

**ANKARA ÜNİVERSİTESİ**  
**MÜHENDİSLİK FAKÜLTESİ**  
**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**



**BLM 4538 PROJE RAPORU**

**MOBILE UBERCLONE APP**

**MUHAMMET TAHA GÜNER**

**19291064**

**ENVER BAĞCI**

**06/2023**



## ÖZET

Özet olarak UberClone, React Native ile geliştirilmiş bir mobile uygulamasıdır. Bu uygulama aynı bir Uber çağırma sitesi gibi çalışabilen IOS ve Android'de sorunsuz şekilde açılan bir uygulamadır. Örneğin İstanbul'da yaşayan biri Ankara'ya gitmek için kendi bütçesine uygun ve en hızlı şekilde gitmek istediği yere ulaşabilecek. Kısaca bahsetmemiz gerekirse bu uygulama sayesinde insanlar taksilere tonlarca para vermek zorunda kalmayacak. Bu sayede kullanıcılar daha rahat ve konforlu bir yolculuk deneyimi kazanabilecek.

# İçindekiler

1. GİRİŞ .....	1
1.1 AMAÇ VE BÖLÜMLER.....	1
2. YARARLANDIĞIM PROGRAMLAR .....	2
2.1 VISUAL STUDIO CODE .....	2
2.2 TAILWIND CSS.....	3
2.3 REDUX.....	4
2.4 GOOGLE AUTOCOMPLETE .....	5
3. PROJE BÖLÜMLERİ.....	6
3.1 APP.JS .....	6
3.2 STORE.JS .....	9
3.3 NAVSLICE.JS .....	10
3.4 HOMESCREEN.JS.....	13
3.5 NAVOPTIONS.JS .....	17
3.6 GOOGLE CLOUD AUTOCOMPLETE.....	21
3.7 BABELCONFIG.JS VE .ENV .....	26
3.8 MAPSCREEN.JS .....	28
3.9 MAP.JS .....	30
3.10 RIDEOPTIONSCARD.JS.....	36
3.11 NAVIGATECARD.JS .....	40
3.12 NAVFAVOURITES.JS .....	45
4. UYGULAMADAN EKRAN GÖRÜNTÜLERİ.....	48
5. SONUÇ .....	56
6. KAYNAKÇA .....	56

# 1.GİRİŞ

## **1.1 AMAÇ VE BÖLÜMLER**

React Native ile oluşturulmuş bu UberClone uygulaması, popüler yolculuk paylaşımı ve taşımacılık hizmeti olan Uber'ın temel işlevlerini taklit etmek ve benzer bir deneyim sunmaktır. Bu uygulama, kullanıcılara yolculuk talepleri oluşturma, sürücülerin bulunması ve seyahatlerin gerçekleştirilmesi gibi Uber'in sağladığı ana özellikleri sunmayı hedeflemektedir. Ayrıca kullanıcılar, nereden nereye gitmek istediklerini belirten yolculuk talepleri oluşturabilirler. Bunun yanın sıra kullanıcılar, yolculuk talepleri sırasında tahmini süre ve ücret bilgilerini görebilirler. Bu, kullanıcının yolculuk maliyetini tahmin etmesine ve seyahat planlamasını buna göre yapmasına olanak sağlar, ve en önemlisi bu uygulamayı kullanan kişiler, sürücünün konumunu gerçek zamanlı olarak takip edebilirler. Bu özellik, kullanıcının sürücünün ne zaman varacağını ve bekleyeceği süreyi görmesine yardımcı olur.

## 2. YARARLANDIĞIM PROGRAMLAR

### 2.1 VISUAL STUDIO CODE

Visual Studio Code (VS Code), Microsoft tarafından geliştirilen ücretsiz ve açık kaynaklı bir metin düzenleyicisidir. Geniş bir kullanıcı tabanına sahip olan VS Code, özellikle web ve yazılım geliştirme alanında popülerdir.

VS Code, modern ve kullanıcı dostu bir arayüze sahiptir. Ana ekranında dosya gezginine, düzenleme alanına ve yan paneldeki araçlara erişim sağlar. Kullanıcılar, temalar ve eklentiler aracılığıyla arayüzün görünümünü ve davranışını özelleştirebilirler.

Bunun yanında birçok programlama dili için entegre dil desteği sunar. Önemli diller arasında JavaScript, TypeScript, HTML, CSS, Python, C#, Java, Go ve daha birçok dil bulunur. Bu sayede, farklı dillerde yazılmış projelerde kolaylıkla çalışabilirsiniz.

VS Code'un başka bir mahareti de, zengin bir kod tamamlama özelliği sunar. Kullanıcılar, değişkenler, fonksiyonlar, sınıflar ve diğer öğeler için otomatik tamamlama önerilerini görebilir ve hızlı bir şekilde kod yazabilirler. Ayrıca, hataları vurgulayarak ve uyarılarla belirterek kod kalitesini artıran bir hata kontrolü mekanizması da bulunur. Birçok programlama dili için entegre hata ayıklama özelliği sağlar. Kullanıcılar, hata ayıklama noktaları belirleyebilir, adım adım kodu çalıştırabilir, değişkenlerin değerlerini kontrol edebilir ve hata ayıklama işlemi sırasında hata izlerini takip edebilirler. Geniş bir eklenti ekosistemine sahiptir. Kullanıcılar, ihtiyaçlarına göre farklı eklentileri kurabilir ve bu sayede işlevselliği genişletebilirler. Örneğin, temalar, dil desteği, özelleştirilmiş hata ayıklama araçları, kod analizi araçları gibi birçok farklı eklenti bulunur.

## **2.2 TAILWIND CSS**

Tailwind CSS, web geliştirme için bir CSS çerçevesidir. Tailwind CSS, HTML dosyalarında bu sınıfları kullanarak stil ve düzen tanımlamanızı sağlar. Tailwind CSS, birincil yaklaşım olarak "utility-first" prensibini benimser. Bu, önceden tanımlanmış bileşenler yerine, küçük ve özgün CSS sınıflarının kullanımını teşvik eder. Örneğin, **.text-center**, **.p-4**, **.bg-gray-500** gibi sınıfları doğrudan HTML elementlerine ekleyerek hızlı bir şekilde stil ve düzen sağlayabilirsiniz.

Tailwind CSS, modüler bir yapıya sahiptir. CSS sınıfları küçük ve bağımsız birimlere ayrıldığından, bu sınıfları yeniden kullanabilir ve birleştirebilirsiniz. Böylece, bileşenlerinizi ve stil öğelerinizi projenizde tekrar tekrar kullanabilirsiniz. Tailwind CSS'in doping sınıflar olarak adlandırılan birçok özelliği vardır. Örneğin, genişlik, yükseklik, kenar boşlukları, yazı tipleri, renkler, pozisyonlar gibi stil özelliklerini hızlı bir şekilde uygulamak için kullanabileceğiniz sınıflar sağlar. Bu, hızlı prototipleme ve stil uygulaması için kolaylık sağlar.

Tailwind CSS, responsive tasarıma yönelik sınıflar sağlar. Farklı ekran boyutlarına göre stil ve düzen ayarlamaları yapmak için önceden tanımlanmış sınıfları kullanabilirsiniz. Örneğin, **.md:flex** sınıfı sadece orta büyüklükteki (md) ekranlarda esnek bir düzen sağlar.

Tailwind CSS, minimalist bir yaklaşım benimseyen, hızlı ve özelleştirilebilir bir CSS çerçevesidir. Çok fazla stil önceden tanımlanmamış olsa da, doping sınıflar ve modüler yapı sayesinde hızlı ve esnek bir şekilde stil oluşturmanıza imkan tanır.

## **2.3 REDUX**

Redux, React Native gibi bir uygulama geliştirme çerçevesiyle birlikte kullanılan bir durum yönetim kütüphanesidir. Redux, uygulama durumunun merkezi bir yerde tutulmasını ve bileşenler arasında veri paylaşımını kolaylaştıran bir mimari sağlar. Redux, özellikle büyük ve karmaşık uygulamalarda durum yönetimini etkili bir şekilde yönetmek için tercih edilir.

Action, bir olayın gerçekleştiğini belirten JavaScript nesneleridir. Bir action, genellikle bir olayı tetikleyen kullanıcı etkileşimi veya veri kaynağından gelen bir değişiklik olabilir. Reducer'lar, mevcut durumu ve bir action'ı kullanarak yeni bir durum döndüren fonksiyonlardır. Her bir reducer, uygulamanın farklı bölümlerindeki durumu yönetir. Redux, tüm reducer'ların çalışması sonucunda toplam durumu günceller. Dispatch, bir action'ın store'a gönderildiği fonksiyondur. Bir bileşen, dispatch işlevini kullanarak bir action'ı tetikleyebilir. Dispatch, action'ı reducer'lara iletir ve durum güncellemelerinin yapılmasını sağlar. Subscribe, store'daki durum değiştiğinde tetiklenen bir fonksiyondur. Bileşenler, subscribe işlevini kullanarak store'daki durum değişikliklerini dinleyebilir ve uygun şekilde yeniden render edilebilir.

Redux, uygulamanın durumunu merkezi bir şekilde yöneterek bileşenler arasında veri paylaşımını kolaylaştırır. Bu, uygulama ölçeklendirildikçe durumun karmaşık hale geldiği durumlarda önemli bir avantaj sağlar. Redux, ayrıca zamanla değişen durumu kaydetme, geri alma veya tekrar oynatma gibi gelişmiş durum yönetimi özelliklerini de sağlar.



## **2.4 GOOGLE AUTOCOMPLETE**

Google AutoComplete, Google'ın arama motoru hizmetinde kullanılan bir özelliktir. Bu özellik, kullanıcının arama sorgusunu tamamlamasına yardımcı olmak amacıyla öneriler sunar. Kullanıcı bir arama sorgusu yazarken, AutoComplete, kullanıcının girişini tamamlamak için öneriler sunar ve kullanıcının istediği sonuca daha hızlı ulaşmasını sağlar.

Google AutoComplete, kullanıcıların popüler arama terimlerini keşfetmelerine ve sorgularını daha spesifik hale getirmelerine yardımcı olur. Bu özelliğin arkasında Google'ın arama verileri, kullanıcı davranışları ve popüler arama terimleri gibi faktörler bulunur. Google, bu verileri kullanarak kullanıcıların en olası arama terimlerini tahmin eder ve bu tahminleri AutoComplete özelliği altında sunar.

AutoComplete, kullanıcının yazdığı kısmi arama terimine göre dinamik olarak değişen öneriler sunar. Öneriler, en yaygın veya popüler arama terimleri, tamamlanan sorgunun baş harfleriyle eşleşen terimler, son aramalar, konum bilgisi ve kullanıcının geçmiş arama davranışı gibi faktörlere dayanabilir.

AutoComplete, kullanıcılara aradıkları bilgilere hızlıca ulaşma imkanı sağlarken aynı zamanda arama sürecini kolaylaştırır. Kullanıcılar tamamlanan sorguları seçerek istedikleri sonuçları daha hızlı bulabilir veya sorgularını daha spesifik hale getirerek arama sonuçlarını daraltabilir.

Google AutoComplete, geniş bir kullanıcı tabanına dayanan veriler ve karmaşık algoritmalar kullanarak kullanıcı deneyimini iyileştirmeyi hedefler. Bu özellik, kullanıcıların arama yaparken daha etkili, hızlı ve doğru sonuçlara ulaşmalarını sağlamak amacıyla sürekli olarak geliştirilmektedir.

### 3. PROJE BÖLÜMLERİ

#### **3.1 APP.JS**

React Native uygulamalarında, genellikle "app.js" veya "App.js" olarak adlandırılan bir başlangıç noktası veya ana bileşen dosyası bulunur. Bu dosya, React Native uygulamasının giriş noktasıdır ve uygulamanın genel yapısını, bileşenlerini ve davranışını tanımlar.

App.js dosyası, React Native uygulamasının temel bileşenlerini içerir ve uygulamanın genel işlevselliğini ve görünümünü belirler. Bu dosya genellikle JavaScript veya TypeScript dilinde yazılır ve React Native tarafından sunulan bileşenlerin kullanımını içerir.

App.js dosyası, uygulamanın ana bileşeni olarak kabul edilir ve genellikle uygulamanın kök bileşeni olan "App" bileşenini içerir. Bu dosya, kullanıcı arayüzünü tanımlayan JSX (JavaScript XML) sözdizisini kullanarak bileşenlerin yapısını ve düzenini oluşturur. Ayrıca, uygulamanın durumunu yönetmek, API isteklerini yapmak, kullanıcı etkileşimlerini ele almak ve diğer işlevleri gerçekleştirmek için JavaScript kodları içerir.

React Native'de App.js dosyası, uygulamanın başlangıç noktasıdır ve uygulamanın tüm bileşenlerini yönetir. Diğer bileşenler genellikle app.js içinde dahil edilir ve uygulama bu dosyanın başlatılmasıyla çalışmaya başlar.

Öncelikle uygulamaya App.js dosyası içine bazı eklentileri import ederek (içine aktarma) başladım. Bu sayede uygulama arayüzünde ve diğer kodlamalarda herhangi bir sorunla karşılaşmayacağız.

```
App.js M X
UberClone > JS App.js > App
1 import React from "react";
2 import { KeyboardAvoidingView, Platform } from 'react-native';
3 import { Provider } from "react-redux";
4 import { store } from "../expo/store";
5 import HomeScreen from "../screens/HomeScreen";
6 import MapScreen from "../screens/MapScreen";
7 import { SafeAreaProvider } from 'react-native-safe-area-context';
8 import 'react-native-gesture-handler';
9 import { NavigationContainer } from '@react-navigation/native';
10 import { createStackNavigator } from '@react-navigation/stack';
```

Şekil 3.1.1.

Yukarıdaki kod bloğunda görüldüğü üzere İlk olarak, React ve gerekli React Native bileşenlerini import ediyoruz. React, React Native uygulamaları için temel kütüphane, KeyboardAvoidingView ise klavyeyle etkileşimdeki bileşenlerin düzenini ayarlamak için kullanılan bir bileşendir. Ardından, React Redux için Provider bileşenini import ediyoruz ve Redux store'umuza erişim sağlıyoruz. Bu, Redux ile state yönetimi sağlamak ve Redux store'una erişim sağlamak için kullanılan bileşenlerdir. Redux, uygulama genelindeki durumu (state) yönetmek için kullanılan bir kütüphanedir. Sonraki satırda, SafeAreaProvider'ı import ediyoruz. Bu, cihazın güvenli alanını kullanarak bileşenlerin uygun bir şekilde yerleştirilmesini sağlayan bir bileşendir. **'react-native-gesture-handler'** kütüphanesini import ediyoruz. Bu kütüphane, dokunma hareketleri ve jestlerle ilgili işlevselliği sağlamak için kullanılır. **@react-navigation/native** paketini import ediyoruz. Bu paket, gezinme (navigation) işlevselliği için gerekli olan temel bileşenleri sağlar. Son olarak, **@react-navigation/stack** paketini import ediyoruz. Bu paket, yığın tabanlı gezinme için kullanılan StackNavigator bileşenini sağlar. Bu kodlar, başlangıç noktası dosyasında kullanılan temel bağımlılıkları ve bileşenleri içerir. Bu kodlar, uygulamanın navigasyon yapısını oluşturmak, Redux store'a erişim sağlamak, güvenli alan düzenlemesini kullanmak ve diğer önemli kütüphaneleri import etmek için kullanılır.

```
js App.js M X
UberClone > js App.js > App
12 export default function App() {
13   const Stack = createStackNavigator();
14
15   return (
16     <Provider store={store}>
17       <NavigationContainer>
18         <SafeAreaProvider>
19           <KeyboardAvoidingView
20             behavior={Platform.OS === "ios" ? "padding" : null }
21             style={{ flex: 1 }}
22             keyboardVerticalOffset={Platform.OS === "ios" ? -64 : 0}
23           >
24             <Stack.Navigator>
25               <Stack.Screen
26                 name='HomeScreen'
27                 component={HomeScreen}
28                 options={{
29                   headerShown: false,
30                 }}
31               />
32               <Stack.Screen
33                 name='MapScreen'
34                 component={MapScreen}
35                 options={{
36                   headerShown: false,
37                 }}
38               />
39             </Stack.Navigator>
40           </KeyboardAvoidingView>
41         </SafeAreaProvider>
42       </NavigationContainer>
43     </Provider>
44   );
}
```

Şekil 3.1.2.

