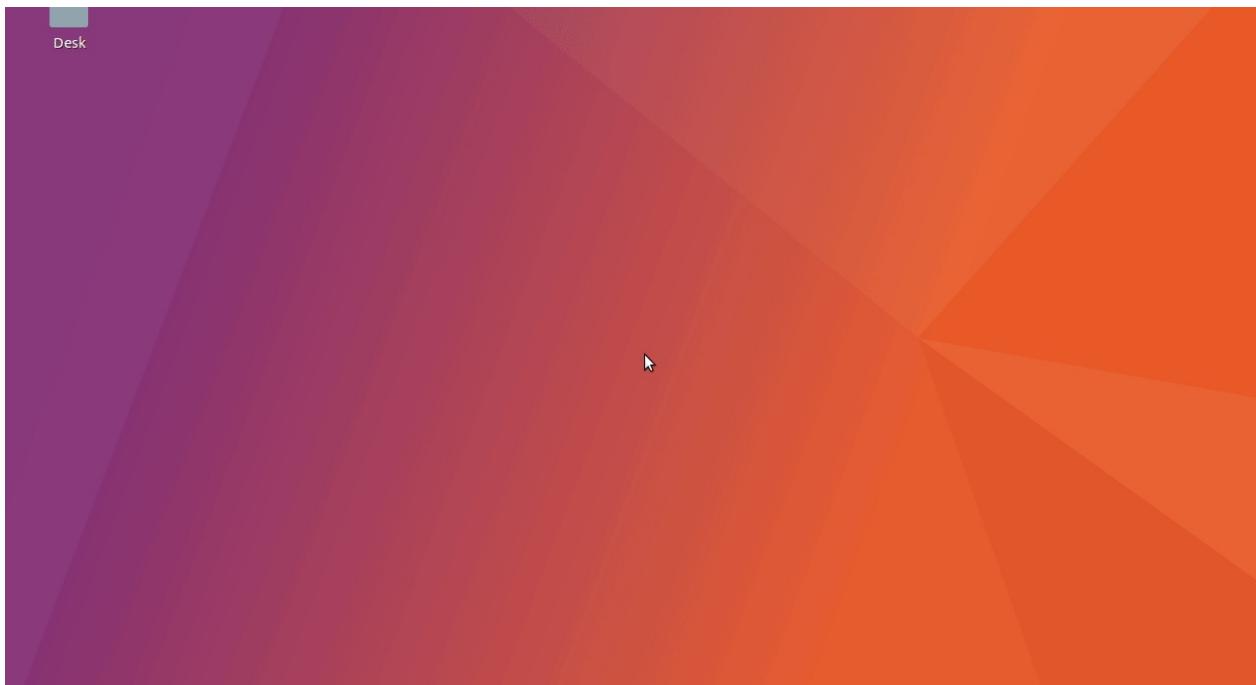
Öncelikle terminal ekranınızı açın. Daha sonra şu komutları çalıştırın. Bu komut Home dizinize 'helloNative' adlı bir klasör açıp içine React Native için gerekli dosyaları koyacaktır. İkinci komut ise proje dosyamızın içine girmenizi sağlayacaktır.

```
react-native init helloNative
```

```
cd helloNative
```

### - AVD'yi çalıştırma

AVD'nin açılması sisteminizin özelliklerine göre uzun sürebilir ana ekran gelene kadar bekleyin. Performans için bilgisayarınızda çalışan gereksiz programları kapatın. AVD çalışmaya başladıkten sonra Android Studio'yu kapatmanız sisteminizi hızlandıracaktır.



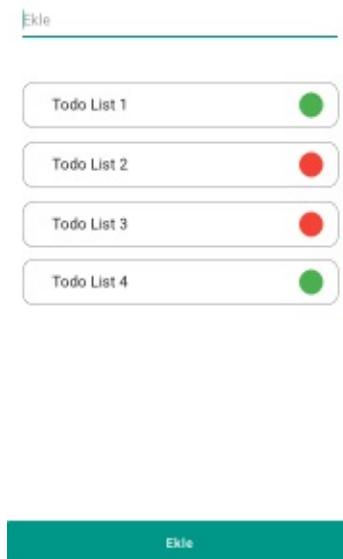
AVD tamamen açıldıysa artık projemizi attach edebiliriz. Bunun için terminale şu komutu yazalım.

```
react-native run-android
```

# REACT NATIVE BAŞLANGIÇ

Şimdi bir tane işin içine Redux, Navigation, rest karıştırmadan klasik olarak basit bir Todo List uygulamasını React Native'de adım adım yapalım.

Uygulamamızın görseli şöyle olsun.



Global olarak RN cli'yi yüklememişseniz

```
npm install -g react-native-cli
```

Projenizde bir üst versiyona geçmek için `react-native upgrade` komutunu çalıştırabilirsiniz.

Workspace klasörümüzde projemizi başlatalım

```
react-native init mytodolist
```

Projemizin içine girelim (bunu yazıyorum çünkü olduğu yerde run etmeye çalışıp hata alıyor diyenler oluyor )

```
cd mytodolist
```

Projemizi run ederken, aşağıdaki komutları kullanıyoruz.

Android için ( Genymotion i veya emulatorü açdın mı, java adresini doğru verdin mi ? )

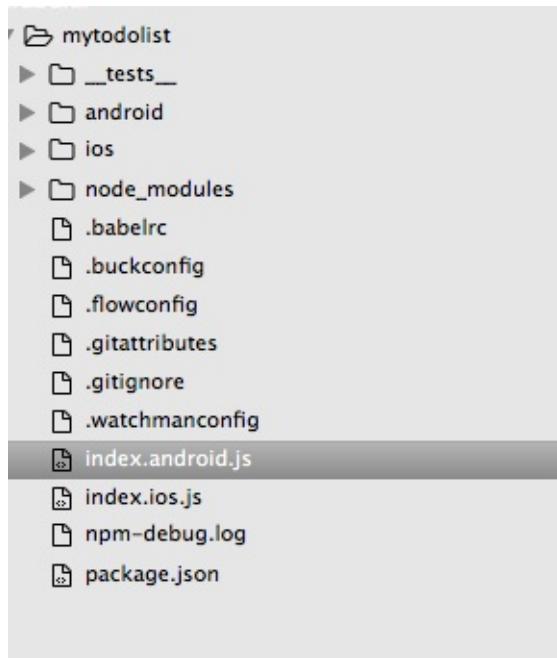
```
react-native run-android
```

IOS için ( Xcode yüklü ise sadece aşağıdaki kodu çalıştır )

```
react-native run-ios
```

Şimdi ilk yapmamız gereken Hot ve Live Reloading' leri telefonun seçenekler menusunden açmak. IOS'da cmd+R ile de açabilirsiniz.

React Native CLI , aşağıdaki gibi bir proje verecek bize.



Burada android ve ios klasörlerimiz var. Olaya native deneyimi yaşatan kısım burası. Başta bahsettiğimiz gibi bu klasörler içinde bize native componentleri React Native, bir köprü ile bize sunuyor. Bu klasörlere nasıl dokunacağımıza sonra bakacağız.

Burada `index.android.js` ve `index.ios.js` isimli iki dosya görüyoruz. React Native aynı isimli iki dosyayı uzantısı `.ios.js` ise IOS da, `.android.js` ise Android de derler. Bu ne işimize yarar derseniz. Diyelimki siz bir tane `navigationBar` isimli bir component yaptınız. IOS ve Android de farklı davranışını bekliyorsunuz. Normalde `navigationBar.js` diye bir componente bu farklı davranışları tanımladığınızda ortaya çok karmaşık bir yapı çıkıyorsa `navigationBar.ios.js` ve `navigationBar.android.js` olarak ikiye bölün ve rahat rahat `import NavigationBar from './navigationBar'` olarak çağrırip kullanın.

Hem android hem de IOS da çalışmasını beklediğimiz kod bloğunu bir klasörde toplayalım. `index.android.js` ve `index.ios.js` de de sadece bu en üst componenti import edelim.

**index.android.js || index.ios.js**

```

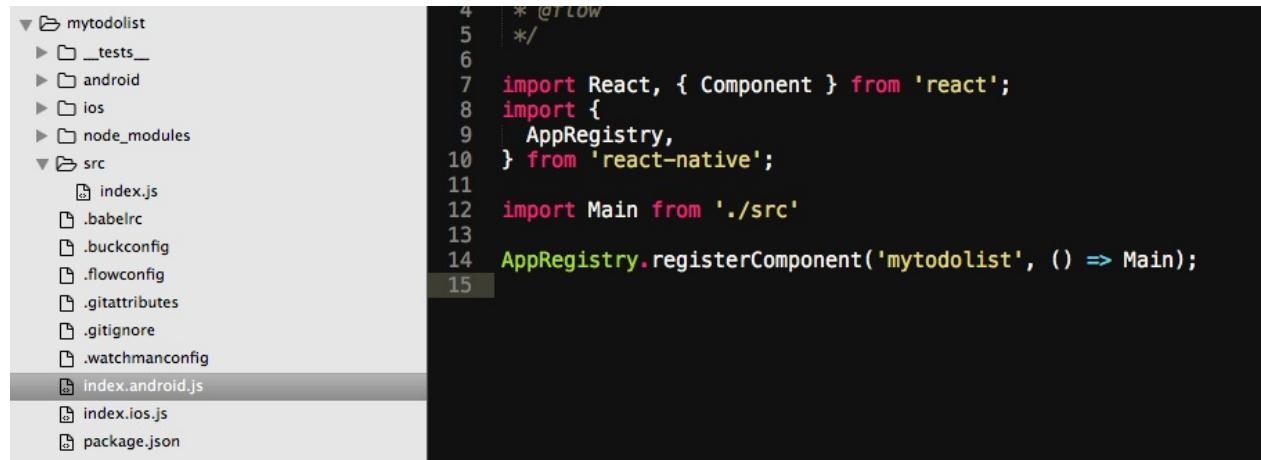
import React, { Component } from 'react';
import {
  AppRegistry,
} from 'react-native';

import Main from './src'

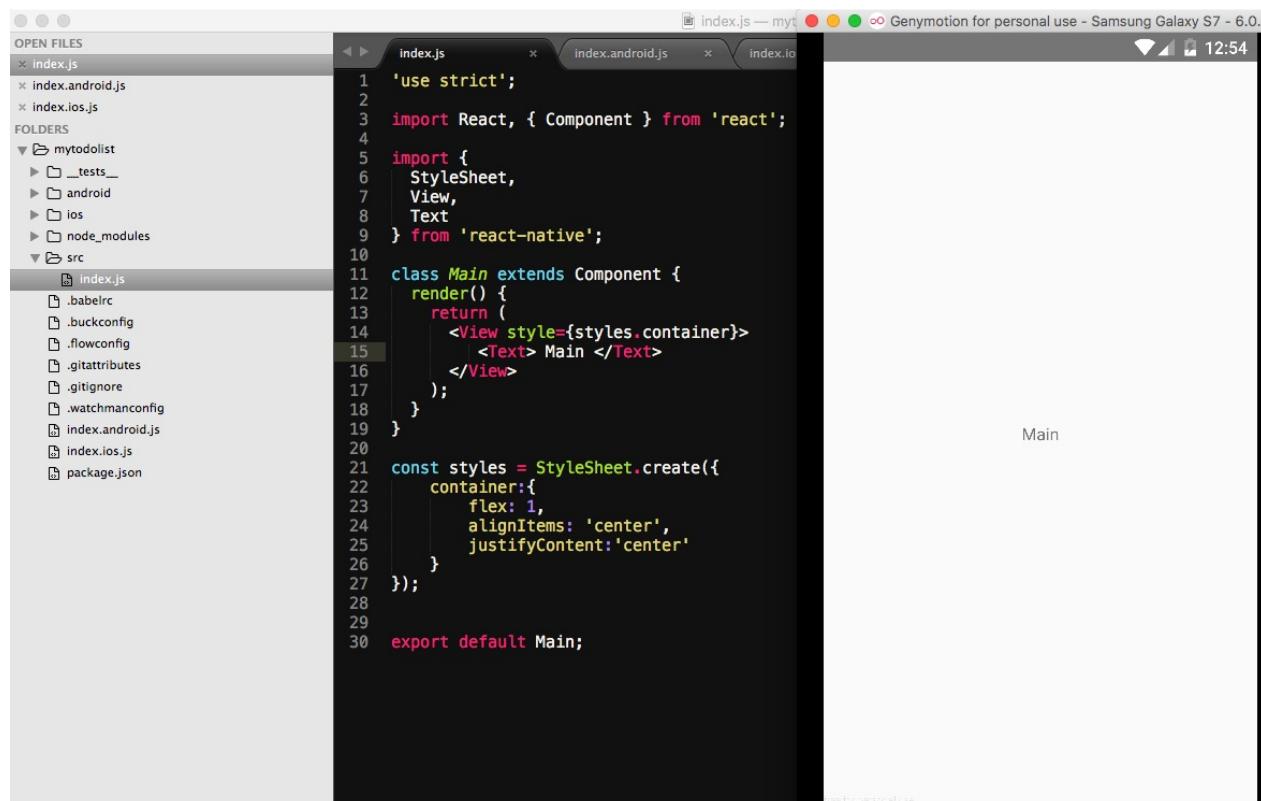
AppRegistry.registerComponent('mytodolist', () => Main);

```

Yeni klasör yapımız aşağıdaki gibi oldu,



Bir sonraki adımlımızda Main componentimizi aşağıdaki gibi düzenlemeye başlayabiliriz. En yukarıdaki görseli elde etmek için biraz flexbox olayına el atalım.





# Style ve FlexBox

CSS'de kullandığımız, `display:flex` ile benzer.

	<b>Options (Default:Bold)</b>	<b>Description</b>
<code>flexDirection</code>	<code>row, column</code>	<b>(column)</b> seçerseniz elementleri dikey, <b>(row)</b> seçerseniz elementleri yatay sıralar. Bu seçim çok önemli çünkü bizim <b>primary axis</b> 'imiz ne olacak onu belirler. Diğer <b>secondary axis</b> olur
<code>justifyContent</code>	<code>flex-start, center, flex-end, space-around, space-between</code>	Child elementlerimiz <b>primary axis</b> boyunca Container içinde nasıl dağılsın.
<code>alignItems</code>	<code>flex-start, center, flex-end, stretch</code>	Child elementlerimiz <b>secondary axis</b> boyunca Container içinde nasıl dağılsın.

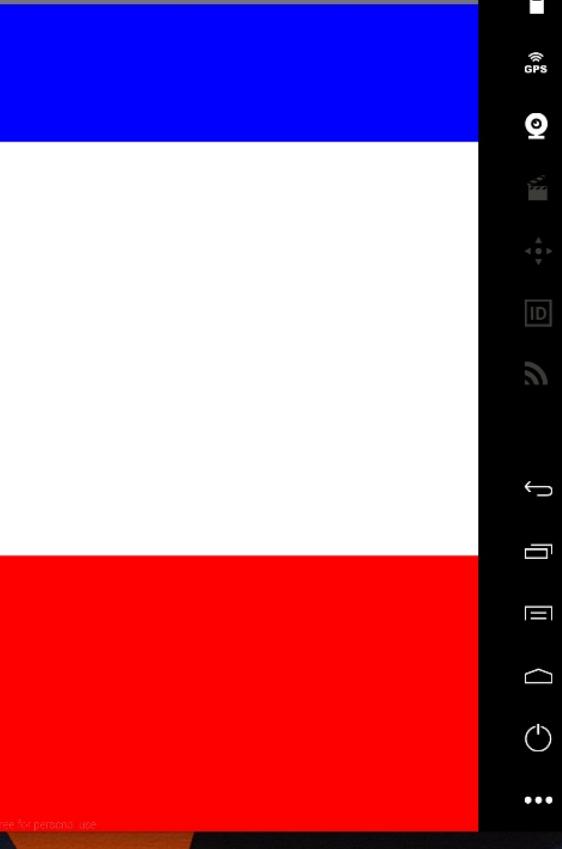
```
import { StyleSheet } from 'react-native';

const styles = StyleSheet.create({
  container: {
    flex: 1
  }
});
```

React Native de style tanımlamalarını, kendi kütüphanesinin içinde bulunan **StyleSheet** ile yapıyoruz. Bilindik bir obje tanımlaması yapıyoruz aslında. Yukarıdaki örnekte tanımladığımız **container** objesini aşağıdaki gibi kullanıyoruz.

```
<View style={styles.container} />
```

**flex:1** demek içinde bulunduğu componenti enine boyuna %100 kapla demek. En üsteki componente bu değeri verirsek ekranımızın tamamını kaplamış oluruz. Aşağıdaki örneği inceleyelim



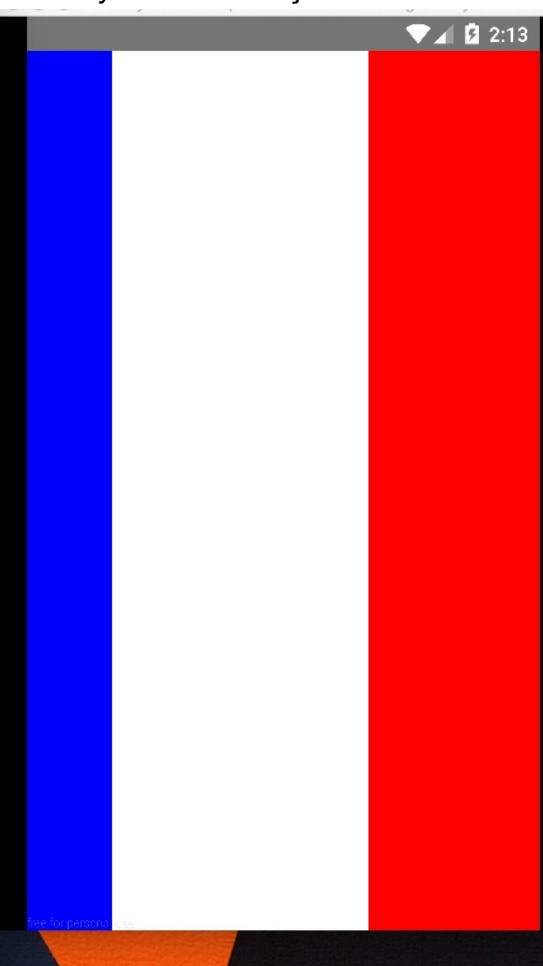
```
index.js      index.android.js    index.ios.js
1 'use strict';
2
3 import React, { Component } from 'react';
4
5 import { StyleSheet, View, Text } from 'react-native'
6
7 class Main extends Component {
8   render() {
9     return (
10       <View style={styles.container}>
11         <View style={styles.cardBlue} />
12         <View style={styles.cardWhite}/>
13         <View style={styles.cardRed}/>
14       </View>
15     );
16   }
17 }
18
19 const styles = StyleSheet.create({
20   container: {
21     flex: 1,
22     flexDirection: 'column'
23   },
24   cardBlue: {
25     flex: 1,
26     backgroundColor: 'blue'
27   },
28   cardWhite: {
29     flex: 3,
30     backgroundColor: 'white'
31   },
32   cardRed: {
33     flex: 2,
34     backgroundColor: 'red'
35 }
36 });
37
38
39 export default Main;
```

Container kartımız **flex:1** ve tüm ekranı kaplıyor. Ayrıca **flexDirection** olarak **column** almış **primary axis** dikey.. Böylece child elementler dikey olarak sıralanıyor.

Mavi Kart **flex:1**, Beyaz Kart **flex:3**, Kırmızı Kart **flex:2** değerlerini almış.

Burada Container componenti primary axis boyunca oransal olarak 6x parça bölmüşler. Mavi'ye 1x, Beyaz'a 3x, Kırmızı'ya 2x oranında pay düşmüş. Resimde de bu oranı rahatlıkla görebilirsiniz.

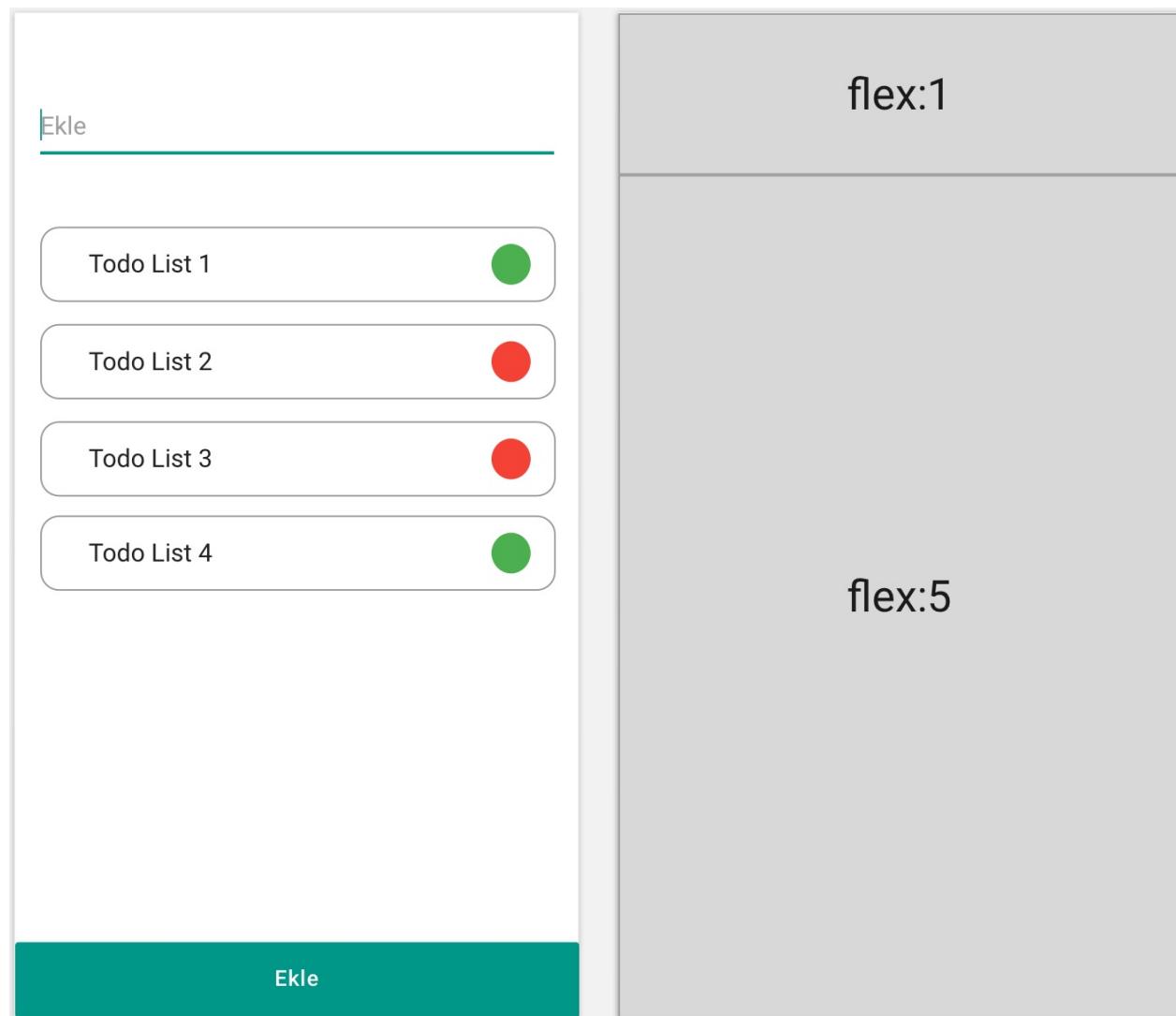
Aynı oranı **primary axis'mız row** olsaydı bu kez şöyle bir layout elde etmiş olacaktık.



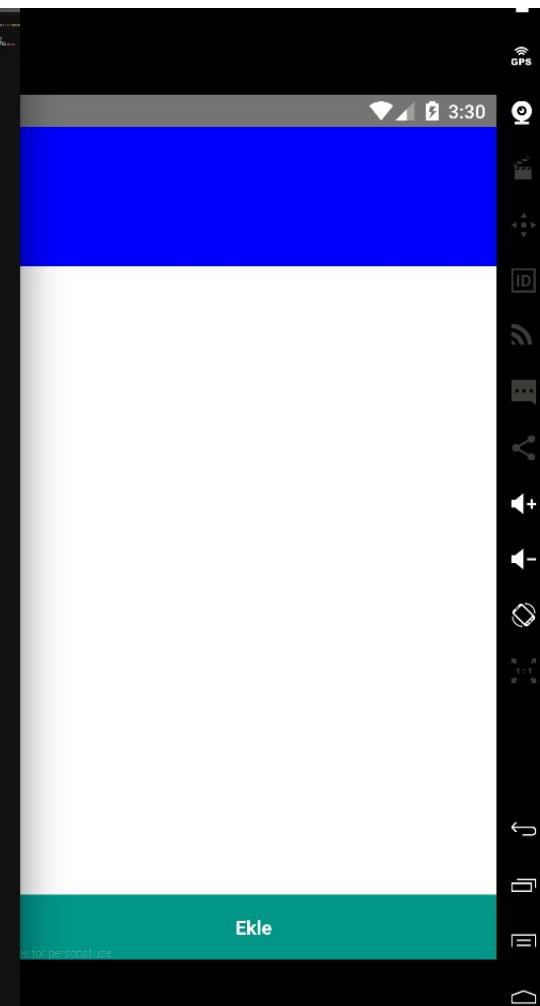
The screenshot shows an Android application running in an emulator. The screen displays a horizontal layout containing three colored cards: blue, white, and red. The blue card is on the left, the white card is in the middle, and the red card is on the right. The cards are evenly spaced and have equal widths, demonstrating a horizontal layout where the primary axis is 'row'.

```
index.js      index.android.js    index.ios.js
1 'use strict';
2
3 import React, { Component } from 'react';
4
5 import { StyleSheet, View, Text } from 'react-native'
6
7 class Main extends Component {
8   render() {
9     return (
10       <View style={styles.container}>
11         <View style={styles.cardBlue} />
12         <View style={styles.cardWhite} />
13         <View style={styles.cardRed} />
14       </View>
15     );
16   }
17 }
18
19 const styles = StyleSheet.create({
20   container: {
21     flex: 1,
22     flexDirection: 'row'
23   },
24   cardBlue: {
25     flex: 1,
26     backgroundColor: 'blue'
27   },
28   cardWhite: {
29     flex: 3,
30     backgroundColor: 'white'
31   },
32   cardRed: {
33     flex: 2,
34     backgroundColor: 'red'
35 }
36 });
37
38
39 export default Main;
```

Şimdi yapmak istediğimiz uygulamaya bir göz atalım.



Burada temel container elementimiz ekranı kaplasın, 2 tane de container elementimiz olsun ve telefonun ekranını 1 e 5 oranında paylaşsınlar. Button componentinin pozisyonu fixed olduğundan onu bu durumdan ben ayrı tutuyorum. Aşağıdaki resimde bunu nasıl gerçekleştirdik inceleyebilirsiniz.



```

1 'use strict';
2
3 import React, { Component } from 'react';
4
5 import { StyleSheet, View, Text, TouchableOpacity } from 'react-native'
6
7 class Main extends Component {
8   render() {
9     return (
10       <View style={styles.container}>
11         <View style={styles.inputContainer} />
12         <View style={styles.listContainer} />
13         <TouchableOpacity style={styles.button} >
14           <Text style={styles.buttonLabel} > Ekle </Text>
15         </TouchableOpacity>
16       </View>
17     );
18   }
19 }
20
21 const styles = StyleSheet.create({
22   container: {
23     flex: 1,
24   },
25   inputContainer: {
26     flex: 1,
27     backgroundColor: 'blue'
28   },
29   listContainer: {
30     flex: 5,
31     backgroundColor: 'white'
32   },
33   button: {
34     position: "absolute",
35     bottom: 0,
36     left: 0,
37     right: 0,
38     height: 48,
39     alignItems: 'center',
40     justifyContent: 'center',
41     backgroundColor: '#009688',
42   },
43   buttonLabel: {
44     fontSize: 14,
45     fontWeight: 'bold',
46     color: '#FFFFFF'
47 }
48 });
49
50
51 export default Main;

```

TouchableOpacity componentinin style'ı button objesine yazılmış. Pozisyonu fixed olacak demistīk. React Native'de position:fixed methodu yok. Onun yerine position özellīğine 'relative' veya 'absolute' değerlerini verebilirsiniz. Bilindik left, right, bottom değerleriyle de ekranın en altına çakılmış durumda. TouchableOpacity gördüğünüz gibi props olarak children kabul ediyor. HTML de ki <button>Ekle</button> yerine biz burada <Text /> componentini kullanmak zorundayız. Ayrıca TouchableOpacity elementine de flexBox stilleri yazdığımızı dikkat edin. ( alignItems ve justifyContent )

Şimdi inputContainer'a bir TextInput ekleyelim. TextInput, container'ın en sonunda dursun diye, justifyContent'e 'flex-end' diyelim.

justifyContent primary axis'e etki ediyordu. primary axis, default olarak flexDirection: 'column' yani dikey. Burada justifyContent: 'flex-end' demek, child elementi dikey olarak sona at demek.

```
...
<View style={styles.inputContainer}>
  <TextInput />
</View>
...

inputContainer:{
  flex:1,
  justifyContent:'flex-end',
  paddingLeft:16,
  paddingRight:16
},
....
```

Şimdi bir de listemize eklemeler yapılmıça çoğaltacağımız, ListItem componenti yapalım.



Bu componentte de flexbox'ın nimetlerinden faydalanalım. Flex bir container ımız olsun. Bunun içindeki child componentler merkezi olarak hizalansın. Ancak sağa doğru yaşlı duran bir butonumuz olsun. Bu yazılanların kodu şöyle oluşuyor.

```
const ListItem=(props)=>{
  return(
    <View style={styles.listItemContainer} >
      <Text> Todo List 1 </Text>
      <TouchableOpacity style={ styles.listItemButton } />
    </View>
  )
}

....  

listItemContainer:{
  height:48,
  borderRadius:12,
  borderWidth:1,
  borderColor:'#979797',
  margin:14,
  position:'relative',

  flexDirection:'row', //yatay hizalansın
  alignItems:'center', //child componentler merkezi hizalansın
  paddingLeft:31,
},
listItemButton:{
  position:'absolute',
  right:16,           //sağa 16px lik uzaklıkta yaslı olsun
  width:26,
  height:26,
  borderRadius:13,
  backgroundColor:'green',
}
....
```

Sonra da main componentimiz de aşağıdaki gibi ListItem componentimizi çağrıralım

```
...
<View style={styles.listContainer}>
  <ListItem />
</View>
...
```

Şimdi ListItem da buttonumuzun props alarak rengini belirleyelim.

```
<TouchableOpacity style={ styles.listItemButton } />
```

Hali hazırda listItemButton isimli bir style'ı var. bu listItemButton objesinde bir backgroundColor tanımı yapılmış. O tanım kalsa bir sorun yaratmaz ama kaldırıralım.

