

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



**BLM4062-M PROJE RAPORU**

**Kriptoparalar İçin Mobil Uygulama(IOS)**

**Seyfettin Kılınç**

**18290035**

**Enver Bağcı**

**GitHub Linki:** <https://github.com/dahazil/BLM4062-M-Bahar-donemi-bitirme-projesi-KriptoApp-IOS->

**5.2022**

## ÖZET

Mobil uygulamalar hayatımızın her noktasında yer almaktadır. Bu uygulamalar sayesinde birçok işimizi kolaylıkla gerçekleştirebiliyor, bilgiye çok hızlı bir şekilde erişebiliyoruz. Raporda yer alan proje de kripto paralar ile ilgili detaylı grafik, fiyat bilgisi ve deneme amaçlı alım satım imkânı sunan bir mobil uygulamadır. Proje IOS işletim sisteminde çalışmaktadır ve swift programlama dili kullanılarak geliştirildi. Uygulama kullanıcı kayıt sistemine sahiptir. Bu sayede kişiye özgü verilerle işlemler gerçekleştirilmesine olanak sağlamaktadır. Her kullanıcı kendine özgü işlemler yapabilecektir.

## İÇİNDEKİLER

### İçindekiler Tablosu

ÖZET.....	2
İÇİNDEKİLER.....	3
1. GİRİŞ.....	4
2. TASARIM.....	5
2.1. Kayıt Ol-Giriş Ekranı.....	5
2.2. Ana sayfa.....	5
2.3. Coin Bilgi Ekranı.....	8
2.4. Cüzdan Ekranı.....	10
2.5. Alım- Satım Ekranı.....	11
2.3. Favori Coinler Ekranı.....	14
2.3. Kullanıcı Bilgileri Ekranı.....	15
3. SWİFTUI İLE UYGULAMANIN ANA YAPISINI OLUŞTURMA .....	16
3.1. Swift Programlama Dili.....	16
3.2. Kripto Modeli Oluşturma.....	16
3.3. Modele Verileri Çekme.....	17
3.4.Cüzdan Modeli Oluşturma.....	18
3.5. İnit Fonksiyonu.....	19
3.4. Ekranda Gösterilen Cüzdan Bilgilerini Güncelleme.....	19
4. LOGIN EKRANI.....	21
5. ANASAYFA.....	24
5.1. Coin List View.....	24
5.2. Popüler Coin View.....	26
5.3. Hareketli Haber Fotoğrafı.....	27
6. COIN BİLGİ EKRANI.....	28
7. CÜZDAN EKRANI.....	30
8. ALIM- SATIM EKRANI.....	32
9. FAVORİ COİNLER EKRANI .....	37
10. KULLANICI BİLGİLERİ EKRANI .....	38
11. VERİTABANI .....	40
11.1. Kullanıcı Verileri.....	40
11.2. Cüzdan Verileri.....	41
11.3. Sonuç.....	42
11.4. Kaynakça.....	43

## 1. GİRİŞ

Kriptoparalar günümüz dünyasında oldukça popülerlik kazanmış olan teknolojik bir yatırım aracıdır. Özellikle 2008 yılında Bitcoin'inin ortaya çıkması ile günümüze kadar gelen kriptoparalar sürekli gelişmelerini sürdürmüş ve sayıları devamlı artmıştır. Merkeziyetsiz olması sebebiyle de insanların güvenini kazanması kolay olmuştur.

Cep telefonlarının hayatımızdaki yeri her geçen gün artmaktadır. Birçok işlemlerimizi artık cep telefonlarımızdaki uygulamalar sayesinde gerçekleştirebilmekteyiz. Bankacılık, sağlık, oyunlar vs. birçok örnek gösterilebilir. Bu uygulamalara ek olarak kripto paraların popülerlik kazanması ile kripto paralar ile ilgili geliştirilen mobil uygulamaların da sayıları oldukça hızlı bir şekilde artmıştır. Bu uygulamalar ile insanlar takip ettiklerini coinler hakkında bilgi alabilmektedir. Bilginin yanı sıra bazı uygulamalarla anlık ve gerçek olarak alım- satım yapıp kar elde edebilmektedir. Bu uygulama da bunu amaçlayarak hazırlanmıştır.

Uygulama kullanıcı kayıt ve giriş yap ekranı ile karşılamaktadır kullanıcıyı. Kullanıcı kaydolarak sisteme giriş yapabilecektir. Ancak burada kullanıcıya kolaylık sağlamak için uygulamanın bazı temel fonksiyonlarını kaydolmadan da görebilme fırsatı sunmaktadır.