İlk olarak, **createStackNavigator** fonksiyonunu kullanarak bir Stack Navigator oluşturuyoruz. Bu, uygulamanın gezinme yapısını yönetmek için kullanılan Stack Navigator'ı temsil eder. Bileşen, **<Provider>** bileşeni ile Redux store'a erişim sağlar. Redux store, uygulama durumunun yönetildiği yerdir. **<Provider>** bileşeni, Redux store'u tüm bileşenlere sağlar, böylece bileşenler durumu okuyabilir ve güncelleyebilir. **<NavigationContainer>**, gezinme yapısını sağlamak için kullanılır. Bu, tüm navigasyon bileşenlerini sarmalar ve gezinme işlevselliğini sağlar. **<SafeAreaProvider>** bileşeni, cihazın güvenli alanını kullanarak bileşenlerin uygun bir şekilde yerleştirilmesini sağlar. **<KeyboardAvoidingView>**, klavyeyle etkileşimdeki bileşenlerin düzenini ayarlamak için kullanılır. Klavye açıldığında bileşenlerin kaymasını sağlar.

**Behavior** prop'u, platforma bağlı olarak klavye davranışını belirler. **style** prop'u, bileşenin görünümünü tanımlar. **KeyboardVerticalOffset** prop'u, klavyenin açıldığında bileşenlerin kaymasını kontrol eder. **<Stack.Navigator>** ise, Stack Navigator içindeki ekranları yönetmek için kullanılır. Stack Navigator içinde gezinilecek ekranları içerir. **<Stack.Screen>** bileşeni, Stack Navigator içindeki bir ekranı temsil eder. Bu bileşen, **HomeScreen** adında bir ekran bileşenini temsil eder. **name** prop'u, ekranın adını belirtir. **component** prop'u, ekrana karşılık gelen bileşeni belirtir. **options** prop'u, ekran seçeneklerini belirtir. Buna benzer şekilde, **<Stack.Screen>** bileşeni kullanarak başka bir ekranı tanımlarız. Son olarak yukarıda bahsettiğim bütün bileşenleri kapatıyoruz ve bu şekilde **'App'** Redux Store'a erişim sağlayan, navigasyon yapısını tanımlayan ve ekran bileşenlerini yöneten bir yapıya sahip olur. Bu kod sayesinde uygulamanın temel yapılandırılmasını ve navigasyonu için gerekli olan özellikleri eklemiş oluruz.

### 3.2 STORE.JS



```
JS App.js M JS store.js X
UberClone > .expo > JS store.js > [🔍] store
1 import { configureStore } from "@reduxjs/toolkit";
2 import navReducer from "../slices/navSlice";
3
4 export const store = configureStore({
5   reducer: {
6     nav: navReducer,
7   },
8 });
```

Şekil 3.2.1.

Bu kod, Redux Toolkit'in **configureStore** fonksiyonunu kullanarak Redux store'unu yapılandırır. İlk olarak, Redux Toolkit'in **configureStore** fonksiyonunu import ediyoruz.

Bu fonksiyon, önceden yapılandırılmış bir Redux store oluşturmak için kullanılır. **NavReducer**'ı import ediyoruz. Bu, Redux store'umuza ekleyeceğimiz bir parçadır ve navigasyon durumunu yönetmek için kullanılır. **configureStore** fonksiyonuna bir yapılandırma nesnesi geçiyoruz. Bu nesnenin **reducer** özelliği, kullanacağımız dilimleri (slices) içerir. Burada, **nav** anahtarı altında **navReducer**'ı belirtiyoruz.

Son olarak, **store** değişkenini **export** ediyoruz, böylece bu Redux store'u diğer bileşenlerden erişilebilir hale geliyor.

Bu kod, Redux Toolkit kullanarak basit bir Redux store'u yapılandırır. **navReducer** parçası, navigasyon durumunu yönetmek için kullanılır. Bu yapı, **configureStore** fonksiyonuyla oluşturulan Redux store'u, uygulama boyunca kullanılabilir hale getirir.

### **3.3 NAVSLICE.JS**

**navSlice.js** dosyası, Redux uygulamasında kullanılan bir Redux dilimi (slice)'ni tanımlar. **navSlice** dilimi genellikle navigasyon durumunu yönetmek için kullanılır.

```
Appjs M navSlice.js x
UberClone > .expo > slices > navSlice.js > navSlice > reducers > setDestination
1 import { createSlice } from "@reduxjs/toolkit";
2
3 const initialState = {
4   origin: null,
5   destination: null,
6   travelTimeInformation: null,
7 };
8
9 export const navSlice = createSlice({
10   name: 'nav',
11   initialState,
12   reducers: {
13     setOrigin: (state, action) => {
14       state.origin = action.payload;
15     },
16     setDestination: (state, action) => {
17       state.destination = action.payload;
18     },
19     setTravelTimeInformation: (state, action) => {
20       state.travelTimeInformation = action.payload;
21     },
22   },
23 });
24
25 export const { setOrigin, setDestination, setTravelTimeInformation } = navSlice.actions;
26
27 // Selectors
28 export const selectOrigin = (state) => state.nav.origin;
29 export const selectDestination = (state) => state.nav.destination;
30 export const selectTravelTimeInformation = (state) => state.nav.travelTimeInformation;
31
32 export default navSlice.reducer;
```

Şekil 3.3.1.

```
const initialState = {  
  origin: null,  
  destination: null,  
  travelTimeInformation: null,  
};
```

Şekil 3.3.2.

Yukarıdaki kod parçasında ‘**origin**’ özelliği seyahatin başlangıç noktasını temsil eder. Başlangıç noktası genellikle bir adres veya konum bilgisi olarak saklanır. Başlangıçta, **origin** özelliği **null** olarak ayarlanmıştır.

‘**destination**’ ise, seyahatin varış noktasını temsil eder. Varış noktası da genellikle bir adres veya konum bilgisi olarak saklanır. Başlangıçta, **destination** özelliği **null** olarak ayarlanmıştır.

‘**travelTimeInformation**’ ise, seyahat süresi ve bilgilerini temsil eder. Örneğin, yolculuğun tahmini süresi veya trafik durumu gibi bilgiler burada saklanabilir. Başlangıçta, **travelTimeInformation** özelliği de **null** olarak ayarlanmıştır.

İlk etapta, seyahat bilgileri ve konum bilgileri boş olarak başlatılır. Daha sonra, uygulama çalıştıkça veya kullanıcı etkileşimleri gerçekleştirdikçe bu başlangıç durumu güncellenebilir ve yönetilebilir.

```
export const navSlice = createSlice({
  name: 'nav',
  initialState,
  reducers: {
    setOrigin: (state, action) => {
      state.origin = action.payload;
    },
    setDestination: (state, action) => {
      state.destination = action.payload;
    },
    setTravelTimeInformation: (state, action) => {
      state.travelTimeInformation = action.payload;
    },
  },
});
```

Şekil 3.3.3.

Yukarıdaki kod bloğunda ise İlk olarak, Redux Toolkit'ten **createSlice** fonksiyonunu ve başlangıç durumu (**initialState**) nesnesini import ediyoruz.

**createSlice** fonksiyonuna bir yapılandırma nesnesi geçiyoruz. Bu nesnenin **name** özelliği, dilimin adını belirtir. Burada dilim adı "nav" olarak belirlenmiştir. **initialState** ise başlangıç durumu nesnesidir. **reducers** özelliği, dilime ait eylem yaratıcıları (action creators) ve işleyicilerini tanımlar.

**setOrigin**, **setDestination** ve **setTravelTimeInformation** adında üç farklı eylem yaratıcısı (action creator) ve işleyici (reducer) tanımlanmıştır. Her bir eylem yaratıcısı, ilgili özelliği (**origin**, **destination** veya **travelTimeInformation**) güncellemek için kullanılır. **action.payload**, eyleme geçirilen veriyi (payload) temsil eder.

Son adım olarak ise navSlice'ı import ediyorum. Bu sayede diğer bileşenler, **navSlice'a** ait eylem yaratıcılarını ve işleyicilerini kullanabilir. Yukarıdaki kod bloğu sayesinde navigasyon durumu yönetilebilir ve **'setOrigin'**, **'setDestination'** ve **'setTravelTimeInformation'** sürekli olarak güncellenebilir, sabit kalmaz. Kod bloğunda yer alan diğer değişkenler sayesinde ise navigasyon durumu sürekli olarak değişken bir yapıda kalabilir.



```
export const { setOrigin, setDestination, setTravelTimeInformation } = navSlice.actions;

// Selectors
export const selectOrigin = (state) => state.nav.origin;
export const selectDestination = (state) => state.nav.destination;
export const selectTravelTimeInformation = (state) => state.nav.travelTimeInformation;

export default navSlice.reducer;
```

Şekil 3.3.4.

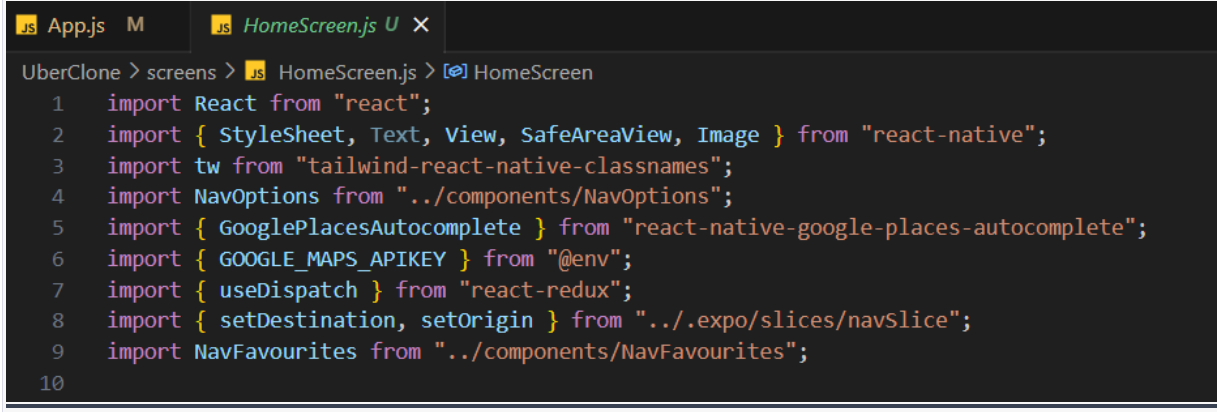
Şekil 3.3.4'te görülen kod bloğundaki satırlar **navSlice** dilimine ait seçicileri (selectors) export eder. Seçiciler, Redux store'unda tutulan veriye erişmek için kullanılır. **selectOrigin**, **selectDestination** ve **selectTravelTimeInformation** adında üç seçici tanımlanır. Bu seçicileri kullanarak, ilgili özelliklere erişebiliriz.

Son satır ise **navSlice** dilimine ait işleyiciyi (reducer) export eder. İşleyici, dilime ait eylemleri alır ve bu eylemlere göre dilimdeki durumu günceller. Bu sayede Redux store'u güncellenir.

Bu kod, **navSlice.js** dilimine ait eylem yaratıcılarını, seçicileri ve işleyiciyi dışarıya aktarır. Bu şekilde diğer bileşenler, bu dilime ait eylemleri tetikleyebilir, dilimdeki durumu okuyabilir ve güncelleyebilir.

### **3.4 HOMESCREEN.JS**

Homescreen, kullanıcının uygulamayı başlattığında karşılaştığı ana ekran veya giriş noktasıdır. Bu ekran, kullanıcının bir sürücü çağırması veya bir yolculuk talep etmesi gibi temel eylemleri gerçekleştirebileceği bir arayüz sunar. Homescreen, kullanıcının hızlı ve kullanıcı dostu bir şekilde yolculuk talep etmesini sağlar.



```
JS App.js M JS HomeScreen.js U X
UberClone > screens > JS HomeScreen.js > HomeScreen
1 import React from "react";
2 import { StyleSheet, Text, View, SafeAreaView, Image } from "react-native";
3 import tw from "tailwind-react-native-classnames";
4 import NavOptions from "../components/NavOptions";
5 import { GooglePlacesAutocomplete } from "react-native-google-places-autocomplete";
6 import { GOOGLE_MAPS_APIKEY } from "@env";
7 import { useDispatch } from "react-redux";
8 import { setDestination, setOrigin } from "../expo/slices/navSlice";
9 import NavFavourites from "../components/NavFavourites";
10
```

Şekil 3.4.1.

Şekil 3.4.1.'de ki kod bloğunda görüldüğü üzere gerekli kütüphanelerin import edilmesi işlemi yapılıyor. **StyleSheet** ve **SafeAreaView** gibi bileşenler, uygulama için stil ve güvenli alan sağlamak için kullanılır. **Image** bileşeni, resimleri görüntülemek için kullanılır. **tw** fonksiyonu, Tailwind CSS ile uyumlu stilleri uygulamak için kullanılır. **GooglePlacesAutocomplete** bileşeni, Google Haritalar (İlerleyen bölümlerde Google Cloud Hesabımı oluşturup Uydu üzerinden Dünya haritası kullandığım kısım) hizmetini kullanarak yer seçimini sağlar. **@env** paketi, ortam değişkenlerine erişim sağlar. **useDispatch** fonksiyonu, Redux eylemlerini tetiklemek için kullanılır. **'NavOptions'**, kullanıcının farklı seyahat seçeneklerini görmesini ve kendine uygunu seçmesini sağlar. **'GooglePlacesAutocomplete'** kullanıcının yer seçimini kolaylaştırır ve Google Haritalar API'sini kullanarak yer araması yapar. **GOOGLE\_MAPS\_APIKEY** değişkeni, Google Haritalar API anahtarını içerir. API anahtarı ise, GOOGLE CLOUD kısmında daha detaylı anlattım. **setDestination** ve **setOrigin** eylemleri, Redux'ta tanımlanan **navSlice** kısmından import ediliyor. Bu ifadeler, kullanıcının seçtiği hedefi ve başlangıç noktasını depolamak için kullanılır. **'NavFavourites'** ise, kullanıcının sık kullandığı konumları ve güzergahları görmesini ve seçmesini sağlar.

Bu kod, React Native UberClone uygulamasının ana ekranının bir kısmını oluşturur. Kullanıcılar, **NavOptions** bileşenindeki seyahat seçeneklerini görebilir, **GooglePlacesAutocomplete** bileşeni ile yer seçimi yapabilir ve **NavFavourites** bileşenindeki sık kullanılan konumları görüntüleyebilir. Redux kullanılarak, kullanıcının seçtiği hedef ve başlangıç noktası **setDestination** ve **setOrigin** eylemleriyle saklanır ve diğer bileşenlerde kullanılabilir.

```

App.js M HomeScreen.js U
UberClone > screens > HomeScreen.js > HomeScreen
11 const HomeScreen = () => {
12   const dispatch = useDispatch();
13
14   return (
15     <SafeAreaView style={tw`bg-white h-full`}>
16       <View style={tw`p-5`}>
17         <Image
18           style={{
19             width: 100,
20             height: 100,
21             resizeMode: 'contain',
22           }}
23           source={{
24             uri: "https://links.papareact.com/gzs",
25           }}
26         />
27         <GooglePlacesAutocomplete
28           placeholder="Where from?"
29           styles={{
30             container: {
31               flex: 0,
32             },
33             textInput: {
34               fontSize: 18,
35             },
36           }}
37           onPress={({data, details = null}) => {
38             dispatch(setOrigin({
39               location: details.geometry.location,
40               description: data.description
41             })))
42

```

Şekil 3.4.2.

Şekil 3.4.2. de ki kod bloğu uygulamanın ana ekranını oluşturur.

**useDispatch** fonksiyonunu kullanarak **dispatch** fonksiyonunu tanımlarız. Bu, Redux eylemlerini tetiklemek için kullanılır.

**return** ifadesiyle başlayan blok, bileşenin döndüreceği JSX (JavaScript XML) yapısını oluşturur.

**SafeAreaView** bileşeni, içindeki bileşenlerin cihaz ekranına tam oturmasını ve güvenli bir alanda görüntülenmesini sağlar.

**View** bileşeni, içindeki diğer bileşenleri gruplamak ve düzenlemek için kullanılır.

**Image** bileşeni, bir resmi görüntülemek için kullanılır. **style** özelliği ile resmin genişliği, yüksekliği ve yeniden boyutlandırma özellikleri ayarlanır. **source** özelliği ile resmin kaynağı belirtilir.

**GooglePlacesAutocomplete** bileşeni, kullanıcının yer seçimini kolaylaştırır ve Google Haritalar API'sini kullanarak yer araması yapar. **placeholder** özelliği, yer seçim alanına yer tutucusu metni ekler. **styles** özelliği ile bileşenin görünümünü özelleştiririz.