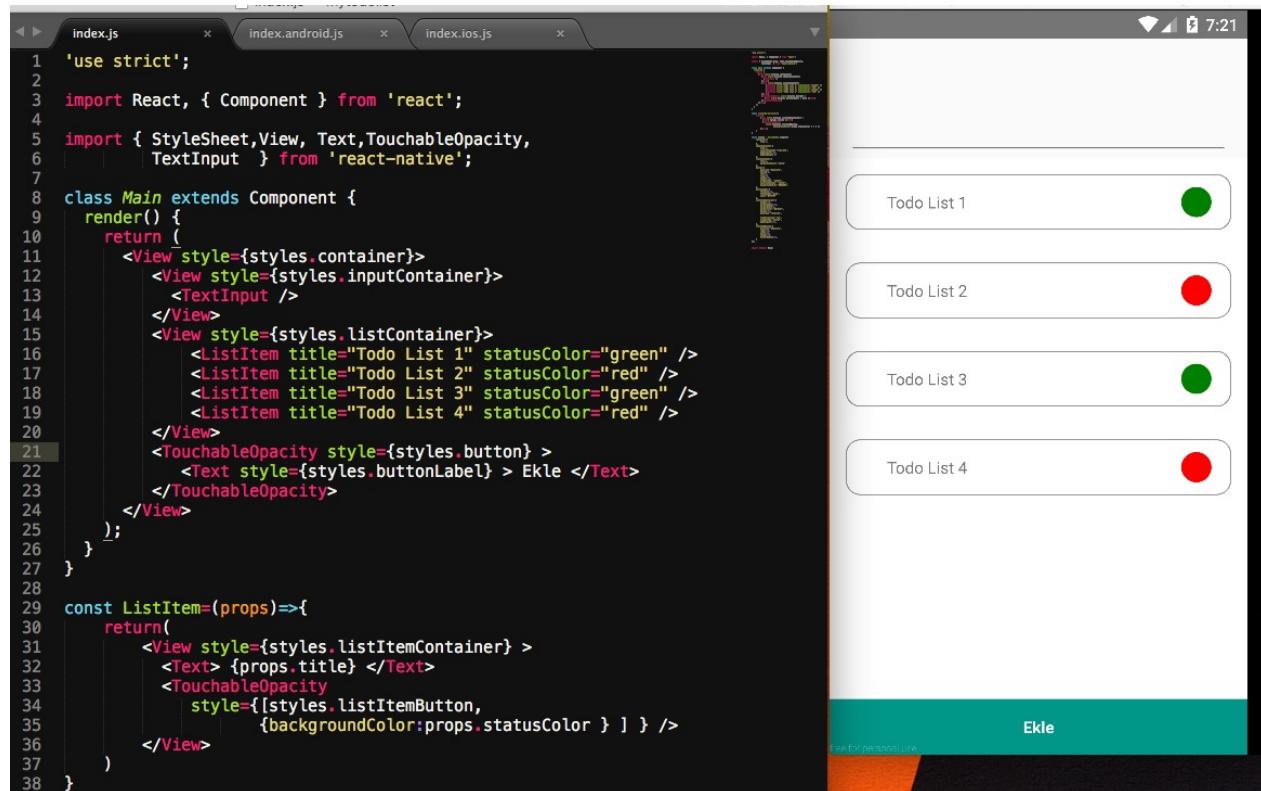
```

const ListItem=(props)=>{
    ...
        <TouchableOpacity style={[styles.listItemButton,{backgroundColor:props.statusColor}]} />
    ...
}

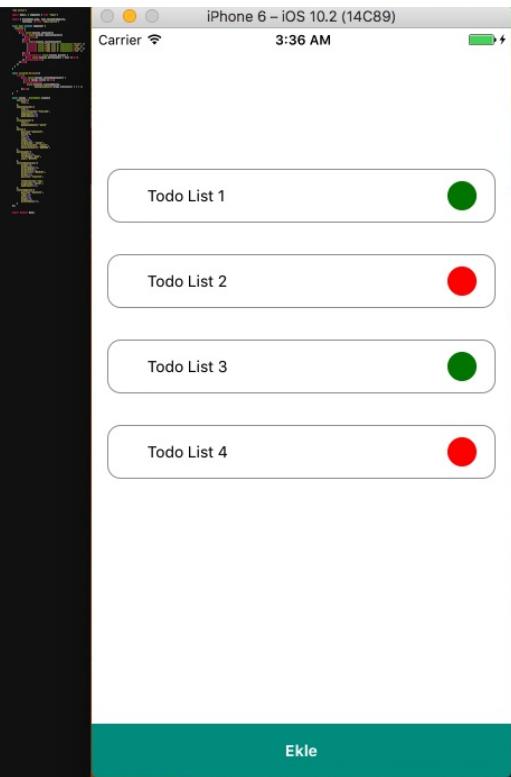
```

Yukarıdaki koda dikkat edin. style propsu bir obje alıyor { }. Bu objenin içine biz array tanımlıyoruz { [ ] }. bu array içine biz istediğimiz kadar style objesi yerleştirebiliriz.{ [ { }, { } ] }

Son olarak şekilledirdiğimiz ekran bu şekilde olmuş oldu,



Şimdi IOS da da istediğimiz gibi görünüyor mu ona bakalım.



```

1  'use strict';
2
3  import React, { Component } from 'react';
4
5  import { StyleSheet, View, Text, TouchableOpacity,
6         TextInput } from 'react-native';
7
8  class Main extends Component {
9    render() {
10      return (
11        <View style={styles.container}>
12          <View style={styles.inputContainer}>
13            <TextInput/>
14          </View>
15          <View style={styles.listContainer}>
16            <ListItem title="Todo List 1" statusColor="green" />
17            <ListItem title="Todo List 2" statusColor="red" />
18            <ListItem title="Todo List 3" statusColor="green" />
19            <ListItem title="Todo List 4" statusColor="red" />
20          </View>
21          <TouchableOpacity style={styles.button} >
22            <Text style={styles.buttonLabel} > Ekle </Text>
23          </TouchableOpacity>
24        </View>
25      );
26    }
27  }
28
29 const ListItem=(props)=>{
30   return(
31     <View style={styles.listItemContainer} >
32       <Text> {props.title} </Text>
33       <TouchableOpacity
34         style={[{backgroundColor:props.statusColor}]} >
35           <Text style={styles.listItemButton} >
36             {props.title}
37           </Text>
38     </View>

```

Maalesef görünmüyör :( . %90 oranında aynısı da olsa, TextInput ortalıkta yok. Çünkü IOS da height alanı zorunlu. Ve IOS tasarımda genelde materialden farklı olarak, text girişlerinde bir border oluyor. Şimdi TextInput'a ios da ayrı, android de ayrı çalışacak şekilde bir style ekleyelim. Bunun için React Native'de **Platform** isimli bir API var. Bu bizim hangi işletim sistemi kodu çalıştırıyor söyleyecek.

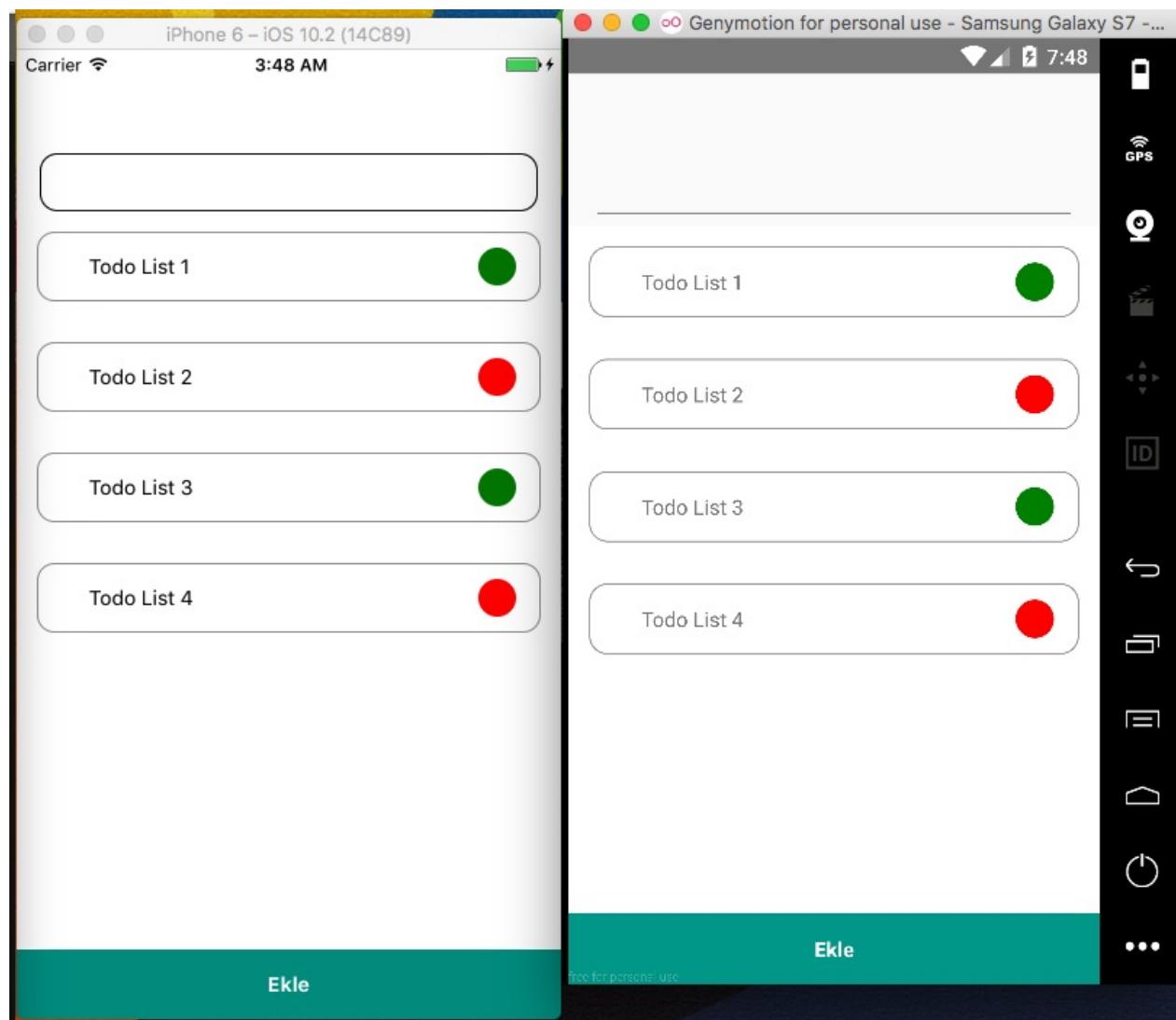
```

import { StyleSheet, View, Text, TouchableOpacity,
         TextInput, Platform } from 'react-native';
...
<View style={styles.inputContainer}>
  <Text style={ styles.textStyle} />
</View>
...
textStyle:{ ...
  ...Platform.select({
    ios: {
      height:40,
      paddingLeft:12,
      borderWidth:1,
      borderRadius:12
    },
    android:{ ...
      ...
    }
  },
  ...
}

```

Şimdi tam istediğimiz gibi oldu. Kodun tamamı için

<https://gist.github.com/ysfzrn/a825488f65fabcf6e41ede93a678d83c>



# Temel Component'ler

- [View](#)
- [Text](#)
- [Image](#)
- [ScrollView](#)
- [ListView](#)
- [FlatList](#)

# View

Belki de en temel component. Android'deki `android.view`, IOS'daki `UIView`. HTLM'deki karşılığı `div` diyebiliriz. Ekranımızı temel anlamda şekillendiren UI elementi.

[Resmi dökümanda](#) tüm propslarını görebilirsiniz. Zaten yapacağımız örneklerde bir çok kez kullanacağız. Burada gözden kaçabilecek bir özelliğinden bahsetmek istiyorum.

View componenti mount edilir edilmez, **onLayout** eventi tetiklenir.

```
<View onLayout={this._handleLayout} />
```

Burada fonksiyonumuzun bir tane obje tipinde parametresi olur.

```
_handleLayout=(e)=>{
  const { x, y, width, height } = e.nativeEvent.layout
}
```

Burada View componentimizin, parent elemente göre x ve y konumunu, genişlik ve uzunluk bilgilerine erişebilirsiniz. Animasyon kısmında göreceğimiz, LayoutAnimation yöntemini kullanırken bayağı işe yarıyor.

# Text

Normalde HTML de `<div> Merhaba Dünya </div>` tag'ler arasında yazı yazıp o element tag'in fontuna style verebiliyorduk. Mobile'de durum biraz farklı. Bunun için `<Text>` componentini kullanıyoruz. font stillerimizi bu elemente tanımlıyoruz.

Güzel özelliklerinden birisi iç içe `<Text>` elementlerini kullanabiliyoruz.

```
<Text style={{...}}>
  Merhaba <Text style={{...}}> Dünya </Text>
</Text>
```

Font stillerini Text componentinde tanımlıyoruz dedik, tabi durum böyle olunca varsayılanın dışında belli bir font kullanmaya karar verdiysek, o zaman kendimize bir tane özel bir Text componenti yapıp, mesela ismi MyText olsun, bunun stillerinde fontFamily özelliğini verip, diğer Text componentleri de bu componentimizin child'ı olarak çağırabiliriz

```
<MyText>
  Merhaba <Text style={{...}}> Dünya </Text>
</MyText>
```

`<Text>` componenti de View gibi onLayout eventini desteklemektedir.

# Image

Resimlerimizi derlemek için kullandığımı component. İster bir network bağlantısından, isterse local'den resimleri render edebiliriz.

Bu örnekte server'dan dinamik olarak resimleri fetch eden bir componenti görüyorsunuz.

```
<Image resizeMode = "contain"  
      style={styles.image}  
      source={{uri: `${API_PUBLIC}${category.name}.png`} }} />
```

Mesela diyelim ki sizin assets klasörünüzde boyutuna göre **test.png**, **test@2x.png** ve **test@test3.png** adında 3 tane resminiz olsun. Siz bunları **require('./test.png')** yazarak çağrıdığınızda, cihazın boyutuna göre resmi seçili, render edilecektir.

React native'in 0.39 ve alt sürümlerinde Image componenti için mutlaka width ve height değerleri verilmek zorundaydı. Üst ve şimdiki sürümde (0.42) default olarak Image componentinin boyutu **flex:1** olarak derlenmektedir.

# ScrollView

Kaydırılabilir içerikler kullanabileceğimiz en basit component. Eğer sizin 30 ve altında bir içeriğiniz varsa ScrollView çok kullanışlı. Hem yatay hem de dikey olarak içerikleri kaydırabiliyoruz.

```
<ScrollView style={styles.container}>
  <View style={styles.boxLarge} />
  <ScrollView horizontal>      //default olarak vertical seçilidir.
    <View style={styles.boxSmall} />
    <View style={styles.boxSmall} />
    <View style={styles.boxSmall} />
  </ScrollView>
  <View style={styles.boxLarge} />
  <View style={styles.boxSmall} />
  <View style={styles.boxLarge} />
</ScrollView>
```

30 ve üzerinde bir içeriği sıralamak istiyorsanız, performans açısından bunun için bize önerilen, ScrollView yerine, ListView 'i kullanmamız.

# ListView (Deprecated, Önerilen FlatList kullanmanız)

ListView vertical olarak sırf performans gözterek hazırlanmış bir component. Sadece ekranda görünen componentler render, edilir, diğer componentler cihazın performansını zorlamaz. Bir update işlemi geldiyse tüm liste tekrardan render edilmez, sadece ekranda görünenler render edilir. Bu da ciddi bir akıcılık kazandırır. Ama kabul edelim, biraz syntax karışık gelebilir. Tek tek nasıl yazarsınız ona bakalım.

`ListView`, `ScrollView` gibi kendi altındaki elementleri yani `children` elementleri render etmez. Onun yerine `renderRow` propsunu kullanır. `renderRow` aslında `dataSource` 'dan dönen elementleri derleyen bir çıktı üreten bir fonksiyon. Örneğin, `dataSource` bir array dönüyorsa, `rowData`'yı aşağıdaki gibi kullanırız.

```
(rowData) => <Text>{rowData}</Text>
```

## DataSource

```
const ds = new ListView.DataSource({  
  rowHasChanged: (r1, r2) => r1 !== r2  
});
```

ListView elementleri render ederken DataSource API'sini kullanır. DataSource input olarak bir array alır. Bu array her türlü veriyi içerebilir. `rowHasChanged` bizim verdığımız koşula göre verileri karşılaştırır ve hangilerinin render edileceğine karar verir. Yukarıdaki örnekte `(r1, r2) => r1 !== r2` olarak verilmiş ama biz `(r1, r2) => r1.id !== r2.id` olarak tanımlayabiliyoruz.

Son olarak datayı `DataSource.cloneWithRows(inputArray)` olarak çağrılmalısınız.

Şimdi her şeyi baştan olarak yazalım.

**1-Bizim bir elimizde array olsun.**

```
import React, { Component } from "react";
import { View, Text, ListView, StyleSheet } from "react-native";

const rows = [
  { id: 1, text: "row1" },
  { id: 2, text: "row2" },
  { id: 3, text: "row3" },
  { id: 4, text: "row4" },
  { id: 5, text: "row5" },
  { id: 6, text: "row6" }
];
```

### 2-Şimdi constructor ' da DataSource' u yaratalım

```
constructor() {
  super();
  const ds = new ListView.DataSource({
    rowHasChanged: (r1, r2) => r1 !== r2
  });
}
```

### 3-Data yi `DataSource.cloneWithRows(inputArray)` olarak çağırıalım

```
constructor() {
  super();
  const ds = new ListView.DataSource({
    rowHasChanged: (r1, r2) => r1 !== r2
  });
  this.state = {
    list: ds.cloneWithRows(rows)
  };
}
```

### 4- ListView kullanıma hazır

```
<ListView
  dataSource={this.state.list}
  renderRow={rowData =><Text>{rowData.text}</Text>}
/>
```

Örneğin tamamı için burayı inceleyebilirsiniz

<https://sketch.expo.io/SJxhqQRoe>



# FLATLIST

## Pattern

```
<FlatList
  data={[]}
  ref={ref => {this.flatListRef = ref;}}
  renderItem={({ item }) => ( <Text>{item.name}</Text>)}
  ItemSeparatorComponent={this.renderSeparator}
  ListHeaderComponent={this.renderHeader}
  ListFooterComponent={this.renderFooter}
  keyExtractor={item => item.id}
  onEndReached={this.getMoreData}
  refreshing={this.state.refreshing}
  onRefresh={this.handleRefresh}
  onEndReachedThreshold={100}
/>
```

FlatList, performans açısından ve kullanım kolaylığı bakımından ListView'den daha iyi bir component.

## Methodlar

**scrollToEnd ({ })**

**scrollToIndex({ })**

**scrollToItem({ })**

**scrollToOffset({ })**

# Component API'lar

- Animasyon
- PanResponder

# ANİMASYON

React JS ile uygulama geliştiren insanlardan en çok şikayet edilen konulardan biri animasyon. Ama react native'de pek öyle değil. React Native'in kendi sunduğu Animated API, gerçekten çok etkileyici. Bunu web tarafında kullanmak isterseniz, [animated](#) veya [react-native-web](#) kütüphanesinin içindeki Animated API gibi çeşitli implementasyonlar mevcut. (react-native-web, react native codebase'i ile web geliştirmek için yazılmış bir kütüphane, belki denemek istersiniz)

React Native'de animasyonlar için bize 2 tane API sunuluyor.

- LayoutAnimation
- Animated

## LayoutAnimation

`LayoutAnimation` , `Animated` API'a göre çok daha az konfigurasyonu olan ve componentinizde ne özellik etkilenmişse animasyon olarak kullanabileceğiniz ama `Animated` API'a göre de sınırlı olan bir API.

Burada göz önünde bulunduracağınız şey bir sonraki render'da ne etkilenecek. Örneğin `width` ve `height` için oldukça kullanışlı oluyor.

`LayoutAnimation` ayrıca native bir çözüm. JavaScript'in içinde yaratılmış bir API değil. O yüzden smooth yani akıcı animasyonlar için kullanışlı.

## Animated

`Animated` API, `LayoutAnimation`'a göre oldukça fazla konfigurasyonu olan ve karmaşık animasyonları yazacağınız bir API. Ve maalesef JavaScript tarafından çalışan bir API bu da daha çok animasyonlar da takılmalara sebep olabiliyor. Daha akıcı animasyonlar yapmak için takılmalarda `setNativeProps` ile native dünyada değişiklikler yapabilirsiniz.

# LayoutAnimation

Herkesin basit layoutAnimation'ı anlatırken yaptığı basit bir örnekle başlayalım. Örneğimizde bir tane karemiz var. Üzerine basınca state'e göre height değeri 250 veya 450 oluyor, o kadar.

```
class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      selected: false
    };
  }

  handleSelect = () => {
    this.setState({ selected: !this.state.selected });
  };

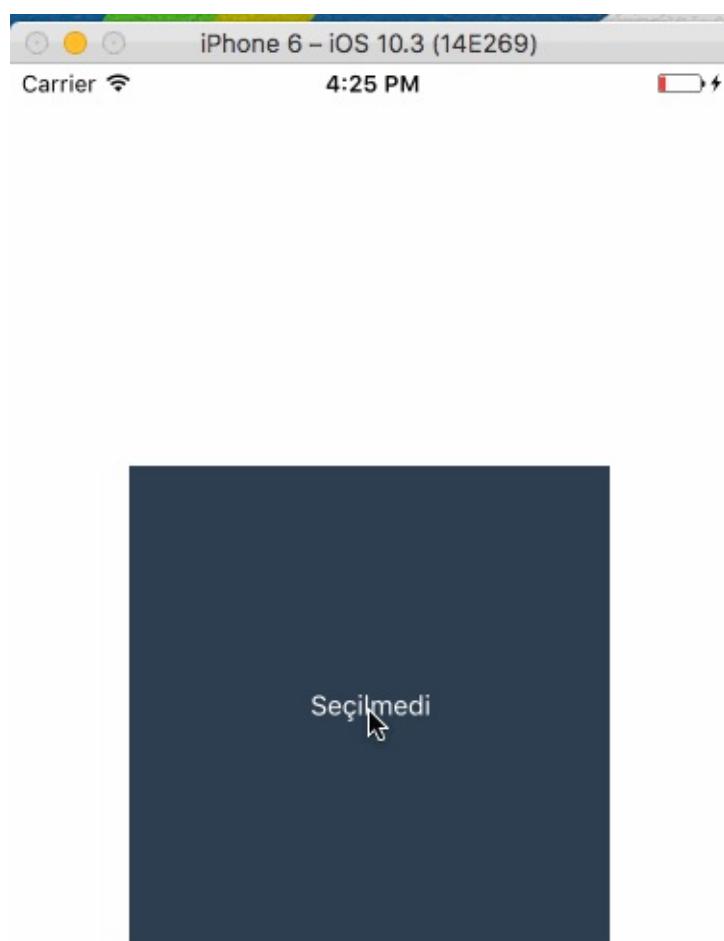
  renderRectangle = () => {
    const customStyle = {
      height: this.state.selected ? 450 : 250
    };
    return (
      <View style={[styles.rectangleStyle, customStyle]}>
        <TouchableWithoutFeedback onPress={() => this.handleSelect()}>
          <View>
            <Text style={{ color: "white" }}>
              {" "}{this.state.selected ? "Seçildi" : "Seçilmedi"}{" "}
            </Text>
          </View>
        </TouchableWithoutFeedback>
      </View>
    );
  };

  render() {
    const { selected } = this.state;
    return (
      <View style={styles.container}>
        {this.renderRectangle()}
      </View>
    );
  }
}

// define your styles
const styles = StyleSheet.create({
  container: {
```

```
flex: 1,
justifyContent: "center",
alignItems: "center"
},
rectangleStyle: {
width: 250,
backgroundColor: "#2c3e50",
justifyContent: "center",
alignItems: "center"
}
});

export default App;
```



Şimdi LayoutAnimation'ın magic kısmına gelelim.

Tek yapmamız gereken LayoutAnimation'ı import edip, state'i değiştirdiğimiz yerde LayoutAnimation konfigurasyonunu yapmak. Baştaki kodda handleSelect fonksiyonuna aşağıdaki kodu eklemeniz yeterli.

```
...
handleSelect = () => {
  LayoutAnimation.configureNext(LayoutAnimation.Presets.spring);
  this.setState({ selected: !this.state.selected });
};
...
```



(yukarıdaki gif sizi aldatmasın gayet smooth bir animasyon sadece gif de takılma varmış gibi gözüüyor)

Burada `LayoutAnimation.spring();` olarak da ekleyebilirdik. Ama illâ hazır animasyonları kullanmak yerine custom animasyonları da eklemek isterseniz, `configureNext`'i kullanmalısınız. `duration`, `create`, `update` methodlarını kullanabilirsiniz. Hazır fonksiyonlardan `spring` dışında `easeInEaseOut` ve `linear` var.

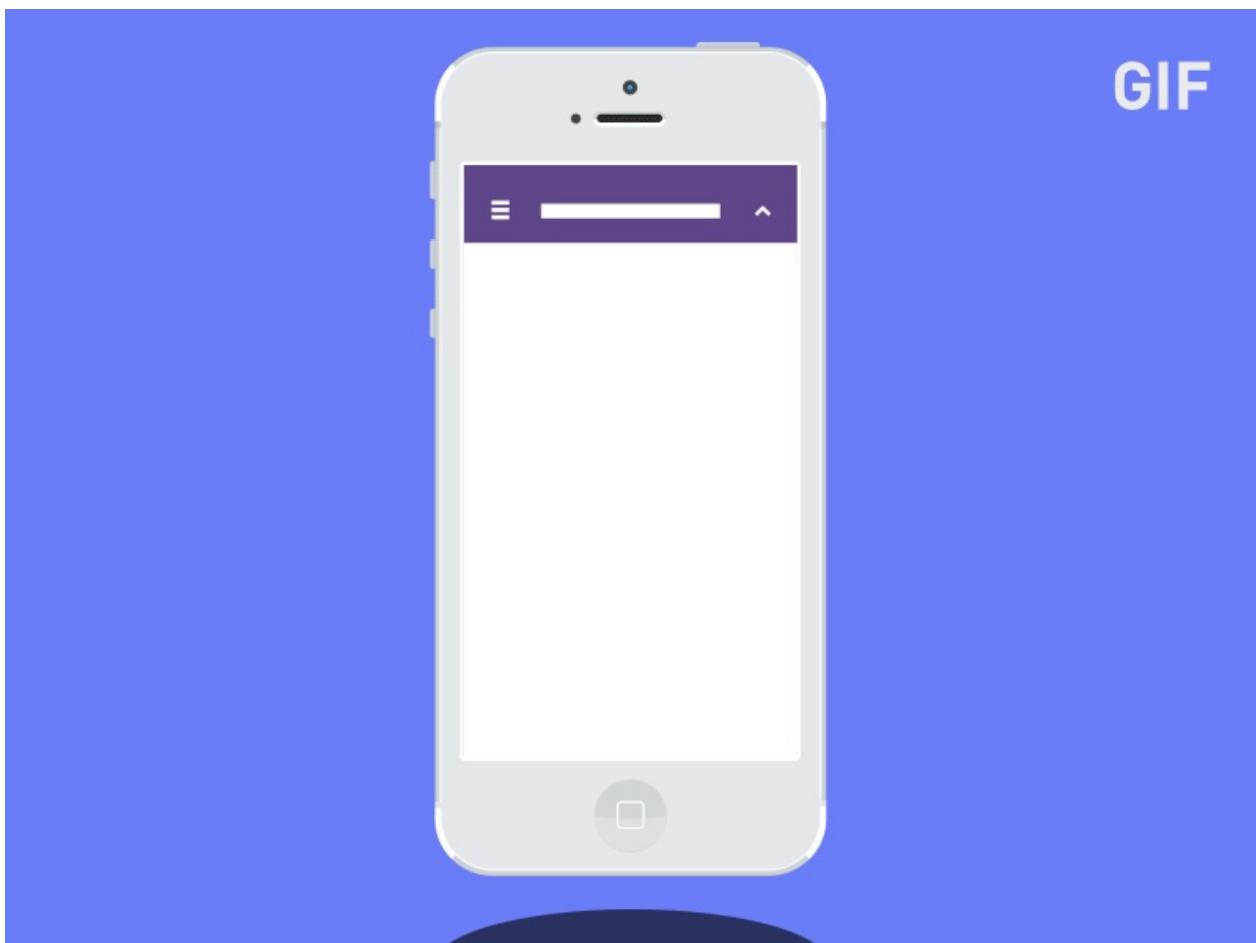
`configureNext` 'e ikinci parametre olarak bir callback fonksiyonu da geçirebilirsiniz fakat bu durum sadece ios için geçerli.

Önemli Not: Android'de LayoutAnimation kullanırken mutlaka yapmanız gereken, 2 şey var.

```
var UIManager = require('UIManager'); //UIManager'ı import edin

//componentDidMount aşağıdaki configurasyonu yapın
componentDidMount() {
    UIManager.setLayoutAnimationEnabledExperimental &&
    UIManager.setLayoutAnimationEnabledExperimental(true);
}
```

Şimdi şuradaki örneğe bir göz atalım.<sup>1</sup>



Biz de benzer bir örnek yaratalım.

```

index.ios.js      app.js      header.js      flexTab.js
18     { id: 7, text: "item7", selected: false, backgroundColor: '#E8AA8C' },
19     { id: 8, text: "item8", selected: false, backgroundColor: '#F03861' },
20     { id: 9, text: "item9", selected: false, backgroundColor: '#FFDDE1' },
21     { id: 10, text: "item10", selected: false, backgroundColor: '#1EE494' }
22   ]
23 }
24 }

25 handleSelectedItem = item => {
26   //LayoutAnimation.configureNext(LayoutAnimation.Presets.linear);
27   const _list = this.state.list;
28   for (let i = 0; i < _list.length; i++) {
29     if (_list[i].id === item.id) [
30       _list[i].selected = _list[i].selected ? false : true;
31     ] else {
32       _list[i].selected = false;
33     }
34   }
35 }
36 this.setState({ list: _list });
37 };

38 renderflexTabs = () => {
39   return this.state.list
40   .map((item, i) => {
41     return (
42       <FlexTab
43         key={i}
44         item={item}
45         onPress={() => this.handleSelectedItem(item)}
46     )
47   )
48 }

```

Burada tek yaptığımız baştaki koddaki Rectangle'i component haline getirip, ScrollView içinde map edip, container componentte ki item'ları flexBox kullanarak wrap etmek Animasyonu sağlamak için de LayoutAnimation satırının commentini açıp son haline bakalım.

```

index.ios.js      app.js      header.js      flexTab.js
1 //import libraries
2 import React, { Component } from "react";
3 import { View, Text, StyleSheet, ScrollView, LayoutAnimation, StatusBar } from "react-native";
4 import FlexTab from "./flexTab";
5 import Header from './header'
6 // create a component
7 class App extends Component {
8   constructor() {
9     super();
10    this.state = {
11      list: [
12        { id: 1, text: "item1", selected: false, backgroundColor: '#A5F2E7' },
13        { id: 2, text: "item2", selected: false, backgroundColor: '#8983F3' },
14        { id: 3, text: "item3", selected: false, backgroundColor: '#432C51' },
15        { id: 4, text: "item4", selected: false, backgroundColor: '#535474' },
16        { id: 5, text: "item5", selected: false, backgroundColor: '#54A4AF' },
17        { id: 6, text: "item6", selected: false, backgroundColor: '#ECBC55' },
18        { id: 7, text: "item7", selected: false, backgroundColor: '#E8AA8C' },
19        { id: 8, text: "item8", selected: false, backgroundColor: '#F03861' },
20        { id: 9, text: "item9", selected: false, backgroundColor: '#FFDDE1' },
21        { id: 10, text: "item10", selected: false, backgroundColor: '#1EE494' }
22      ]
23    };
24 }

25 handleSelectedItem = item => {
26   LayoutAnimation.configureNext(LayoutAnimation.Presets.linear);
27   const _list = this.state.list;
28   for (let i = 0; i < _list.length; i++) {
29

```

Tabii ki benim yaptığım gif sizi yanlışın aslında gayet smooth bir animasyon elde ettik burada. Zaten performans monitorunda UI ve JS tarafınd 60 fps nin altına düşmediğini görebilirsiniz.

Kodun tamamı için şuraya bakabilirsiniz.

<https://gist.github.com/ysfzrn/e540a072e639656269e22e53f9b7da2d>

---

Kaynak

[https://dribbble.com/shots/1279908-UI-Animation-experiment-1<sup>1</sup>](https://dribbble.com/shots/1279908-UI-Animation-experiment-1)

# Animated

Anlat anlat bitmeyecek, kesin bir hiyerarşisi olmayan, resmi dökümda dahi tüm özellikleriyle anlatılmamış en sevdiğim API. Burası benim kendi yorumum. Şimdi üçüncü ağızdan anlatmaya başlayalım.

Animated API özellikle syntax itibariyle karışık gelebilir. Bu dökümda tüm özellikleriyle bahsedebilmem çok zor. Sadece buraya sınırlı kalamayacağınız için kendi dökümanı hariç önerdiğim kaynaklar aşağıdaki gibi;

(Bahsedilen tüm kaynaklar [browniefed](#) mahlaslı arkadaşa ait)

- [İngilizce döküman](#)
- [Temel anlamda Animated API egghead eğitim videoları](#)
- [Gerçek case'lere göre advanced animasyon egghead eğitim videoları](#)

Temel felsefe, bir tane animatedValue() yarat. Bu değişkenin değerini değiştir ve componentin style objesine de değiştirdiğin değeri yaz, animatedValue'ya göre animasyon yarat.