Ayrıca bu kod, **HomeScreen** bileşeninin görünümünü tanımlar. **SafeAreaView** içindeki **View** bileşeni, bir resim ve bir yer seçim alanı içerir. Resim, belirtilen URL'den alınır ve yer seçim alanı, kullanıcının başlangıç noktasını belirlemesine olanak tanır.

```
JS App.js M JS HomeScreen.js U X
UberClone > screens > JS HomeScreen.js > [0] HomeScreen
38 dispatch(setOrigin({
39   location: details.geometry.location,
40   description: data.description
41 })))
42
43 dispatch(setDestination(null))
44 }}
45 fetchDetails={true}
46 returnKeyType={"search"}
47 enablePoweredByContainer={false}
48 minLength={2}
49 query={{
50   key: GOOGLE_MAPS_APIKEY,
51   language: "en",
52 }}
53 nearbyPlacesAPI="GooglePlacesSearch"
54 debounce={400}
55 />
56
57 <NavOptions />
58 <NavFavourites />
59 </View>
60 </SafeAreaView>
61 );
62 };
63
64 export default HomeScreen
65
66 const styles = StyleSheet.create({
67   text: {
68     color: 'blue'
69   }
70 });
```

Şekil 3.4.3.

Yukarıda görülen kod parçası '**HomeScreen**' dosyasının JSX yapısının devamı niteliğindedir.

**onPress** özelliği, kullanıcının bir yer seçtiğinde tetiklenen işlevi belirtir. Bu işlev, Redux eylemlerini kullanarak kullanıcının seçtiği başlangıç noktasını **setOrigin** ile saklar ve hedefi **setDestination** ile sıfırlar. **fetchDetails** özelliği, seçilen yerin detaylarını almak için API çağrısının yapılmasını sağlar. **returnKeyType** özelliği, klavyede arama düğmesi görüntüler. **enablePoweredByContainer** özelliği, Google tarafından desteklenen bir göstergeyi etkinleştirir veya devre dışı bırakır. **minLength** özelliği, arama için minimum karakter uzunluğunu belirler. **query** özelliği, Google Haritalar API'sine yapılan sorgunun ayarlarını içerir. **nearbyPlacesAPI** özelliği, yakındaki yerleri aramak için kullanılan API'yi belirtir. **debounce** özelliği, kullanıcının yazarken yapılan istekler arasında gecikme sağlar. **NavOptions** bileşeni, seyahat seçeneklerini görüntülemek için kullanılır. Bu bileşen, kullanıcının farklı seyahat türlerini seçmesine olanak tanır. **NavFavourites** bileşeni, kullanıcının sık kullanılan konumlarını görüntülemek için kullanılır. Bu bileşen, kullanıcının önceden kaydettiği ve sıkça kullanılan konumları gösterir. **View** bileşeni, içindeki diğer bileşenleri gruplamak ve düzenlemek için kullanılır. **SafeAreaView** bileşeni, içindeki bileşenlerin cihaz ekranına tam oturmasını ve güvenli bir alanda görüntülenmesini sağlar. Son olarak, **HomeScreen** bileşeni **export default** ifadesiyle dışa aktarılır ve **StyleSheet.create** ile oluşturulan **styles** objesi, bileşenin stilini içerir. Bu kod, **HomeScreen** bileşeninin geri kalanını tamamlar ve JSX yapısını tamamlayarak ana ekranın tasarımını oluşturur. **GooglePlacesAutocomplete** bileşeni, yer seçimini sağlar, ardından **NavOptions** ve **NavFavourites** bileşenleri diğer özellikleri ve seçenekleri gösterir.

### **3.5 NAVOPTIONS.JS**

Şekil 3.5.1. de görülen kod bloğunda bir dizi öge içeren bir düz liste (**FlatList**) oluşturur ve her bir öge için bir kart görünümü oluşturur.

```
App.js M NavOptions.js U X
UberClone > components > NavOptions.js > NavOptions > <function>
1 import React from "react"
2 import { FlatList, Image, Text, TouchableOpacity, View } from "react-native"
3 import tw from "tailwind-react-native-classnames";
4 import { Icon } from "react-native-elements";
5 import { useNavigation } from "@react-navigation/native";
6 import { useSelector } from "react-redux";
7 import { selectOrigin } from "../expo/slices/navSlice";
8
9 const data = [
10   {
11     id: "123",
12     title: "Get a ride",
13     image: "https://links.papareact.com/3pn",
14     screen: "MapScreen",
15   },
16   {
17     id: "456",
18     title: "Order Food",
19     image: "https://links.papareact.com/28w",
20     screen: "EatScreen", // Change in future...
21   },
22 ];
23
```

Şekil 3.5.1.

İlk olarak, gerekli kütüphaneler ve bileşenler import edilir. **React** kütüphanesi, React bileşenlerini kullanmamızı sağlar. **FlatList**, bir dizi veri üzerinde kaydırılabilir bir liste oluşturmak için kullanılır. **Image**, resimleri görüntülemek için kullanılır. **Text**, metinleri görüntülemek için kullanılır. **TouchableOpacity**, dokunulabilir bir bileşen oluşturur. **View**, içindeki bileşenleri gruplamak ve düzenlemek için kullanılır. **tw** fonksiyonu, Tailwind CSS ile uyumlu stilleri uygulamak için kullanılır. **Icon** bileşeni, ikonları görüntülemek için kullanılır. **useNavigation** kancası, navigasyon işlemlerini gerçekleştirmek için kullanılır. **useSelector** kancası, Redux deposundan belirli bir durumu seçmek için kullanılır. **selectOrigin** fonksiyonu, **navSlice** kesitinden başlangıç noktasını seçmek için kullanılır.

**data** adında bir dizi tanımlanır. Bu dizi, her bir öğenin **id**, **title**, **image** ve **screen** özelliklerini içerir. Her bir öğe, bir kartın içeriğini temsil eder.

Bir **FlatList** bileşeni oluşturulur. Bu bileşen, **data** dizisini kullanarak bir dizi öğe oluşturur. Her bir öğe için bir kart görünümü oluşturulur.

Her bir kartın içeriği, bir **TouchableOpacity** bileşeni içinde yer alır. Bu, kullanıcının kartlara dokunarak etkileşimde bulunmasını sağlar.

Kartın içeriği, bir **View** bileşeniyle başlar. İçinde bir **Image** bileşeni, bir **Text** bileşeni ve bir **Icon** bileşeni bulunur. **Image** bileşeni, kartın üzerinde bir resim görüntüler. **Text** bileşeni, kartın başlığını görüntüler. **Icon** bileşeni, bir ok simgesi içerir ve kullanıcının işlem yapmak için kartı tıklatmasını teşvik eder.

Her bir kartın stilini **tw** fonksiyonu kullanarak belirliyoruz. Böylece Tailwind CSS'in sağladığı stilleri kullanabiliriz.

Özetle bu kod bloğu, **FlatList** kullanarak bir dizi öğeyi kartlar şeklinde görüntüler. Her kartın içeriği, bir resim, başlık ve bir işlem simgesi içerir. Bu bileşen, kullanıcının farklı ekranlara (harita ekranı, yemek siparişi ekranı vb.) geçmesini sağlar.

```
App.js M NavOptions.js U X
UberClone > components > NavOptions.js > NavOptions > <function>
24 const NavOptions = () => {
25   const navigation = useNavigation();
26   const origin = useSelector(selectOrigin);
27
28   return (
29     <FlatList
30       data={data}
31       horizontal
32       keyEx (alias) class TouchableOpacity
33       rende import TouchableOpacity
34       <TouchableOpacity
35         onPress={() => navigation.navigate(item.screen)}
36         style={tw`p-2 pl-6 pb-8 pt-4 bg-gray-200 m-2 w-40`}
37         disabled={!origin}
38       >
39         <View style={tw`${!origin} && "opacity-20"}`>
40           <Image
41             style={{ width: 120, height: 120, resizeMode: "contain" }}
42             source={{ uri: item.image }}
43           />
44           <Text style={tw`mt-2 text-lg font-semibold`}>{item.title}</Text>
45           <Icon
46             style={tw`p-2 bg-black rounded-full w-10 mt-4`}
47             name="arrowright"
48             color="white"
49             type='antdesign' />
50         </View>
51       </TouchableOpacity>
52     </FlatList>
53   );
54 };
55
56
57 export default NavOptions;
```

Şekil 3.5.2.

Şekil 3.5.2. ‘ de görüldüğü üzere ilk olarak gerekli kütüphaneler ve bileşenler import edilir. **useNavigation**, navigasyon işlemlerini gerçekleştirmek için kullanılır. **useSelector**, Redux deposundan belirli bir durumu seçmek için kullanılır. **selectOrigin** fonksiyonu, **navSlice** kısmından başlangıç noktasını seçmek için kullanılır. **data** adında bir dizi tanımladım. Bu dizi, seçeneklerin bilgilerini içerir. Her bir seçenek, bir **id**, **title**, **image** ve **screen** özelliğine sahiptir. Bu bilgiler, seçeneklerin kart görünümünde kullanılacak olan şeyleri döndürür.

**NavOptions** bileşeni, bir **FlatList** bileşeni içinde yer alır. **FlatList**, **data** dizisi üzerinde bir yatay (horizontal) liste oluşturur. Her bir öğe için bir kart görünümü oluşturulur. ‘**Flatlist**’ deki özellikleri şöyle açıklayabilirim. ‘**data**’ gösterilecek veriyi, ‘**horizontal**’ oluşturulacak yatay listeyi, ‘**keyExtractor**’ ise her bir öğe için ona özgü anahtarı belirlemede kullanılır. ‘**renderItem**’ ise her öğe için bir kart görünümü oluşturan kısımdır.

Her bir kartın içeriği, bir **TouchableOpacity** bileşeni içinde yer alır. Bu, kullanıcının kartlara dokunarak etkileşimde bulunmasını sağlar.

Kartın içeriği, bir **View** bileşeni ile başlar. İçinde bir **Image** bileşeni, bir **Text** bileşeni ve bir **Icon** bileşeni bulunur. **Image** bileşeni, kartın üzerinde bir resmi görüntüler. **Text** bileşeni, kartın başlığını görüntüler. **Icon** bileşeni, bir ok simgesi içerir ve kullanıcının işlem yapmak için kartı tıklatmasını teşvik eder.

Kartın stili, **tw** fonksiyonunu kullanarak belirlenir. Kartın boyutu, arka plan rengi ve diğer özellikleri **style** özelliği ile belirlenir. Ayrıca, kartın etkinlik durumuna (**disabled**) göre stilini değiştirmek için **!origin** koşulu kullanılır.

Blok, **return** ifadesiyle tamamlanır ve **NavOptions** bileşeni **export default** ifadesiyle dışa aktarılır.

Son olarak bu kod, **NavOptions** bileşenini oluşturur ve **FlatList** kullanarak bir dizi öğeyi kartlar şeklinde görüntüler. Her bir kartın içeriği, bir resim, başlık ve bir ok simgesi içerir. Kullanıcı kartlara dokunarak ilgili ekrana geçebilir.



### **3.6 GOOGLE CLOUD AUTOCOMPLETE**

Google Cloud Autocomplete, Google Cloud Platform'un bir hizmetidir ve geliştiricilere otomatik tamamlama (autocomplete) özelliği sağlar. Bu hizmet, kullanıcıların metin girişi yaparken gerçek zamanlı öneriler sunmak için kullanılır. Özellikle adres veya yer adı gibi metin tabanlı verilerle çalışırken kullanışlıdır.

Google Cloud Autocomplete, Google Haritalar'ın veri tabanına dayanarak metin girişi yapılırken kullanıcıya tamamlama önerileri sağlar. Kullanıcılar, metin girişi yaparken gerçek zamanlı olarak potansiyel tamamlama seçenekleri görebilirler. Örneğin, bir kullanıcı bir adrese veya yer adına başladığında, Google Cloud Autocomplete hizmeti, giriş tamamlanırken otomatik olarak ilgili adresleri veya yerleri önerir. Bu, kullanıcı deneyimini iyileştirir, giriş hatalarını azaltır ve veri girişini hızlandırır.

Google Cloud Autocomplete hizmeti, uygulama geliştiricilerinin metin girişi alanlarına otomatik tamamlama özelliği eklemelerine olanak tanır. Hizmet, bir API aracılığıyla erişilebilir ve geliştiricilerin istemci tarafı uygulamalarında kullanabilmesi için uygun bir programlama arabirimi sunar.

Google Cloud Autocomplete, yerel arama sonuçları, adresler, işletme adları, yer adları, konumlar ve daha fazlası gibi çeşitli metin tabanlı veri türlerini destekler. Ayrıca, dil desteği ve coğrafi konum bilgilerine dayalı özelleştirme seçenekleri sunar.

Bu hizmet, özellikle lokasyon tabanlı uygulamalar, haritalar, seyahat rezervasyonları, e-ticaret platformları ve daha birçok alanda kullanılabilir. Kullanıcıların hızlı ve doğru metin girişi yapmalarını sağlamak için Google Cloud Autocomplete hizmeti kullanılabilir.

Ayrıca Google Cloud üzerinde çeşitli API destekleriyle uygulama için Uydu haritası ve gerçek zamanlı haritalar indirdim.

API	Requests	Errors	Avg latency (ms)
<a href="#">Directions API</a>	103	16	172
<a href="#">Distance Matrix API</a>	79	-	95
<a href="#">Places API</a>	400	1	49

Şekil 3.6.1.

Şekil 3.6.1. de de görüldüğü üzere, uygulama için gerekli olan navigasyon desteğini Google Cloudun sağlamış olduğu 90 günlük deneme sürümü üzerinden **‘Directions API’** , **‘Distance Matrix API’** ve **‘Places API’** isimli API’lerden sağladım. Bu API’lerin işlevlerine gelecek olursak Distance Matrix API, Google Haritalar’ın bir hizmetidir ve iki veya daha fazla konum arasındaki mesafeleri ve seyahat sürelerini hesaplamak için kullanılır. Bu API, yerel veya küresel düzeyde yolculuk sürelerini, mesafeleri ve tahmini trafik bilgilerini sağlar.

Distance Matrix API, yolculuk planlaması, trafik analizi, rota optimizasyonu ve yerleşim planlaması gibi senaryolarda kullanışlıdır. API, farklı ulaşım modlarına (yürüme, bisiklet, araç veya toplu taşıma) ve trafik durumuna bağlı olarak tahmini seyahat sürelerini döndürebilir.

Places API, Google Haritalar’ın bir parçasıdır ve yerel işletmeler, ilgi noktaları ve coğrafi yerler hakkında bilgi almak için kullanılır. Bu API, yer adları, adresler, koordinatlar ve diğer parametreler aracılığıyla yerler hakkında ayrıntılı bilgileri sağlar. Places API, konum tabanlı uygulamalar, restoran rezervasyonları, seyahat planlaması ve diğer yer tabanlı senaryolarda yaygın olarak kullanılır.

Ayrıca Places API farklı hizmetlerde sunar. Bunlar şöyle açıklanabilir:

**Place Search:** Belirli bir konuma veya kategoriye dayalı olarak yerleri aramak için kullanılır. Örneğin, restoranlar, oteller, mağazalar gibi yerlerin listesini alabilirsiniz. Sonuçlar, yer adları, adresler, koordinatlar, değerlendirmeler, fotoğraflar ve diğer ayrıntılar gibi çeşitli bilgiler içerir.

**Place Details:** Bir yer hakkında ayrıntılı bilgi almak için kullanılır. Yer adı, yer kimliği veya koordinatlar gibi bir tanımlayıcı kullanılarak belirli bir yerin ayrıntılarını alabilirsiniz. Bu ayrıntılar, adres, açık saatler, telefon numarası, web sitesi, fotoğraflar, değerlendirmeler ve diğer bilgileri içerebilir.

**Place Photos:** Bir yerin fotoğraflarını almak için kullanılır. Yer kimliği kullanılarak fotoğrafların bir koleksiyonunu alabilir ve bu fotoğrafları görüntüleyebilirsiniz.

**Directions API,** Google Haritalar'ın bir hizmetidir ve rota bilgilerini almak ve yol tarifleri oluşturmak için kullanılır. Bu API, iki veya daha fazla nokta arasındaki en kısa rota, seyahat süresi, yönlendirmeler ve diğer ilgili bilgileri sağlar. Directions API, yolculuk planlaması, navigasyon, lojistik ve taşımacılık gibi senaryolarda yaygın olarak kullanılır.

Directions API'de de kendine has özellikler mevcuttur. Bunları şu şekilde sıralayabiliriz:

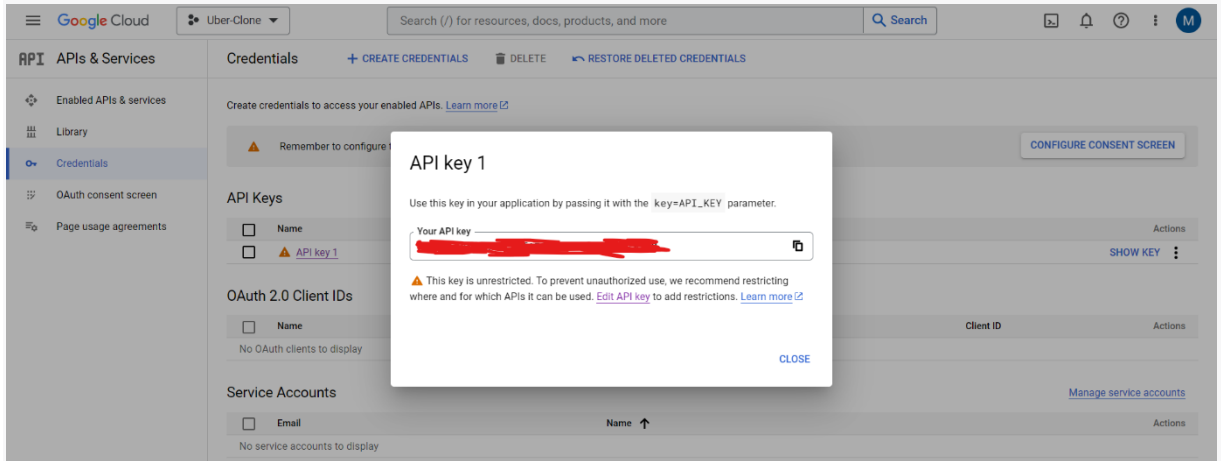
**Route Calculation:** İki veya daha fazla nokta arasındaki en kısa rota bilgisini almak için kullanılır. Başlangıç ve varış noktaları, ara noktalar ve yolculuk modu gibi parametreler belirtilir. API, rota bilgilerini, seyahat süresini, mesafeyi, yönlendirmeleri ve diğer ayrıntıları içeren bir yanıt döndürür.

**Route Alternatives:** Birden fazla rota seçeneği sunan alternatif rota bilgilerini almak için kullanılır. API, farklı rota seçeneklerini ve bu seçeneklerin seyahat süresini, mesafeyi ve yönlendirmeleri içeren bir yanıt döndürür.

Traffic Information: Trafik durumuna göre rota bilgilerini almak için kullanılır. API, mevcut trafik bilgilerine dayalı olarak tahmini seyahat süresini, trafik durumunu ve alternatif rotaları içeren bir yanıt döndürür.

Directions API, isteklerde başlangıç ve varış noktaları, ulaşım modu, trafik bilgisi, dil ve diğer parametrelerin belirtilmesini gerektirir. İstekler HTTP üzerinden yapılır ve API yanıtları JSON veya XML formatında alınır.

Bu hizmetler, Google Haritalar platformunun güçlü özelliklerini kullanarak, yer tabanlı uygulamaların ve hizmetlerin daha zengin ve işlevsel olmasını sağlar. Geliştiriciler, bu API'ları kullanarak konum tabanlı verileri işleyebilir, kullanıcı deneyimini iyileştirebilir ve farklı senaryolarda etkili çözümler sunabilir.



Şekil 3.6.2.

Şekil 3.6.2. de görülen Google Cloud API Key, Google Cloud Platform (GCP) hizmetlerine erişmek ve kimlik doğrulaması için kullanılan bir anahtardır. Bu anahtar, uygulamaların Google Cloud API'lerini kullanabilmesi ve yetkilendirme süreçlerini tamamlaması için gereklidir.

Google Cloud API Key'i kullanmanın iki temel işlevi vardır:

**Kimlik Doğrulama:** Google Cloud API Key, uygulamaların Google Cloud API'lerine erişim yetkisi kazanmasını sağlar. API Key, uygulamanın kimliğini doğrular ve kimlik doğrulama sürecini tamamlar. Bu sayede, uygulama istekleri API'lerin tarafından tanınır ve yetkilendirilir.

**API Kullanımı:** Google Cloud API Key, uygulamanın Google Cloud API'lerine erişmesine olanak tanır. API Key, API isteklerinde kullanılarak API'ye erişim sağlar ve uygulamanın hizmetleri kullanabilmesini sağlar. Her bir Google Cloud API Key, belirli bir GCP projesine ve API'ye erişimi temsil eder. Bu sayede, API Key'in kontrolü altında hangi hizmetlere erişileceği ve ne tür işlemlerin gerçekleştirileceği belirlenebilir.

Google Cloud API Key'i oluştururken, API Key ayarları belirlenebilir. Bu ayarlar arasında, API Key'in erişim kontrolü, kısıtlamalar, güvenlik önlemleri ve hız sınırları gibi konular yer alır. Bu ayarlar, API Key'in nasıl kullanılacağını ve hangi hizmetlere erişileceğini belirler.

Google Cloud API Key'inin gizli tutulması önemlidir, çünkü bu anahtar, yetkisiz kişilerin API'leri kötüye kullanmasını veya uygulama kaynaklarını etkilemesini engeller. API Key'in güvenli bir şekilde saklanması ve gereksiz paylaşımından kaçınılması önemlidir. Bu yüzden de bu kısımdaki görseli karalayarak dosyaya koymak durumundayım.

Google Cloud API Key, Google Cloud Platform'daki birçok hizmet ve API için kullanılabilir. Örneğin, Google Maps API, Google Cloud Translation API, Google Cloud Vision API gibi API'ler, uygulamaların API Key kullanarak erişebileceği hizmetler arasındadır.

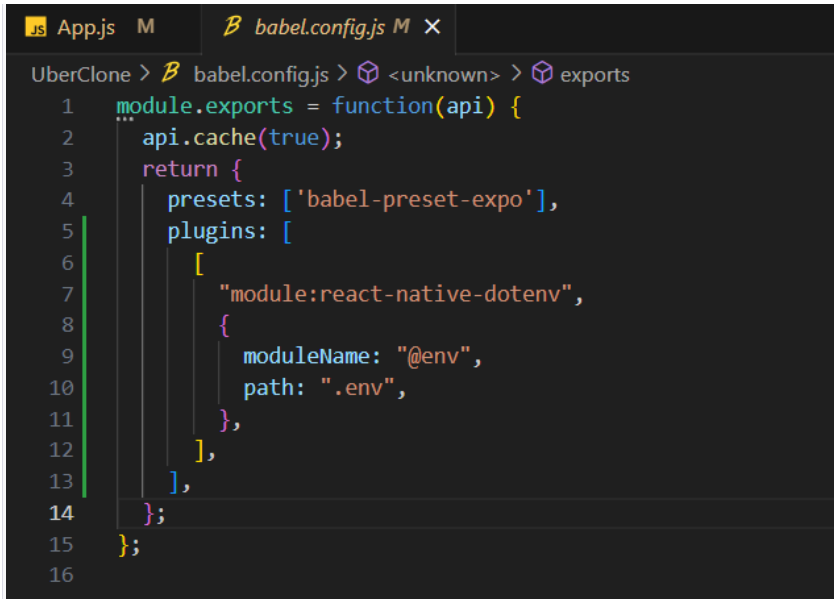
### **3.7 BABELCONFIG.JS VE .ENV**

React Native projelerinde Babel Config ve ENV (Environment) dosyaları, uygulama geliştirmesi sırasında çeşitli yapılandırma ve ortam değişkenlerini ayarlamak için kullanılan dosyalardır.

Babel Config: Babel, JavaScript kodunu farklı tarayıcılarda ve ortamlarda çalışacak şekilde dönüştüren bir transpilerdir. Babel Config dosyası, Babel transpilerinin nasıl yapılandırılacağını belirler. React Native projelerinde **.babelrc** veya **babel.config.js** dosyası olarak adlandırılır.

Babel Config dosyası, Babel eklentilerini, dönüşümleri ve yapılandırmaları içeren bir JavaScript nesnesi içerir. Bu dosya, JavaScript dil özelliklerinin belirli bir tarayıcıda veya React Native platformunda nasıl destekleneceğini belirlemek için kullanılır. Örneğin, modern JavaScript özelliklerini eski tarayıcılarda çalışacak şekilde dönüştürmek veya React Native özelliklerini uyumlu hale getirmek için Babel Config dosyası kullanılabilir.

Babel Config dosyası, Babel eklentilerini ve yapılandırmalarını özelleştirme imkanı sağlar. Böylece geliştiriciler, projelerine özgü ihtiyaçlarına göre Babel'in dönüştürme sürecini yapılandırabilirler.



```
1 module.exports = function(api) {  
2   api.cache(true);  
3   return {  
4     presets: ['babel-preset-expo'],  
5     plugins: [  
6       [  
7         "module:react-native-dotenv",  
8         {  
9           moduleName: "@env",  
10          path: ".env",  
11        }],  
12      ],  
13    ],  
14  };  
15 };  
16
```

Şekil 3.7.1.

ENV (Environment) dosyası: ENV dosyası, uygulama ortamında kullanılan değişkenlerin ve yapılandırmaların depolandığı bir dosyadır. React Native projelerinde **.env** veya **.env.\*** (örneğin **.env.development**, **.env.production**) olarak adlandırılır. ENV dosyası, uygulama için ortama özgü yapılandırmaları, API anahtarlarını, veritabanı bağlantı bilgilerini ve diğer gizli bilgileri içerebilir.

ENV dosyası, projenin farklı ortamlarda (geliştirme, üretim, test vb.) çalışabilmesini sağlar. Her bir ortam için ayrı bir ENV dosyası oluşturulabilir ve bu dosyalarda ilgili ortama özgü değişkenler tanımlanabilir. ENV dosyası, uygulama kodunda bu değişkenlere erişmek için kullanılır.

ENV dosyasındaki değişkenler, projenin her yerinde **process.env** nesnesi aracılığıyla erişilebilir hale gelir. Bu değişkenler, genellikle API anahtarları, hedef URL'ler, etkinleştirme bayrakları ve diğer yapılandırmaları içerir. ENV dosyası, projenin farklı ortamlarda çalışabilmesini ve yapılandırmaların kolayca değiştirilebilmesini sağlayarak uygulama geliştirmesini daha esnek hale getirir.

```
UberClone > .env
1  GOOGLE_MAPS_APIKEY = [REDACTED]
2
3
```

Şekil 3.7.2.

Şekil 3.7.2. de görüldüğü üzere Google Cloud tarafından şahsıma verilen bana özgü Google API KEY im görölüyor, daha önce de belirttiğim gibi bu kişisel ve özel bir bilgi olduğu için bu kısmı karalamak zorunda kaldım. Bu API KEY ile Google Cloud üzerinden çeşitli API'leri uygulamama entegre edebilme şansını yakalıyorum.

Babel Config ve ENV dosyaları, React Native projelerinde yapılandırmayı kolaylaştıran ve farklı ortamlarda çalışmayı mümkün kılan önemli araçlardır. Bu dosyaları doğru bir şekilde yapılandırmak, uygulamanın gereksinimlerine uygun olarak çalışmasını sağlar ve geliştirme sürecini daha esnek hale getirir.

### **3.8 MAPSCREEN.JS**

```
JS App.js M JS MapScreen.js U X
UberClone > screens > JS MapScreen.js > ...
1  import React from "react";
2  import { StyleSheet, Text, View } from "react-native";
3  import tw from "tailwind-react-native-classnames";
4  import Map from "../components/Map";
5  import MapView from 'react-native-maps';
6  import { createStackNavigator } from "@react-navigation/stack";
7  import NavigateCard from "../components/NavigateCard";
8  import RideOptionsCard from "../components/RideOptionsCard";
9  import { TouchableOpacity } from "react-native-gesture-handler";
10 import { useNavigation } from "@react-navigation/native";
11 |
```

Şekil 3.8.1.



Şekil 3.8.1. de görülen kod bloğunda aynı diğer dosyalarda olduğu gibi öncelikle import etme (içe aktarma) işlemlerini gerçekleştirdim çünkü bu tool'ları içe aktarmazsam uygulama düzgün çalışmayacak ve sürekli hata verecektir.

Görüldüğü üzere ilk 3 satır React ve React Native kütüphanelerini içe aktarır. Ayrıca, stil işlemleri için **tailwind-react-native-classnames** kütüphanesi de içe aktarılır.

4 ve 5. Satırlarda ise **Map** bileşeni, uygulamanın harita görüntülemesiyle ilgili işlevleri sağlar. **react-native-maps** kütüphanesinden gelen **MapView** bileşeni ise haritanın görüntülenmesi ve etkileşimli özelliklerin sağlanması için kullanılır.

Ardından gelen 3 satırda ise **createStackNavigator**, StackNavigator'ın kullanılmasını sağlayan bir fonksiyondur. **TouchableOpacity**, dokunma etkinlikleri için kullanılan bir bileşendir. **useNavigation**, bir bileşen içinde navigasyon işlevlerini kullanmak için kullanılmıştır.

```
App.js  M  MapScreen.js  U  X
UberClone > screens > MapScreen.js > ...
12  const MapScreen = () => {
13    const Stack = createStackNavigator();
14
15
16    return (
17      <View>
18        <View style={tw`h-1/2`}>
19          <Map />
20        </View>
21
22        <View style={tw`h-1/2`}>
23          <Stack.Navigator>
24            <Stack.Screen
25              name="NavigateCard"
26              component={NavigateCard}
27              options={{
28                headerShown: false,
29              }}
30            />
31            <Stack.Screen
32              name="RideOptionsCard"
33              component={RideOptionsCard}
34              options={{
35                headerShown: false,
36              }}
37            />
38          </Stack.Navigator>
39        </View>
40      </View>
41    );
42  };
43
44  export default MapScreen;
```

Şekil 3.8.2.

```
46 const styles = StyleSheet.create({});
```

Şekil 3.8.3

Şekil 3.8.2. ve Şekil 3.8.3. te görülen kod parçacıkları birbirinin devamı niteliğindedir. Kod parçacıklarının tamamı **'MapScreen.js'** 'i oluşturur.

Üst bölüm: Harita görüntülemesini içeren Map bileşeni, yarım ekran boyutunda yer alır.

Alt bölüm: StackNavigator kullanarak NavigateCard ve RideOptionsCard bileşenlerini içeren bir gezinme kartı sağlar. Bu kartlar, kullanıcının yol tarifi ve sürüş seçeneklerini görüntülemesine olanak tanır. Alt bölüm de yarım ekran boyutunda yer alır.

**Stack.Navigator** ve **Stack.Screen** kullanarak NavigateCard ve RideOptionsCard bileşenlerini birbirine bağlar ve gezinme işlemlerini sağlar. **headerShown** seçeneği, başlık çubuğunun görüntülenmesini kontrol eder ve burada **false** olarak ayarlanarak başlık çubuğunun gizlenmesi sağlanır.

Şekil 3.8.2. de 44. Satırda görülen 'export default MapScreen;' ifadesi sayesinde uygulama dışa aktarılır ve bu sayede başka yerlerde ve cihazlarda kullanılabilir hale gelir.

Şekil 3.8.3. te görülen kod parçacığı ise stil tanımlamaları içermektedir. Ben boş bıraktım fakat istenilen tarzda kullanıcıya göre doldurulabilir.

Sonuc olarak MapScreen, harita görüntülemesi, gezinme kartları ve ilgili bileşenlerin düzenlenmesini sağlar.

### **3.9 MAP.JS**

**Map.js** bir React Native bileşenidir ve harita görüntülemesi için kullanılır. Bu bileşen, uygulamanızda haritalarla etkileşimli özellikler sunmanızı sağlar.

**Map.js** bileşeni genellikle aşağıda belirttiğim özelliklere sahiptir:

**Harita Görüntülemesi:** **Map.js** bileşeni, bir harita görüntülemesini kullanıcıya sunar. Bu görüntüleme, gerçek zamanlı bir harita, uydu görüntüleri veya başka bir harita türü olabilir. Kullanıcılar haritayı yakınlaştırabilir, uzaklaştırabilir, sürükleyebilir veya döndürebilir.

**Konum Gösterme:** Bileşen, kullanıcının mevcut konumunu veya belirli bir konumu gösterebilir. Bu, kullanıcıların kendilerini harita üzerinde takip etmelerine veya belirli bir konumu seçmelerine olanak tanır.

**Markörler ve Etiketler:** Harita üzerinde markörler ve etiketler oluşturabilirsiniz. Bu, belirli konumları veya ilgi noktalarını göstermek için kullanışlıdır. Markörler ve etiketler, dokunulabilir olabilir ve tıklanabilir olduklarında belirli bir eylemi tetikleyebilir.

**Yol Tarifi:** Bileşen, kullanıcılara belirli bir başlangıç ve varış noktası arasındaki yol tarifini gösterebilir. Bu, kullanıcıların yolculuklarını planlamalarına ve rotalarını takip etmelerine yardımcı olur.