1 - animatedValue() yarat

```
this._animatedValue = new Animated.Value(0);
```

2 - animatedValue() değerini değiştir

```
this._animatedValue.setValue(150);
```

3 - style objesine set et

```
<Animated.View style={{left: this._animatedValue}} />
```

Şimdi [şuradaki](#) layoutAnimation örneğini bu kez Animated API ile yapalım.

ilk önce constructor'da bir animatedValue yaratalım

```

constructor() {
  super();
  this.state = {
    animValue: new Animated.Value(250)
  };
}

```

Sonra bir tane dikdörtgen bir obje yaratalım. Ve bu objemizin height değeri, yarattığımız `animatedValue()` 'a göre değişsin. Burada dikkat etmeniz gereken şey, `Animated.View`. Her component animated API'ı kullanabilir. Bunu `createAnimatedComponent()` fonksiyonuyla sağlayabilirsiniz. Bunun yanında react native sık kullanılan componentler için bazılarını bize hazır olarak Animated API içinde veriyor. Bunlar; **Animated.View**, **Animated.Image**, **Animated.ScrollView** ve **Animated.Text**.

```

renderRectangle = () => {
  const customStyle = {
    height: this.state.animValue,
  };

  return (
    <Animated.View style={[styles.rectangle, customStyle]}>
      <TouchableWithoutFeedback onPress={() => this.handleSelect()}>
        <View style={{ flex: 1 }} />
      </TouchableWithoutFeedback>
    </Animated.View>
  );
};

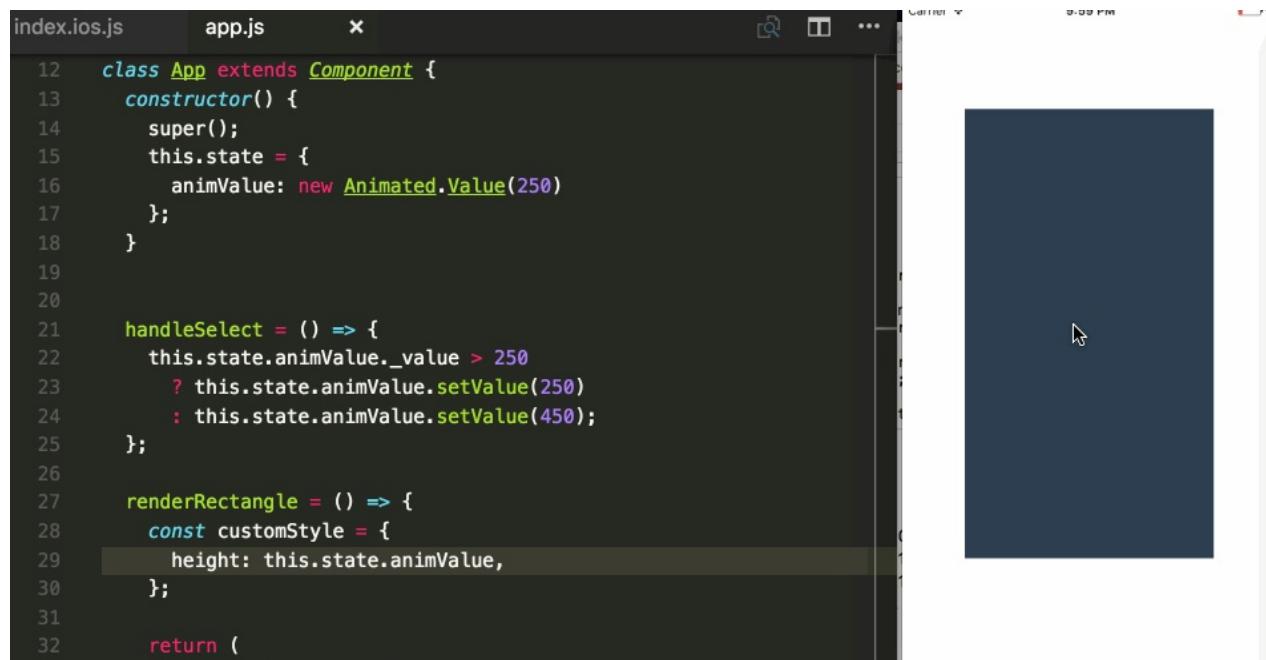
```

Dikdörtgen objemize dokununca, height değeri 250 ise 450, 450 ise 250 olsun.

```

handleSelect = () => {
  this.state.animValue._value > 250
    ? this.state.animValue.setValue(250)
    : this.state.animValue.setValue(450);
};

```



```

index.ios.js    app.js      *
12   class App extends Component {
13     constructor() {
14       super();
15       this.state = {
16         animValue: new Animated.Value(250)
17       };
18     }
19
20
21     handleSelect = () => {
22       this.state.animValue._value > 250
23         ? this.state.animValue.setValue(250)
24         : this.state.animValue.setValue(450);
25     };
26
27     renderRectangle = () => {
28       const customStyle = {
29         height: this.state.animValue,
30       };
31
32       return (

```

Oley, setState olmadan tekrar tekrar render ettik. Ama bir saniye hiç animasyona benzer bir tarafı yok bunun :(

Animasyonu sağlamak için bize 3 tane animasyon tipi sunuluyor ve bir çok ihtiyacı karşılıyor daha doğrusu ben karşılamadığı bir case görmedim.Bunlar;

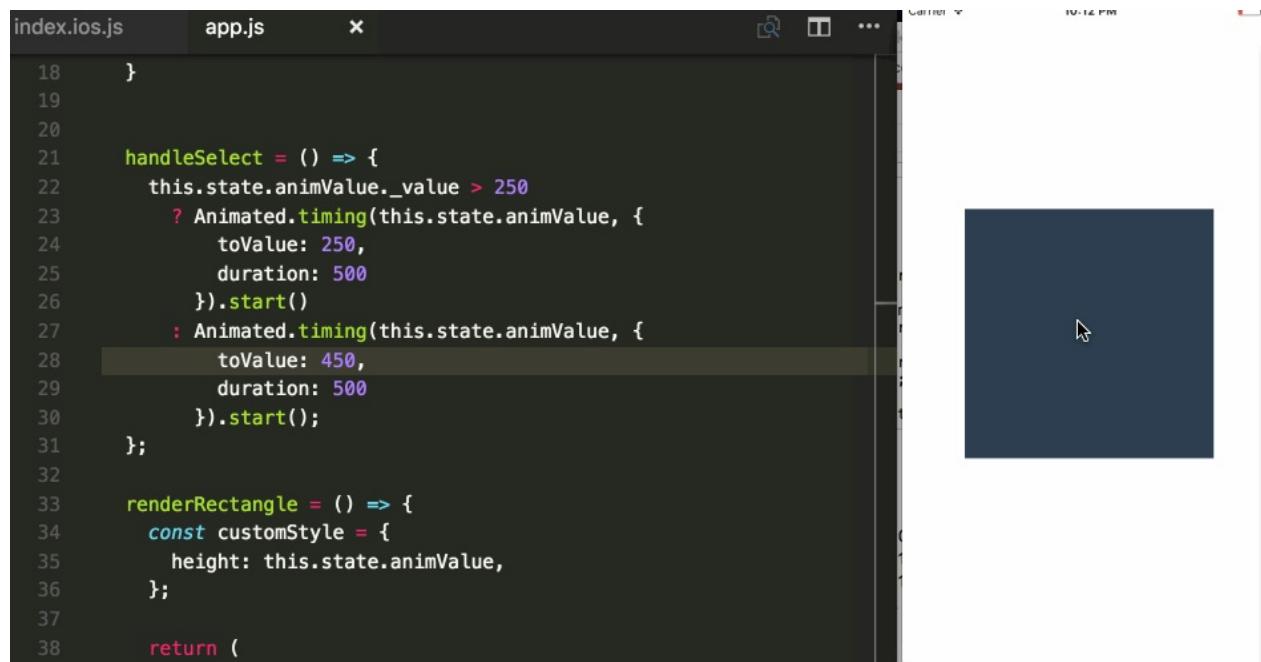
- `Animated.decay()` : Bir başlangıç hız değeri ile başlar, kademe kademe yavaşlayan animasyonlara yarar
- `Animated.spring()` : Fizik kurallarına göre animasyonlar hazırlamaya yarar
- `Animated.timing()` :Zamana göre animasyon hazırlamaya yarar

Şimdi bu örneğimiz de handleSelect fonksiyonunu Animated.timing() yardımı ile değiştirelim.

```

handleSelect = () => {
  this.state.animValue._value > 250
    ? Animated.timing(this.state.animValue, {
        toValue: 250,
        duration: 500
      }).start()
    : Animated.timing(this.state.animValue, {
        toValue: 450,
        duration: 500
      }).start();
};


```



```

18     }
19
20
21     handleSelect = () => {
22       this.state.animValue._value > 250
23         ? Animated.timing(this.state.animValue, {
24           toValue: 250,
25           duration: 500
26         }).start()
27         : Animated.timing(this.state.animValue, {
28           toValue: 450,
29           duration: 500
30         }).start();
31     };
32
33     renderRectangle = () => {
34       const customStyle = {
35         height: this.state.animValue,
36       };
37
38       return (

```

burada kullandığımız `this.state.animValue`'ya göre sadece `height` özelliğini animasyon yaptıktı. Bununla beraber rengini değiştirip biraz da döndürelim.

Bunun için **interpolation** yapacaz. İşin burasını anlamak çok önemli. Mesela aşağıdaki kodu inceleyerek anlamaya çalışalım;

```

let rotateAnimation = this.state.animValue.interpolate({
  inputRange: [250, 450],
  outputRange: ['0deg', '360deg']
});

```

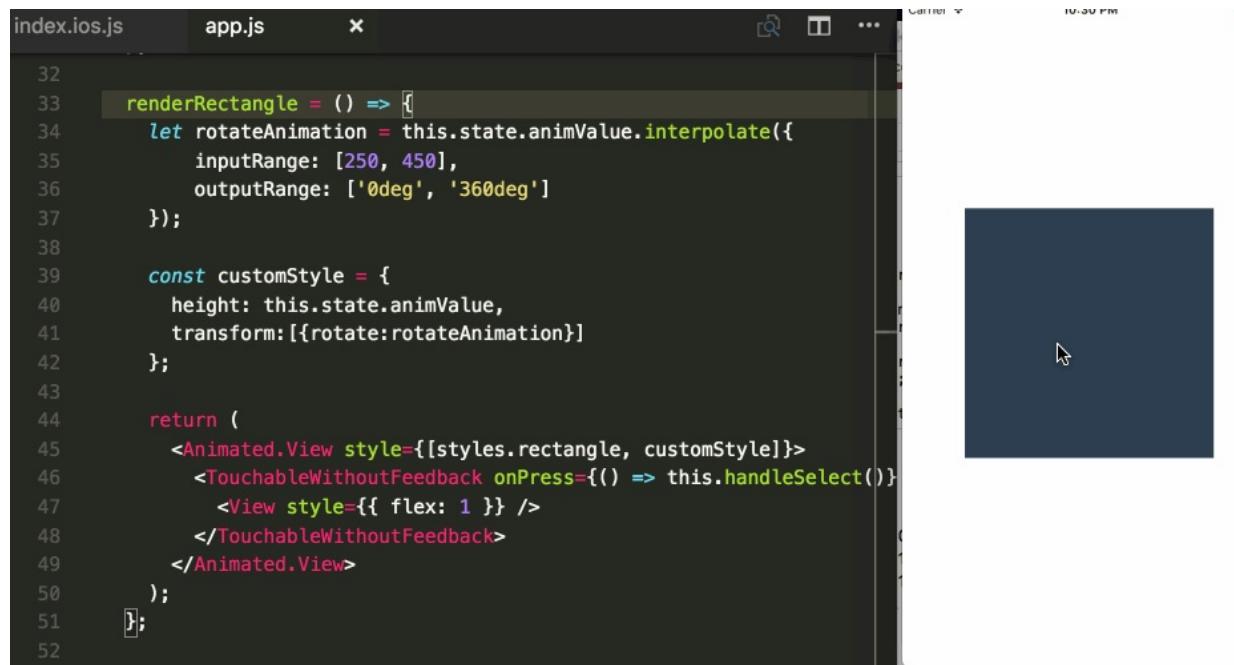
`animationValue` olan `this.state.animValue` içinden çağrıdığımız `interpolate` fonksiyonu içine `inputRange` ve `outputRange` keylerine sahip bir obje alıyor. `inputRange` **250'den --> 450'ye** doğru giderken, `outputRange` 'de **0deg'den --> 360deg'e** doğru gidiyor. Bu arada mesela `inputRange` **350 civarındayken** `outputRange` 'de **180deg'ye** yakın oluyordur. Biz araya girip (*interpolate'in kelime anımlarından biri araya girmek*) `inputRange` **350 iken 720deg** ol sonra **350 ve 450 arasında**, kendini ayarla ve **360deg'e** doğru hızla ilerle diyebiliriz.

*Bu kadar basit bir mevzuyu anlatamadıysam lütfen beni uyarın.*

inputRange	outputRange
250	0deg
...	...
350	180deg
...	...
450	360deg

Şimdi yarattığımız rotateAnimation'ı style objesine atayalım.

```
const customStyle = {
  height: this.state.animValue,
  transform:[{rotate:rotateAnimation}]
};
```



```
index.ios.js    app.js    x
32
33     renderRectangle = () => [
34       let rotateAnimation = this.state.animValue.interpolate({
35         inputRange: [250, 450],
36         outputRange: ['0deg', '360deg']
37       });
38
39       const customStyle = {
40         height: this.state.animValue,
41         transform:[{rotate:rotateAnimation}]
42       };
43
44       return (
45         <Animated.View style={[styles.rectangle, customStyle]}>
46           <TouchableWithoutFeedback onPress={() => this.handleSelect()}>
47             <View style={{ flex: 1 }} />
48           </TouchableWithoutFeedback>
49         </Animated.View>
50       );
51     ];
52   }
```

Maalesef gif yapmak için kullandığım tool pek animasyonları doğru yansıtmadı. Kodun tamamını canlı olarak [şuradan](https://snack.expo.io/rk0oekzGb) izleyebilirsiniz. ( <https://snack.expo.io/rk0oekzGb> )

# Animated.Event()

Ekranı kaydırırken, scroll yaparken, Animated.Event( ) kullanılarak animated değişkeni direkt olarak değiştirebiliriz.

Örnek olarak, biz ekranı dikey olarak scroll yaparken animated değişken olan, animatedValue'yu aşağıdaki gibi değiştirebiliriz

```
<ScrollView
    style={styles.container}
    scrollEventThrottle={16}
    onScroll={Animated.event([
        { nativeEvent: {
            contentOffset: {
                y: animatedValue //yatay scroll için, y yerine x yazabilirsiniz
            }
        }
    ])}
>
...
```

Animated.Event( ), bizim yerimize animated değişkeni, kendi içinde setValue() kullanarak değiştirmektedir.

PanResponder ile nasıl kullandığını bakalım,

```
onPanResponderMove: Animated.event([
    null, // raw event'i null geçiyoruz
    {dx: this._pan.x, dy: this._pan.y}, // gestureState , _pan => referans
]),
```

Burada, ikinci parametre `gestureState` , yukarıdaki örnekte state'ler dx ve dy değerleri. Kullanıcı ekrana ilk dokunuşundan, parmağını ne kadar kaydırırsa, ardışık olarak her harekette, state'in değerini güncelleyecektir.

animated değer değişikten sonra, onu istediğiniz gibi interpolate ederek animasyonlarınızı oluşturabilirsiniz

# PANRESPONDER

PanResponder API ile React Native'de touch(dokunma) eventleri, PanResponder'a ait lifecycle methodlar ile düzenleyip, drag and drop işlemlerini gerçekleştirebiliriz.

```
componentWillMount() {
  this._panResponder = PanResponder.create({
    onStartShouldSetPanResponder: (evt, gestureState) => true,
    onMoveShouldSetPanResponder: (evt, gestureState) => true,

    onPanResponderGrant: (evt, gestureState) => this.handleResponderGrant,
    onPanResponderMove: this.handleResponderMove,
    onPanResponderRelease: this.handleResponderRelease
  });
}

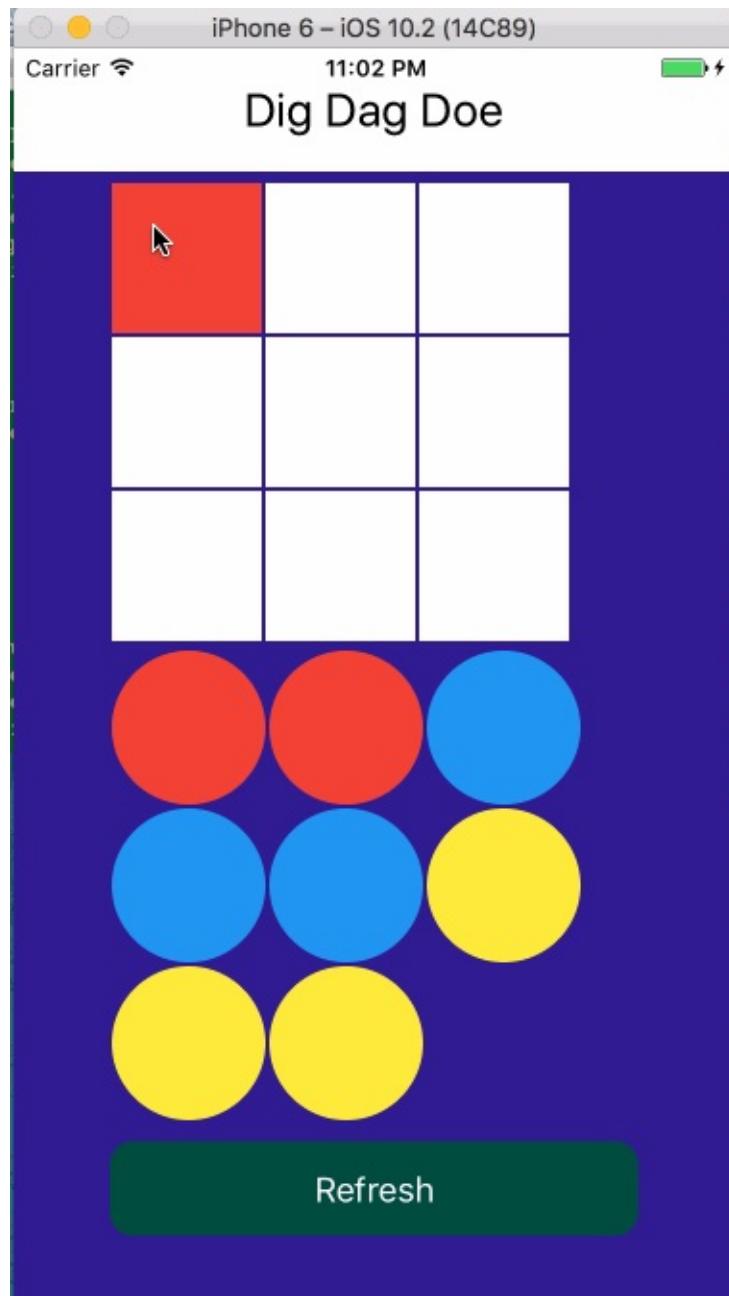
render: function() {
  return (
    <View {...this._panResponder.panHandlers} />
  );
},
```

Bir elemente drag-drop özelliği eklemek istediğimizde, yukarıdaki kod parçası sabit bir yaklaşım direkt bunu kullanabiliriz. Hatta kullanığınız editörde bir snippet kısayolu ayarlamanız size pratiklik sağlayacaktır. ( Bu arada react-native'in gesture sisteminde sadece bu methodlar yok. Ben buraya en çok kullanılanları ve aşağıdaki örnekte kullanacaklarımızı yazdım)

Şimdi bu API'ı bir örnekle açıklamak istiyorum. Bizim bir görevimiz olsun. 3x3 kare board ve altında 3 kırmızı, 3 mavi, 3 sarı topumuz olsun. Aşağıdaki topları sürükleyip, board'a bırakalım.

1. Sürükleyip bıraktığımız kutu boş ise , topun rengini alsın.
2. Sürükleyip bıraktığımız kutu dolu ise, top tekrar yerine gitsin
3. Bütün board kareleri dolduğunda yan yana olan tüm renkler aynı ise Kazandınız, değil ise Kaybettiniz diye bir Alert çıkarsın.

Kodun tamamı için <https://github.com/ysfzrn/react-native-panresponder-demo>



Görevimizi anladıysak şimdi yapılacakları adım adım sıralayalım.

1. Board çizelim. Board'ın koordinatlarını ve özelliklerini state'te tutalım.
2. Bir PanResponder Element Yaratım.
3. Hover effect ekleyelim
4. Sürüklediğimiz componenti drop ettikten sonraki kontrolleri ekleyelim.

## 1.BOARD

Bizden istenen 3x3 bir board. Bu board'daki her bir karede sayfada kendine ait koordinatlarını, içinin boş olup olmadığını, üstünde bir componentin sürüklendiği sürüklendiğini bilecek. Bunun için bir başlangıç objesi set edelim.

```
const initObj = {
  x: 0,
  y: 0,
  hovering: false,
  filled: false,
  color: MyStyle.color.white,
  value: null
};
```

Uygulamamız açılır açılmaz board'ı çizmek için componentDidMount methodunda 3x3 lük board bilgisini tutacak bir array oluşturalım. Bu array'in her bir elemanı az önceki yarattığımız, **initObj** objesinden oluşsun. Oluşturduğumuz geçici array'i state'e set edelim. Bunu aşağıda **digHole** isimli fonksiyonda topladım. Bunu componentDidMount methodunda çağırıralım.

```
// ./src/app.js

digHole = () => {
  const initObj = {
    x: 0,
    y: 0,
    hovering: false,
    filled: false,
    color: '#FFFFFF'
    value: null
  };
  let _holes = [];

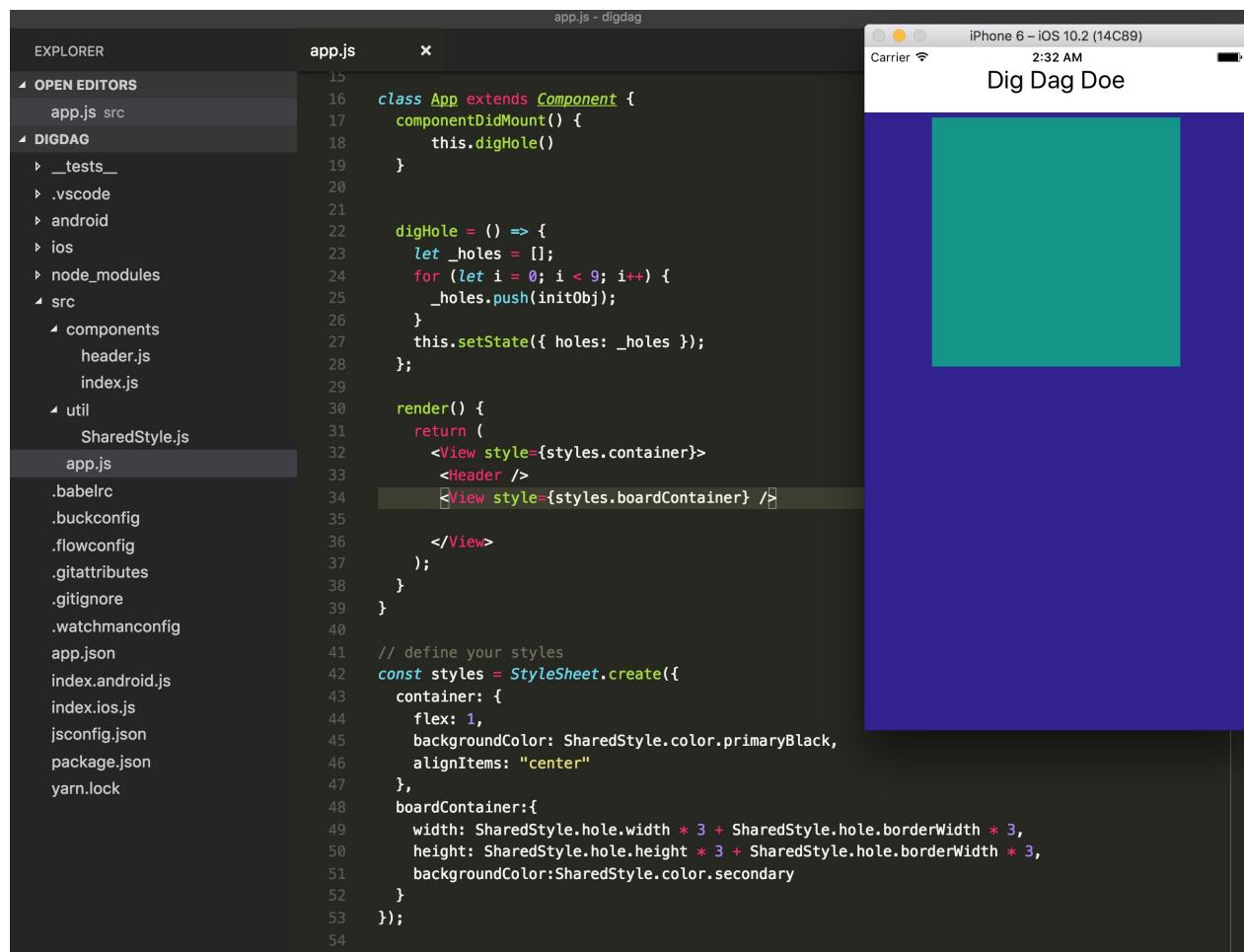
  for (let i = 0; i < 9; i++) {
    _holes.push(initObj);
  }
  this.setState({ holes: _holes });
};
```

Güzel artık elimde görselleştirebileceğim bir data'm var. Ama ondan önce bizim componentlerimiz arası width, height, konum ilişkisi oldukça fazla olacak. Bu yüzden bir tane componentlerin bu değerlerini bir araya toplamak adına bir kendimize ait customStyle objesini bir javascript dosyasında tutalım ve kullanacağımız yerlerde import edelim. Bunun için **src** klasörümüzde bir tane util klasörü yaratıp içine de **SharedStyle.js** isimli dosya ekleyelim. (initObj objesindeki color değerini de bu style'a göre değiştirelim)

```
// ./src/util/SharedStyle.js
const SharedStyle ={
  text:{
    small:12,
    regular:18,
    large:24,
    xlarge:32
  },
  color:{
    primary: '#673AB7',
    primaryBlack: '#311B92',
    secondary: '#009688',
    secondaryBlack: '#004D40',
    white: '#FFFFFF',
    disable: 'gray',
    red: '#f44336',
    yellow: '#2196F3',
    blue: '#ffeb3b'
  },
  hole:{
    width:80,
    height:80,
    borderWidth:1
  },
  header:{
    height:64,
    marginBottom:5
  }
}

export default SharedStyle
```

Tekrar app.js' e dönelim. Bir adet header componenti ve Board'umuz için bir boardContainer yaratalım. boardContainer style'a dikkat edin. SharedStyle'da verdığımız **width** ve **borderWidth** değerlerinin 3 katının toplamını verdik ki boardContainer'ımız tam anlamıyla karelerimiz için kapsayıcı olsun.



The screenshot shows a VS Code interface with the following details:

- EXPLORER:** Shows project files including `app.js`, `DIGDAG`, `src`, `components`, `util`, and configuration files like `.babelrc` and `.gitignore`.
- app.js:** The code is as follows:

```
15  class App extends Component {
16    componentDidMount() {
17      this.digHole()
18    }
19
20
21
22    digHole = () => {
23      let _holes = [];
24      for (let i = 0; i < 9; i++) {
25        _holes.push(initObj);
26      }
27      this.setState({ holes: _holes });
28    }
29
30    render() {
31      return (
32        <View style={styles.container}>
33          <Header />
34          <View style={styles.boardContainer} />
35        </View>
36      );
37    }
38  }
39
40 // define your styles
41 const styles = StyleSheet.create({
42   container: {
43     flex: 1,
44     backgroundColor: SharedStyle.color.primaryBlack,
45     alignItems: "center"
46   },
47   boardContainer: {
48     width: SharedStyle.hole.width * 3 + SharedStyle.hole.borderWidth * 3,
49     height: SharedStyle.hole.height * 3 + SharedStyle.hole.borderWidth * 3,
50     backgroundColor: SharedStyle.color.secondary
51   }
52 });
53 }
```

The right side shows an iPhone 6 simulator running iOS 10.2 (14C89) at 2:32 AM. The app title is "Dig Dag Doe". The screen displays a large teal square centered on a dark blue background.

Önce bir tane **Hole.js** isimli bir component yapıp holes state'inde tuttuğumuz datayı görselleştirmeye başlayalım.

```
// ./src/components/hole.js

import React, { Component } from "react";
import { View, Text, StyleSheet } from "react-native";
import SharedStyle from "../util/SharedStyle";

// create a component
class Hole extends Component {

  render() {
    const { hole } = this.props;

    return (
      <View style={styles.container} />
    );
  }
}

// define your styles
const styles = StyleSheet.create({
  container: {
    alignItems: "center",
    justifyContent: "center",
    width: SharedStyle.hole.width,
    height: SharedStyle.hole.height,
    backgroundColor: SharedStyle.color.white,
    borderWidth: SharedStyle.hole.borderWidth,
    borderColor: SharedStyle.color.primaryBlack
  }
});

//make this component available to the app
export default Hole;
```

The screenshot shows the Xcode interface with three tabs: app.js, index.js, and hole.js. The app.js tab is active, displaying the following code:

```
14  };
15
16 class App extends Component {
17   constructor(props) {
18     super(props);
19     this.state = {
20       holes: []
21     };
22   }
23
24   componentDidMount() {
25     this.digHole();
26   }
27
28   digHole = () => {
29     let _holes = [];
30     for (let i = 0; i < 9; i++) {
31       _holes.push(initObj);
32     }
33     this.setState({ holes: _holes });
34   };
35
36   render() {
37     const { holes } = this.state;
38     return (
39       <View style={styles.container}>
40         <Header />
41         <View style={styles.boardContainer}>
42           {holes.map((hole, i) => {
43             return <Hole key={i} />;
44           })}
45         </View>
46       </View>
47     );
48   }
49 }
```

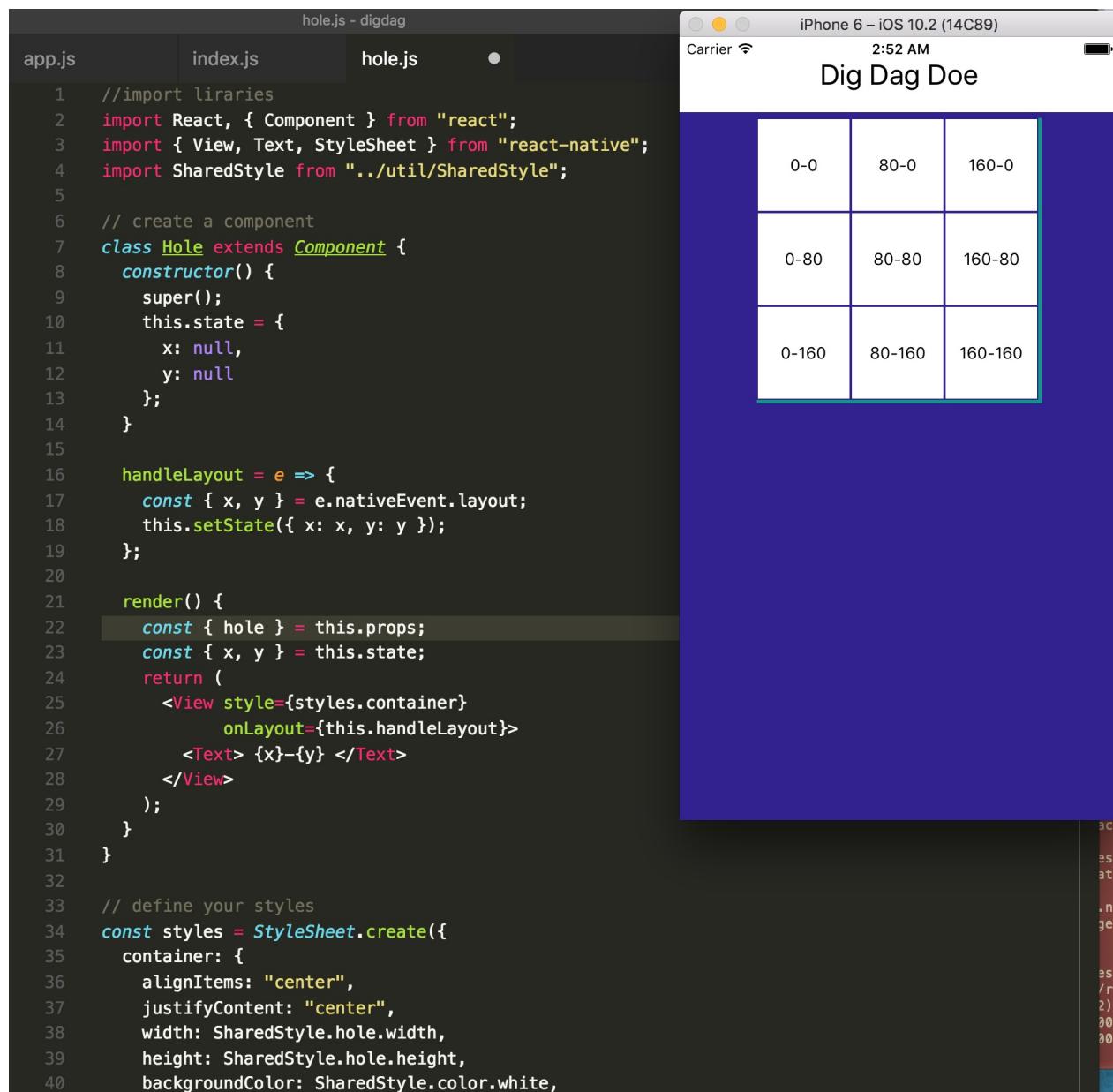
The iPhone 6 simulator window shows an application titled "Dig Dag Doe". The screen displays a 3x3 grid of holes. The top-left hole is white, while the other eight holes are teal. The background of the board area is dark blue.

**boardContainer tam bir kapsayıcı olmuşa benzemiyor. Bunun için boardContainer'a, flexWrap'i eklememiz gerekiyor.**

The screenshot shows the Xcode interface with the file 'app.js' open. The code defines a component with a 3x3 grid layout. The first two columns of the grid are white, while the third column is blue. The code uses React Native's styling and PanResponder API.

```
app.js - digdag
app.js      x  index.js  hole.js
44          });
45      </View>
46
47      </View>
48    );
49  }
50 }
51 // define your styles
52 const styles = StyleSheet.create({
53   container: {
54     flex: 1,
55     backgroundColor: SharedStyle.color.primaryBlack,
56     alignItems: "center"
57   },
58   boardContainer: {
59     width: SharedSt create<T extends StyleSheet.NamedS1
60     height: SharedS
61     backgroundColor Creates a StyleSheet style reference from th
62     flexWrap: 'wrap'
63   }
64 });
65
66 //make this component available to the app
67 export default App;
68
69
```

Sıra geldi karelerimizin sayfadaki pozisyonlarını bulmaya. Bunun için react-native'de birkaç yöntem var. Ama en pratigi her element yaratıldığında tetiklenen **onLayout** methodu.



The screenshot shows the Xcode interface with three files open: app.js, index.js, and hole.js. The hole.js file contains the following code:

```

hole.js - digdag
app.js | index.js | hole.js •

1 //import libraries
2 import React, { Component } from "react";
3 import { View, Text, StyleSheet } from "react-native";
4 import SharedStyle from "../util/SharedStyle";
5
6 // create a component
7 class Hole extends Component {
8     constructor() {
9         super();
10        this.state = {
11            x: null,
12            y: null
13        };
14    }
15
16    handleLayout = e => {
17        const { x, y } = e.nativeEvent.layout;
18        this.setState({ x: x, y: y });
19    };
20
21    render() {
22        const { hole } = this.props;
23        const { x, y } = this.state;
24        return (
25            <View style={styles.container}
26                onLayout={this.handleLayout}>
27                <Text> {x}-{y} </Text>
28            </View>
29        );
30    }
31 }
32
33 // define your styles
34 const styles = StyleSheet.create({
35     container: {
36         alignItems: "center",
37         justifyContent: "center",
38         width: SharedStyle.hole.width,
39         height: SharedStyle.hole.height,
40         backgroundColor: SharedStyle.color.white,

```

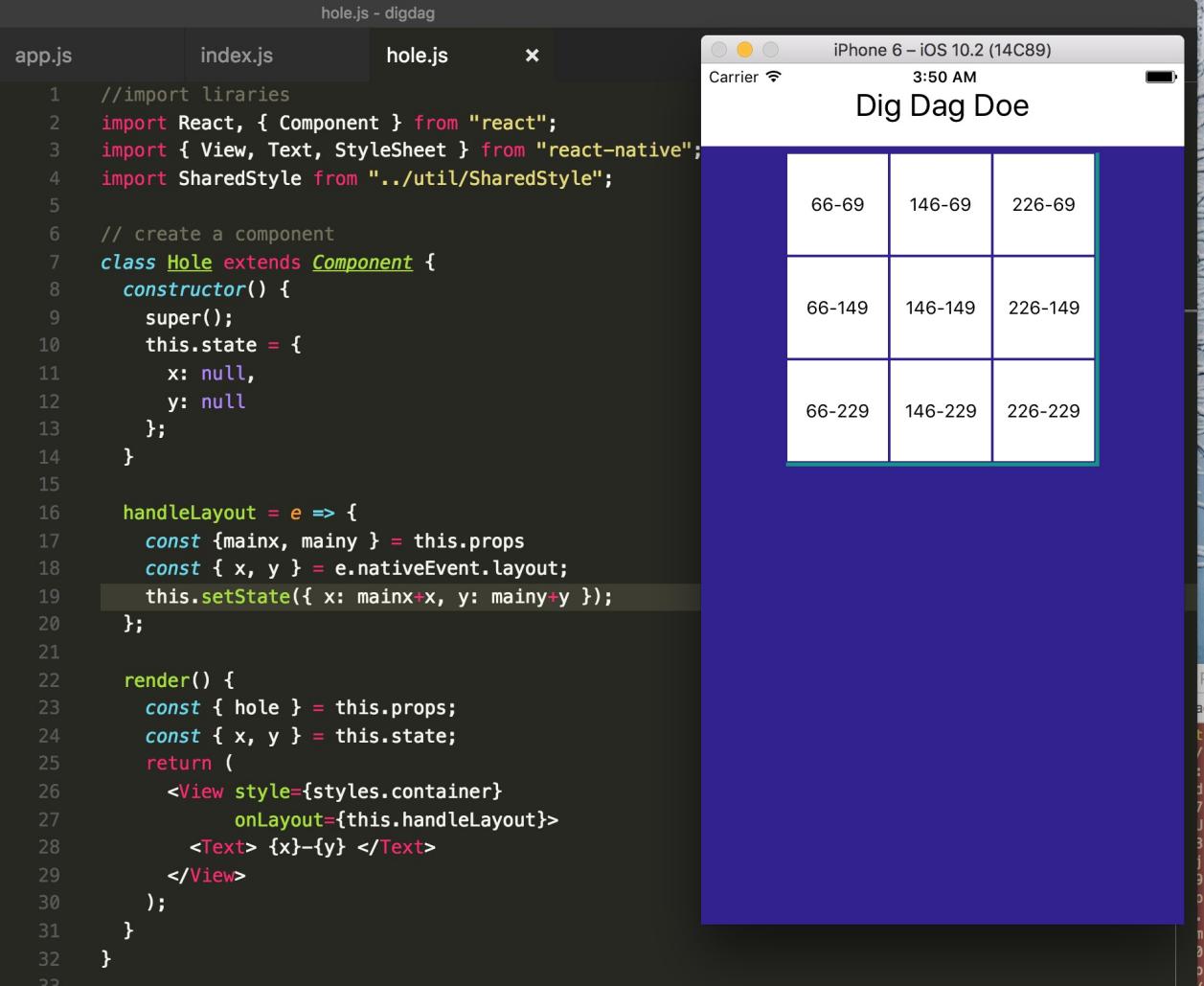
The iPhone 6 simulator window shows a 3x3 grid of coordinates. The columns are labeled 0-80, 80-80, and 160-80. The rows are labeled 0-0, 80-0, and 160-0. The grid is defined by purple lines.

Bir şeyle ters gidiyor. onLayout methodunu çağrırdık koordinatları aldık ama bu koordinatlar hole componentin kendi üstündeki View'e göre boardContainer'a göre değerleri. Bize sayfaya göre olanı lazım. Bunun için bizim boardContainer'ın koordinatlarına da ihtiyacımız var. app.js'e dönüp boardContainer'in koordinatlarını alıp, hole componentin her birine bunları gönderelim. onLayout methodu component mount edildikten sonra çağrıldığı için bir render koşulu ekleyelim. mainx ve mainy belli değil ise ekranda bir loader gösterelim.

```
///...src/app.js
...
handleLayout = e => {
  const { x, y } = e.nativeEvent.layout;
  this.setState({ mainx: x, mainy: y });
};

render() {
  const { holes, mainx, mainy } = this.state;
  return (
    <View style={styles.container}>
      <Header />
      <View style={styles.boardContainer} onLayout={this.handleLayout}>
        {mainx !== null && mainy !== null
          ? holes.map((hole, i) => {
              return <Hole key={i} mainx={mainx} mainy={mainy} />;
            })
          : <ActivityIndicator />}
      </View>
    </View>
  );
}
...
```

Şimdi hole.js componentimizde koordinat bulma işlemine mainx ve mainy props'larını da katalım. Aşağıdaki koordinatlar gayet doğru gözükmüyor.



The screenshot shows the Xcode interface with three files open: app.js, index.js, and hole.js. The hole.js file contains the following code:

```

hole.js - digdag
app.js | index.js | hole.js | x
1 //import libraries
2 import React, { Component } from "react";
3 import { View, Text, StyleSheet } from "react-native";
4 import SharedStyle from "../util/SharedStyle";
5
6 // create a component
7 class Hole extends Component {
8   constructor() {
9     super();
10    this.state = {
11      x: null,
12      y: null
13    };
14  }
15
16  handleLayout = e => {
17    const {mainx, mainy} = this.props
18    const { x, y } = e.nativeEvent.layout;
19    this.setState({ x: mainx+x, y: mainy+y });
20  };
21
22  render() {
23    const { hole } = this.props;
24    const { x, y } = this.state;
25    return (
26      <View style={styles.container}
27        onLayout={this.handleLayout}>
28        <Text>{x}-{y}</Text>
29      </View>
30    );
31  }
32}
33

```

The simulator window shows an iPhone 6 running iOS 10.2 with the title "Dig Dag Doe". It displays a 3x3 grid of numbers:

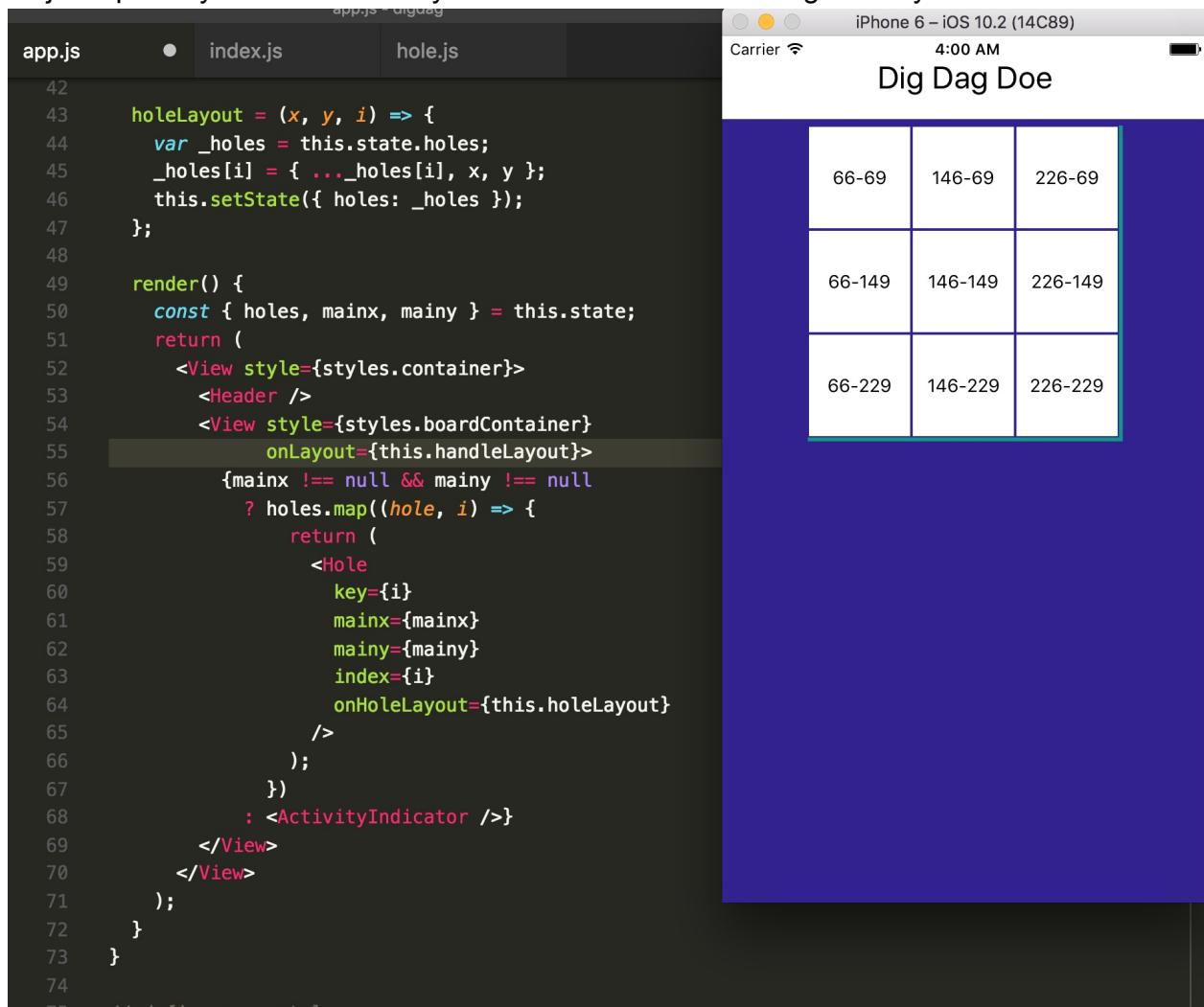
66-69	146-69	226-69
66-149	146-149	226-149
66-229	146-229	226-229

Yukarıda karelere ait array'i yaratırken bir başlangıç objesi set etmiştık ve onda pozisyon değerleri {x:0, y:0} şeklindeydi. Şimdi o state'deki array'i değiştirelim. Bunun için hole.js içinde handleLayout methodun da üstte kullanılmak üzere bir function props fırlatalım.

```
// ./src/components/hole.js

handleLayout = e => {
  const {mainx, mainy, index} = this.props
  const { x, y } = e.nativeEvent.layout;
  this.setState({ x: mainx+x, y: mainy+y });
  this.props.onHoleLayout(mainx + x, mainy + y, index);
};
```

Object spread yardımı ile holeLayout methodunda state'ımızı güncelleyelim.



The screenshot shows the Xcode interface with two tabs open: 'app.js' and 'index.js'. The 'app.js' tab is active, displaying the following code:

```

42     holeLayout = (x, y, i) => {
43       var _holes = this.state.holes;
44       _holes[i] = { ..._holes[i], x, y };
45       this.setState({ holes: _holes });
46     };
47
48
49     render() {
50       const { holes, mainx, mainy } = this.state;
51       return (
52         <View style={styles.container}>
53           <Header />
54           <View style={styles.boardContainer}
55             onLayout={this.handleLayout}>
56             {mainx !== null && mainy !== null
57               ? holes.map((hole, i) => {
58                 return (
59                   <Hole
60                     key={i}
61                     mainx={mainx}
62                     mainy={mainy}
63                     index={i}
64                     onHoleLayout={this.holeLayout}
65                   />
66                 );
67               })
68               : <ActivityIndicator />}
69           </View>
70         </View>
71       );
72     }
73   }
74
75 // define your styles

```

The iPhone 6 simulator window shows a 3x3 grid of numbers. The grid is defined by a border and contains the following values:

66-69	146-69	226-69
66-149	146-149	226-149
66-229	146-229	226-229

## 2. PanResponder Element Yaratalım

Toplamda bizim 9 tane topumuz olacak. Bunların kendine özgü renkleri olacak. Ve sürüklendiğinde bırakılan top bir daha sürüklenemeyecek. Bu verilere dayanarak kendimize yine state'te tutacağımız bir array yaratalım.

```

balls:[
  { id: 1, value: "R", color: SharedStyle.color.red, selected: false },
  { id: 2, value: "R", color: SharedStyle.color.red, selected: false },
  { id: 3, value: "R", color: SharedStyle.color.red, selected: false },
  { id: 4, value: "Y", color: SharedStyle.color.yellow, selected: false },
  { id: 5, value: "Y", color: SharedStyle.color.yellow, selected: false },
  { id: 6, value: "Y", color: SharedStyle.color.yellow, selected: false },
  { id: 7, value: "B", color: SharedStyle.color.blue, selected: false },
  { id: 8, value: "B", color: SharedStyle.color.blue, selected: false },
  { id: 9, value: "B", color: SharedStyle.color.blue, selected: false }
]

```

Bu array'i görselleştireceğimiz bir ball.js isimli bir component yaratalım sonra da ona drag&drop özellikleri ekleyelim.

```
// ./src/components/Ball.js
import React, { Component } from "react";
import { View, Text, StyleSheet, PanResponder } from "react-native";

// create a component
class Ball extends Component {

  componentWillMount() {
    this.panResponder = PanResponder.create({
      onStartShouldSetPanResponder: (evt, gestureState) => true,
      onMoveShouldSetPanResponder: (evt, gestureState) => true,
    })
  }

  render() {
    return <View style={styles.container} {...this.panResponder.panHandlers}>;
  }
}
```

Başlangıçta componentimiz touch eventlere tepki versin ve touch eventlerle hareket edebilmesi için `onStartShouldSetPanResponder` ve `onMoveShouldSetPanResponder` fonksiyonlarının `true` döndürmesini sağladık. Bu fonksiyonlar tetiklendikten sonra iki tane daha fonksiyon tetiklenecektir. Bunlardan birincisi başlangıç değerlerini set edebileceğimiz, `onPanResponderGrant`; parmağımız componenti sürüklemeye başlayınca ekranda dokunduğumuz yerlerin coordinatlarını almamızı sağlayan `onPanResponderMove`.

Önemli: Biz PanResponer API'ı kullanırken elementin değerlerini filan alıyoruz. Biz ekranda dokunduğumuz yerlerin koordinat değerlerini alıp, elementin style'ını değiştiriyoruz.

Şimdi başlangıç değerlerimizi set edelim. Burada Animated API kullanacağız. bunun için ilk önce constructor'da bir Animated state yaratalım.

```
// ./src/components/Ball.js
...
class Ball extends Component {
  constructor(props) {
    super(props);
    this.state = {
      pan: new Animated.ValueXY()
    };
  }
  ...
}
```

Şimdi onResponderGrant'da bu state'imize değer verelim.

```
// ./src/components/Ball.js
...
componentWillMount() {
  this.panResponder = PanResponder.create({
    onStartShouldSetPanResponder: (evt, gestureState) => true,
    onMoveShouldSetPanResponder: (evt, gestureState) => true,
    onPanResponderGrant: (e, gestureState) => {
      this.state.pan.setValue({ x: 0, y: 0 });
    },
  })
}
...
...
```

Component'i tutup elimizle sürükleyince tetiklenecek olan `onPanResponderMove` için bir fonksiyon atayalım.

```
// ./src/components/Ball.js
...
componentWillMount() {
  this.panResponder = PanResponder.create({
    onStartShouldSetPanResponder: (evt, gestureState) => true,
    onMoveShouldSetPanResponder: (evt, gestureState) => true,
    onPanResponderGrant: (e, gestureState) => {
      this.state.pan.setValue({ x: 0, y: 0 });
    },
    onPanResponderMove: this.handleResponderMove,
  })
}

handleResponderMove = (evt, gestureState) => {
  this.state.pan.setValue({ x: gestureState.dx, y: gestureState.dy });
};}
...
...
```

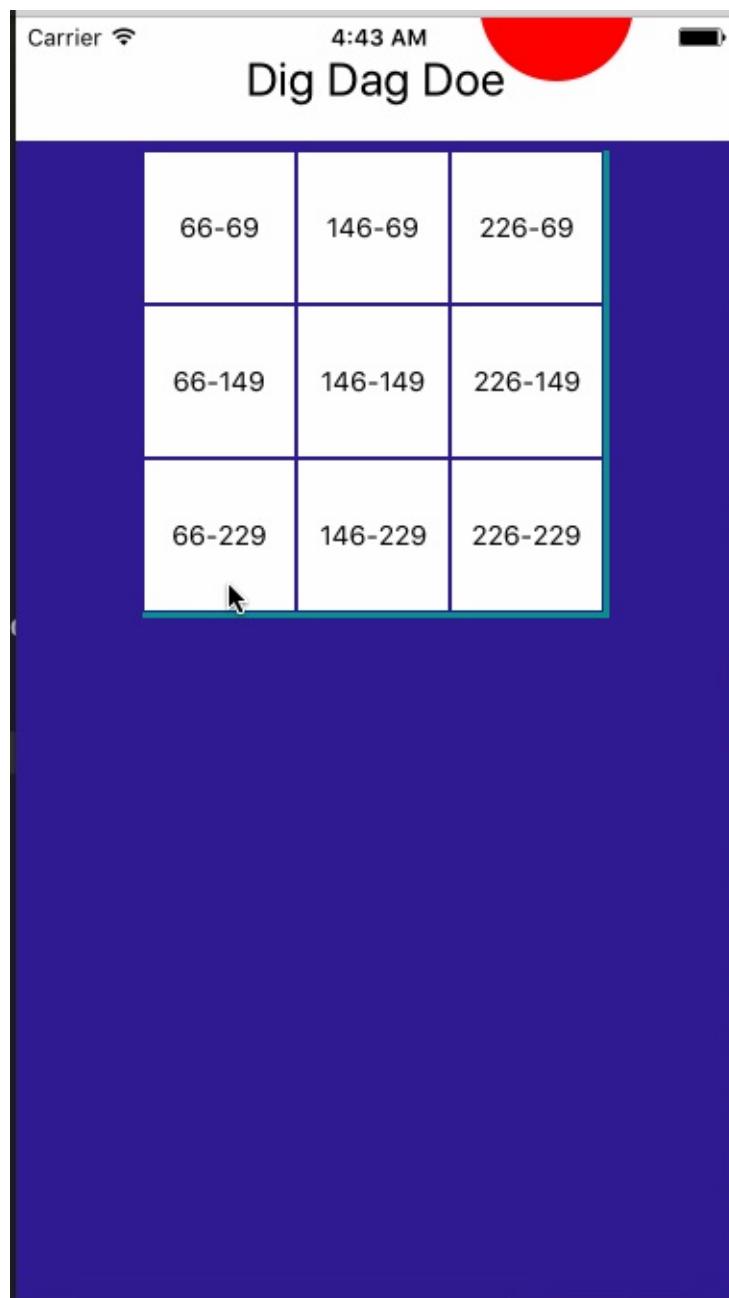
`this.state.pan` değişikçe değişecek olan style yapıp, View'den oluşan ball componentini Animated.View'e çevirelim.

```
// ./src/components/Ball.js
... render() {
  const animatedStyle = {
    transform: this.state.pan.getTranslateTransform(),
    backgroundColor: "red"
  };

  return (
    <Animated.View
      style={[styles.container, animatedStyle]}
      {...this.panResponder.panHandlers}
    />
  );
}
...
```

getTranslateTranform, tranform:[{translateX: value }, {translateY:value} ] işini yapan  
Animated value ile kullanılabilen hazır bir fonksiyon

Şimdi örnek tek bir Ball componentini app.js' e import edip neler oluyor bir bakalım.



Yine bir şeyler ters gidiyor. İlk defa sürüklerken bir sorun olmuyor fakat ikinci kez topa dokunduğumuzda topun pozisyonu uzaklara kaçıyor. Bunu çözmek için 2 tane global değişken yaratacağız (`this._animatedValueX`, `this.animatedValueY`) ve bu iki değişkeni dinleyen listenerlarımız olacak. Topu son bıraktığımız yerin değerini kendilerinde tutup, ikinci kez dokunduğumuzda tekrar tetiklenen, `onPanResponderGrant` methodunda bunlarda tuttuğumuz değerleri `this.state.pan` state'ine atacağız. O da dolayısıyla elementin style'ını update edecek.

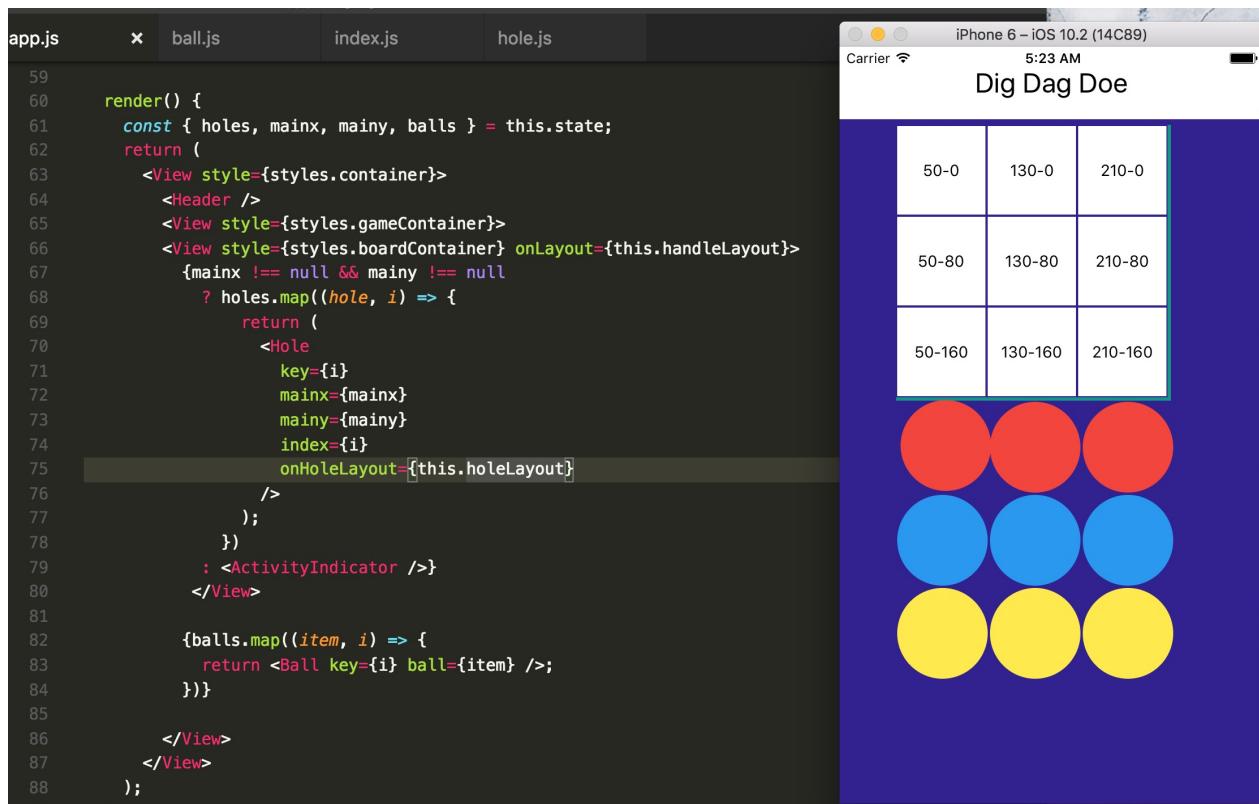
*Listenerları `componentWillUnmount`'da remove etmeyi unutmayın*

The screenshot shows a code editor with several files: p.js, ball.js, index.js, and hole.js. The p.js file contains code for a PanResponder. A red rectangle highlights the line `onPanResponderMove: this.handleResponderMove`. To the right is an iPhone 6 simulator running iOS 10.2 (14C89) at 4:51 AM. The app title is "Dig Dag Doe". The screen displays a 3x3 grid of numbers representing coordinates. A red circle highlights the top-left cell (66-69, 146-69, 226-69). A cursor arrow is visible on the screen.

```
13     }
14
15     componentWillMount() {
16       this._animatedValueX = 0;
17       this._animatedValueY = 0;
18       this.state.pan.x.addListener(value => (this._animatedValueX = value.value));
19       this.state.pan.y.addListener(value => (this._animatedValueY = value.value));
20       this.panResponder = PanResponder.create({
21         onStartShouldSetPanResponder: (evt, gestureState) => true,
22         onMoveShouldSetPanResponder: (evt, gestureState) => true,
23         onPanResponderGrant: (e, gestureState) => {
24           this.state.pan.setOffset({
25             x: this._animatedValueX,
26             y: this._animatedValueY
27           });
28           this.state.pan.setValue({ x: 0, y: 0 });
29         },
30         onPanResponderMove: this.handleResponderMove
31       });
32     }
33
34     handleResponderMove = (evt, gestureState) => {
35       this.state.pan.setValue({ x: gestureState.dx, y: gestureState.dy });
36     };
37
38     render() {
39       const animatedStyle = {
40         transform: this.state.pan.getTranslateTransform(),
41         backgroundColor: "red"
42       };
43     }
44   }
45 }
```

Şimdi tüm toplarımızı map edelim

```
// ./src/app.js
...
{balls.map((item, i) => {
  return <Ball key={i} ball={item} />;
})}
...
```



### 3-Hover Effect Ekleme

PanResponder sisteminde `onPanResponderMove` ile ekranın nerdesinde sürüklendirme işlemi oluyor anlayabiliyoruz. Hover effect yaparken de bunu kullanacağız. `onPanResponderMove` aktif iken gerekli değerleri alıp, üst component'e function props fırlatıp, effect oluşturma işlemini de üst component'e halledeceğiz.

```
// ./src/components/ball.js
...
handleResponderMove = (evt, gestureState) => {
  this.state.pan.setValue({ x: gestureState.dx, y: gestureState.dy });
  this.props.onPositionChanging(evt.nativeEvent.pageX, evt.nativeEvent.pageY);
};
...
```

```
// ./src/app.js
...
{balls.map((item, i) => {
    return (
        <Ball
            key={i}
            ball={item}
            onPositionChanging={this.handlePositionChanging}
        />
    );
})}
...
```

nativeEvent ve gestureState kavramları için; Panresponder sisteminde her bir method bu iki değişkeni bize otomatik verir. Bu iki değişken (evt.nativeEvent ve gestureState) aslında bir obje ve biz burada dokunduğumuz yerin nativeEvent içerisinde sayfaya göre konumunu pageX ve pageY değişkenlerinden alıyoruz. Daha fazla ayrıntı için resmi dökümanda [şuraya](#) bakabilirsiniz

Şimdi bir kaç util fonksiyonu yazalım. Bunlardan biri ben bir karenin üstünde miyim değil miyim onu söyleyecek olan isDropZone fonksiyonu diğer üstündeyken holes state'ini değiştirecek olan hoverDropZone fonksiyonu. İşin logic tarafına girmiyorum. Çünkü gayet koddan okunabilir diye düşünüyorum.

```
// ./src/util/index.js

export const isDropZone = (hole, itemX, itemY, width, height, headerHeight) => {
  if (
    !hole.filled &&
    (hole.x <= itemX && hole.x + width >= itemX) &&
    (hole.y+headerHeight < itemY && hole.y + height+headerHeight > itemY)
  ) {
    return true;
  } else {
    return false;
  }
};

export const hoverDropZone = (holes, itemX, itemY, width, height, headerHeight) => {
  for (let i = 0; i < holes.length; i++) {
    if (isDropZone(holes[i], itemX, itemY, width, height, headerHeight)) {
      holes[i].hovering = true;
    } else {
      holes[i].hovering = false;
    }
  }

  return holes;
};
}
```

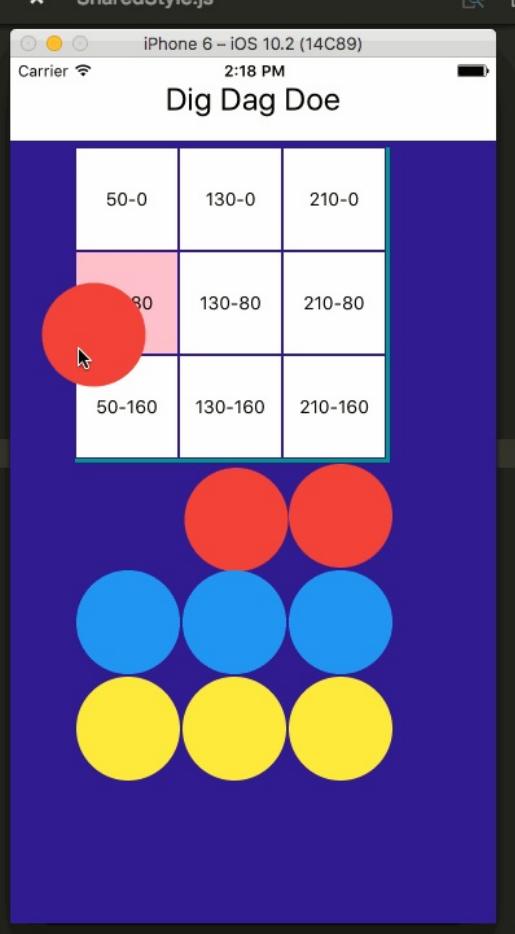
Burada ayrıntı belki de algoritmayı kurarken yaptığım bir hata, sayfadaki header boyutuna da ihtiyacım var. İşin açıkçası karelerin koordinatlarını hesaplarken onu da içine katıp bu parametreye burada ihtiyaç duymayabilirdik. Ama sorun değil.

Şimdi app.js'de handlePositionChanging methodunda hoverDropZone'u çağıralım. Ordan dönen holes array'i ile state'i güncelleyelim.

```
// ./src/app.js
...
handlePositionChanging = (itemX, itemY) => {
  let _holes = this.state.holes;

  _holes = hoverDropZone(
    _holes,
    itemX,
    itemY,
    SharedStyle.hole.width,
    SharedStyle.hole.height,
    SharedStyle.header.height
  );
  this.setState({ holes: _holes });
};
...
```

Şimdi this.state.holes state'i sorumlu olduğu kareye kendi üzerinde neler döndüğünü anlatabilir. Kareye ona göre style ekleyelim.



```

p.js          index.js .../util    index.js .../components    hole.js      SharedStyle.js
11         x: null,
12         y: null
13     };
14 }
15
16 handleLayout = e => {
17     const { mainx, mainy, index } = this.props;
18     const { x, y } = e.nativeEvent.layout;
19     this.setState({ x: mainx + x, y: mainy + y });
20     this.props.onHoleLayout(mainx + x, mainy + y, index);
21 };
22
23 render() {
24     const { hole } = this.props;
25     const { x, y } = this.state;
26     const holeHoverStyle = [
27         backgroundColor: hole.hovering
28             ? SharedStyle.color.hoverColor
29             : SharedStyle.color.white
30     ];
31
32     return (
33         <View
34             style={[styles.container, holeHoverStyle]}
35             onLayout={this.handleLayout}
36         >
37             <Text> {x}-{y} </Text>
38         </View>
39     );
40 }
41 }
42 // define your styles
43

```

## 4-Drop - Release - Bırakma İşlemi

Tekrar PanResponder componentimize dönüp yeni bir method ekliyoruz, `onPanResponderRelease:this.handleRelease` methodu. Bu method ne yapacak ? Topun bırakıldığı koordinatları ve hangi topun bırakıldığını üst componente haber verecek.

```

// ./src/components/ball.js
...
handleRelease = (evt, gestureState) => {
    const {ball} = this.props;
    this.state.pan.flattenOffset();
    this.props.onDrop(ball, evt.nativeEvent.pageX, evt.nativeEvent.pageY);
};
...

```

Üst component'te onDrop props'unu karşıyalalım

```
// ./src/app.js
...
{balls.map((item, i) => {
    return (
        <Ball
            key={i}
            ball={item}
            onPositionChanging={this.handlePositionChanging}
            onDrop={this.handleDrop}
        />
    );
})}
...
```

Burada bir util fonksiyon daha yazalım, selectDropZone diye. Buda bizim balls ve holes array'ımız değiştirsün. Ve bizde o değiştirdiği array'ler ile state'lerimizi güncelleyelim.

```
// ./src/util.js

export const selectDropZone = (holes, itemX, itemY, width, height, balls, ball, headerHeight) => {
    for (let i = 0; i < holes.length; i++) {
        if (isDropZone(holes[i], itemX, itemY, width, height, headerHeight)) {

            holes[i].hovering = false;
            holes[i].filled = true;
            holes[i].color = ball.color;
            holes[i].value = ball.value;

            balls = selectBall(balls, ball);
        }
    }

    return {holes, balls};
};

export const selectBall=(balls ,ball)=>{
    for (let i = 0; i < balls.length; i++) {
        if (balls[i].id === ball.id) {
            balls[i].selected = true;
        }
    }

    return balls
}
```

Yukarıdaki algoritma çok basit. for döngüsü ile ilk önce hangi karenin üzerinde olduğunu karar veriyor. Sonra o karenin renk ve ilgili değerlerini değiştiriyor sonra da, hangi topun seçildiğine karar verip, o topun bir daha seçilmemesi adına selected özelliğini false'a çekiyor.