**Etkileşimli Özellikler:** Bileşen, haritayı etkileşimli hale getirmek için çeşitli özellikler sağlayabilir. Bu, kullanıcıların harita üzerinde çizim yapmalarını, alanları vurgulamalarını veya belirli bir alana yakınlaşmalarını sağlayabilir.

**Map.js** bileşeni, uygulamanın gereksinimlerine göre özelleştirilebiliriz ve stilize edilebiliriz. Harita öğelerinin görünümü, harita türü, renkler ve etkileşimli özellikler, uygulama tasarımı ve kullanıcı deneyimi hedeflerine uygun olarak ayarlanabilir.

**Map.js**, genellikle diğer bileşenlerle birlikte kullanılarak, uygulamanızın harita tabanlı özelliklerini sağlamak için entegre edilir. Örneğin, yol tarifi veya yer arama gibi işlevler, harita bileşeniyle birlikte kullanılarak daha kapsamlı bir deneyim sunabilir.

```
Appjs M Mapjs U X
UberClone > components > Mapjs > Map > useEffect() callback > getTravelTime > then() callback
1 import React, { useEffect } from "react";
2 import { StyleSheet, Text, View } from "react-native";
3 import MapView, { Marker } from "react-native-maps";
4 import { useDispatch, useSelector } from "react-redux";
5 import tw from "tailwind-react-native-classnames";
6 import { selectDestination, selectOrigin, setTravelTimeInformation } from "../expo/slices/navSlice";
7 import MapViewDirections from "react-native-maps-directions";
8 import { GOOGLE_MAPS_APIKEY } from "@env";
9 import { useRef } from "react";
10
```

Şekil 3.9.1.

Şekil 3.9.1. de görüldüğü üzere önceki bölümlerde anlattığım gibi çeşitli kütüphaneleri import (içe aktarma) işlemini uyduladım. Burada önemli olan kısım ‘GOOGLE\_MAPS\_APIKEY’ bileşenini de import etmem. Bu anahtar, Google Cloud un bana vermiş olduğu kişisel API KEY’idir. Bu keyi .env dosyası içinde GOOGLE\_MAPS\_APIKEY ifadesine atayarak kişiselleştirdim. Diğer bileşenlerin hepsini önceki bölümlerde import ettiğim için bu bölümde tekrar üzerinde durmayacağım.

```
Appjs M Mapjs U X
UberClone > components > Mapjs > Map > useEffect() callback > getTravelTime > then() callback
11
12 const Map = () => {
13   const origin = useSelector(selectOrigin);
14   const destination = useSelector(selectDestination);
15   const mapRef = useRef(null);
16   const dispatch = useDispatch();
17
18   useEffect(() => {
19     if (!origin || !destination) return;
20
21     // Zoom & fit to markers
22     mapRef.current.fitToSuppliedMarkers(['origin', 'destination'], {
23       edgePadding: { top: 50, right: 50, bottom: 50, left: 50 }
24     });
25   }, [origin, destination]);
26
27   useEffect(() => {
28     if (!origin || !destination) return;
29
30     const getTravelTime = async() => {
31       fetch(`https://maps.googleapis.com/maps/api/distancematrix/json?
32         units=imperial&origins=${origin.description}&destinations=${destination.description}&key=${GOOGLE_MAPS_APIKEY}`
33       )
34       .then(res => res.json())
35       .then(data => {
36         dispatch(setTravelTimeInformation(data.rows[0].elements[0]));
37       });
38     };
39
40     getTravelTime();
41   }, [origin, destination, GOOGLE_MAPS_APIKEY]);
42 }
```

Şekil 3.9.2.

Şekil 3.9.2. de görüldüğü üzere kod bloğunun açıklaması şu şekildedir:

**origin**: Redux store'dan seçilen başlangıç noktası bilgisini içerir.

**destination**: Redux store'dan seçilen varış noktası bilgisini içerir.

**mapRef**: Haritanın referansını tutmak için kullanılan bir referans nesnesidir.

**dispatch**: Redux store'a eylem göndermek için kullanılır.

İki **useEffect** bloğu kullanılır ve işlevleri aşağıdaki gibidir:

İlk **useEffect** bloğu, **origin** veya **destination** değiştiğinde tetiklenir. Eğer **origin** veya **destination** bilgisi yoksa (**null** veya **undefined**), bu etkileşimliğin gerçekleşmesini önler. Aksi takdirde, **mapRef** üzerindeki **fitToSuppliedMarkers** işlevi kullanılarak haritanın yakınlaştırılması ve işaretçilerin sığdırılması sağlanır. **Origin** ve **destination** işaretçileri, haritada **origin** ve **destination** olarak tanımlanan yerlerin konumunu belirten işaretçilerdir.

İkinci **useEffect** bloğunda ise, **origin** veya **destination** değiştiğinde ve **GOOGLE\_MAPS\_APIKEY** değişkeni kullanılabilir olduğunda tetiklenir. Yine, **origin** veya **destination** bilgisi eksikse işlemler gerçekleştirilmez. Aksi takdirde, **fetch** işlevi kullanılarak Google Distance Matrix API'sine bir istek yapılır. Bu istek, **origin** ve **destination** konumları arasındaki mesafe ve süre bilgilerini döndürür. Sonuçlar, **setTravelTimeInformation** eylemini tetikleyerek Redux store'a gönderilir. Bu bilgiler daha sonra uygulamada yolculuk süresini göstermek için kullanılabilir.

Bu kod parçası, geri kalan JSX kodu oluşturulmadan önce gerekli hesaplamaları yapar ve gerekli verileri Redux store'a gönderir.

```
App.js M Map.js U X
UberClone > components > Map.js > Map
43 return (
44   <MapView
45     ref={mapRef}
46     style={tw`flex-1`}
47     mapType="mutedStandard"
48     initialRegion={{
49       latitude: origin.location?.lat,
50       longitude: origin.location?.lng,
51       latitudeDelta: 0.005,
52       longitudeDelta: 0.005,
53     }}
54   >
55     {origin && destination && (
56       <MapViewDirections
57         origin={origin.description}
58         destination={destination.description}
59         apiKey={GOOGLE_MAPS_APIKEY}
60         strokeWidth={3}
61         strokeColor="black"
62       />
63     )}
64
65     {origin?.location && (
66       <Marker
67         coordinate={{
68           latitude: origin.location?.lat,
69           longitude: origin.location?.lng,
70         }}
71         title="Origin"
72         description={origin.description}
73         identifier="origin"
74       />
75     )}
76   </MapView>
77 )
```

Şekil 3.9.3.

Yukarıdaki kod bloğunda görüldüğü üzere bu kod parçası JSX içindeki **<MapView>** kısmını oluşturur ve harita üzerindeki işaretçileri ve yol tarifini gösterir.

Bu kodun detaylı açıklamasına gelecek olursak:

**ref={mapRef}**: Haritaya referans vermek için **mapRef** kullanılır.

**style={tw`flex-1`}**: Harita bileşenine tam ekran genişliği ve yüksekliği vermek için kullanılır.

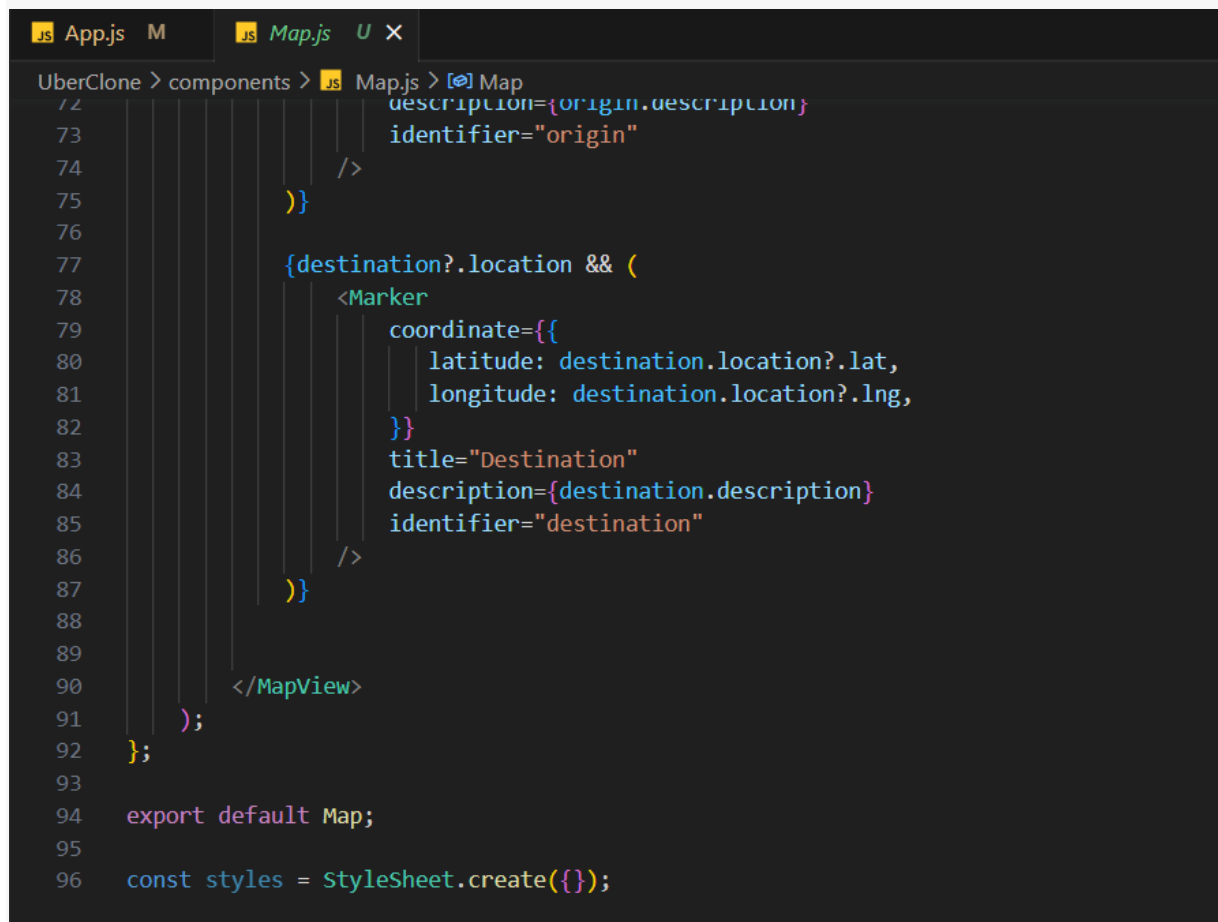
**mapType="mutedStandard"**: Harita türünü belirlemek için kullanılır. **mutedStandard**, haritanın hafif renkli bir görünüme sahip olmasını sağlar.

**initialRegion**: Haritanın başlangıç bölgesini belirler. Bu, **origin** konumunun latitud ve longitud koordinatlarına dayanır. **latitudeDelta** ve **longitudeDelta**, haritanın yakınlaştırma düzeyini kontrol eder.

**<MapViewDirections>**: **origin** ve **destination** konumlarını temel alarak harita üzerinde yol tarifi gösterir. **apikey** özelliği, Google Maps API anahtarını belirtir. **strokeWidth** ve **strokeColor**, yol tarifi çizgi genişliğini ve rengini belirler.

**<Marker>**: **origin** konumunu göstermek için bir işaretçi oluşturur. **coordinate** özelliği, işaretçinin latitud ve longitud koordinatlarını belirler. **title** ve **description**, işaretçinin başlık ve açıklamasını belirler. **identifier**, işaretçiyi tanımlayan bir kimlik olarak kullanılır.

Kısaca özetleyecek olursak Şekil 3.9.3.'deki kod bloğu **origin** ve **destination** bilgilerine bağlı olarak haritayı oluşturur ve işaretçileri ve yol tarifi gösterir.



```
UberClone > components > Map.js > Map
72      description={origin.description}
73      identifier="origin"
74    />
75  )}
76
77  {destination?.location && (
78    <Marker
79      coordinate={{
80        latitude: destination.location?.lat,
81        longitude: destination.location?.lng,
82      }}
83      title="Destination"
84      description={destination.description}
85      identifier="destination"
86    />
87  )}
88
89
90  </MapView>
91 );
92 };
93
94 export default Map;
95
96 const styles = StyleSheet.create({});
```

Şekil 3.9.4

Son olarak **'Map.js'** dosyasının tamamlayıcı kod bloğu olan Şekil 3.9.4.'de görüldüğü üzere bu blok, **destination** konumu mevcutsa haritaya bir hedef (destination) işaretçisi ekler.

**coordinate**: İşaretçinin latitud ve longitud koordinatlarını belirler. Bu, **destination** konumunun latitud ve longitud koordinatlarına dayanır.

**title**: İşaretçiye başlık (title) verir. Bu durumda, başlık "Destination" olarak ayarlanır.

**description**: İşaretçiye açıklama (description) verir. Bu, **destination** konumunun açıklamasına dayanır.

**identifier**: İşaretçiyi tanımlayan bir kimlik (identifier) olarak kullanılır. Bu durumda, işaretçi "destination" olarak tanımlanır.

**'export default Map;'** kısmı, map bileşenini dışa aktarma görevi üstlenir ve böylelikle başka bileşenlerde de kullanılabilir hale gelir.

**'const styles = StyleSheet.create({});'** bloğu ise, boş bir stiller nesnesi oluşturur.

### **3.10 RIDEOPTIONSCARD.JS**

**RideOptionsCard.js**, yolculuk seçeneklerini gösteren bir kısımdır. Bu kısım, kullanıcının farklı yolculuk seçenekleri arasında seçim yapmasını sağlar.



```
App.js M RideOptionsCard.js U X
UberClone > components > RideOptionsCard.js > data
1 import React, { useState } from "react";
2 import { StyleSheet, Text, View } from "react-native";
3 import { FlatList, TouchableOpacity } from "react-native-gesture-handler";
4 import { SafeAreaView } from "react-native-safe-area-context";
5 import tw from "tailwind-react-native-classnames";
6 import { Icon } from "react-native-elements";
7 import { useNavigation } from "@react-navigation/native";
8 import { Image } from "react-native-elements/dist/image/Image";
9 import { useSelector } from "react-redux";
10 import { selectTravelTimeInformation } from "../expo/slices/navSlice";
11
```

Şekil 3.10.1.

Yukarıdaki kod bloğunda daha önceden anlatmış olduğum gibi bu dosya için gerekli olan kütüphaneleri ve tool'ları import etme işleminin gerçekleştiği kısımdır. Bu yüzden bu kısımda onlara tekrar değinmeyeceğim. Eğer bu bileşenleri içe aktarma işlemini yapmaz isek uygulamada hatalar meydana gelecektir.

```
App.js M RideOptionsCard.js U X
UberClone > components > RideOptionsCard.js > data
11
12 const data = [
13   {
14     id: "Uber-X-123",
15     title: "UberX",
16     multiplier: 1,
17     image: "https://links.papareact.com/3pn",
18   },
19   {
20     id: "Uber-XL-456",
21     title: "UberXL",
22     multiplier: 1.2,
23     image: "https://links.papareact.com/5w8",
24   },
25   {
26     id: "Uber-X-789",
27     title: "Uber LUX",
28     multiplier: 1.75,
29     image: "https://links.papareact.com/7pf",
30   },
31 ];
32
33 const SURGE_CHARGE_RATE = 1.5;
34
```

Şekil 3.10.2.

Şekil 3.10.2’de ki kod dizisi, farklı yolculuk seçeneklerinin bilgilerini içerir. Her bir öge, bir yolculuk seçeneğini temsil eder. Özelliklerin açıklamaları şu şekildedir:

**id**: Yolculuk seçeneğinin kendine özgü bir kimliğini temsil eder.

**title**: Yolculuk seçeneğinin başlığını içerir, örneğin "UberX" veya "UberXL".

**multiplier**: Yolculuk seçeneğinin fiyat çarpanını temsil eder. Bu, yolculuk seçeneğinin standart fiyatına uygulanacak bir çarpan olabilir. Bu kısmın detayını Şekil 3.10.4.’de daha detaylı anlattım.

**image**: Yolculuk seçeneğinin resmini temsil eden bir URL içerir.

33. satırda yer alan **‘const SURGE\_CHARGE\_RATE = 1.5;’** ifadesi ise yolculuk sırasında uygulanabilecek bir artış ücretinin çarpanını temsil eder. Örneğin, bazı durumlarda yoğun trafik veya talep artışı nedeniyle fiyatlar yükseltilir. **SURGE\_CHARGE\_RATE** sabiti, standart fiyatın üzerine uygulanacak bir çarpan olarak kullanılabilir. Bu sabit, yolculuk fiyatlarının hesaplanması sırasında kullanılabilir.

```
App.js M X RideOptionsCard.js U X
UberClone > components > RideOptionsCard.js > data
34
35 const RideOptionsCard = () => {
36   const navigation = useNavigation();
37   const [selected, setSelected] = useState(null);
38   const travelTimeInformation = useSelector(selectTravelTimeInformation)
39
40
41   return (
42     <SafeAreaView style={tw`bg-white flex-grow`} >
43       <View>
44         <TouchableOpacity
45           onPress={() => navigation.navigate("NavigateCard")}
46           style={tw`flex-row top-0 left-5 z-50 p-1 rounded-full`}
47         >
48           <Icon name="chevron-left" type="font-awesome" />
49         </TouchableOpacity>
50         <Text style={tw`text-center pt-0 text-xs mb-3 mt-0 z-50`} >
51           Select a Ride - {travelTimeInformation?.distance?.text}
52         </Text>
53       </View>
54
55       <FlatList data={data} keyExtractor={item => item.id}
56         renderItem={({ item: { id, title, multiplier, image }, item }) => (
57           <TouchableOpacity
58             onPress={() => setSelected(item)}
59             style={tw`flex-row justify-between items-center px-10 ${id === selected?.id && 'bg-gray-200'}}` >
60             <Image
61               style={{
62                 width: 100,
63                 height: 100,
64                 resizeMode: "contain",
65               }}
66               source={{ uri: image }}
67             />
68           </TouchableOpacity>
69         )
70       </FlatList>
71     </SafeAreaView>
72   );
73 }
```

Şekil 3.10.3.

Yukarıda yer alan kod bloğunun işlevi, kullanıcının kendi isteğine göre seçebileceği bir liste döndürür. Daha fazla detay vermek gerekirse şöyle açıklanabilir:

**navigation**: React Navigation kütüphanesini kullanarak gezinme işlemlerini gerçekleştirmek için kullanılan bir nesnedir.

**selected** ve **setSelected**: Seçilen yolculuk seçeneğinin durumunu izlemek ve güncellemek için kullanılan bir durum ve durum güncelleme fonksiyonudur.

**travelTimeInformation**: **selectTravelTimeInformation** işlevi kullanılarak Redux deposundan seyahat süresi bilgilerini seçer.

**SafeAreaView**: Ekranın güvenli bir alanında yer alan bir bileşendir. Bu, içeriğin ekranın üst ve alt kenarlarına doğru uzanmasını sağlar ve kesinti olmadan görüntülenebilir.

**TouchableOpacity**: Dokunmatik olaylara yanıt veren bir bileşendir. Geri düğmesi olarak kullanılarak kullanıcıyı NavigateCard ekranına yönlendirir.

**Icon**: React Native Elements kütüphanesinden alınan bir ikon bileşenidir.

**Text**: Metin görüntülemek için kullanılan bir bileşendir. Burada yolculuk seçeneklerini gösteren bir başlık metni bulunur.

**FlatList**: Veri dizisi üzerinde liste görüntülemek için kullanılan bir bileşendir. **data** dizisini döngüye alır ve her öğeyi **renderItem** işlevine geçirir.

**'renderItem'** kısmı içerisindeki Image bloğu ise, her yolculuk seçeneği için bir resim gösterir. **Image** bileşeni, **uri** özelliği aracılığıyla resmin kaynağını belirtir. Resim, 100x100 boyutunda ve içeriği koruyacak şekilde ayarlanır.

```
App.js M RideOptionsCard.js U X
UberClone > components > RideOptionsCard.js > data
67 />
68 <View style={tw`ml-6`}>
69 <Text style={tw`text-xl font-semibold`}>{title}</Text>
70 <Text>{travelTimeInformation?.duration?.text} Travel Time</Text>
71 </View>
72 <Text style={tw`text-xl`}>
73
74 {new Intl.NumberFormat('en-gb', {
75   style: 'currency',
76   currency: 'GBP'
77 }).format(
78
79   (travelTimeInformation?.duration.value * SURGE_CHARGE_RATE * multiplier) / 100
80
81 )}
82
83 </Text>
84 </TouchableOpacity>
85 </View>
86 )}
87 />
88
89
90 <View style={tw`mt-auto border-t border-gray-200`}>
91 <TouchableOpacity disabled={!selected} style={tw`bg-black py-0 m-0 ${!selected && "bg-gray-300"}}>
92 <Text style={tw`text-center text-white text-xl`}>Choose {selected?.title}</Text>
93 </TouchableOpacity>
94 </View>
95 </SafeAreaView>
96 );
97 };
98
99 export default RideOptionsCard;
100
```

Şekil 3.10.4.

Şekil 3.10.4’ de görüldüğü üzere RideOptionsCard.js dosyasının son kısmıdır. Bu kısımda format içinde yolculuk ücretini hesaplayan denklemini görebiliriz. Ek olarak **new Intl.NumberFormat()**: Yolculuk seçeneğinin tahmini maliyetini biçimlendiren bir küresel fonksiyondur. İngiliz poundu (GBP) para birimi kullanılır. Uygulamayı İngilizce evrensel dil olduğu için ücret hesaplamada da İngiliz poundu olan ‘**GBP**’ ‘yi kullanmayı uygun gördüm. Son olarak **disabled**: Seçili bir yolculuk seçeneği olmadığında onay düğmesinin devre dışı bırakılmasını sağlar.

### **3.11 NAVIGATECARD.JS**

**NavigateCard**, kullanıcının yolculuk için başlangıç noktası ve varış noktası seçebileceği bir kart bileşenidir. İşlevi, kullanıcının navigasyon işlemi için gerekli bilgileri girmesine olanak tanımadır. Örneğin, kullanıcı bir yolculuk başlatmak için bir konum seçebilir veya bir adres arayabilir.

```
JS App.js M JS NavigateCard.js U X
UberClone > components > JS NavigateCard.js > [0] NavigateCard > <function>
1 import React from "react";
2 import { StyleSheet, Text, View, SafeAreaView } from "react-native";
3 import tw from "tailwind-react-native-classnames";
4 import { GooglePlacesAutocomplete } from "react-native-google-places-autocomplete";
5 import { GOOGLE_MAPS_APIKEY } from "@env";
6 import { useDispatch } from "react-redux";
7 import { setDestination } from "../expo/slices/navSlice";
8 import { useNavigation } from "@react-navigation/native";
9 import NavFavourites from "../NavFavourites";
10 import { TouchableOpacity } from "react-native-gesture-handler";
11 import { Icon } from "react-native-elements";
12
```

Şekil 3.11.1.

Şekil 3.11.1.'de ki görselden de anlaşılacağı üzere dosya içine uygulamanın çalışması için gerekli olan bütün kütüphane ve araçları import etme işlemini gerçekleştirdim. Bu tooları daha önceden anlattığım için burayı direkt geçiyorum.

```
JS App.js M JS NavigateCard.js U X
UberClone > components > JS NavigateCard.js > [0] NavigateCard
13 const NavigateCard = () => {
14   const dispatch = useDispatch();
15   const navigation = useNavigation();
16
17   return (
18     <SafeAreaView style={tw`bg-white flex-1`}>
19       <Text style={tw`text-center py-5 text-xl`}>Good Morning, Muhammet Taha</Text>
20       <View style={tw`border-t border-gray-200 flex-shrink`}>
21         <View>
22           <GooglePlacesAutocomplete
23             placeholder='Where to?'
24             styles={toInputBoxStyles}
25             fetchDetails={true}
26             returnKeyType={"search"}
27             minLength={2}
28             onPress={(data, details = null) =>{
29               dispatch(
30                 setDestination({
31                   location: details.geometry.location,
32                   description: data.description,
33                 })
34               );
35               navigation.navigate('RideOptionsCard');
36             }}
37             enablePoweredByContainer={false}
38             query={{
39               key : GOOGLE_MAPS_APIKEY,
40               language: "en",
41             }}
42             nearbyPlacesAPI="GooglePlacesSearch"
43             debounce={400}
44           />
45         </View>
46       </View>
47     </SafeAreaView>
48   );
49 }
```

Şekil 3.11.2.

Şekil 3.11.2.'den anlaşılacağı üzere bu kod, kullanıcının varış noktasını seçmesine ve ardından seçilen varış noktasına göre yolculuk seçeneklerini görüntülemesine olanak tanıyan bir bileşeni temsil etmektedir.

Kodda kullanılan **NavigateCard** bileşeni, **react-native-google-places-autocomplete** kütüphanesini kullanarak Google Places API'yi entegre eder. Böylece, kullanıcı varış noktası için bir yer seçebilir veya arama yapabilir.

Kod örneğinde, **GooglePlacesAutocomplete** bileşeni **placeholder**, **styles**, **fetchDetails**, **onPress**, **query**, **nearbyPlacesAPI**, **debounce** gibi özelliklerle yapılandırılmıştır. Kullanıcının bir yer seçtiğinde **onPress** olayı tetiklenir ve seçilen yerin bilgileri (**data** ve **details**) alınır.

Daha sonra, **dispatch** fonksiyonu kullanılarak **setDestination** eylemi tetiklenir ve seçilen varış noktasının konumu ve açıklaması saklanır. Ardından, **navigation.navigate** fonksiyonuyla **RideOptionsCard** ekranına yönlendirme yapılır. Bu ekran, kullanıcıya farklı yolculuk seçeneklerini gösterir.

**GooglePlacesAutocomplete** bileşenine verilen **textInputBoxStyles** stili, giriş alanlarının görünümünü yapılandırır. Örneğin, arka plan rengi, kenar yuvarlaklığı, metin boyutu vb. ayarlanabilir.

Bu şekilde, **NavigateCard** bileşeni kullanıcının varış noktasını seçmesine ve ardından uygun yolculuk seçeneklerini görüntülemesine olanak tanır.

```
App.js M  NavigateCard.js U x
UberClone > components > .js NavigateCard.js > [e] NavigateCard
46     </View>
47
48     <NavFavourites />
49   </View>
50
51   <View style={tw`flex-row bg-white justify-evenly py-2 mt-auto border-t border-gray-100`}>
52     <TouchableOpacity
53       onPress={() => navigation.navigate('RideOptionsCard')}
54       style={tw`flex flex-row justify-between bg-black w-24 px-4 py-3 rounded-full`}>
55       <Icon name="car" type="font-awesome" color="white" size={16} />
56       <Text style={tw`text-white text-center`}>Rides</Text>
57     </TouchableOpacity>
58
59     <TouchableOpacity style={tw`flex flex-row justify-between w-24 px-4 py-3 rounded-full`}>
60       <Icon
61         name="fast-food-outline"
62         type="ionicon"
63         color="black"
64         size={16}
65       />
66       <Text style={tw`text-center`}>Eats</Text>
67     </TouchableOpacity>
68   </View>
69 </SafeAreaView>
70 );
71 };
72
```

Şekil 3.11.3.

Yukarıda görülen kod bloğunda, kullanıcının araç veya yiyecek seçenekleri arasında geçiş yapmasını sağlayan bir bileşeni temsil etmektedir.

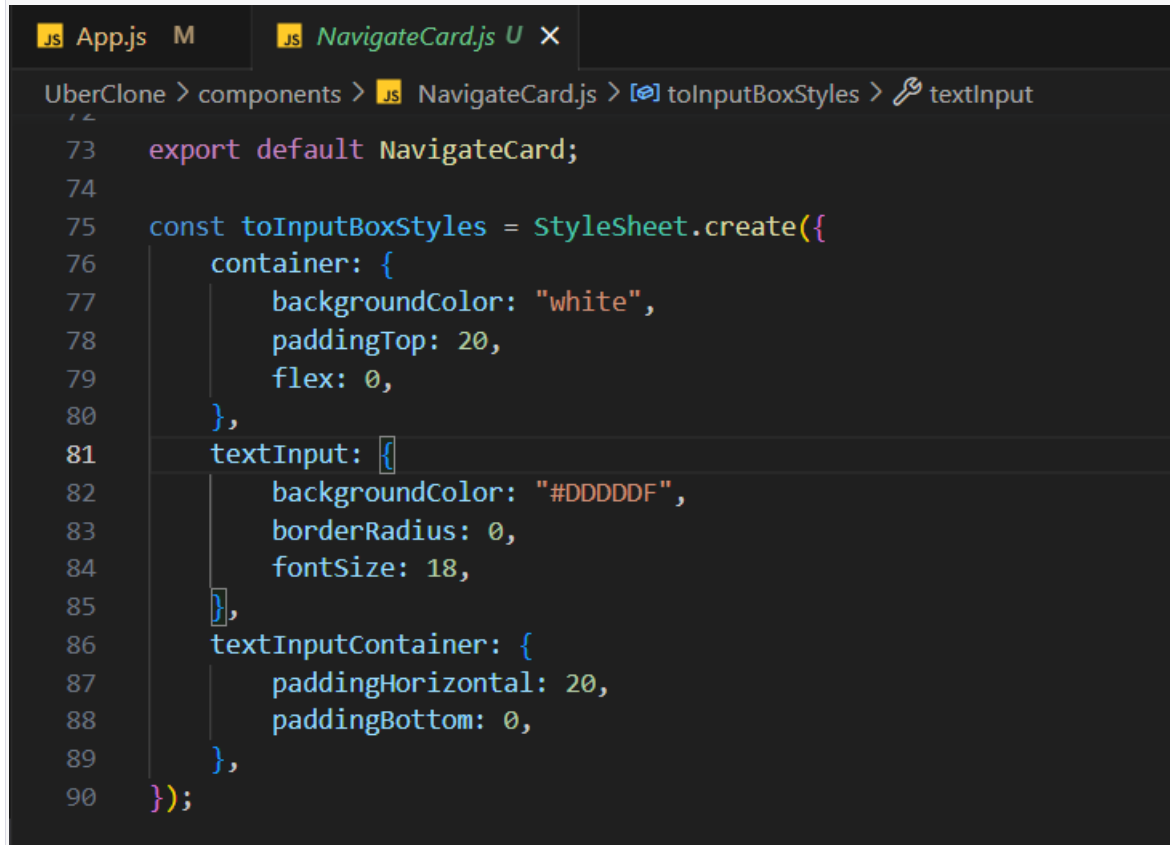
Kodda kullanılan **NavigateCard** bileşeni, kullanıcıya araç seçeneklerini gösterir ve aynı zamanda kullanıcının daha önce kaydedilen favori konumlarını gösteren **NavFavourites** bileşenini içerir.

Kod örneğinde, kullanıcı **Rides** düğmesine tıkladığında **navigation.navigate('RideOptionsCard')** fonksiyonu çağrılır ve kullanıcıyı **RideOptionsCard** ekranına yönlendirir. Bu ekran, kullanıcıya farklı araç seçeneklerini gösterir.

Aynı şekilde, **Eats** düğmesine tıkladığında ilgili işlevsellik henüz eklememiş olsam da, **TouchableOpacity** bileşenine ilgili simge ve metin ekledim. Eats kısmı da kullanıcının zevklerine göre doldurulabilir vaziyettedir.

Bununla birlikte, kodun geri kalanı **SafeAreaView**, **View** ve **TouchableOpacity** bileşenlerini kullanarak bileşenin düzenini ve stilini yapılandırmaktadır. Örneğin, arka plan rengi, kenar yuvarlaklığı, metin boyutu, düzen vb. ayarlanabilir.

Bu şekilde, **NavigateCard** bileşeni kullanıcının araç veya yiyecek seçenekleri arasında geçiş yapmasını ve farklı işlemlere erişmesini sağlar.



```
JS App.js M JS NavigateCard.js U X
UberClone > components > JS NavigateCard.js > toInputBoxStyles > textInput
73 export default NavigateCard;
74
75 const toInputBoxStyles = StyleSheet.create({
76   container: {
77     backgroundColor: "white",
78     paddingTop: 20,
79     flex: 0,
80   },
81   textInput: {
82     backgroundColor: "#DDDDDF",
83     borderRadius: 0,
84     fontSize: 18,
85   },
86   textInputContainer: {
87     paddingHorizontal: 20,
88     paddingBottom: 0,
89   },
90 });
```

Şekil 3.11.4.

Şekil 3.11.4.'de belirtilen kod parçasında **NavigateCard** bileşeninin stilini yapılandırmak için kullanılan **toInputBoxStyles** adında bir StyleSheet nesnesi ve **NavigateCard** bileşeninin dışa aktarılmasını içerir.



**textInputBoxStyles** adlı StyleSheet nesnesi, Google Places Autocomplete bileşeninin görünümünü yapılandırmak için kullanılır. Bu stil nesnesi, **container**, **textInput** ve **textInputContainer** olmak üzere üç farklı stil tanımı içerir.

**container** stil tanımı, otomatik tamamlama alanının genel konteynerini ayarlar. Arkaplan rengini beyaz (**white**) olarak ayarladım fakat bu isteğe göre değiştirilebilir ve üstten boşluk (**paddingTop**) 20 birimdir.(Telefonun ekran büyüklüğüne göre değişebilir.)