Şimdi bu util fonksiyonlarını handleDrop methodunda çağıralım.

```
// ./src/app.js
...
handleDrop = (ball, itemX, itemY) => {
  let _holes = this.state.holes;
  let _balls = this.state.balls;

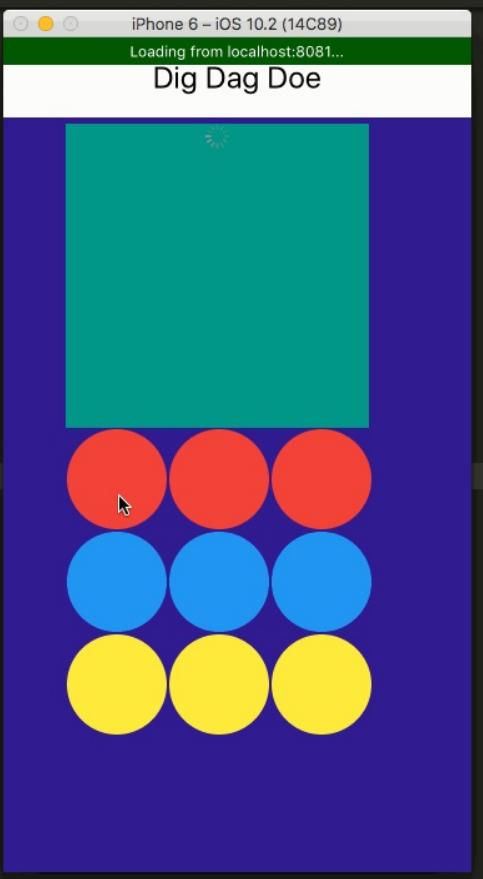
  const result = selectDropZone(
    _holes,
    itemX,
    itemY,
    SharedStyle.hole.width,
    SharedStyle.hole.height,
    _balls,
    ball,
    SharedStyle.header.height
  );
  _holes = result.holes;
  _balls = result.balls;

  this.setState({ holes: _holes, balls: _balls });
};

...
...
```

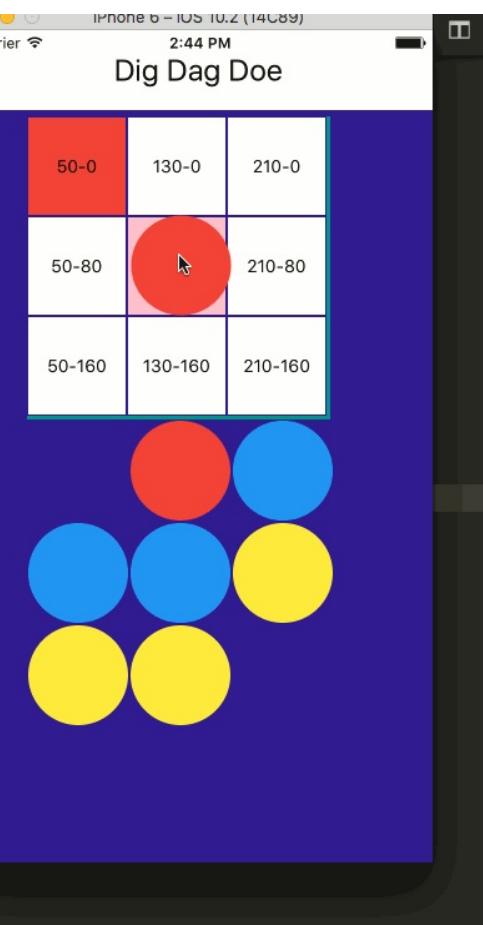
hole ve ball componentlerimizin style'larının yeni state'e göre şekilaması lazım. Onları ekleyelim. İlk önce hole componentine bakalım. Hovering ise hoverColor, filled ise toptan aldığı renk, hiçbirdeğilse beyaz gözükecek. ( Ne dersiniz bu yöntemi biraz değiştirerek basit bir kalemlle çizim uygulaması yapılabilir mi ? )

## PanResponder



```
p.js      index.js .../util    index.js .../components    hole.js    ball.js    SharedStyle.js
11      x: null,
12      y: null
13  };
14
15
16  handleLayout = e => {
17    const { mainx, mainy, index } = this.props;
18    const { x, y } = e.nativeEvent.layout;
19    this.setState({ x: mainx + x, y: mainy + y });
20    this.props.onHoleLayout(mainx + x, mainy + y, index);
21  };
22
23  render() {
24    const { hole } = this.props;
25    const { x, y } = this.state;
26    const holeExtraStyle = {
27      backgroundColor: hole.hovering
28        ? SharedStyle.color.hoverColor
29        : hole.filled ? hole.color : SharedStyle.color.white
30    };
31
32    return (
33      <View
34        style={[styles.container, holeExtraStyle]}
35        onLayout={this.handleLayout}
36      >
37        <Text> {x}-{y} </Text>
38      </View>
39    );
40  }
41}
42
43 // define your styles
```

Şimdi ball.js componentine sen seçildinse ortadan kaybol diyelim.



```
app.js     index.js .../util    index.js .../components    hole.js    ball.js    SharedStyle.js
40  handleResponderMove = (evt, gestureState) => {
41    this.state.pan.setValue({ x: gestureState.dx, y: gestureState.dy });
42    this.props.onPositionChanging(evt.nativeEvent.pageX, evt.nativeEvent.pageY);
43  };
44
45  handleRelease = (evt, gestureState) => {
46    const {ball} = this.props;
47    this.state.pan.flattenOffset();
48    this.props.onDrop(ball, evt.nativeEvent.pageX, evt.nativeEvent.pageY);
49  };
50
51  render() {
52    const { ball } = this.props;
53    const animatedStyle = {
54      transform: this.state.pan.getTranslateTransform(),
55      backgroundColor: ball.color,
56      display: ball.selected ? "none" : "flex",
57    };
58
59    return (
60      <Animated.View
61        style={[styles.container, animatedStyle]}
62        {...this.panResponder.panHandlers}
63      />
64    );
65  }
66}
67
68 // define your styles
69 const styles = StyleSheet.create({
70   container: {
71     width: SharedStyle.hole.width,
72     height: SharedStyle.hole.height,
```

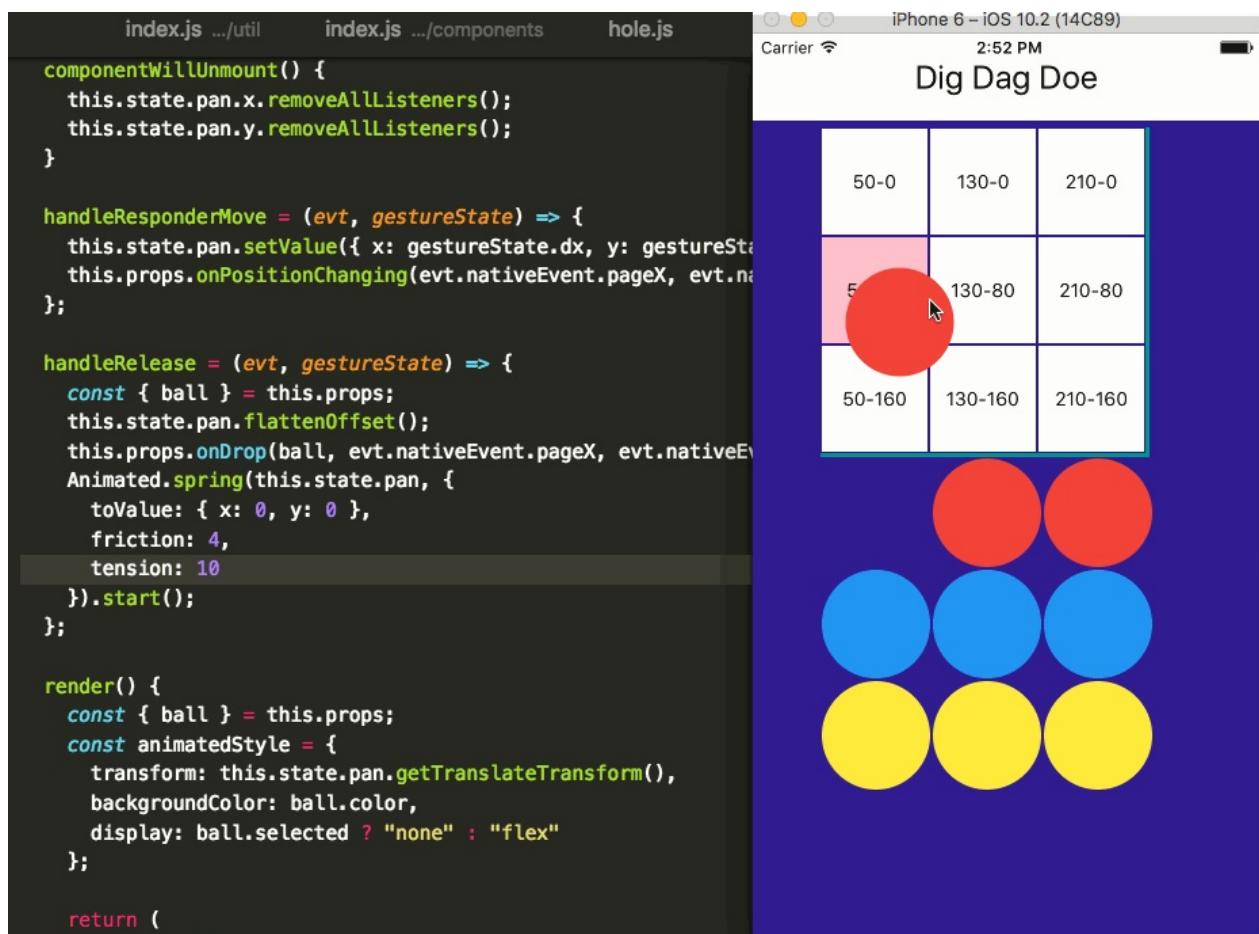
Şimdi top bırakıldığında boardContainer'da değilse yada dolu bir karenin üzerindeyse yerine geriye dönmesini isteyelim. İçin burası çok basit sadece release methoduna Animated fonksiyonu ekleyeceğiz.

```
// ./src/components/ball.js
...
handleRelease = (evt, gestureState) => {
  const {ball} = this.props;
  this.state.pan.flattenOffset();
  this.props.onDrop(ball, evt.nativeEvent.pageX, evt.nativeEvent.pageY);
  Animated.spring(this.state.pan, {
    toValue: {x:0 , y:0},
    friction:4,      //canlılık değeri diyeceğim ama çok bir şey anlatmıyor
    tension:10       //hız kontrolü diyelim buna da
  }).start()
};

...

```

Animated.spring'de diyoruz ki; top bırakıldığında this.state.pan başlangıç değerini {x:0 , y:0} değerine çek. Bunu da kendine özel animasyonunla yap.



Bu kadar, kodun tamamı için <https://github.com/ysfzrn/react-native-panresponder-demo>



# Navigation

React Native'de sayfalar arası geçiş, router için söylenecek çok şey var. Önceki versiyonlarda sunulan direkt react-native kütüphanesinde yer alan Navigator pek sevilmedi. Redux implementasyonu neredeyse imkansızdı. Sonra Navigation-Experimental diye bir şey denendi. Adı üstünde experimental (deneysel) bir çalışmaya ve bana göre başarısız oldu. Daha sonra bu bilgi birikimini toparlayıp, react-native kominitesi tarafından [react-navigation](#) piyasaya sürdü. Oldukça kapsamlı ve güzel bir kütüphane oldu. Ancak yine tatmin edici değildi. Normal IOS ve Android uygulamalardaki kullanıcı deneyimini sağlamıyordu. Bunun için native bir çözüm daha mantıklı ve gerekliydi. [wix.com](#) da çalışan geliştiricilerde böyle düşünmüş olacaklar ki tamamen native bir kütüphane geliştirmişler, [wix/react-native-navigation](#) .

Bir kaç, navigation kütüphanesi

- [react-navigation](#) ( Javascript çözümler için de en iyisi )
- [react-native-router-flux](#)
- [react-native-router](#)
- [react-native-router-redux](#)
- [react-native-redux-router](#)
- [react-router](#) (Bu web tarafında kullandığımız react-router. V4 ile native özelliği geldi )

## wix/react-native-navigation

Gerçekten kullanımı diğer kütüphanelere çok daha kolay. Redux entegrasyonu diğerlerine çok çok daha kolay. Tek problem native bir component olduğu için, yükleme sırasında manuel link yapmak durumunda sizi bırakıyor. Dökümantasyonu da kütüphanenin büyülüğüne göre biraz basit kaçmış gibi. Discord'da kütüphanenin yaratıcılarıyla konuştugunuzda aslında kütüphanenin dökümante edilmemiş bir çok özelliğiyle karşılaşıyorsunuz. Örneğin, Android'de header tabbar ve ona stil vermekten henüz bahsedilmemiş durumda. Bunu yaratıcıları biz bu özelliği kendi uygulamamızda kullanıyoruz ve implemantasyonu tam istediğimiz gibi olmadı bir sonraki versiyonlarda bu kısım değişebilir o yüzden dökümante etmedik diyorlar. Bu sizi korkutmasın, kütüphane oldukça sağlam, stable durumda ve arkasında güzel güçlü bir kominitesi var.

## airbnb/native-navigation

2017 React Konferansında airbnb'den [Leland Richardson](#) yeni bir native navigation component tanıttı, [airbnb/native-navigation](#). React-Native de ki NavigatorIOS ve wix/react-native-navigation'dan sonra 3. native route kütüphanesi. Yalnız şimdilik beta sürümü yayında, airbnb kendi uygulamasında sorunsuz kullanmaya başladıkten sonra ilk versiyonu yayinallyayacağını açıkladı. Sunumu izleyince çoğu geliştiriciyi bu haber heyecanlandırdı. Özellikle sayfa geçişlerinde animasyon eklemek hayli zorlayıcı olabiliyor. Bu tip native ve smooth geçişleri bize vaaden bir kütüphane olacak. Bu kütüphane hazır olana kadar wix/react-native-navigation'ı kullanmanızı öneririm. Daha fazla bilgi ve navigation sorunsalına geniş bir açıdan bakmak için, aşağıdaki sunumu izleyebilirsiniz



Not: Sunum videosu gözükmüyorsa, adres çubuğundan **load unsafe script**'i aktif etmelisiniz.

# wix/react-native-navigation

[IOS Kurulumu](#)

[Android Kurulumu](#)

[Temel Kullanım](#)

[Icon Ekleme - TabBar](#)

# IOS Kurulumu

1- react native versiyonu olarak 0.43 ve üstünü kullanıyor olduğunuzdan emin olun.

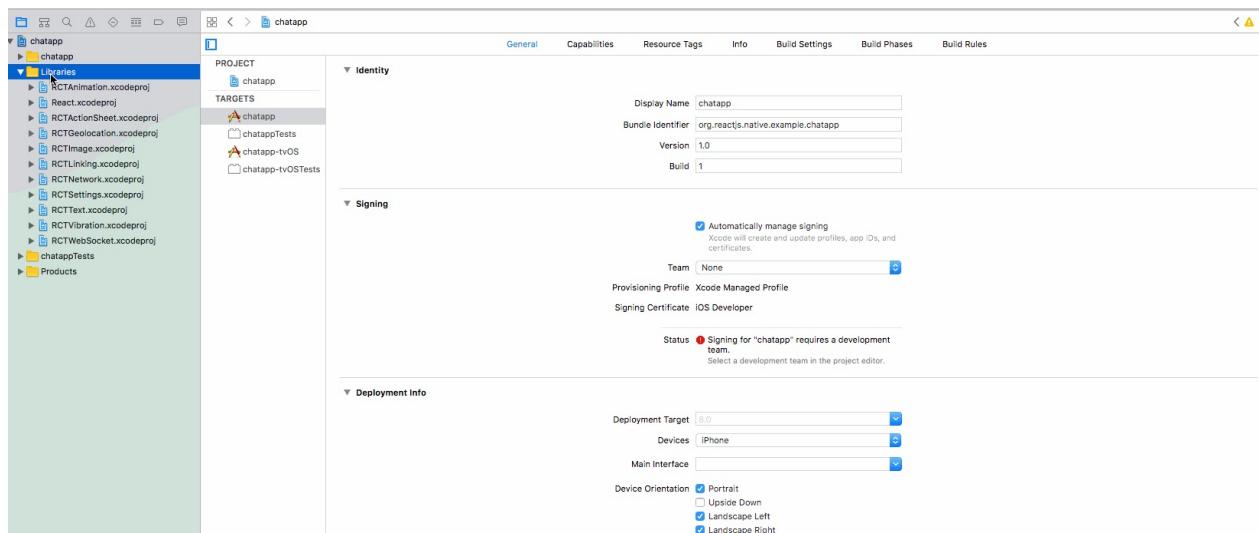
2- npm kullanıyorsanız 3 ve üstü versiyon kullandığınızdan emin olun.

```
npm install react-native-navigation@latest
```

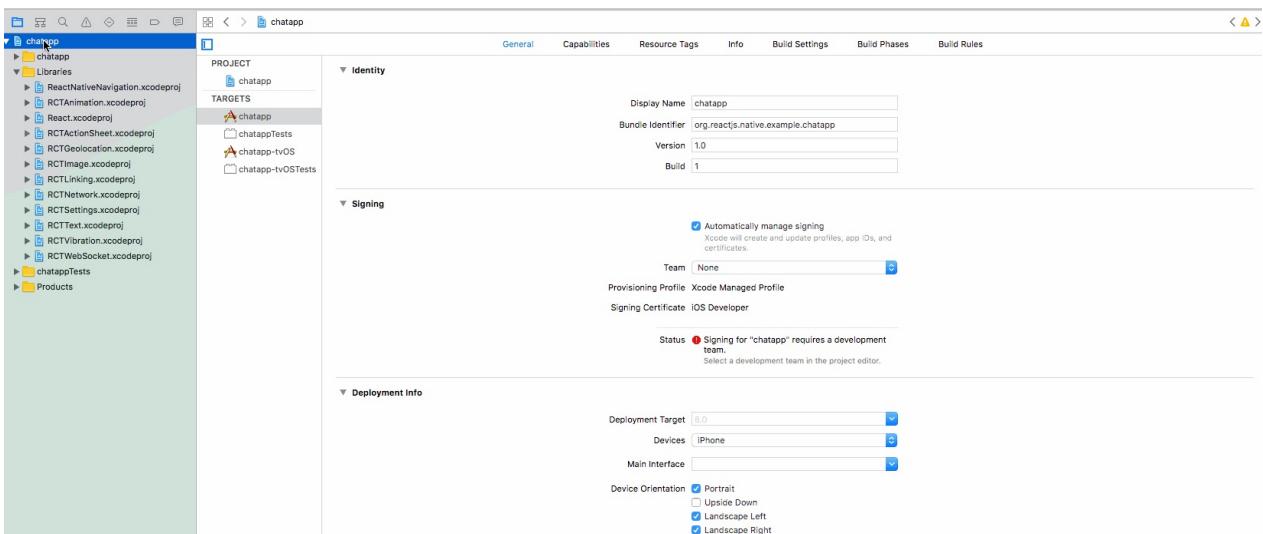
3- yarn kullanıyorsanız

```
yarn add react-native-navigation@latest
```

4-react-native init ile kurduğunuz projede ios bölümünü **XCode** ile açın. **Libraries** kısmına sağ tıklayıp **Add Files** seçeneğini seçin. **node\_modules** klasörüne kaydedilen, **react-native-navigation/ios/ReactNativeNavigation.xcodeproj** projesini **Libraries** kısmına ekleyin.

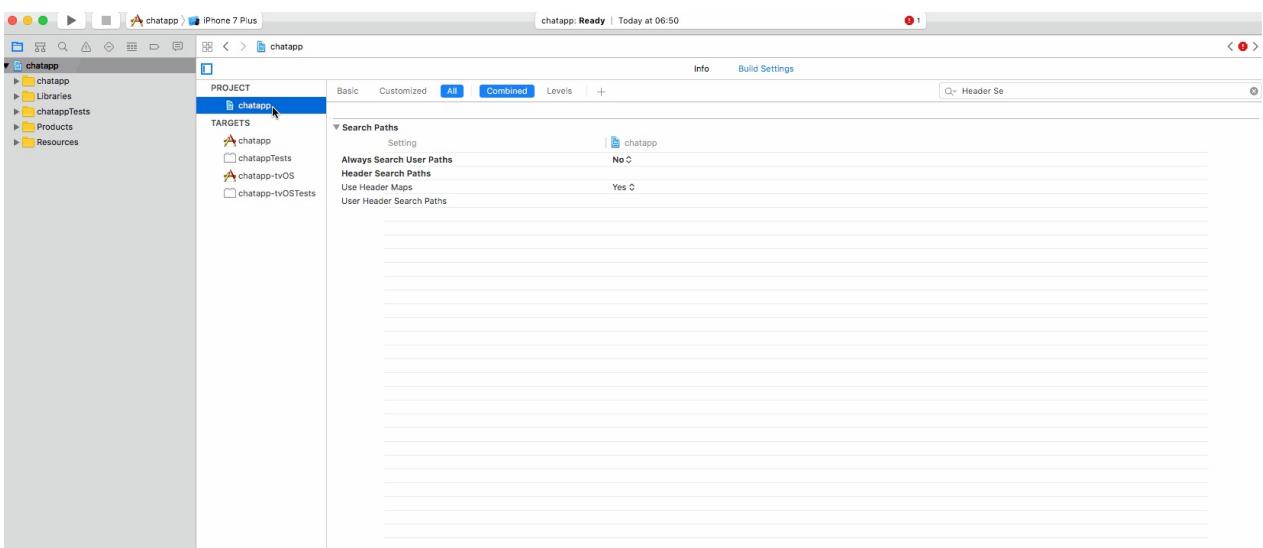


5- XCode'da navigator'de projenizi seçin ve **Build Phases** kısmına gelin. **Link Binary with Libraries** kısmında **libReactNativeNavigation.a** seçip ekleyin.



**6-** Yine Project Navigator'de bu kez **Build Settings**'i seçin. Search kısmına **Header Search Paths** yazın. ve aşağıdaki kodu kopyalayıp bu kısma ekleyin ve mutlaka **recursive** seçeneğini seçin.

```
$(SRCROOT)/../../node_modules/react-native-navigation/ios
```



**7-** Son olarak aşağıdaki kodu projenizin **AppDelegate.m** dosyasının tamamını ezerek kopyalayın.( Evet tamamını ezerek )

```
#import "AppDelegate.h"
#import <React/RCTBundleURLProvider.h>

// *****
// *** DON'T MISS: THE NEXT LINE IS IMPORTANT ***
// *****

#import "RCCManager.h"

// IMPORTANT: if you're getting an Xcode error that RCCManager.h isn't found, you've probably ran "npm install"
```

```
// with npm ver 2. You'll need to "npm install" with npm 3 (see https://github.com/wix/react-native-navigation/issues/1)

#import <React/RCTRootView.h>

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    NSURL *jsCodeLocation;
#ifdef DEBUG
//    jsCodeLocation = [NSURL URLWithString:@"http://localhost:8081/index.ios.bundle?platform=ios&dev=true"];
    jsCodeLocation = [[RCTBundleURLProvider sharedSettings] jsBundleURLForBundleRoot:@"index.ios" fallbackResource:nil];
#else
    jsCodeLocation = [[NSBundle mainBundle] URLForResource:@"main" withExtension:@"jsbundle"];
#endif

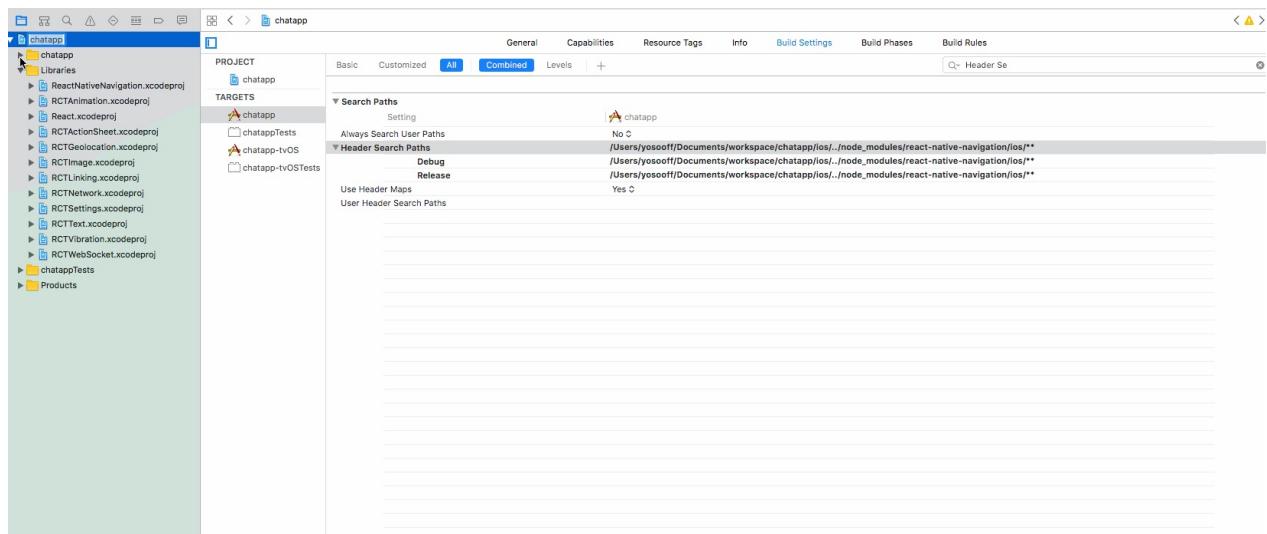
    // *****
    // *** DON'T MISS: THIS IS HOW WE BOOTSTRAP ***
    // *****

    self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen].bounds];
    self.window.backgroundColor = [UIColor whiteColor];
    [[RCCManager sharedInstance] initBridgeWithBundleURL:jsCodeLocation];

    /*
    // original RN bootstrap - remove this part
    RCTRootView *rootView = [[RCTRootView alloc] initWithBundleURL:jsCodeLocation
                                                       moduleName:@"example"
                                                       initialProperties:nil
                                                       launchOptions:launchOptions];
    self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen].bounds];
    UIViewController *rootViewController = [UIViewController new];
    rootViewController.view = rootView;
    self.window.rootViewController = rootViewController;
    [self.window makeKeyAndVisible];
    */

    return YES;
}

@end
```



(Burada yaptığımız işlem IOS için manuel linking yapmak. Resmi dökümda geçen [Linking Libraries](#) kısmının bir benzeri sadece. Küçük bir tavsiye kolayınıza gelmese bile react-native link kullanmak yerine oldukça manuel linkleme seçeneğini seçip ne olup bittiğini görün, projenizde kontrolü kaybetmeyin. Ne olup bittiğini tam anladıktan sonra react-native link kullanmaya başlayın hata alırsanız o zaman düzeltebilecek halde olursunuz )

# Android Kurulumu

Android kurulumu ve kullanımı IOS implemantasyonuna göre daha çetrefilli. Kullanım açısından, Android, material design ile tasarlandığından sahne geçişleri daha çok animasyon içeren daha komplike bir özelliğe sahip ama estetik açısından da daha güzel. Hem android kullanıcılarının aşina olduğu özellikler zor diye de atlamamak lâzım. Kullanıcı deneyimi uygulamalarınızda asla göz ardı edemeyeceğiniz bir özellik. Tabi burada hiçbir kütüphaneden, react-native'in kendisinden bile tam anlamıyla bahsetmek, örnekler hazırlamak oldukça zor. react-native-navigation'ın android'e özel kullanıcıları için lütfen [resmi dökümantasyonuna](#) göz atın.

**1-** react native versiyonu olarak 0.43 ve üstünü kullanıyor olduğunuzdan emin olun.

**2-** npm kullanıyorsanız 3 ve üstü versiyon kullandığınızdan emin olun.

```
npm install react-native-navigation@latest
```

**3-** yarn kullanıyorsanız

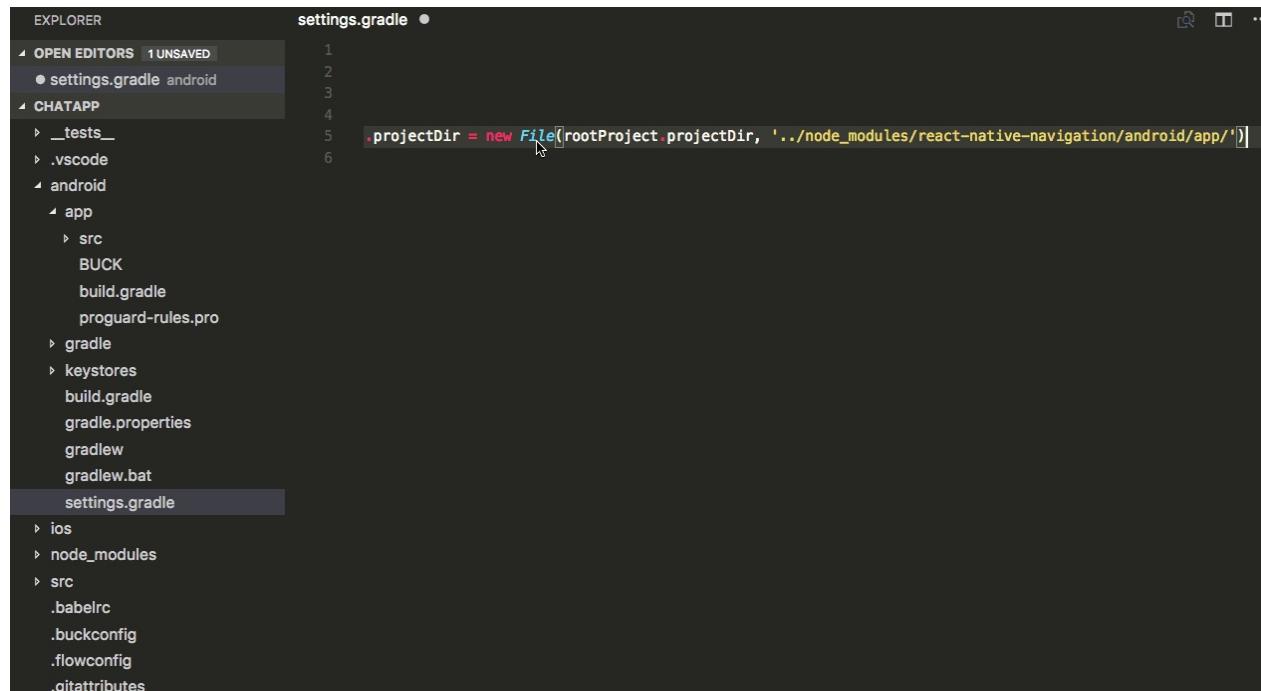
```
yarn add react-native-navigation@latest
```

**4- android/settings.gradle** içine aşağıdaki kod parçaslığını ekleyin.

```
include ':react-native-navigation'  
project(':react-native-navigation').projectDir = new File(rootProject.projectDir, '../node_modules/react-native-navigation/android/app/')
```

**5- android/app/build.gradle** içinde Android sdk versyonunu 25 olarak set edin ve project dependency kısmına react-native-navigation'ı ekleyin.

```
android {  
    compileSdkVersion 25  
    buildToolsVersion "25.0.1"  
    ...  
}  
  
dependencies {  
    compile fileTree(dir: "libs", include: ["*.jar"])  
    compile "com.android.support:appcompat-v7:23.0.1"  
    compile "com.facebook.react:react-native:+"  
    compile project(':react-native-navigation')  
}
```



7- **android/app/src/main/java/com/projenizin\_ismi/MainApplication.java** dosyanıza aşağıdaki java kodu kopyalayın, tümünü ezin.

```
package com.projenizin_ismi; // !!! bunu değiştirmeyi unutma

import android.app.Application;

//import com.facebook.react.ReactApplication;
import android.support.annotation.NonNull;
import com.reactnativavigation.NavigationApplication;
import com.facebook.react.ReactNativeHost;
import com.facebook.react.ReactPackage;
import com.facebook.react.shell.MainReactPackage;
import com.facebook.soloader.SoLoader;
import java.util.Arrays;
import java.util.List;

public class MainApplication extends NavigationApplication {

    @Override
    public boolean isDebug() {
        return BuildConfig.DEBUG;
    }

    @NonNull
    @Override
    public List<ReactPackage> createAdditionalReactPackages() {
        return Arrays.<ReactPackage>asList(
            //Yeni native kütüphanelere buraya eklenecek, örneğin new VectorIconsPackage()
        );
    }

    @Override
    public void onCreate() {
        super.onCreate();
        SoLoader.init(this, /* native exopackage */ false);
    }

}
```

**8- android/app/src/main/java/com/projenizin\_ismi/MainActivity.java** dosyanıza aşağıdaki java kodu kopyalayın, tümünü ezin.

```
package com.projenizin_ismi; // !!! bunu değiştirmeyi unutma

//import com.facebook.react.ReactActivity;
import android.widget.LinearLayout;
import android.graphics.Color;
import android.widget.TextView;
import android.widget.ImageView;
import android.view.Gravity;
import android.util.TypedValue;
import com.reactnativavigation.controllers.SplashActivity;

public class MainActivity extends SplashActivity {

    @Override
    public LinearLayout createSplashLayout() {
        LinearLayout view = new LinearLayout(this);
        TextView textView = new TextView(this);
        ImageView imageView = new ImageView(this);

        view.setBackgroundColor(Color.parseColor("#FFFFFF"));
        view.setGravity(Gravity.CENTER);

        view.addView(imageView);
        return view;
    }

}
```

# Temel Kullanım

**1-** İlk yapmamız gereken her zaman olduğu gibi bir **src** isimli source klasörü oluşturmak. Ve onun içinde de **FirstScreen.js** isimli ilk ekranımızı ekleyelim. (Bildiğimiz react-native component)

**2-** Aynı yerde bir tane de ekranlarımızı navigator'e register edeceğimiz, **screen.js** isimli bir dosya ekleyelim.

**3-** **screen.js**'den export ettiğimiz fonksiyonu kullanacağımız, projemizin ana çıkış kaynağı olan **index.ios.js** ve **index.android.js** isimli iki dosya oluşturalım.