**textInput** stil tanımı, otomatik tamamlama alanındaki metin giriş alanının stilini ayarlar. Arkaplan rengini açık gri (**#DDDDDF**) olarak ayarladım fakat bu da aynı şekilde değişebilir. Tamamen zevke bağlıdır, kenar yuvarlaklığını ise (**borderRadius**) 0 olarak ayarladım ve metin boyutunu (**fontSize**) 18 olarak uygun gördüm.

**textInputContainer** stil tanımı, otomatik tamamlama alanının metin giriş alanını içeren konteynerini ayarlar. Yatay kenar boşluğunu (**paddingHorizontal**) 20 birim olarak ayarlarken, alt kenar boşluğunu (**paddingBottom**) sıfıra indirdim.

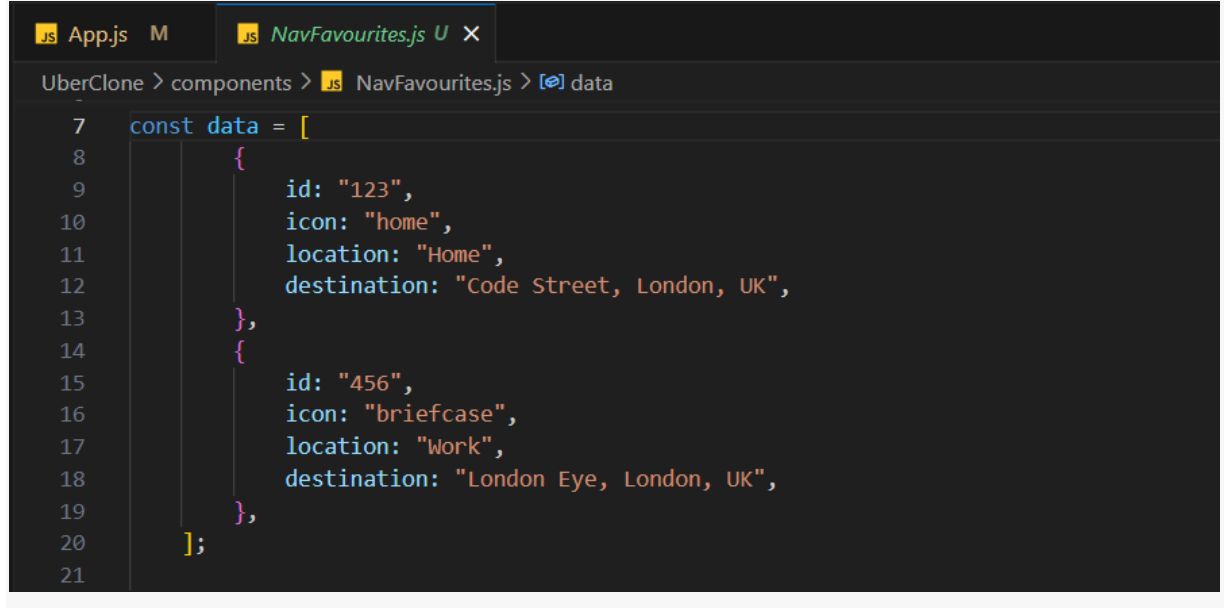
Bu şekilde, **textInputBoxStyles** StyleSheet nesnesi, **NavigateCard** bileşeninin otomatik tamamlama alanının görünümünü özelleştirmek için kullanılır. Kodun sonunda **NavigateCard** bileşeni **export default** ifadesiyle dışa aktarılır.

### 3.12 NAVFAVOURITES.JS

```
JS App.js M JS NavFavourites.js U X
UberClone > components > JS NavFavourites.js > [data]
1 import React from "react";
2 import { StyleSheet, Text, TouchableOpacity, View } from "react-native";
3 import { FlatList } from "react-native-gesture-handler";
4 import { Icon } from "react-native-elements";
5 import tw from "tailwind-react-native-classnames";
6
```

Şekil 3.12.1.

Şekil 3.12.1.'de her zaman olduğu gibi öncelik olarak dosya içine, uygulamanın hatasız çalışması için gerekli kütüphaneleri ve tooları import ettim. ( İçerik aktarma )



```
7  const data = [
8      {
9          id: "123",
10         icon: "home",
11         location: "Home",
12         destination: "Code Street, London, UK",
13     },
14     {
15         id: "456",
16         icon: "briefcase",
17         location: "Work",
18         destination: "London Eye, London, UK",
19     },
20 ];
21
```

Şekil 3.12.2.

### Şekil 3.12.2.

Yukarıda belirtilen kod parçasında, **data** adında bir dizi oluşturur. Her bir öğe, kullanıcının favori konumlarını temsil eden bir nesneyi içerir. Her öğe, aşağıdaki özellikleri içerir:

**id**: Favori konumun benzersiz kimliğini temsil eden bir dizedir.

**icon**: Favori konumu sembolize eden bir ikonun adını içerir. Örneğin, "home" ikonu evi temsil ederken "briefcase" ikonu çalışma yeri gibi işyerlerini temsil edebilir.

**location**: Favori konumun adını veya tanımını içeren bir dizedir. Örneğin, "Home" veya "Work" gibi.

**destination**: Favori konumun hedef adresini içeren bir dizedir. Örneğin, "Code Street, London, UK" veya "London Eye, London, UK" gibi. Bu lokasyonlar isteğe bağlı olarak değiştirilebilir.

Bu **data** dizisi genellikle kullanıcının favori konumlarını saklamak ve **NavFavourites** bileşeninde kullanmak için kullanılır. Böylece kullanıcı, favori konumlarından herhangi birini seçebilir ve ilgili işlevselliği başlatmak için kullanabilir.

```
App.js M NavFavourites.js U X
UberClone > components > JS NavFavourites.js > [0] data
21
22 const NavFavourites = () => {
23   return <FlatList data={data} keyExtractor={item => item.id}
24   ItemSeparatorComponent={() => (
25     <View
26       style={tw`bg-gray-200`, { height: 0.5}}
27     />
28   )}
29   renderItem={({item: { location, destination, icon} }) => (
30     <TouchableOpacity style={tw`flex-row items-center p-3`}>
31       <Icon
32         style={tw`mr-4 rounded-full bg-gray-300 p-3`}
33         name={icon}
34         type="ionicon"
35         color="white"
36         size={18}
37       />
38       <View>
39         <Text style={tw`font-semibold text-lg`}>{location}</Text>
40         <Text style={tw`text-gray-500`}>{destination}</Text>
41       </View>
42     </TouchableOpacity>
43   )}
44 />;
45
46 };
47
48 export default NavFavourites
49
50 const styles = StyleSheet.create({})
```

Şekil 3.12.3.

Şekil 3.12.3.'de belirtilen kod bloğunda **NavFavourites** adında bir bileşen oluşturur. Bu bileşen, **data** adında bir diziye erişir ve her bir öge için bir favori konum ögesi oluşturur. **FlatList** bileşenini kullanarak favori konumları liste şeklinde görüntüler.

Bileşen, **data** dizisindeki her bir öge için aşağıdaki özellikleri içeren bir görünüm oluşturur:

**location**: Favori konumun adını veya tanımını içeren bir metin.

**destination**: Favori konumun hedef adresini içeren bir metin.

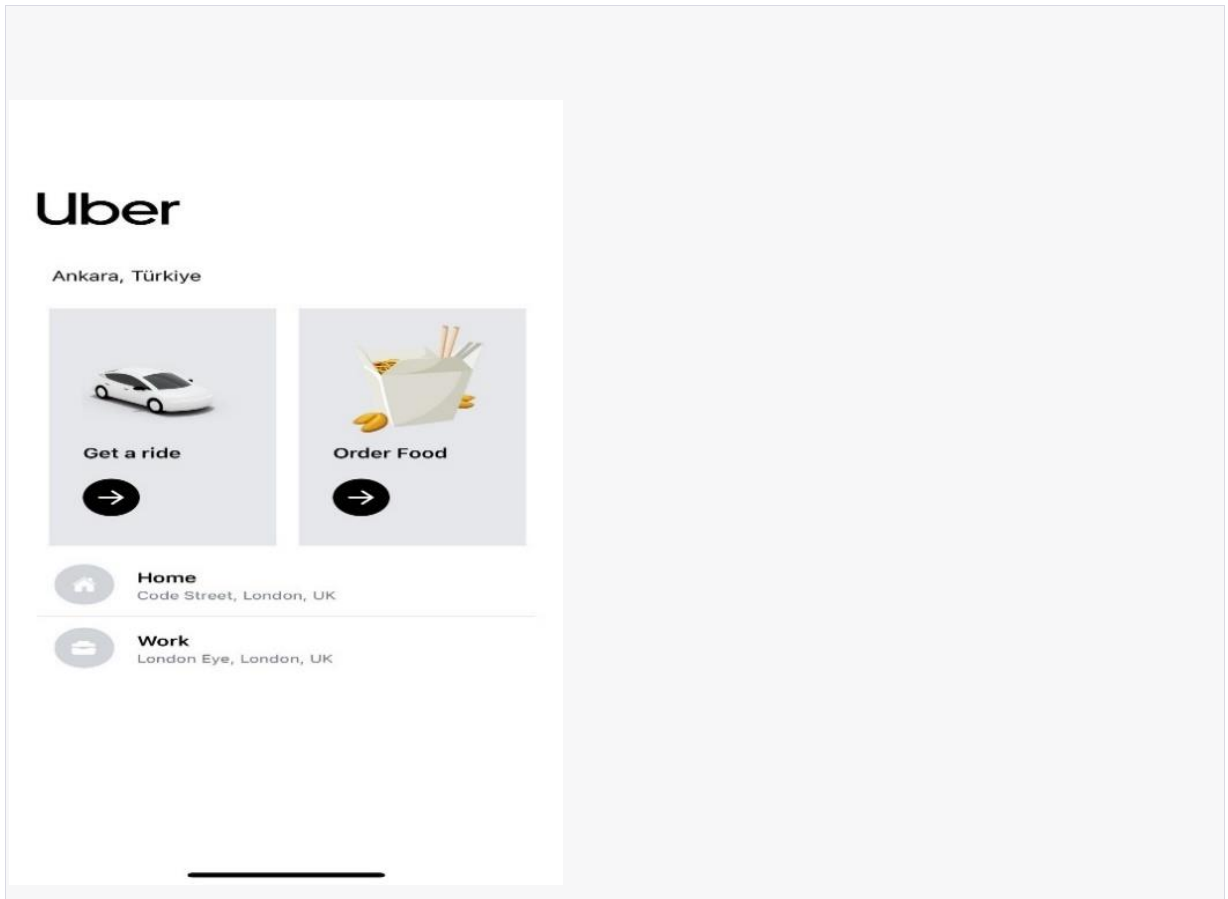
**icon**: Favori konumu sembolize eden bir ikonun adı.

Her favori konum ögesi, bir **TouchableOpacity** bileşeni içinde görüntülenir. Bu, kullanıcıların favori konumlarına dokunarak ilgili işlevselliği başlatmalarını sağlar. İkon, konum adı ve hedef adresi metinleri yan yana görüntülenir.

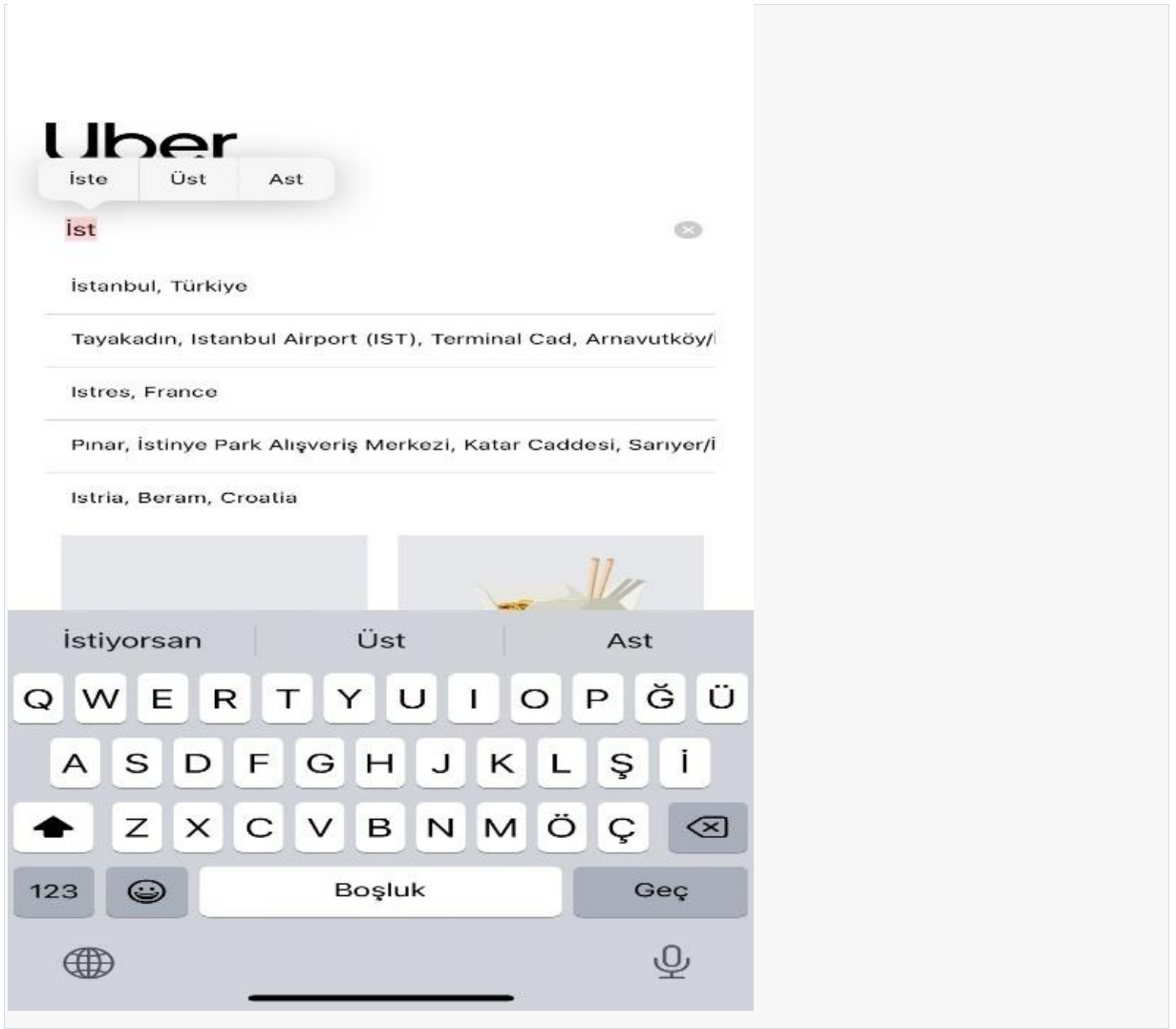
Ayrıca, **ItemSeparatorComponent** özelliği kullanılarak her öge arasında bir ayırıcı çizgi oluşturulur.

Bileşen, **FlatList** bileşenine **data** dizisini veri kaynağı olarak sağlar. **keyExtractor** özelliği, her bir ögenin benzersiz bir kimliğe sahip olduğunu belirtir. **renderItem** fonksiyonu, her bir ögenin nasıl görüntüleneceğini tanımlar.

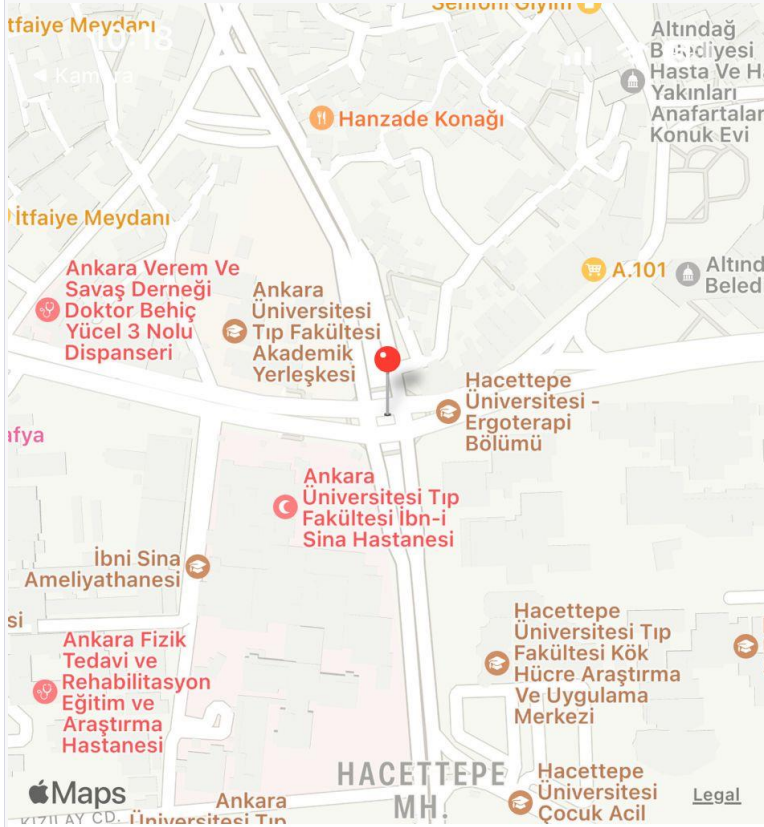
#### 4. UYGULAMADAN EKRAN GÖRÜNTÜLERİ



Şekil 4.1.1



Şekil 4.1.2.



Good Morning, Muhammet Taha

Where to?



**Home**

Code Street, London, UK



**Work**

London Eye, London, UK



Rides



Eats

Şekil 4.1.3.

# Uber

İst



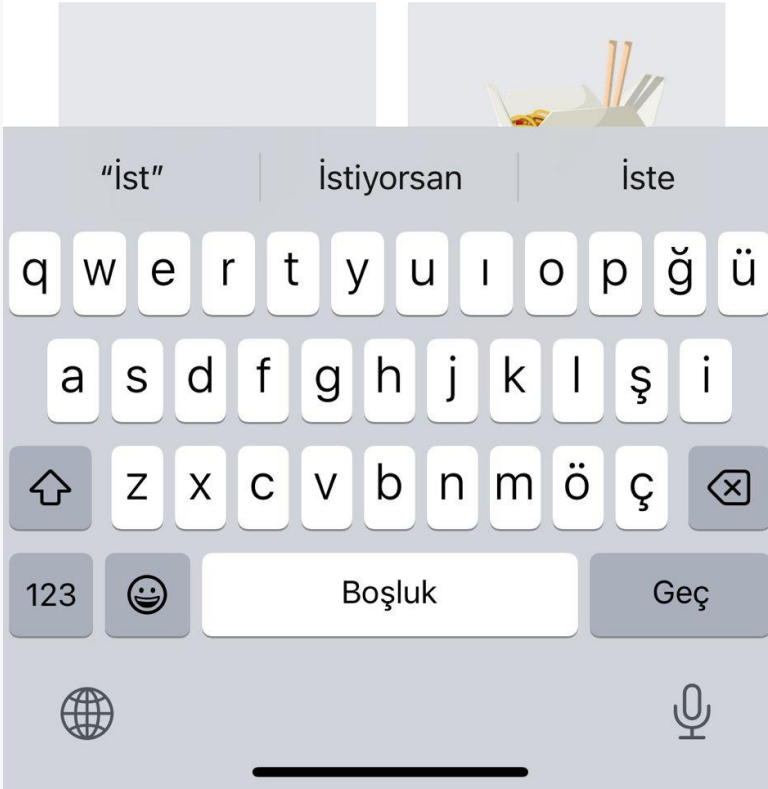
İstanbul, Türkiye

Tayakadın, Istanbul Airport (IST), Terminal Cad, Arnavutköy/İ

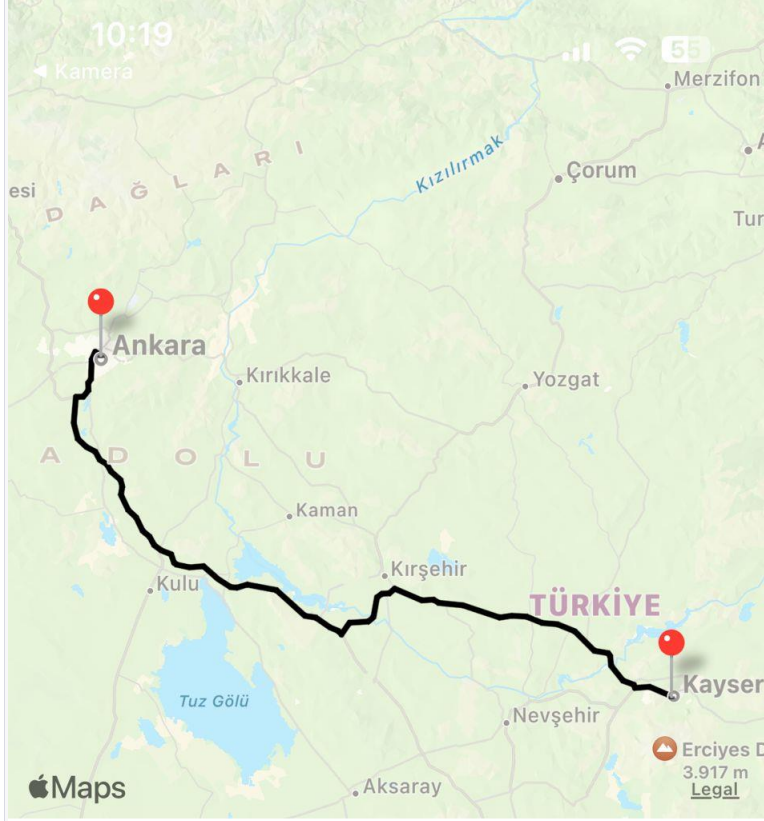
Istres, France

Pınar, İstinye Park Alışveriş Merkezi, Katar Caddesi, Sarıyer/İ

Istria, Beram, Croatia






Şekil 4.1.4.



←

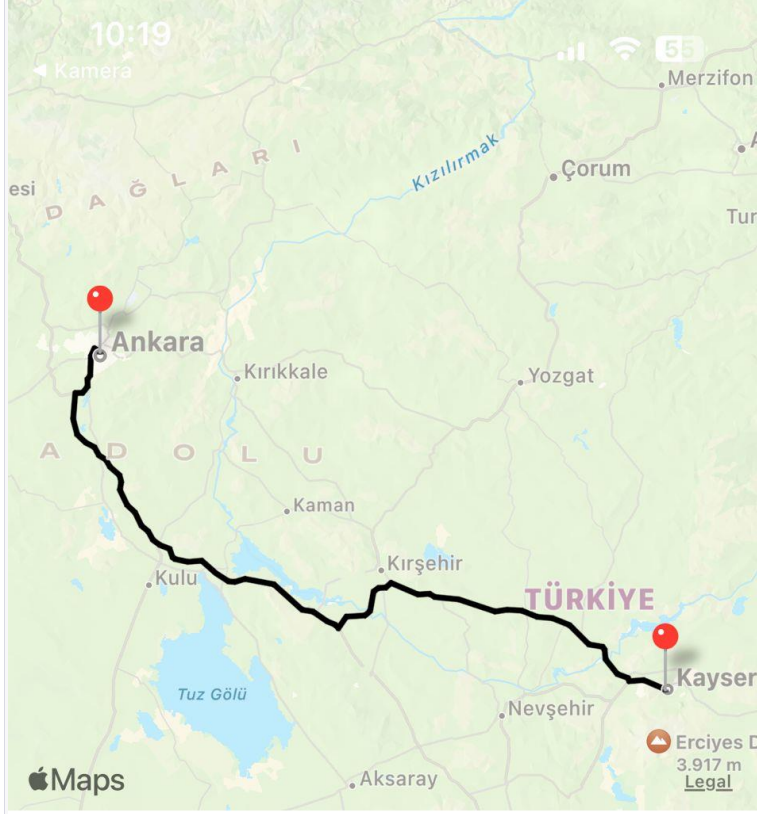
Select a Ride - 346 km

	<b>UberX</b> 3 saat 47 dakika Travel Time	<b>£204.42</b>
	<b>UberXL</b> 3 saat 47 dakika Travel Time	<b>£245.30</b>
	<b>Uber LUX</b> 3 saat 47 dakika Travel Time	<b>£357.74</b>

Choose


Şekil 4.1.5.






←


Select a Ride - 346 km



**UberX**  
3 saat 47 dakika Travel Time  
£204.42



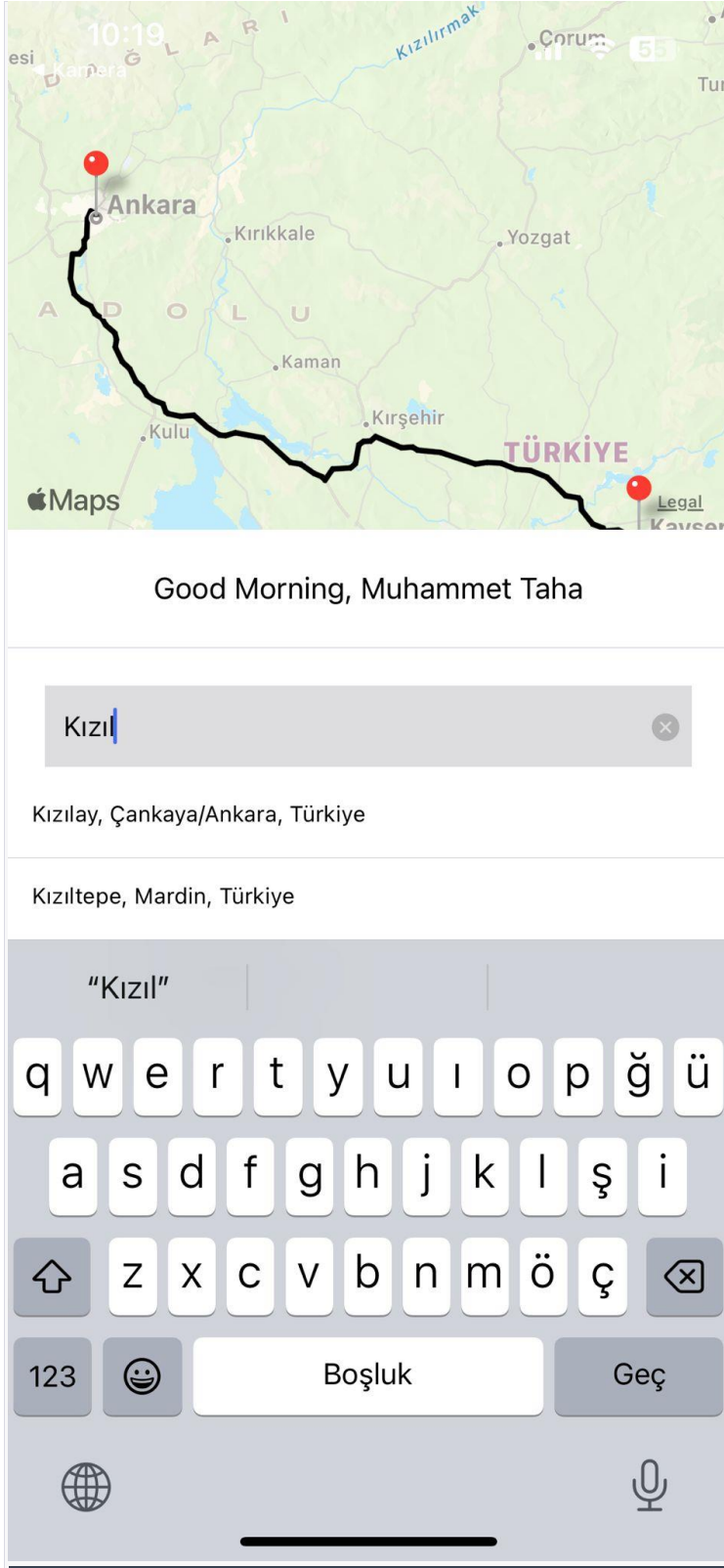
**UberXL**  
3 saat 47 dakika Travel Time  
£245.30



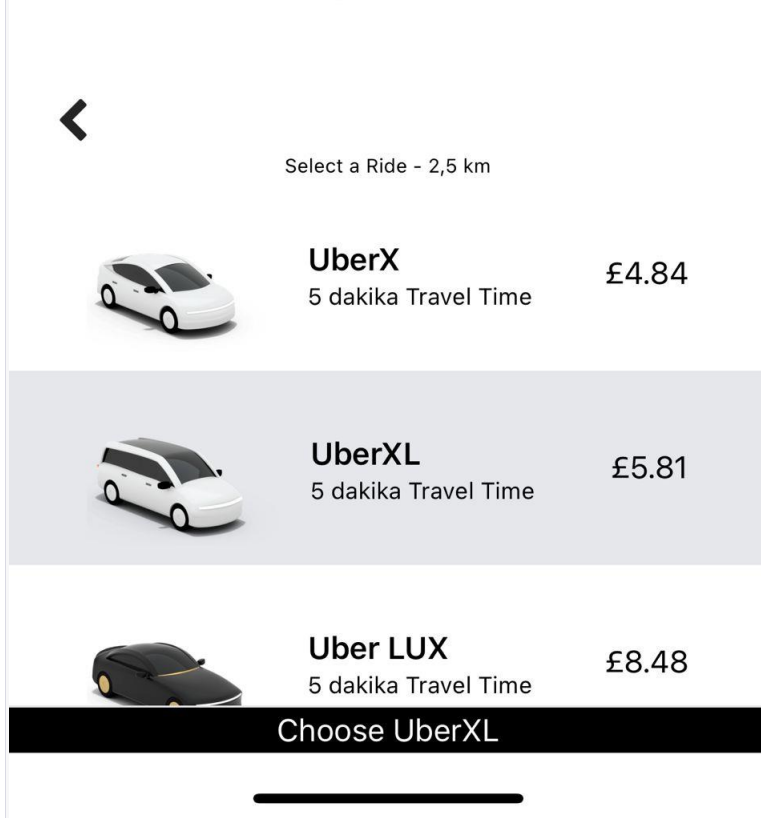
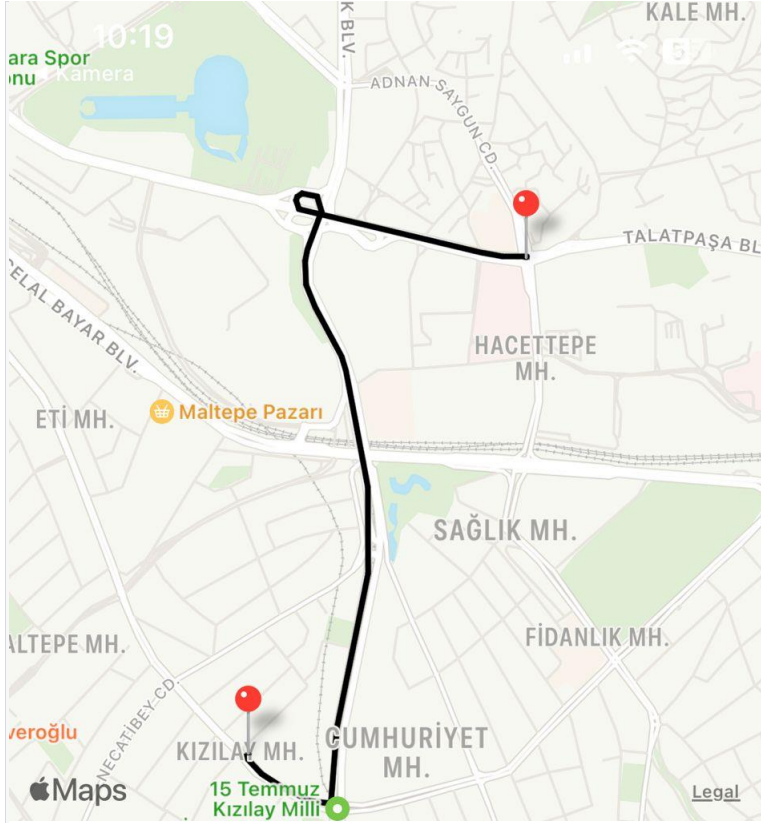
**Uber LUX**  
3 saat 47 dakika Travel Time  
£357.74

**Choose Uber LUX**

Şekil 4.1.6.



Şekil 4.1.7.



Şekil 4.1.8.

## 5. SONUÇ

Sonuç olarak yapmış olduğum bu UberClone uygulaması seyahat etmek isteyen kullanıcılar için bütçelerine ve zevklerine en uygun şekilde rota ve araçlarla yolculuk etme fırsatı sunmaktadır. Güzel tasarlanmış bu arayüzle beraber artık insanlar taksi dışında da seyahat edebilme özgürlüğünü kazanmaları amaçlanmış ve bir dönemlik bir proje yapılmıştır.

## 6. KAYNAKÇA

GitHub Resmî Sitesi: <https://github.com/>

GeeksForGeeks Resmî Sitesi: <https://www.geeksforgeeks.org/>

Udemy Resmî Sitesi: <https://www.udemy.com/>

Google Cloud Resmî Sitesi: <https://console.cloud.google.com/>

StackOverflow Resmî Sitesi: <https://stackoverflow.com/>