**4 -** **FirstScreen.js** bildiğimiz bir react-native component onu **screen.js** de import edip, Navigasyon'da kullanacağımız bir ekran olduğunu aşağıdaki gibi belirtelim yani register edelim.

```
// ./src/screen.js

import { Navigation } from 'react-native-navigation';
import FirstScreen from './FirstScreen';

export function registerScreens() {
  Navigation.registerComponent('chatapp.FirstScreen', () => FirstScreen);
}
```

**5-** Oluşturduğumuz **index.ios.js** ve **index.android.js** ' de ekranlarımızı App isimli class'da set edelim.

```
// ./src/index.ios.js veya ./src/index.android.js

import { Navigation } from "react-native-navigation";

import { registerScreens } from "./screen";

registerScreens();

export default class App {
  constructor() {
    this.startApp();
  }

  startApp() {
    Navigation.startSingleScreenApp({
      screen: {
        screen: "chatapp.FirstScreen", //screen.js deki ile aynı olmak zorunda
        title: "Chat App"
      }
    });
  }
}
```

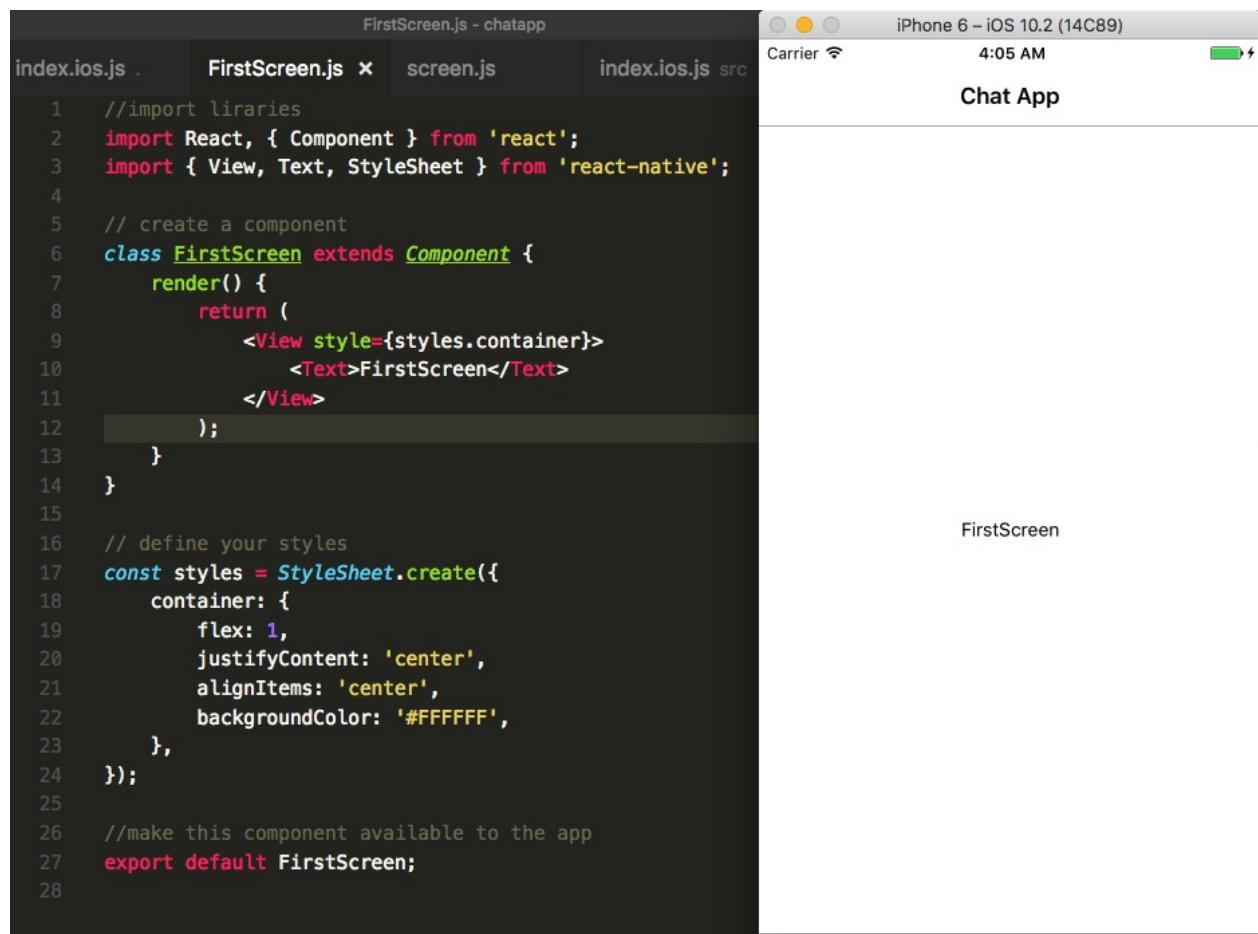
**6-** App class'ını proje root path'inde yer alan index.ios.js ve index.android.js'de çalıştıralım. Tabi bu bir class olduğu için bunu normal fonksiyon olarak çağrıramayız o yüzden bir değişkene set edip çağrırmalıyız.

```
// index.ios.js veya index.android.js

import App from './src'

const app = new App();
```

**7-** Şimdi react-native run-ios ile ( Android kurulumunu yaptıysanız react-native run-android ile ) uygulamamızı çalıştaralım. Aşağıdaki gibi bir ekran görmeliyiz.



8- Yukarıda resimdeki görüntüyü ios ve android emulatorde elde ettiyseniz şimdi **SecondScreen.js** isimli ikinci ekran ekleyelim.Bunu da screen.js ' de import edip, Navigator'a aşağıdaki gibi register edelim.

```

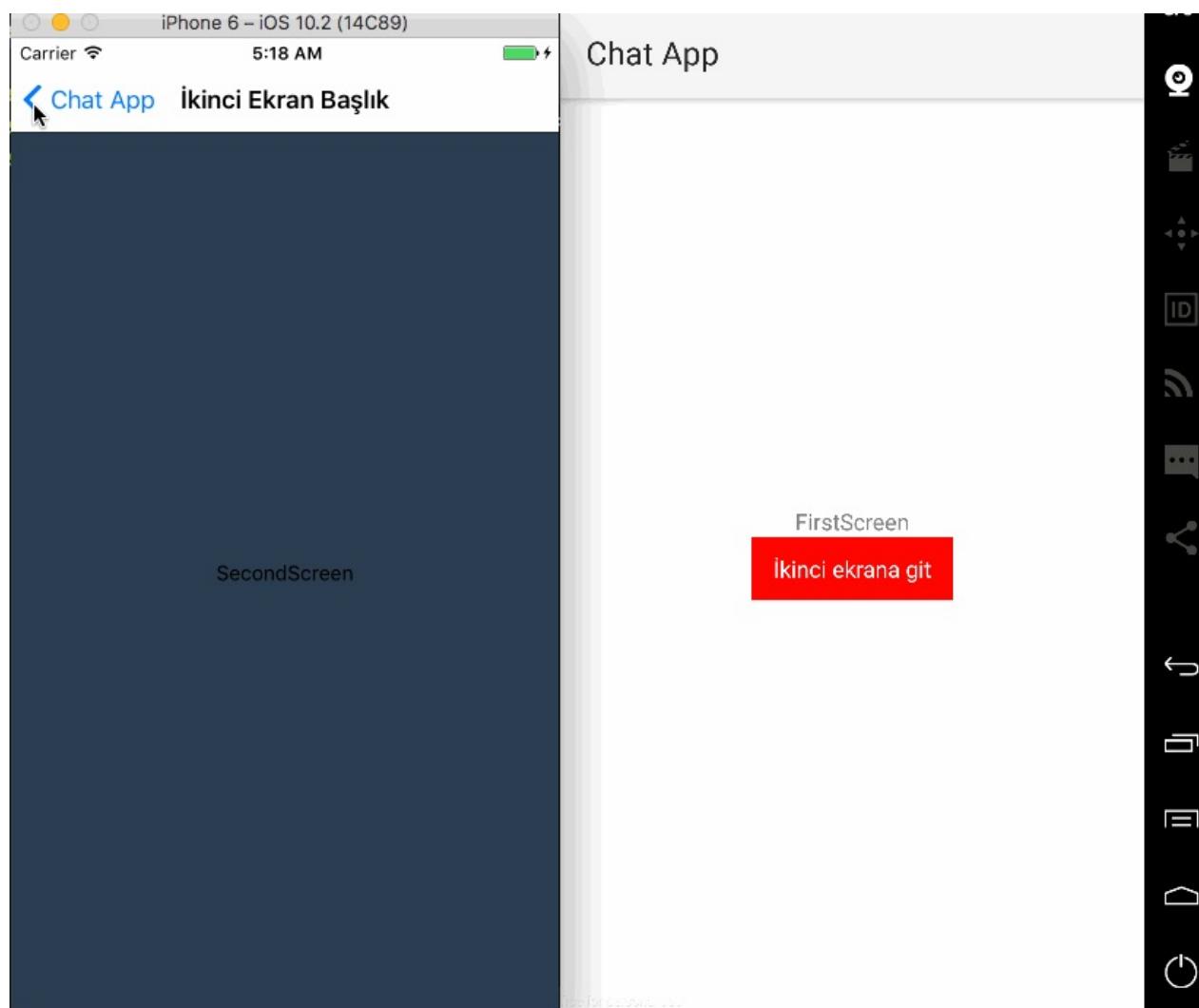
import { Navigation } from 'react-native-navigation';
import FirstScreen from './FirstScreen';
import SecondScreen from './SecondScreen';

export function registerScreens() {
  Navigation.registerComponent('chatapp.FirstScreen', () => FirstScreen);
  Navigation.registerComponent('chatapp.SecondScreen', () => SecondScreen);
}

```

9 - **FirstScreen.js** componentine bir tane button ekleyelim ve bastığımızda uygulamayı ikinci ekrana route edelim. Navigator'a register ettiğimiz her component **navigator** isimli props'a sahip oluyor. Route ederken de navigator propsunda yer alan `push` u kullanıyoruz.

```
// ./src/FirstScreen.js
...
class FirstScreen extends Component {
  handleRoute = () => {
    this.props.navigator.push({
      screen: "chatapp.SecondScreen",
      title: "İkinci Ekran Başlık"
    });
  };
  render() {
    return (
      <View style={styles.container}>
        <Text>FirstScreen</Text>
        <TouchableOpacity style={styles.button} onPress={this.handleRoute}>
          <Text style={{ color: "#FFFFFF" }}> İkinci ekrana git </Text>
        </TouchableOpacity>
      </View>
    );
  }
}
...
```





# Icon Ekleme - TabBar

React Native'de icon deyince akla gelen ilk kütüphane, [react-native-vector-icons](#). Gerçekten harikulade bir kütüphane. react-native-navigation ile nasıl kullanılıyor bir bakalım.

Aşağıdaki komut ile indirelim.

```
npm install react-native-vector-icons --save
```

Sonra react-native link ile linkleme yapalım.

```
react-native link react-native-vector-icons
```

**android/app/src/main/java/com/projenizin\_ismi/MainApplication.java** dosyasına gidelim ve VectorIconsPackage'i aşağıdaki set edelim.

```
...
import com.oblador.vectoricons.VectorIconsPackage;
...
public List<ReactPackage> createAdditionalReactPackages() {
    return Arrays.<ReactPackage>asList(
        new VectorIconsPackage()
    );
}
...
```

Bizim arkadaş listesini göreceğimiz bir FriendList.js isimli bir ekranımız ve mesajları görebileceğimiz ProfileScreen.js isimli bir componentimiz olsun. Ve aşağıdaki gibi register edelim.

```
// ./src/screen.js

import { Navigation } from 'react-native-navigation';
import FirstScreen from './FirstScreen';
import SecondScreen from './SecondScreen';
import FriendListScreen from './FriendListScreen';
import ProfileScreen from './ProfileScreen';

export function registerScreens() {
  Navigation.registerComponent('chatapp.FirstScreen', () => FirstScreen);
  Navigation.registerComponent('chatapp.SecondScreen', () => SecondScreen);
  Navigation.registerComponent('chatapp.FriendListScreen', () => FriendListScreen);
  Navigation.registerComponent('chatapp.ProfileScreen', () => ProfileScreen);
}
```

./src/index.ios.js dosyasına dönelim ve uygulamamızın başlangıç sayfasını bir tabBar'a dönüştürelim. Bunun için react-native-navigation'da `Navigation.startTabBasedApp` api'sini kullanacağımız. Ama daha önce icon belirleyelim. FriendList için, IonicIcons'daki ios-people ve profil sayfası için ios-person iconlarını kullanalım. ./src/index.ios.js 'de App class'ımıza bir method ekleyelim adı populatelcons olsun. Bu bize Promise dönsün, Iconların yüklemesi tamamlandığında uygulamamız belirlediğimiz ilk ekrandan çalışmaya başlasın. Bunları yaptığımız aşağıdaki kodu dikkatle lütfen inceleyin.

```
// ./src/index.ios.js

import { Navigation } from "react-native-navigation";
import { registerScreens } from "./screen";
import Icon from "react-native-vector-icons/Ionicons";

registerScreens();

export default class App {
  constructor() {
    this.populateIcons().then(() => {
      this.startApp();
    }).catch((error) => {
      console.error(error);
    });
  }

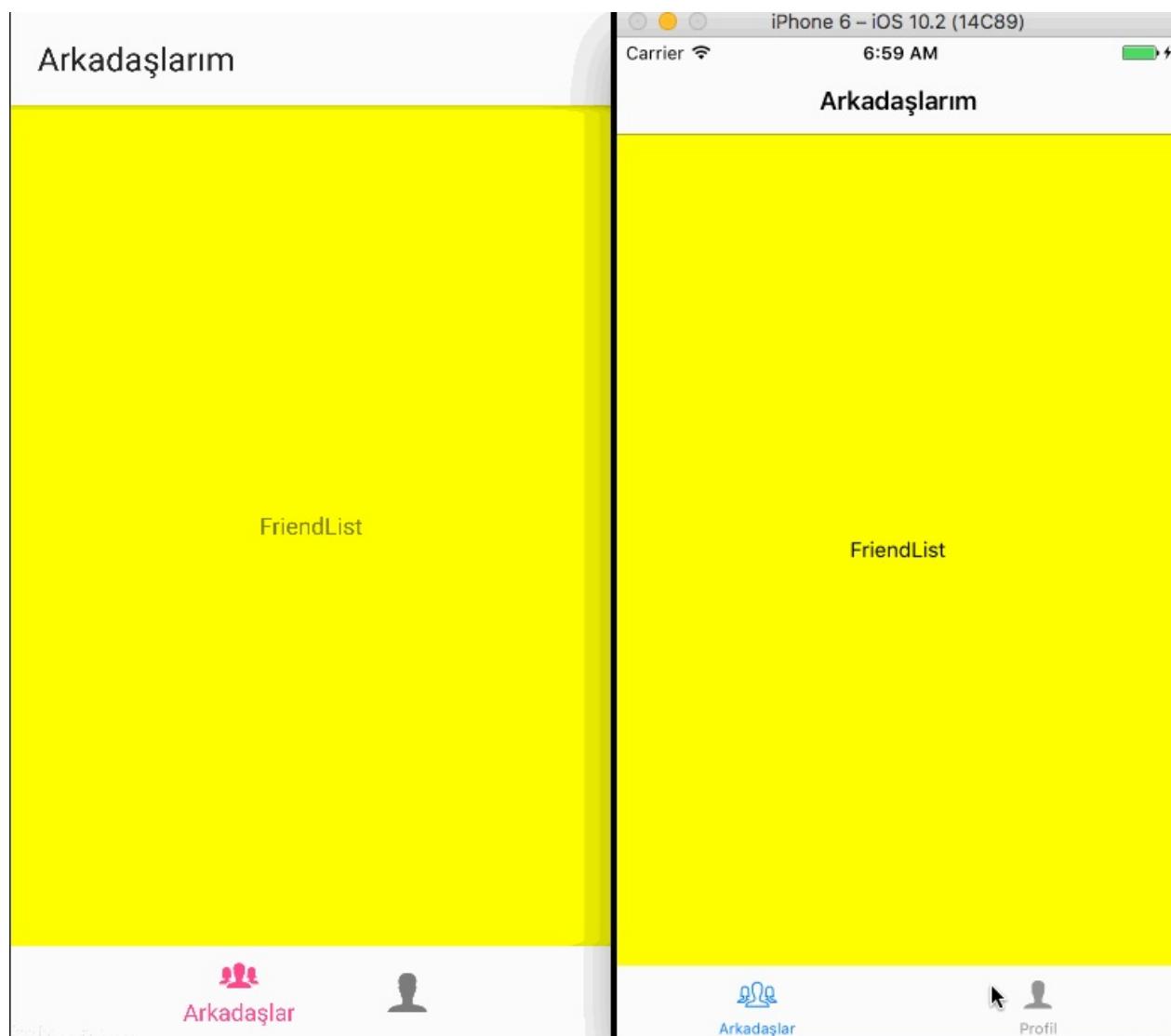
  populateIcons = function() {
    return new Promise(function(resolve, reject) {
      Promise.all([
        Icon.getImageSource("ios-people", 30),
        Icon.getImageSource("ios-people-outline", 30),
        Icon.getImageSource("ios-person", 30),
        Icon.getImageSource("ios-person-outline", 30)
      ])
      .then(values => {
        friendListIcon = values[0];
        friendListOutlineIcon = values[1];
        profileIcon = values[2];
        profileOutlineIcon = values[3];
        resolve(true);
      })
      .catch(error => {
        console.log(error);
        reject(error);
      })
      .done();
    });
  };

  startApp() {
    Navigation.startSingleScreenApp({
      screen: {
        screen: "chatapp.FirstScreen",
        title: "Chat App"
      }
    });
  }
}
```

Şimdi bir tek ekran olarak çalışan uygulamamızı, aşağıdaki gibi tabBar'a dönüştürelim.

```
// ./src/index.ios.js
...
startApp() {
  Navigation.startTabBasedApp({
    tabs: [
      {
        label: 'Arkadaşlar',
        screen: 'chatapp.FriendListScreen',
        icon: friendListIcon,
        selectedIcon: friendListOutlineIcon,
        title: 'Arkadaşlarım'
      },
      {
        label: 'Profil',
        screen: 'chatapp.ProfileScreen',
        icon: profileIcon,
        selectedIcon: profileOutlineIcon,
        title: 'Profilim'
      }
    ]
  });
}
...
```

index.ios.js'de yazdığımız kodu index.android.js'e de kopyalayalım ve uygulamayı tekrardan iki emulatorde de çalıştıralım.



Android kullanıcıları çok ekranın altındaki tabBar'a alışkın değil o yüzden bunu TopTabBar yapalım. Bunun için index.android.js startApp fonksiyonunun aşağıdaki gibi değiştirelim.

```
// ./src/index.android.js

startApp() {
  Navigation.startSingleScreenApp({
    screen: {
      screen: 'chatapp.FirstScreen',
      title: 'Chat App',
      topTabs: [
        {
          screenId: 'chatapp.FriendListScreen',
          icon: friendListIcon,
        },
        {
          screenId: 'chatapp.ProfileScreen',
          icon: profileIcon,
        }
      ]
    }});
}
```

Burada react-native-navigation yaratıcılarının hoşuna gitmeyen durum, topTabs kullanırken aslında chatapp.FirstScreen çok işe yaramaz halde fazladan yer alması. Bunu değiştirmeyi planlıyorlar ama şimdilik bun componenti biz tabBar'a style vermek için kullanabiliriz. O yüzden FirstScreen componentine aşağıdaki static style'ı tanımlayarak son hale bir bakalım.

```
// ./src/FirstScreen.js
import React, { Component } from "react";
import { View, Text, StyleSheet, TouchableOpacity, PixelRatio } from "react-native";

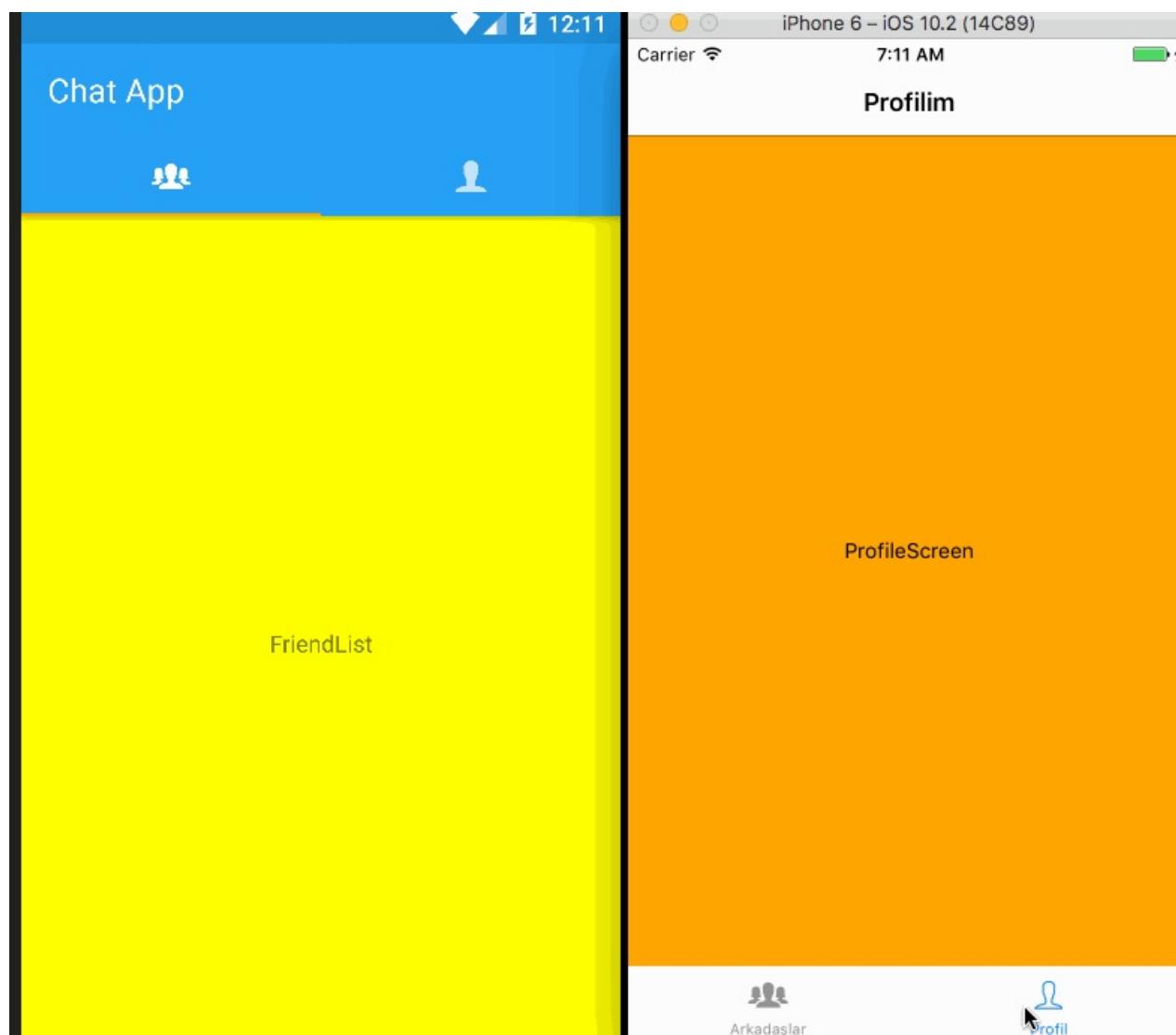
class FirstScreen extends Component {
  static navigatorStyle = {
    statusBarColor: '#1d84d6',
    navBarBackgroundColor: '#2196f3', // Top color
    navigationBarColor: '#1d84d6', // Bottom color
    navBarTextColor: '#ffffff', // Title color

    // Text
    topTabTextColor: 'rgba(255, 255, 255, 0.7)',
    selectedTopTabTextColor: '#ffffff',

    // Icons
    selectedTopTabIconColor: '#ffffff',
    topTabIconColor: 'rgba(255, 255, 255, 0.7)',

    selectedTopTabIndicatorColor: 'orange',
    selectedTopTabIndicatorHeight: PixelRatio.get() * 2,

    topTabScrollable: true,
  }
  ...
}
```



# react navigation

react-native core ekibi navigation, sayfa yönlendirme de herkesi memnun edecek bir navigation kütüphanesini yazmakta ve bunu react-native'in kendisinin içine koymakta zorlanınca, bunu açık kaynağa gönül vermiş insanların halledebileceğine inandığını söyledi. Hal böyle olunca, yüzlerce react-native için router kütüphanesi çıkmaya başladı. Kimileri native çözüm denedi, kimileri javascript ile halledebileceğini düşündü. Bunlardan başarılı olduğuna inandığımız react-native-navigation kütüphanesini bir önceki bölümde anlatmaya çalışmıştık. Şimdi javascript çözümlerin içinde çoğu insana göre en iyi, en popüler olan, github'da react-community grubunun oluşturduğu react-navigation'a bakalım.

react-navigation, sayfa yönlendirme işine yapılan native bir çözüm değil. O yüzden performansına uygulamanızı yazarken her zaman dikkat etmeniz gereklidir. Uygulamanızda mobx veya redux gibi bir state yönetim aracı kullanıyorsanız, sayfalar arası çok yüklü propslar geçirmiyorsanız çok performans sıkıntısı yaşamazsınız.

## Kurulum

Aslında karşımızda bir javascript kütüphanesi var. O yüzden tek yapmamız gereken, npm veya yarn ile uygulamamıza dependency olarak eklemek.

```
//npm kullanıyorsanız
npm install --save react-navigation

//yarn kullanıyorsanız
yarn add react-navigation
```

## Başlangıç

react-navigation'da, mutlaka kendisinin içinde olan 3 navigator'den en az birini yaratmanız gerekmektedir. (Birbiri içinde de bu navigator'leri kullanabilirsiniz, nasıl olduğunu örneklerde açıklamaya çalışacağız.)

- `StackNavigator` : Tüm sayfayı kullanmak istiyorsanız kullanmanız gereken navigator.
- `TabNavigator` : Tab'lardan oluşan bir uygulamanız varsa kullanmanız gereken navigator.
- `DrawerNavigator` : Yandan açılmalı bir sidebar ile sayfa yönlendirmesi yapmak istiyorsanız kullanmanız gereken navigator.

## StackNavigator

Tüm navigator'lerde olduğu gibi StackNavigator, parametre olarak bir obje alıyor. Bu objenin key'leri bizim sayfa route'larımız. Sayfayı yönlendirirken bu key'lerden faydalananacağız.

Aşağıdaki örneğinden Home ve Details key'leri de bir objeyi gösteriyor. Objenin içinde sabit key'lerimiz var. En önemlisi screen key'lerine, container componentlerimiz olan HomeScreen ve DetailsScreen componentlerini atadığımıza dikkat edin.

```
import { StackNavigator } from 'react-navigation';
import HomeScreen from './screens/home'
import DetailsScreen from './screens/details'

const RootNavigator = StackNavigator({
  Home: {
    screen: HomeScreen,
  },
  Details: {
    screen: DetailsScreen,
  },
});

export default RootNavigator;
```

## TabNavigator

StackNavigator örneğini, tab'ları olan bir uygulamaya dönüştürmek istersek yapacağımız şey yukarıdaki ile aynı. Sadece StackNavigator yerine TabNavigator'ı çağırıyoruz.

```
import React from 'react';
import { TabNavigator } from 'react-navigation';
import HomeScreen from './screens/home'
import DetailsScreen from './screens/details'

const RootTabs = TabNavigator({
  Home: {
    screen: HomeScreen,
  },
  Profile: {
    screen: DetailsScreen,
  },
});

export default RootTabs;
```

## DrawerNavigator

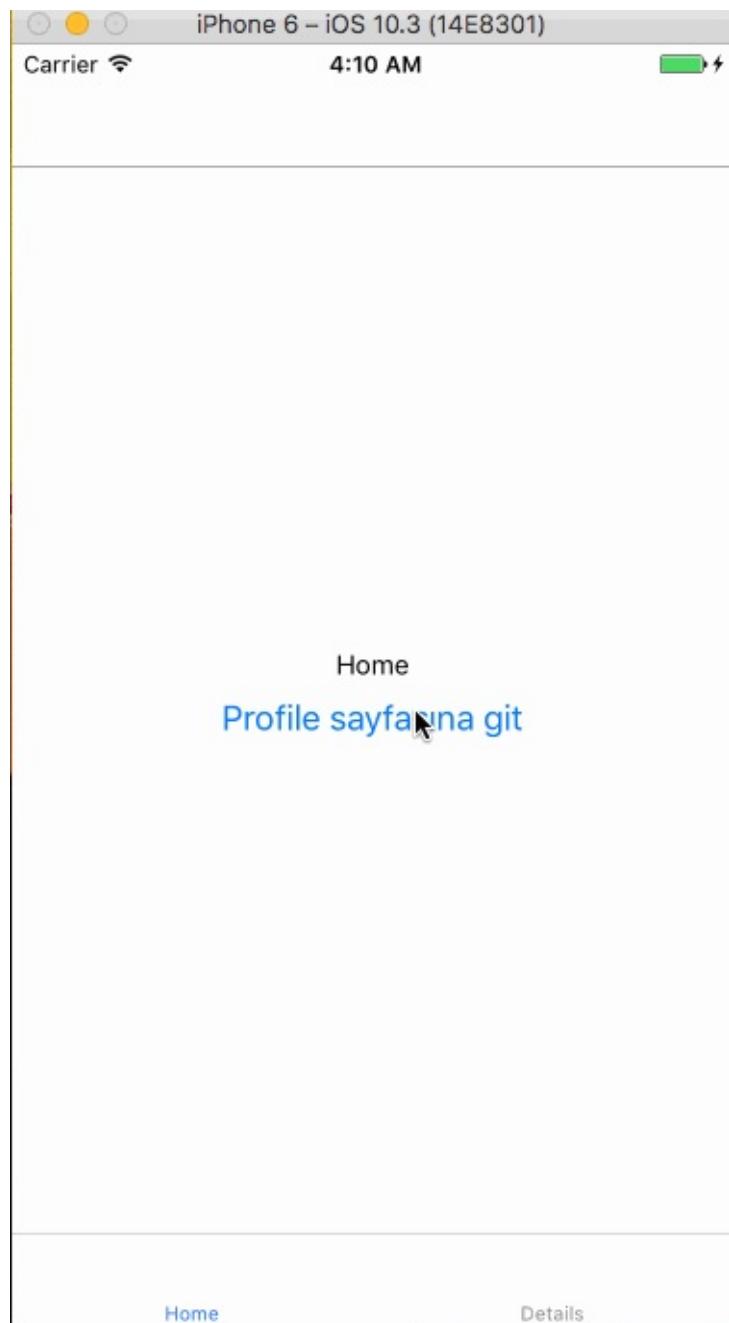
DrawerNavigator kullanırken de temelde farklı bir şey yapmıyoruz.

```
import React from 'react';
import { DrawerNavigator } from 'react-navigation';
import HomeScreen from './screens/home'
import DetailsScreen from './screens/details'

const RootDrawer = DrawerNavigator({
  Home: {
    screen: HomeScreen,
  },
  Profile: {
    screen: DetailsScreen,
  },
});

export default RootDrawer;
```

# İç İçe Navigator Kullanımı (Nesting Navigators)



Navigator'ler iç içe yazılabilir ( composable ) özelliğine sahiptir. Yukarıdaki örnekte TabNavigator ve StackNavigator ün nasıl kullanıldığıni inceleyebilirsiniz.

```
import { StackNavigator, TabNavigator } from "react-navigation";
import HomeScreen from "./src/screens/home";
import DetailsScreen from "./src/screens/details";
import ProfileScreen from "./src/screens/profile";

const MainNavigator = TabNavigator({
  Home: {
    screen: HomeScreen
  },
  Details: {
    screen: DetailsScreen
  }
});

const RootNavigator = StackNavigator({
  Main:{
    screen:MainNavigator // Yukarıdaki TabNavigator
  },
  Profile:{
    screen:ProfileScreen
  }
})

export default RootNavigator;
```

# STATE YÖNETİMİ

- [Mobx](#)
- [Redux](#)

# MOBX ( State yönetmenin en kolay yolu )

Mobx, reactive programlama felsefesiyle yazılmış rakiplerine göre öğrenme süresi çok kısa olan, basitliğiyle dikkat çeken bir state yönetim kütüphanesi.

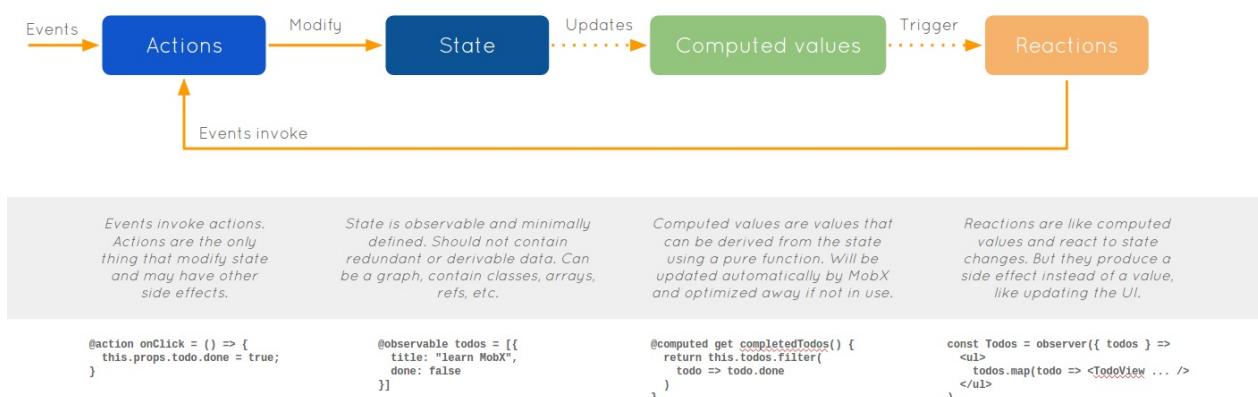
Reactive Programlama nedir ? ( [ekşi sözlükte](#) okuduğum tanım, bayağı sade ve açıklayıcı olduğu gibi aşağıya yazıyorum )

Bir değişikliğin veri akışında yayılımı üzerine kurulmuş programlama paradigmı. Örneğin normalde `a = b+c` ile `b` ile `c` nin toplamı `a` 'ya atanır ve `b` veya `c` degistiginde `a` degismez. Reactive programlamada ise `b` veya `c` degistiginde bu değişiklik `a` 'ya da yansıtılır ve toplam değişmiş olur.

Mobx'in arkasında yatan mantık da aynı. Uygulamanın state'inde herhangi bir şey değişmişse, değişime göre otomatik olarak yeni state üretilir. React içinde de state değiştiğinde önyüz tekrar render edilir. Hal böyle olunca da, Mobx ve React harika bir ikili oluveriyorlar.

Şimdi mobx'de ki temel kavramları basit bir TODO List örneğiyle açıklayalım. Belki TODO List örneği görmekten bir kısmınıza kusma geliyor olabilir ama söz bundan sonraki örneğimizi bir react-native, mobx ve react-native-navigation kullanarak bir oyun geliştirecek yapacağız.

Flux ve redux da olduğu gibi, MobX'de de action'lar var olan state'i değiştirmek için varlar. Ama redux'da olduğu gibi state değişiklerini reducer üzerinden değil, **direkt action fonksiyonunun içinde yapıyoruz.**



## React Native - Mobx Entegrasyonu

- 1) İlk yapmamız gereken mobx ve mobx-react paketlerini projemizin dependency paketlerine eklemek

```
yarn add mobx mobx-react
```

**2)** MobX, javascript'e ES7 ile gelen decoratorler ile genellikle yazılır. ( decorator kullanmadan da yazabilirsiniz ama okunabilirlik açısından tavsiye etmiyorum. ). Bu decoratorları etkin kılmak için babel-plugin-transform-decorators-legacy paketini projemizin devDependencies'e ekliyoruz

```
yarn add --dev babel-plugin-transform-decorators-legacy
```

Paketimizi ekledikten sonra, **.babelrc** dosyasının içini aşağıdaki gibi değiştiriyoruz

```
{
  "presets": ['react-native'],
  "plugins": ['transform-decorators-legacy']
}
```

**3)** Eğer VsCode kullanıyorsanız, editörünüzün decoratorlere kızmaması için projenizde **tsconfig.json** isimli bir dosya oluşturup, aşağıdaki satırları kopyalaip kaydedin.

```
{
  "compilerOptions": {
    "experimentalDecorators": true,
    "allowJs": true
  }
}
```

**4)** Şimdi todoStore diye isimlendireceğimiz sınıfımızı yaratalım. Başlangıç state'leri değişken olarak tanımlayıp, başlarına observable decorator koyduk. Böylece bu state'erde değişiklik olduğu zaman, observable componentlerimiz bunları algılayıp tekrar render edilecekler.

```
import mobx, { observable } from "mobx";

class Store {
  @observable todos=[];
  @observable selectedStatus="all";
}

const todoStore = new Store()
export default todoStore;
```

**5)** State'leri değiştirecek olan action'larımıza, **@action** decorator'u yardımcı ile tanımlayalım. Observable değişkenlere redux'da olduğu gibi bir reducer aracılığıyla değil, zaten class içinde tanımlı olduklarıdan method içinde müdahale edebiliyoruz.

```
import mobx, { observable, action } from "mobx";

class Store {
    @observable todos=[];
    @observable selectedStatus="all";

    @action('adding todo item')
    addTodo(todo){
        this.todos.push({
            id: this.todos[this.todos.length - 1].id +1,
            status: false,
            text: todo
        })
    }

    @action('Selected status changed')
    statusChange(status){
        this.selectedStatus = status;
    }
}

const todoStore = new Store()
export default todoStore;
```

6) MobX'de bir diğer temel kavram `@computed` decorator'ü. Computed methodları, birden fazla observable değerinin değişikliğinden etkilenen yeni bir değer türeteceğimiz zaman kullanıyoruz. Aşağıdaki örneğe bakalım. Status'u `all` olan, ya da `done` olan todo list'de olan elemanları `filteredTodos` değişkeninde tutmak ya da önyüz'de `todos` listesini filtrelemek yerine, çoğu yazılım dilinde hali hazırda olan getter, setter methodları yapıyoruz.

```

import mobx, { observable, action, computed } from "mobx";

class Store {
    @observable todos=[];
    @observable selectedStatus="all";

    @action('adding todo item')
    addTodo(todo){
        this.todos.push({
            id: this.todos[this.todos.length - 1].id +1,
            status: false,
            text: todo
        })
    }

    @action('Selected status changed')
    statusChange(status){
        this.selectedStatus = status;
    }

    @computed get filterTodos() {
        if(this.selectedStatus === 'all'){
            return this.todos;
        } else if( this.selectedStatus === 'done' ){
            return this.todos.filter(
                todo => todo.status === true);
        } else if( this.selectedStatus === 'todo' ){
            return this.todos.filter(
                todo => todo.status === false);
        }
    }

    const todoStore = new Store()
    export default todoStore;
}

```

7) Şimdi react-native ile entegrasyonuna göz atalım. **@observer** decorator'ünü componentin başına ekliyoruz ve Store'u import edip, store içinde ne varsa method, state, her birine erişebiliyoruz.

```
import { observer } from "mobx-react/native";
import TodoStore from './store'

@observer
export default class hellomobx extends Component {
  render() {
    console.log(this.props);
    return (
      <View style={styles.container}>
        <Text>{TodoStore.selectedStatus}</Text>
      </View>
    );
  }
}
```

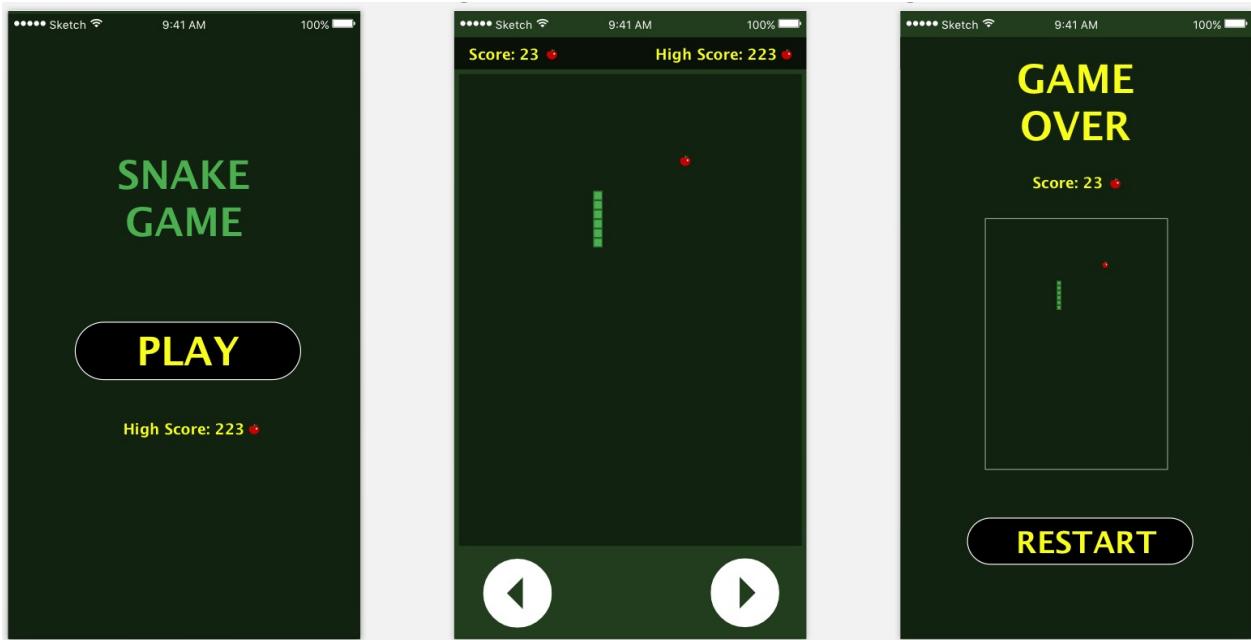


# REACT NATIVE İLE YILAN OYUNU ( PART 1 )

İlk bölümün source kodlarına şuradaki branch'den erişebilirsiniz:

<https://github.com/ysfzrn/crazysnake/tree/firstpart>

React Native, MobX ve react-native-navigation kullanarak, basit bir yılan oyunu yapalım.



Oyunumuz 3 ekrandan oluşacak.

- İlk ekranda kullanıcının telefonunda yapılan en yüksek puanı gösterecek
- Play butonuna basılıncı direkt oyun başlayacak.
- Oyun ekranında 2 tane buton olacak. ( Eski Nokia telefonlardan hatırlarsanız, sadece 3 ve 7 tuşlarıyla, yılanımızı 4 farklı yöne doğru gitmesini yönetebiliyorduk.)
- Yılanımız her bir elmasını ( evet o kırmızı şey çok belli olmasa da bir elma ) yediğinde hızı artarak devam edecek.
- Yılan kendi kuyruğuna çarpınca da oyun bitecek, yeni bir yüksek skor yapılmışsa, yeni skor, telefona kaydedilecek.

Hadi başlayalım :)

1) Projemizi react-native-cli ile yaratalım.

```
react-native init crazysnake
```

2) Projemizin klasörüne girip (cd crazysnake), kullanacağımız dependency paketlerini indirelim.

```
yarn add lodash mobx mobx-react react-native-navigation
```

```
yarn add -d dev babel-plugin-transform-decorators-legacy
```

**3) babelrc dosyamızı düzenleyip, decorator'lerin çalışmasını sağlayalım.**

```
{
  'presets': ['react-native'],
  'plugins': ['transform-decorators-legacy']
}
```

**4)** Mobx için yaptıklarımızdan sonra react-native-navigation paketine geçelim. Bu paket wix ekibinin yazdığı native bir kütüphane olduğundan entegrasyonu android ve ios için ayrı ayrı yapmak gerekiyor. Bu paketin kurulumunu daha önce yazdığımız [şu makalede](#) bulabilirsiniz. Buradaki adımları eksiksiz yaptığına emin olun.

**5)** Geliştirme ortamı kurulumumuzu bitirdik. Şimdi kod yazmaya başlayalım. Bir src isimli source klasör oluşturalım. Ve bunun için de root.js isimli bir dosya oluşturalım. Proje path'inde yer alan index.android.js ve index.ios.js içinde de bu root js'i çağırıralım.

```
EXPLORER
OPEN EDITORS
CRAZYSNAKE
__tests__
.vscode
android
ios
node_modules
src
root.js
.babelrc
.buckconfig
.flowconfig
.gitattributes
.gitignore
.watchmanconfig
app.json
index.android.js
index.ios.js

root.js
1 import App from './src/root';
2
3 const app = new App();
4
5 export default app;
```

**6)** react-native-navigation paketi redux ile kullanılmak üzere tasarlanmıştır. Ama korkmayın, mobx ile de kullanmak için tek yapmamız gereken bir tane fazladan Provider.js isimli dosya yaratıp aşağıdaki kodu değiştirmeden kopyalamanız yeterli.

```
import { Provider } from "mobx-react/native";

const SPECIAL_REACT_KEYS = { children: true, key: true, ref: true };

export default class MobxRnnProvider extends Provider {
  props: {
    store: Object
  };

  context: {
    mobxStores: Object
  };

  getChildContext() {
    const stores = {};

    // inherit stores
    const baseStores = this.context.mobxStores;
    if (baseStores) {
      for (let key in baseStores) {
        stores[key] = baseStores[key];
      }
    }

    // add own stores
    for (let key in this.props.store) {
      if (!SPECIAL_REACT_KEYS[key]) {
        stores[key] = this.props.store[key];
      }
    }
  }

  return {
    mobxStores: stores
  };
}
}
```

**7)** Şimdi store'ları oluşturmaya başlayalım. 2 tane store olacak. 1 tanesi navigation mobx üzerinden de yönetebileceğimiz, navigationStore. Diğereri, oyunun state'lerini yöneteceğimiz gameStore olacak. stores, klasörünün altında dabu iki store'u export edeceğiz. ( Bu arada bu projede navigationStore kullanmak zorunda değildik. Sadece mobx ile nasıl kullanıldığına örnek olması bakımından yazma gereği duydum. )

```

index.js      x  navigationStore.js      ...  gameStore.js  ...
1 //import App from './FirstScreenStore';
2 import nav from './navigationStore';
3 import gameStore from './gameStore';
4
5 export default {
6   nav,
7   gameStore,
8 };

```

```

navigationStore.js
1 "use strict";
2 import mobx, { observable, action } from "mobx";
3
4 class NavStore {
5   @observable route = undefined;
6
7   @action("Route is changing")
8   handleChangeRoute(val) {
9     this.route = val;
10 }
11
12 appInitialized() {
13   this.route = 'root';
14 }
15
16 }
17
18 export default new NavStore()
19
20

```

```

gameStore.js
1 "use strict";
2 import mobx, { observable, action } from "mobx";
3
4 class GameStore {
5
6 }
7
8 export default new GameStore()
9
10

```

**8)** Ana ekranlarımız için **screens** klasörü, componentler için **components** klasörü, resim ve ikonlar için **assets** klasörü, uygulamanın genel kullanılabilecek fonksiyonları için bir tane de **util** fonksiyonu oluşturalım. Ve screens klasörüne home.js isimli ilk ekranımızı koyalım.

**9)** Şimdi ekranlarımızı navigation paketine register edelim. Store ve Provider isimli dosyalarımızı da ekranlara parametre olarak geçirelim. Bunun için source klasöründe **registerScreens.js** isimli bir dosya yaratıp aşağıdaki gibi içini düzenliyoruz.

EXPLORER

- OPEN EDITORS
- CRAZYSNAKE
  - \_\_tests\_\_
  - .vscode
  - android
  - ios
  - node\_modules
  - src
    - assets
    - components
    - screens
      - home.js
    - stores
    - util
    - Provider.js
  - registerScreens.js

registerScreens.js x

```

1 import { Navigation } from "react-native-navigation";
2 import HomeScreen from './screens/home';
3 import Store from './stores'
4 import Provider from './Provider'
5
6 export function registerScreens() {
7   Navigation.registerComponent("crazySnake.HomeScreen", () => HomeScreen, Store, Provider);
8 }
9

```

**10)** Şimdi başta yarattığımız root.js isimli dosyamıza geçelim. Ve aşağıdaki gibi düzenleyelim. Burada root.js bir react componenti değil bir javascript class'ı. Ve bu class'ın içinde başta çalışacak constructor methodunda, mobx'in reaction fonksiyonunu kullanıyoruz. reaction, computed methodlara benzese de, observable state değiştiğinde yeni bir değer üretmek yerine, side effect oluşturmamızı sağlar. Aşağıdaki kodda yapılan işlem, Store.nav.route değeri değiştiğinde, startApp methodunu çalıştır.

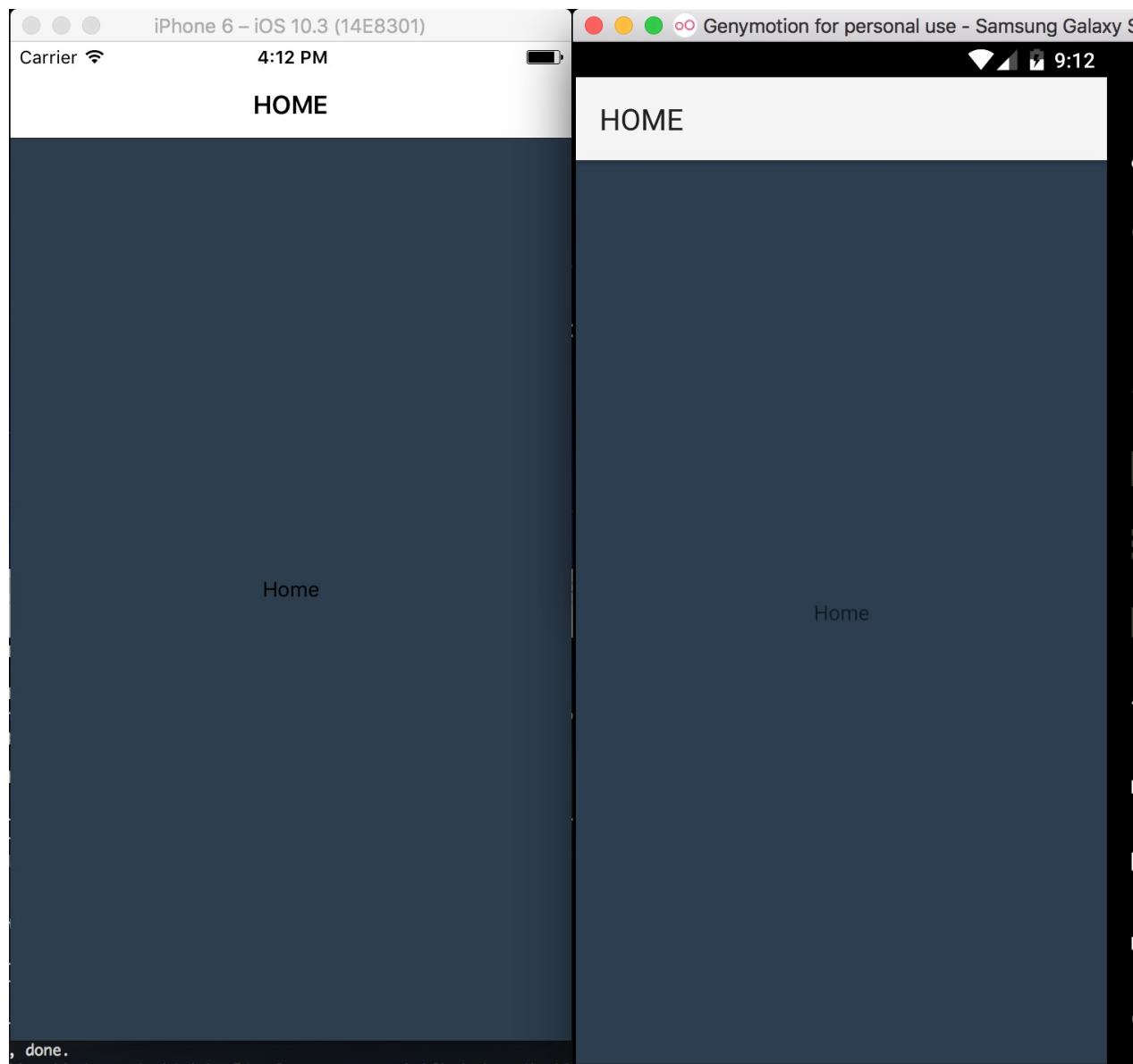
The screenshot shows a code editor interface with a dark theme. On the left, there's an 'EXPLORER' sidebar listing project files and folders. The 'home.js' file is open in the main editor area. The code in 'home.js' is as follows:

```

1 import { Navigation } from "react-native-navigation";
2 import { reaction } from "mobx";
3 import { registerScreens } from "./registerScreens";
4 import { observer } from "mobx-react/native";
5 import Store from "./stores";
6
7 registerScreens();
8
9 export default class App {
10   constructor() {
11     reaction(() => Store.nav.route, () => this.startApp(Store.nav.route));
12     Store.nav.appInitialized();
13   }
14
15   startApp(root) {
16     switch (root) {
17       case "root":
18         Navigation.startSingleScreenApp({
19           screen: {
20             screen: "crazySnake.HomeScreen",
21             title: "HOME"
22           }
23         });
24       return;
25     }
26   }
27 }

```

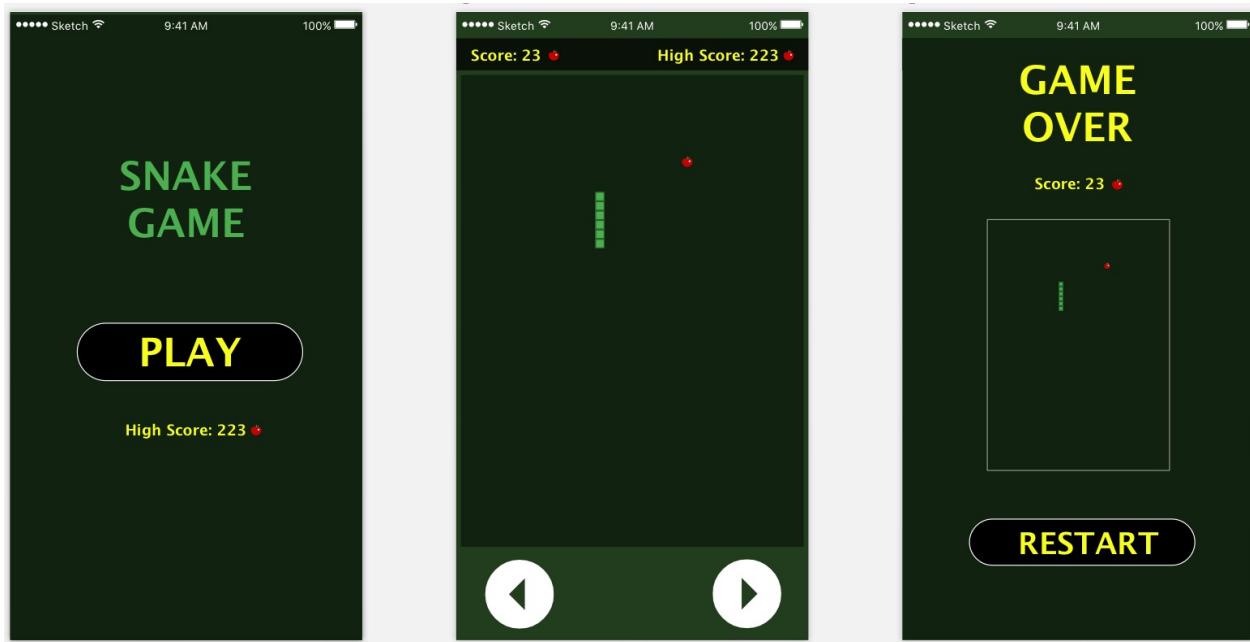
Son olarak react-native run-ios ve react-native run-android komutlarıyla aşağıdaki gibi sonuç alıdysanız, oyunumuzun ikinci bölümüne geçebiliriz.



# REACT NATIVE İLE YILAN OYUNU ( PART 2 )

İkinci bölümün source kodlarına şuradaki branch'den erişebilirsiniz:

<https://github.com/ysfzrn/crazysnake/tree/secondpart>



- 1) Home ekranımızda, gameStore'dan gelecek en yüksek skor bilgisi var. Bu bilgi telefon hafızasında tutulacak. Bunu sağlamak için **AsyncStorage**'ı kullanabiliriz.

EXPLORER

- OPEN EDITORS
- CRAZYSNAKE
- \_\_tests\_\_
- .vscode
- android
- ios
- node\_modules
- src
  - assets
  - components
  - screens
    - home.js
  - stores
    - gameStore.js
    - index.js
    - navigationStore.js
- util

gameStore.js

```

1 "use strict";
2 import mobx, { observable, action } from "mobx";
3 import { AsyncStorage } from "react-native";
4
5 class GameStore {
6   @observable highScore = 0;
7
8   @action("getting high score")
9   getHighScore() {
10     AsyncStorage.getItem("snakeHighScore").then(_highScore => {
11       this.highScore = _highScore;
12       if (_highScore === null) {
13         AsyncStorage.setItem("snakeHighScore", JSON.stringify(0));
14       }
15     });
16   }
17 }
18
19 export default new GameStore();
20
21 
```

AsyncStorage'da iki temel işlem var. `setItem` ve `getItem`. `setItem`'dan `JSON.stringify` kullanmanızın sebebi, AsyncStorage'da yalnızca string ile çalışabiliyoruz. İki temel method

da verilen key'e göre soru çekiyor. Burada bizim key'imiz "snakeHighScore". Buradaki algoritamanın mantığına gelince, telefon hafızasında, snakeHighScore key'ine sahip bir veri var mı. Varsa onu dön. Yoksa sıfır(0) olarak yeni bir tane oluştur. Çünkü normal olarak kullanıcı oyunu ilk açtığında herhangi bir skora sahip olmayacağından emin oluyor. Son olarak bize dönen değeri de, `this.highScore = _highScore` ifadesi ile observable state'imize set ediyoruz.

Şimdi ekranımızın tasarımına geçelim. Ekranlarda ortak ilişkili tutacağım stilleri, bir style dosyasında tutmayı tercih ediyorum. İşimi daha kolaylaştırıyor. Ve daha çok tema opsyonu nasıl olmuş diye deneme şansım oluyor. Burada kullanacağımız stil dosyası aşağıdaki olacak.

Burada önemli olarak yılanımızın her bir parçası yani segment'i 10px değerinin de olacak. Ve yılanın hareket edeceği board'da herhangi bir uyumsuzluk yaşamaması için, board'ın genişliği, telefonun ekran genişliğine göre şekillenmesi ve width değerinin 10'nun katlarından bir değere sahip olması gerekiyor. Bunun için sharedStyle.js içinde definiteWidth adlı bir method yarattım.

```
//src/util/SharedStyle.js
import { Dimensions, Platform } from "react-native";

const { width, height } = Dimensions.get("window");

function definiteWidth(width){
    const remainder = width % 10;
    if( remainder !== 0 ){
        return width - 5;
    }else{
        return width;
    }
}

const SharedStyle ={
    color:{
        primary: '#122210',
        primaryBlack: '#223D1D',
        secondary: '#F9FF1C',
        snake: '#4CAF50',
        scoreColor: 'yellow',
        buttonBackground: '#000000'
    },
    board:{
        height: Platform.OS === 'ios' ? 500 : 400,
        width: definiteWidth(width),
    },
    segment:{
        width:10,
        height:10,
        backgroundColor: '#4CAF50',
        borderWidth:1,
        borderColor: '#285A2A',
    },
    food: {
        width: 10,
        height: 10,
        backgroundColor: 'red',
        borderWidth:1,
        borderColor: 'black',
    },
    scoreBoard: {
        height: 34,
    }
}

export default SharedStyle
```

Home ekranımız da aşağıdaki kodda yazıldığı gibi olacak. Bu yazı genel olarak MobX üzerine yazıldığı için kullanılan componentlerin ve ekranların tasarımı üzerinde çok durmayacağım. Ama şunu söylemek zorundayım. Buradaki **inject** ve **observer**

decoratorlerine dikkat edin.

**@observer** decoratorü ile bu ekranımızın observable değerlerine göre render edilmesini sağlıyoruz. ( İsimlendirme gayet basit, observable, takip edilen değerler, observer, bu değerleri takip eden demek. )

**@inject** decoratorü ise MobX'de store elemanlarını componentlerde kullanabilmemizi sağlayan bir diğer yöntem. İlgili store'u burada import edip de kullanabilirdik. Ama ben **@inject** decoratorünü kullanarak, store elemanlarına direkt erişmek yerine, componentin props'larından erişmeyi tercih ediyorum. Hangi yöntem daha doğru ben de bilmiyorum sadece belki gerektiği zaman lifecycle methodlarını ( `componentWillReceiveProps` gibi ) kullanmaya ihtiyacım olur diye **@inject** ile props'dan almak daha mantıklı geliyor.

Artık props üzerinde store'lara erişebiliyorum. `const { gameStore } = this.props ;` ile `gameStore.highScore` 'u `ScoreText` componentine gönderebilirim.

```
//import liraries
import React, { Component } from 'react';
import { View, Text, StyleSheet, StatusBar } from 'react-native';
import { inject, observer } from "mobx-react/native";
import SharedStyle from "../utils/sharedStyle";
import { Button, ScoreText} from "../components";

// create a component
@inject("nav", "gameStore")
@observer
class Home extends Component {
  static navigatorStyle = {
    navBarHidden: true // default olarak gelen navigationBar'in görünmemesi için
  };

  componentDidMount() {
    const { gameStore } = this.props;
    gameStore.getHighScore(); //highScore değerini al
  }

  handlePlay=()=>{
    const { nav } = this.props;
    nav.handleChangeRoute("gameScreen"); //gameScreen ekranına git
  }

  render() {
    const { gameStore } = this.props;
    return (
      <View style={styles.container}>
        <StatusBar barStyle="light-content"/>
        <View style={{flex:1, justifyContent:'center'}} >
          <Logo />
        </View>
        <View style={{flex:1, flexDirection:'column', }}>
```

```

        <Button text="PLAY" onPress={ this.handlePlay } />
        <ScoreText label="High Score" score={gameStore.highScore} style={{justifyContent:'center', marginTop:42}} />
    </View>
</View>
);
}
}

const Logo = (props) => {
return(
<View style={{alignItems:'center'}}>
<Text style={styles.logoText}>CRAZY SNAKE</Text>
<Text style={styles.logoText}>GAME</Text>
</View>
)
}

// define your styles
const styles = StyleSheet.create({
container: {
flex: 1,
justifyContent: 'center',
alignItems: 'center',
backgroundColor: SharedStyle.color.primary,
},
logoText: {
fontSize: 52,
color: SharedStyle.color.snake,
}
});
}

//make this component available to the app
export default Home;

```

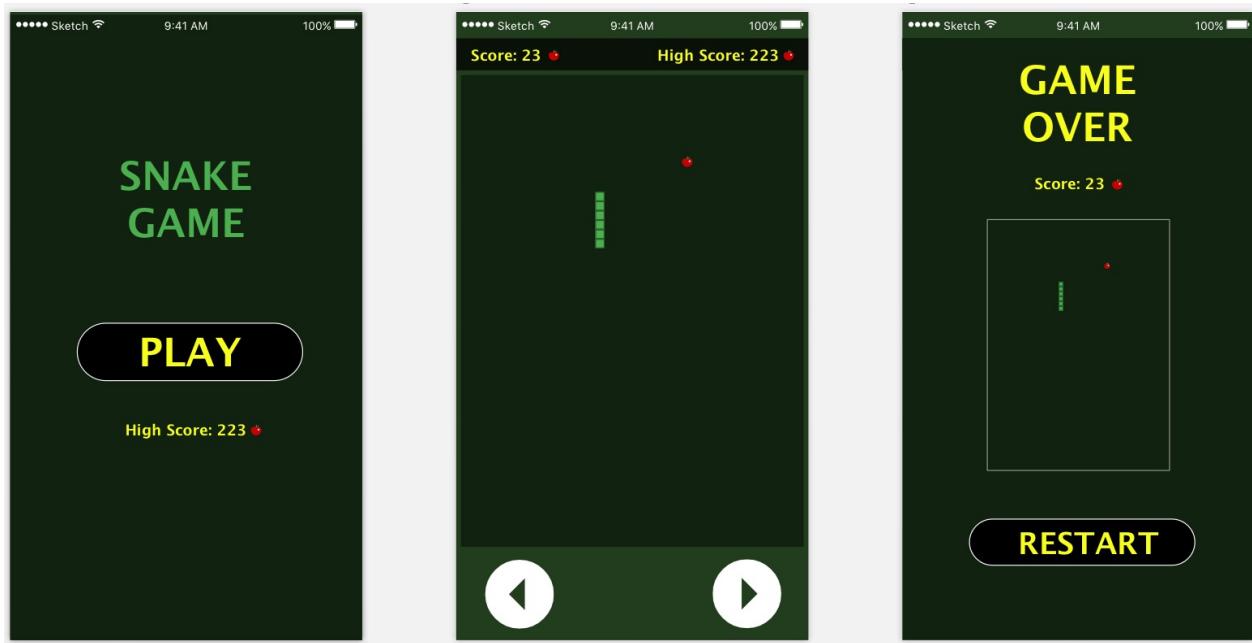
Gelelim PLAY butonuna bastığımızda MobX üzerinden ekranı değiştirmeye. Bu işi yukarıdaki component de `handlePlay` methodunda yapıyoruz. `navigationStore`'da yazdığımız, `handleChangeRoute` methodunu çağırıyoruz. Sadece yaptığı iş `navigationStore`'da route observable değerini değiştirip, değerini "gameScreen" olarak güncellemek. O da ilk bölümde anlattığımız, `root.js` içindeki `reaction`'ı tetikleyip, `this.startApp` methodunu çağıracak.

İlk ekran için söylenecekler bu kadar, şimdi oyunun ana kısmı olan `gameScreen` ekranımıza geçelim. Bu bölüm oldukça uzun olacağından bunu 3. bölümde anlatalım.

# REACT NATIVE İLE YILAN OYUNU ( PART 3 )

Üçüncü bölümün source kodlarına şuradaki branch'den erişebilirsiniz:

<https://github.com/ysfzrn/crazysnake/tree/thirdpart>



İlk olarak gameScreen ekranını registerScreen.js içinde register edelim.

```
import { Navigation } from "react-native-navigation";
import HomeScreen from './screens/home';
import GameScreen from './screens/gameScreen';
import Store from './stores';
import Provider from './Provider'

export function registerScreens() {
  Navigation.registerComponent("crazySnake.HomeScreen", () => HomeScreen, Store, Provider);
  Navigation.registerComponent("crazySnake.GameScreen", () => GameScreen, Store, Provider);
}
```

Daha sonra root.js içinde gameScreen için durum yazalım.

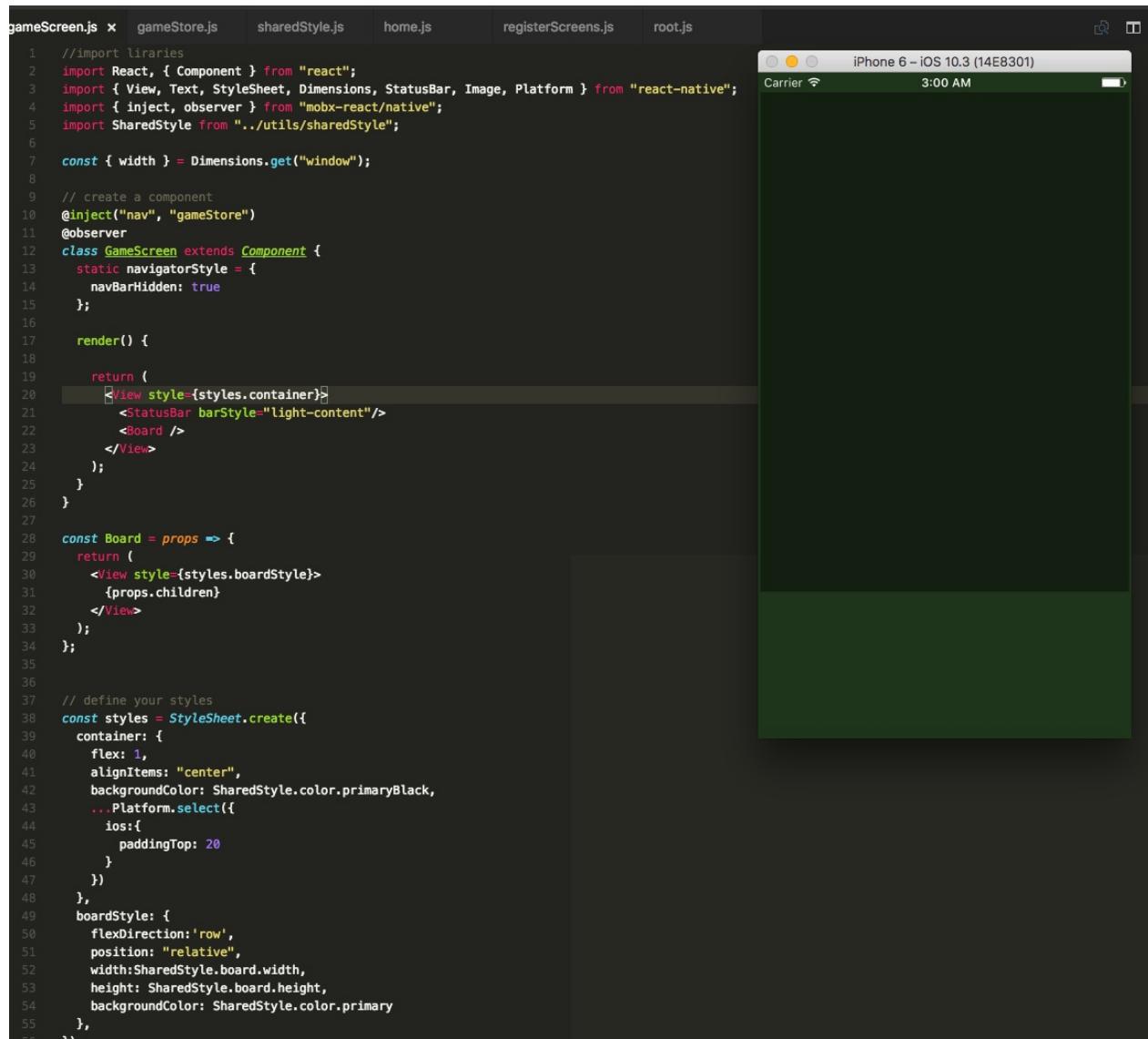
```
import { Navigation } from "react-native-navigation";
import { reaction } from "mobx";
import { registerScreens } from "./registerScreens";
import { observer } from "mobx-react/native";
import Store from "./stores";

registerScreens();

export default class App {
  constructor() {
    reaction(() => Store.nav.route, () => this.startApp(Store.nav.route));
    Store.nav.appInitialized();
  }

  startApp(root) {
    switch (root) {
      case "root":
        Navigation.startSingleScreenApp({
          screen: {
            screen: "crazySnake.HomeScreen",
            title: "HOME"
          }
        });
        return;
      case "gameScreen":
        Navigation.startSingleScreenApp({
          screen: {
            screen: "snakeGame.GameScreen",
            title: "GameScreen"
          }
        });
        return;
    }
  }
}
```

Aşağıdaki gibi board'u oluşturalım.



The screenshot shows a code editor with gameScreen.js open. The code is a React Native component named GameScreen. It imports React, Component, Dimensions, StatusBar, Image, Platform, inject, observer, and SharedStyle. The component has a static navigatorStyle with navBarHidden set to true. The render method returns a View with styles from styles.container, containing a StatusBar with barStyle="Light-content", a Board component, and another View. The Board component takes props and returns a View with styles from styles.boardStyle, containing the prop children. The styles are defined using StyleSheet.create, with container having flex: 1, align-items: "center", and backgroundColor: SharedStyle.color.primaryBlack. The boardStyle has flexDirection: 'row', position: "relative", width: SharedStyle.board.width, height: SharedStyle.board.height, and backgroundColor: SharedStyle.color.primary. The code ends with a closing brace for the styles object.

```
1 //import liraries
2 import React, { Component } from "react";
3 import { View, Text, StyleSheet, Dimensions, StatusBar, Image, Platform } from "react-native";
4 import { inject, observer } from "mobx-react/native";
5 import SharedStyle from "../utils/sharedStyle";
6
7 const { width } = Dimensions.get("window");
8
9 // create a component
10 @inject("nav", "gameStore")
11 @observer
12 class GameScreen extends Component {
13   static navigatorStyle = {
14     navBarHidden: true
15   };
16
17   render() {
18     return (
19       <View style={styles.container}>
20         <StatusBar barStyle="Light-content"/>
21         <Board />
22       </View>
23     );
24   }
25 }
26
27 const Board = props => {
28   return (
29     <View style={styles.boardStyle}>
30       {props.children}
31     </View>
32   );
33 };
34
35
36 // define your styles
37 const styles = StyleSheet.create({
38   container: {
39     flex: 1,
40     alignItems: "center",
41     backgroundColor: SharedStyle.color.primaryBlack,
42     ...Platform.select({
43       ios: {
44         paddingTop: 20
45       }
46     })
47   },
48   boardStyle: {
49     flexDirection: 'row',
50     position: "relative",
51     width: SharedStyle.board.width,
52     height: SharedStyle.board.height,
53     backgroundColor: SharedStyle.color.primary
54   },
55 });
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
878
879
880
881
882
883
884
885
886
887
887
888
889
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
989
990
991
992
993
994
995
996
997
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1167
1168
1169
1170
1171
1172
1173
1174
1175
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1185
1186
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1197
1197
1198
1199
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1276
1277
1278
1279
1279
1279
1280
1281
1282
1283
1284
1285
1286
1286
1287
1288
1289
1289
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1297
1298
1299
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1376
1377
1378
1379
1379
1379
1380
1381
1382
1383
1384
1385
1386
1386
1387
1388
1389
1389
1389
1390
1391
1392
1393
1394
1395
1395
1396
1397
1397
1398
1399
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1476
1477
1478
1479
1479
1479
1480
1481
1482
1483
1484
1485
1486
1486
1487
1488
1489
1489
1489
1490
1491
1492
1493
1494
1495
1495
1496
1497
1497
1498
1499
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1576
1577
1578
1579
1579
1579
1580
1581
1582
1583
1584
1585
1586
1586
1587
1588
1589
1589
1589
1590
1591
1592
1593
1594
1595
1595
1596
1597
1597
1598
1599
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1676
1677
1678
1679
1679
1679
1680
1681
1682
1683
1684
1685
1686
1686
1687
1688
1689
1689
1689
1690
1691
1692
1693
1694
1695
1695
1696
1697
1697
1698
1699
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1786
1787
1788
1789
1789
1789
1790
1791
1792
1793
1794
1795
1795
1796
1797
1797
1798
1799
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1876
1877
1878
1879
1879
1879
1880
1881
1882
1883
1884
1885
1886
1886
1887
1888
1889
1889
1889
1890
1891
1892
1893
1894
1895
1895
1896
1897
1897
1898
1899
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1997
1998
1999
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2137
2138
2
```

```

"use strict";
import mobx, { observable, action } from "mobx";
import { AsyncStorage } from "react-native";

const segmentRate = 10; // yılannın her hareketinde katedeceği mesafe

class GameStore {
  @observable highScore = 0; // yapılan en yüksek skoru tutar
  @observable score = 0; // yiilen elma sayısı
  @observable intervalRate = 15; // yılannın hızı
  @observable currentDirection = "right"; // Yılanın yönü, alacağı değerler: left / right / up / down
  @observable lastSegment = 10; // her segmentin kendinden önce takip edeceği, segment
  @observable
    snake = [
      { id: 1, x: 20, y: 0 },
      { id: 2, x: 10, y: 0 },
      { id: 3, x: 0, y: 0 }
    ];
  ...
}

export default new GameStore();

```

Yılan her bir parçası Segment.js olarak isimlendirdiğim, component'ten olusacak. Segment component'ine bakalım. Aşağıda görüldüğü bir absolute position'a sahip bir component. Ve dışarıdan left ve top, propslarını alıyor. Yani üstte verdigimiz yılann başlangıç koordinatlarına göre yılannız şkil alıyor. Herbir segment sharedStyle'dan gelen 10px lik, yükseklik ve genişliğe sahip. Böylece gameStore'da bulunan segmentRate'e göre uygun bir hareket sergileyecək.

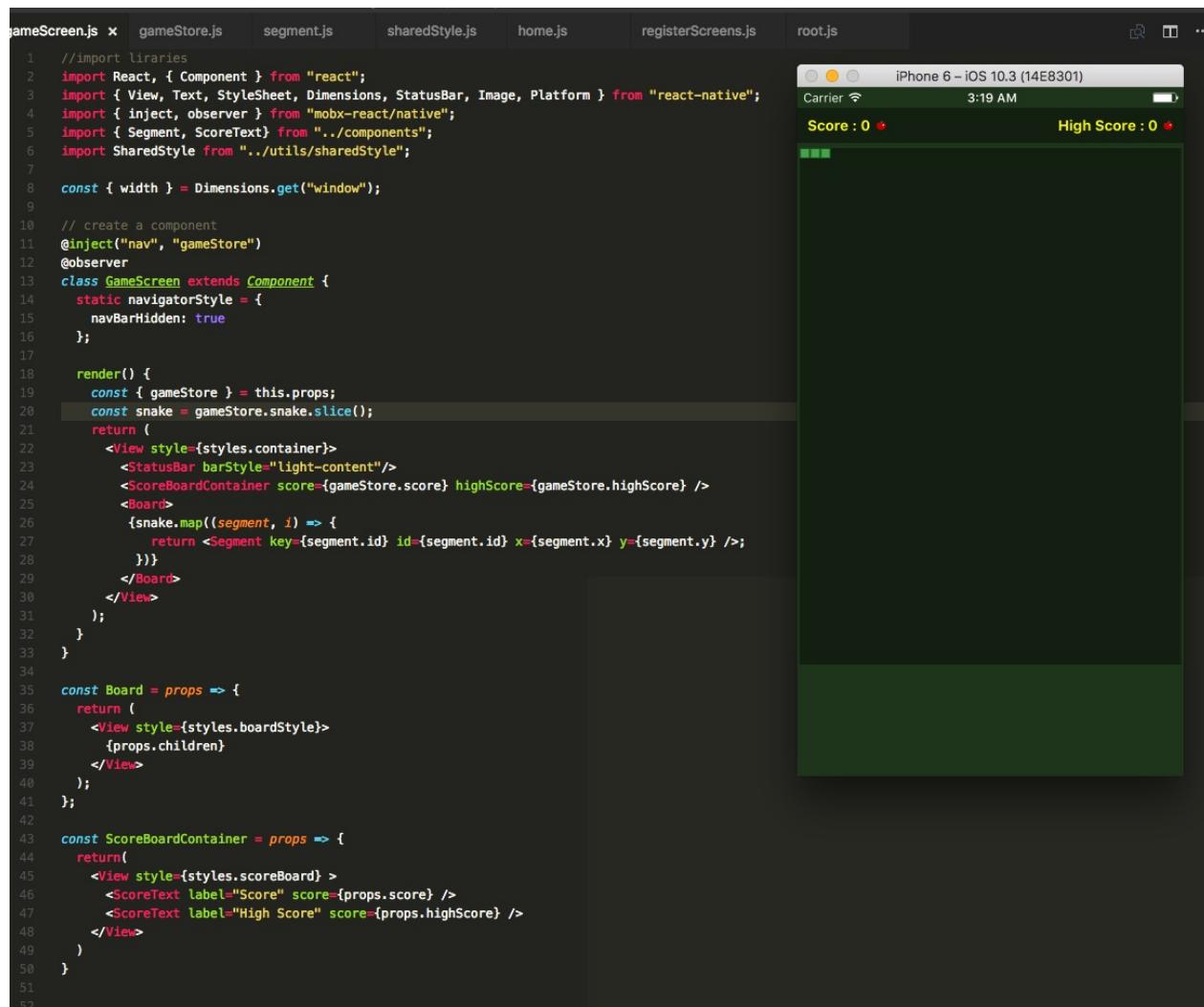
```
//src/components/segment.js
import React, { Component } from "react";
import { View, Text, StyleSheet } from "react-native";
import SharedStyle from "../utils/sharedStyle";

// create a component
class Segment extends Component {
  render() {
    const customStyle = {
      left: this.props.x,
      top: this.props.y
    };
    return <View style={[styles.container, customStyle]} />;
  }
}

// define your styles
const styles = StyleSheet.create({
  container: {
    position: "absolute",
    width: SharedStyle.segment.width,
    height: SharedStyle.segment.height,
    backgroundColor: SharedStyle.color.snake,
    borderWidth: SharedStyle.segment.borderWidth,
    borderColor: SharedStyle.segment.borderColor
  }
});

//make this component available to the app
export default Segment;
```

Şimdi yılanımızı, gameStore'daki snake array'ini map ederek board'umuzun içine koyalım.



```

1 //import libraries
2 import React, { Component } from "react";
3 import { View, Text, StyleSheet, Dimensions, StatusBar, Image, Platform } from "react-native";
4 import { inject, observer } from "mobx-react/native";
5 import { Segment, ScoreText } from "../components";
6 import SharedStyle from "../../utils/sharedStyle";
7
8 const { width } = Dimensions.get("window");
9
10 // create a component
11 @inject("nav", "gameStore")
12 @observer
13 class GameScreen extends Component {
14   static navigatorStyle = {
15     navBarHidden: true
16   };
17
18   render() {
19     const { gameStore } = this.props;
20     const snake = gameStore.snake.slice();
21     return (
22       <View style={styles.container}>
23         <StatusBar barStyle="light-content"/>
24         <ScoreBoardContainer score={gameStore.score} highScore={gameStore.highScore} />
25         <Board>
26           {snake.map((segment, i) => {
27             return <Segment key={segment.id} id={segment.id} x={segment.x} y={segment.y} />;
28           })}
29         </Board>
30       </View>
31     );
32   }
33 }
34
35 const Board = props => {
36   return (
37     <View style={styles.boardStyle}>
38       {props.children}
39     </View>
40   );
41 };
42
43 const ScoreBoardContainer = props => {
44   return(
45     <View style={styles.scoreBoard} >
46       <ScoreText label="Score" score={props.score} />
47       <ScoreText label="High Score" score={props.highScore} />
48     </View>
49   )
50 }
51
52

```

## Peki bu yılan nasıl hareket edecek ?

React Native'de oyun yazmak gerçek anlamda bir challenge olabilir. Çünkü burada biz durmadan render edilecek bir yılanın segmentlerinden bahsediyoruz. Bu yılan büyündüğünde mesela 100 segment'lik bir boyaya sahip olduğunda her bir milisaniyede her bir segmentin bir birim hareket etmesi zaten 100 defa render edilmesi demek. Bunu durmaksızın tüm board boyunca sonsuz kere tekrar edebilmesi, react native'in bridge yapısına ters gelebilir. Bu soru benim de aklımı bir hayli meşgul etti ve ilk denemem olan yıları setInterval ile yapmam ve yılan hareket ettikçe yavaşlaması uygulamanın performansının yerbere düşmesine sebep oldu. Sonra başka hangi timer'i kullanırmış diye düşünürken, **requestAnimationFrame** imdadıma yetişti. requestAnimationFrame, saniyede 1000 kez bir kodu baştan aşağı çalıştırabiliyor.

Yılanımız başlangıç olarak sağa doğru hareket ediyor. Yani snake array'in içindeki ilk elemanın x koordinat +10 olarak artacak ve bir sonraki segment kendinden önceki segmentin koordinat değerlerine sahip olacak. Algoritmanın özeti bu. Şimdi gameStore'da ilgili action methodunu oluşturalım.

```
//src/stores/gameStore.js

const segmentRate = 10; // yılanın her hareketinde katedeceği mesafe
const boardWidth = SharedStyle.board.width;
const BoardHeight = SharedStyle.board.height - 10;
let globalID;

...
@action("Snake is moving")
handleMoveSnake =() => {
  let temp = _.cloneDeep(this.snake.slice());
  this.lastSegment = temp[0];
  for (let i = 0; i < temp.length; i++) {
    if (i !== 0) {
      this.lastSegment = temp[i - 1];
    }

    if (this.currentDirection === "right") {
      if (i === 0) {
        if (this.snake[i].x + segmentRate >= boardWidth ) {
          this.snake[i].x = 0;
        }else{
          this.snake[i].x = this.snake[i].x + segmentRate;
        }
      } else {
        this.snake[i].x = this.lastSegment.x;
        this.snake[i].y = this.lastSegment.y;
      }
    }
  }
}

globalID = setTimeout(()=>{
  requestAnimationFrame(this.handleMoveSnake);
}, 1000 / this.intervalRate);
}
...
```

Buradaki kodu üste yazdığımız paragrafa göre inceleyelim.

İlk önce var olan snake array'inin bir kopyasını oluşturdum. Javascript'in , çoklu levelli array'lerde eşitlemede referans verip, asıl array'i değiştirmemesi için lodash'ın cloneDeep methodunu kullandım.

```
let temp = _.cloneDeep(this.snake.slice());
```

for döngüsü ile tüm array'i dönüyoruz. Eğer array'in ilk elemanı, yani yılanımızın başı değilse segment, yılanın yönü farketmeksızın kendisinden bir önceki segmentin değerlerini almasını istiyoruz. lastSegment değerini bu yüzden tutuyoruz.

```

if (i !== 0) {
    this.lastSegment = temp[i - 1];
}

```

Yılanın yönü, sağa veya sola doğru ise sadece x koordinat düzleminde işlem yapacağımız demek. Sağa giderken yılanın başı, segmentRate kadar ilerleyecek. Eğer board'un sonuna geldiyse, board'un tekrar başına gidecek. Ve diğer segmentler de kendinden önceki segmentin değerlerini alacak.

```

if (this.currentDirection === "right") {
    if (i === 0) {
        if (this.snake[i].x + segmentRate >= boardWidth ) {
            this.snake[i].x = 0;
        }else{
            this.snake[i].x = this.snake[i].x + segmentRate;
        }
    } else {
        this.snake[i].x = this.lastSegment.x;
        this.snake[i].y = this.lastSegment.y;
    }
}

```

Bu işlemleri sürekli tekrar etmemizi sağlayacak kod parçacığına gelelim. requestAnimationFrame, saniyede 1000 kez frame geçisi yapabiliyor. Yılanımızın hızını kontrol etmek için requestAnimationFrame'i setTimeout ile beraber kullanıyoruz. Böylece frame geçiş hızını ayarlayabiliyoruz.

```

...
globalID = setTimeout(()=>{
    requestAnimationFrame(this.handleMoveSnake);
}, 1000 / this.intervalRate);

```

Yılanın yönü sağa doğru iken yaptığımız işlemleri, sola, yukarı, aşağı olarak da aynı mantıkla if koşulları ekleyip çoğaltıyoruz. Kalabalık etmesin diye tekrar eden işlemleri buraya yazmak istemiyorum.

Daha sonra gameScreen ekranının componentDidMount methodu aşağıdaki kod parçacığını ekleyelim. Yılanın hareket ettiğini görelim. ( Tabi burada .gif almak için kullandığım tool'da tam bir smooth animation göremeyebilirsiniz, ama nasıl performanslı çalıştığını performans monitor'de görebilirsiniz. )

```

    ...
componentDidMount() {
  const { gameStore } = this.props;
  gameStore.handleMoveSnake();
}
...

```

```

gameScreen.js x gameStore.js segment.js sharedStyle.js home.js registerScreens.js rod Carrier 4:15 AM Score : 0 * High Score : 0 *
1 //import libraries
2 import React, { Component } from "react";
3 import { View, Text, StyleSheet, Dimensions, StatusBar, Image, Platform } from "react-native";
4 import { inject, observer } from "mobx-react/native";
5 import { Segment, ScoreText } from "./components";
6 import SharedStyle from "../utils/sharedStyle";
7
8 const { width } = Dimensions.get("window");
9
10 // create a component
11 @inject("nav", "gameStore")
12 @observer
13 class GameScreen extends Component {
14   static navigatorStyle = {
15     navBarHidden: true
16   };
17
18   componentDidMount() {
19     const { gameStore } = this.props;
20     gameStore.handleMoveSnake();
21   }
22
23   render() {
24     const { gameStore } = this.props;
25     const snake = gameStore.snake.slice();
26     return (
27       <View style={styles.container}>
28         <StatusBar barStyle="light-content"/>
29         <ScoreBoardContainer score={gameStore.score} highScore={gameStore.highScore} />
30         <Board>
31           {snake.map((segment, i) => {
32             return <Segment key={segment.id} id={segment.id} x={segment.x} y={segment.y} />;
33           })}
34       </Board>

```

Şimdi yön kontrollerini ekleyelim. Yön kontrollerinin mantığı, ilk bölümde belirttiğimiz gibi eski Nokia telefonlarda 3 ve 7 tuşlarıyla oynar gibi olacak. Sağdaki yön tuşu sağa ve yukarı götürecek, soldaki yön tuşu sola ve aşağı götürecek. Bunun için gameStore'a 2 tane daha action ekleyelim, handleRightButton ve handleLeftButton.

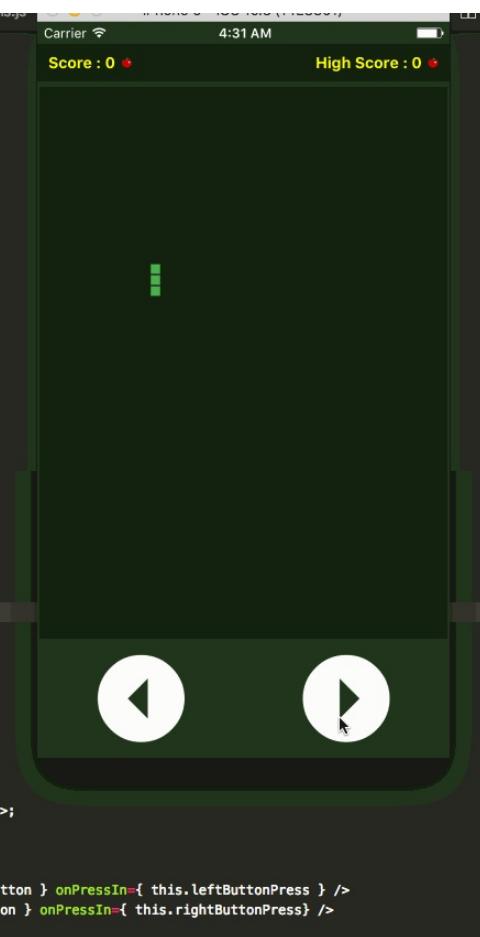
```

@action("leftButton Pressed")
handleLeftButton() {
  if (this.currentDirection === "right") {
    this.currentDirection = "down";
    this.leftButtonText = "left";
    this.rightButtonText = "right";
  } else if (this.currentDirection === "down") {
    this.currentDirection = "left";
    this.leftButtonText = "down";
    this.rightButtonText = "up";
  } else if (this.currentDirection === "left") {
    this.currentDirection = "down";
    this.leftButtonText = "left";
    this.rightButtonText = "right";
  } else if (this.currentDirection === "up") {
    this.currentDirection = "left";
    this.leftButtonText = "down";
    this.rightButtonText = "up";
  }
}

@action("rightButton Pressed")
handleRightButton() {
  if (this.currentDirection === "right") {
    this.currentDirection = "up";
    this.rightButtonText = "right";
    this.leftButtonText = "left";
  } else if (this.currentDirection === "down") {
    this.currentDirection = "right";
    this.rightButtonText = "up";
    this.leftButtonText = "down";
  } else if (this.currentDirection === "left") {
    this.currentDirection = "up";
    this.rightButtonText = "right";
    this.leftButtonText = "left";
  } else if (this.currentDirection === "up") {
    this.currentDirection = "right";
    this.rightButtonText = "up";
    this.leftButtonText = "down";
  }
}

```

Daha sonra gameScreen ekranına da butonları ekleyelim.



```

gameScreen.js X gameStore.js Segments.js SnakeCell.js Home.js RegisterScreen.js
7
8 const { width } = Dimensions.get("window");
9 const rightButton = require('../assets/rightButton.png');
10 const leftButton = require('../assets/leftButton.png');
11 const upButton = require('../assets/upButton.png');
12 const downButton = require('../assets/downButton.png');
13
14 // create a component
15 @inject("nav", "gameStore")
16 @observer
17 class GameScreen extends Component {
18   static navigatorStyle = {
19     navBarHidden: true
20   };
21
22   componentDidMount() {
23     const { gameStore } = this.props;
24     gameStore.handleMoveSnake();
25   }
26
27   leftButtonPress={()=>{
28     const { gameStore } = this.props;
29     gameStore.handleLeftButton();
30   }
31
32   rightButtonPress={()=>{
33     const { gameStore } = this.props;
34     gameStore.handleRightButton();
35   }
36
37   render() {
38     const { gameStore } = this.props;
39     const snake = gameStore.snake.slice();
40     return (
41       <View style={styles.container}>
42         <StatusBar barStyle="light-content"/>
43         <ScoreBoardContainer score={gameStore.score} highScore={gameStore.highScore} />
44         <Board>
45           {snake.map((segment, i) => {
46             return <Segment key={segment.id} id={segment.id} x={segment.x} y={segment.y} />;
47           })}
48         </Board>
49         <ButtonContainer>
50           <DirectionButton icon={gameStore.leftButtonText === 'down' ? downButton : leftButton} onPressIn={ this.leftButtonPress } />
51           <DirectionButton icon={gameStore.rightButtonText === 'up' ? upButton : rightButton} onPressIn={ this.rightButtonPress } />
52         </ButtonContainer>
53       </View>
54     );
55   }
56 }
57
58 
```

Yılan, özgürce hareket etmeye başladı. Şimdi bu yılanı doyurmak için board'da random şekilde belirecek olan elma componentini yaratalım. Bunun için gameStore'da aşağıdaki işlemleri yapalım. elma board içinde segmentRate'in katları ( yani 10'nun katları ) koordinatlarda belirmesi lazım ki, yılanımız elmayı yiyebsin.

```

// src/stores/gameStore.js
class GameStore {
  ...
  @observable food = { x: 50, y: 50 }; //elmanın başlangıçta belireceği yer.
  ...
  @action("make_food")
  handleMakeFood() {
    const frameX = (boardWidth -10) / segmentRate;
    const frameY = BoardHeight / segmentRate;

    this.food.x = this.getInt(0, frameX) * segmentRate;
    this.food.y = this.getInt(0, frameY) * segmentRate;
  }

  getInt(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
  }
}

```

Elma componentini de gameScreen ekranında board içine koyalım.

```

    ...
<Board>
  {snake.map((segment, i) => {
    return <Segment key={segment.id} id={segment.id} x={segment.x} y={segment.y} />;
  })}
  <Food x={ gameStore.food.x } y={ gameStore.food.y }/>
</Board>
...

```

Yılan elmayı yedi mi kontrolü ekleyelim. Bu methodu yılın her bir segment ilerleyişinde çağırıp kontrol edelim. Yılan her elmayı yediğinde handleMakeFood methodunu çağırıp, yeni bir elma oluşturalım ve score state'ini +1 artıralım ve snake array'ini bir tane daha ekleyelim. Son olarak da yılın her 3 elma yediğinde hızını artıralım. Aşağıdaki method bunları yapacak.

```

@action("Did the snake eat the food ?")
handleEatFood() {
  if (this.snake[0].x === this.food.x && this.snake[0].y === this.food.y) {
    this.score = this.score + 1;
    this.snake.push({
      id: this.snake[this.snake.length - 1].id + 1,
      x: this.snake[this.snake.length - 1].x,
      y: this.snake[this.snake.length - 1].y
    });
    if( this.score % 3 === 0 ){
      this.intervalRate = this.intervalRate + 5;
    }
    this.handleMakeFood();
  }
}

```

The screenshot shows a mobile application development interface. On the left, there is a code editor with JavaScript code for a game. On the right, there is a preview window showing a snake game. The code editor has tabs for Screen.js, gameStore.js, segment.js, sharedStyle.js, home.js, registerScreens.js, and root. The preview window shows a dark green background with a small green snake segment at the top center. The status bar at the top right indicates 'Score : 1' and 'High Score : 0'. Below the preview are two circular control buttons.

```

Screen.js      gameStore.js x    segment.js      sharedStyle.js      home.js      registerScreens.js      root      Carrier      4:47 AM      Score : 1      High Score : 0
@action("make food")
handleMakeFood() {
  const frameX = (boardWidth -10) / segmentRate;
  const frameY = BoardHeight / segmentRate;

  this.food.x = this.getRandomInt(0, frameX) * segmentRate;
  this.food.y = this.getRandomInt(0, frameY) * segmentRate;
}

getRandomInt(min, max) {
  return Math.floor(Math.random() * (max - min + 1)) + min;
}

@action("Did the snake eat the food ?")
handleEatFood() {
  if (this.snake[0].x === this.food.x && this.snake[0].y === this.food.y) {
    this.score = this.score + 1;
    this.snake.push({
      id: this.snake[this.snake.length - 1].id + 1,
      x: this.snake[this.snake.length - 1].x,
      y: this.snake[this.snake.length - 1].y
    });
    if( this.score % 3 === 0 ){
      this.intervalRate = this.intervalRate + 5;
    }
    this.handleMakeFood();
  }
}

@action("getting high score")
getHighScore() {
  AsyncStorage.getItem("snakeHighScore").then(_highScore => {
    this.highScore = _highScore;
  })
}

```

Şimdi yılan kendi kuyruğuna çarptığında oyunun bitmesini sağlayalım. Bunun için handleMoveSnake action methoduna bir kontrol daha ekleyelim. Burada da yapacağımız yılanın başı yani ilk segment, yılanın başka herhangi segmentiyle aynı koordinat değerlerine sahip olup olmadığını kontrol etmek. Eğer kuyruğuna çarpmışsa, requestAnimationFrame'i, cancelAnimationFrame ile sonlandıracıız. Elde edilen score, var olan en yüksek skordan daha büyük ise telefon hafızasına AsyncStorage ile yeni skoru set edeceğiz. Ve `NavigationStore.handleChangeRoute( 'gameOverScreen' );` ile de gameOver ekranına yönlendirme yapacağız.

Burada dikkat etmenizi istediğim 2 tane husus var.

1. cancelAnimationFrame, requestAnimationFrame'i durdurması için mutlaka sonunda return yapmanız gerekmektedir. Eğer yapmazsanız kod aşağı doğru derlenmeye devam edecektir.
2. MobX'de store'ları birbiri içinde import edip , birbirlerinin methodlarını çağırmasını sağlayabilirsiniz.

```
//src/stores/gameStore.js
...
class GameStore{
    ...
    @action("Snake is moving")
    handleMoveSnake =() => {
        ...
        //isGameOver control
        for(let i = 1; i < this.snake.slice().length; i++){
            if(this.snake[0].x === this.snake[i].x && this.snake[0].y === this.snake[i].y ){
                if( this.score > this.highScore ){
                    AsyncStorage.setItem('snakeHighScore', JSON.stringify(this.score));
                }
                cancelAnimationFrame(this.handleMoveSnake);
                NavigationStore.handleChangeRoute('gameOverScreen');
                clearTimeout(globalID);
                return;
            }
        }
        globalID = setTimeout(()=>{
            requestAnimationFrame(this.handleMoveSnake);
        }, 1000 / this.intervalRate);
    }
}
```

En son ekranımızın üzerinden de kısa bir şekilde geçelim. Sayfanın başındaki tasarıma bakarsanız, gameOver ekranında oyunu yeniden başlatacak bir tane buton var. Burada yapacağımız tek şey, NavigationStore.handleChangeRoute('gameScreen'); ile ekranı tekrar gameScreen'e çekmek ve başlangıçtaki state'lere geri dönmek. Aşağıdaki gibi;

```
//src/stores/gameStore.js  
...  
class GameStore{  
    ...  
    @action("restart handler")  
    handleRestart(){  
        NavigationStore.handleChangeRoute('gameScreen');  
        this.intervalRate = 5;  
        this.currentDirection = "right";  
        this.lastSegment = 10;  
        this.rightButtonText = "up";  
        this.leftButtonText = "down";  
        this.score = 0;  
        this.highScore = 0;  
        this.food = { x: 50, y: 50 };  
        this.snake = [  
            { id: 1, x: 20, y: 0 },  
            { id: 2, x: 10, y: 0 },  
            { id: 3, x: 0, y: 0 }  
        ];  
    }  
}  
}
```

Hepsi bu kadar. Bir sonraki bölümde bu oyunun Google Play Store'a ve AppStore'a deploy edilmesinden ve aşamalarından bahsedeceğiz.

# Android Apk Oluşturma

Aslında apk oluşturma aşamasına geldiyseniz, bu kısmın react-native ile bir alakası yok. Android geliştiricilerin yaptıklarını yapacağınız burada. Bunun 2 yöntemi var, Android Studio yardımcı ile yapabilirsiniz ya da direkt komut satırını da kullanabilirsiniz. Biz burada, daha kolay olduğu için komut satırıyla adım adım uygulamamızın apk sini oluşturacağımız.

Android'in kendi [resmi sayfasından](#) daha detaylı bakabilirsiniz.

Diyelim ki bizim **myapp** isimli bir uygulamamız olsun.

1. Uygulamamızın klasörüne `cd` komutu ile girelim ve `react-native run-android` komutu ile react-native packager'ı çalıştıralım.
2. `.../workspace/myapp/android/app/src/main`` klasörünün altına eğer yoksa `assets` isimli bir klasör oluşturalım.
3. Uygulamamızın bundle ini assets klasörüne indirmek için şu kodu komut satırında çalıştıralım.

```
curl "http://localhost:8081/index.android.bundle?platform=android" -o  
"android/app/src/main/assets/index.android.bundle"
```

4. Şimdi komut satırında android klasörüne girip `\( cd `.../workspace/myapp/android` \)` aşağıdaki kod ile unsigned edilmemiş apk mizi oluşturalım.

```
./gradlew assembleRelease
```

5. Şimdi oluşturduğumuz apk'yı signed edebilmek için kendimize bir keytool üretelim ve \*\*bu keytool'u her zaman bulabileceğimiz, şifresini asla unutmayacağımız biçimde\*\* bir klasörde muhafaza edelim. Nerde hangi klasörde yaratacağınız tamamen size kalmış.Keytool üretmek için aşağıdaki kodu komut satırında çalıştıralım.

```
keytool -genkey -v -keystore my-keystore.keystore -alias name_alias -keyalg RSA -  
validity 10000
```

my-keystore: keytool'unuz klasör yapısında göreceğiniz ismi, başka isim verebilirsiniz. name\_alias: bu çok spesifik bir şey olmalı. Bir sonraki adımda ihtiyacımız olacak.

6. Şimdi uygulamamızı ürettiğimiz keytool ile signed edelim. (`<keytool-alias>` 'i bir önceki adımdan hatırlayın)

```
jarsigner -verbose -keystore <keystore_path.keystore>  
<...workspace/myapp/android/app/build/outputs/apk/app-release-unsigned.apk> <keytool-  
alias>
```

7. Şimdi sıra uygulamamızın apk'sına isim vermeye. Bunun için de sadece yapmanız gereken tabi ki yine aşağıdaki kodu çalıştırmanız. (Bizim uygulamamızın ismi myapp olduğu için ben myapp verdim)

```
cd ...app/build/outputs/apk
```

```
zipalign -f -v 4 app-release-unsigned.apk myapp.apk
```

8. Bu kadar, artık ürettiğiniz apk'yi ister telefonunuza yükler kullanırsınız isterseniz bir [Google Play Developer](#) hesabı açıp store'a koyabilirsiniz.