Kullanıcı sisteme giriş yaptıktan sonra ekranda coinleri görebilmektedir. Bu ekranda tüm coinlerin fiyat bilgisinin listelendiğini görecektir. Üç farklı liste yer almaktadır. Popülerliğine göre azalan, fiyatına göre en çok artan ve fiyatına göre en çok azalan olacak şekilde sıralanmıştır. Bu sıralamalar sayesinde en çok artan ve azalan coinleri görebilecektir ve buna göre strateji geliştirebilmektedir. Bu listedeki tüm coinlerin isimleri tıklanabilir yapılmıştır. Kullanıcı listedeki coinin ismine dokunduğunda o coinin detaylı bilgilerinin yer aldığı coin ekranına yönlendirilecektir. Bu ekranda coinin animasyonlu bir grafiği, altında coinin bazı detaylı bilgileri ve en alta da şayet isterse daha detaylı bir grafik görmesini sağlayacak buton yer almaktadır.

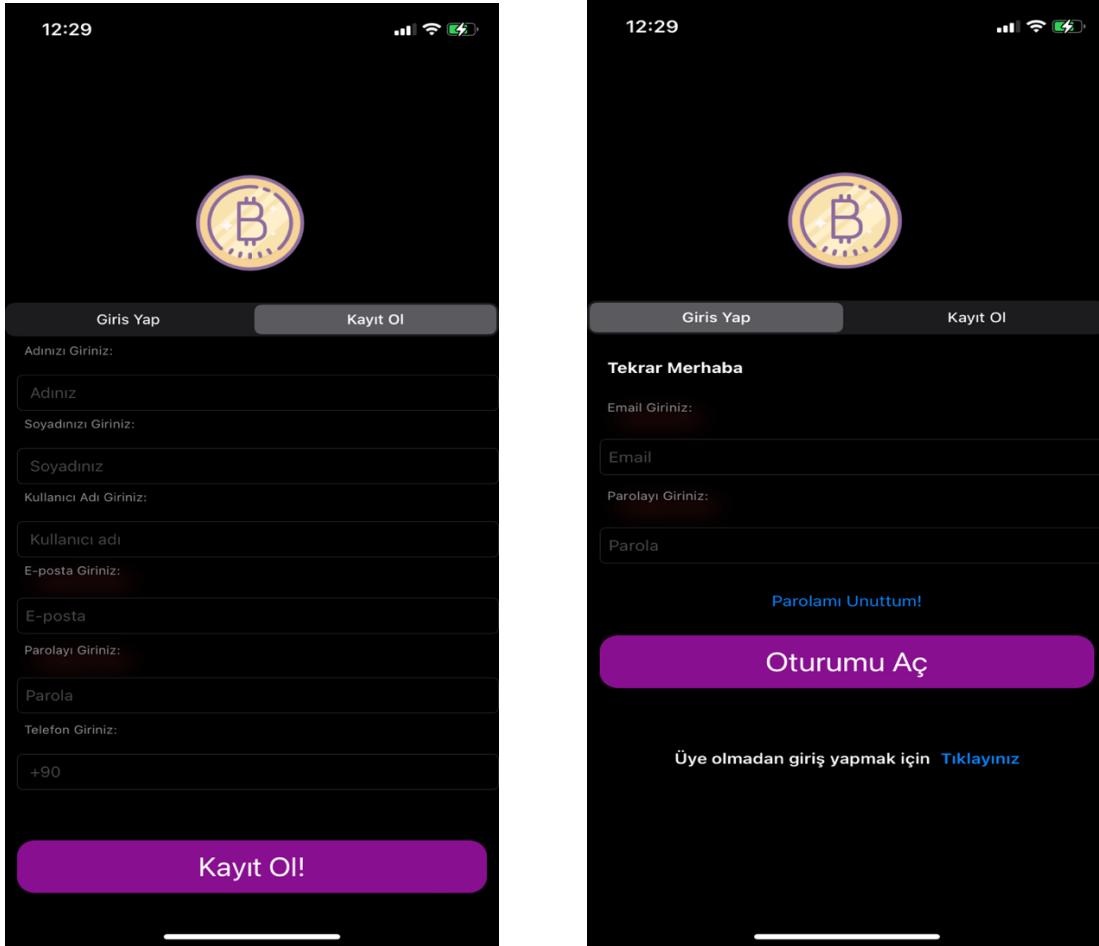
Uygulama ekranının en altında bir tabbar yer almaktadır. Bu tabbar üzerinde anasayfa, cüzdan, alım- satım, favoriler ve son olarak kullanıcı bilgisi ekranlarına yönlendirecek olan iconlar mevcuttur. Cüzdan ekranında kullanıcının sahip olduğu coinler ve bunların kaç adet olduğu, alım satım ekranında al - sat işlemlerini yapabileceği araçlar, favorilerde yakın takip ettiği coinler ve son olarak bilgiler kısmında da sisteme kayıt olduğu zaman verdiği bilgiler yer almaktadır.

## 2. TASARIM

Uygulamanın tasarımı gerçekleştirilirken olabildiğince özgün bir hava yaratılmaya çalışıldı. Özgün hava yaratılırken tabii ki işlevsellik unutulmadı. Tema olarak ise koyu tema seçildi.

### 2.1. Kayıt Ol- Giriş Ekranı

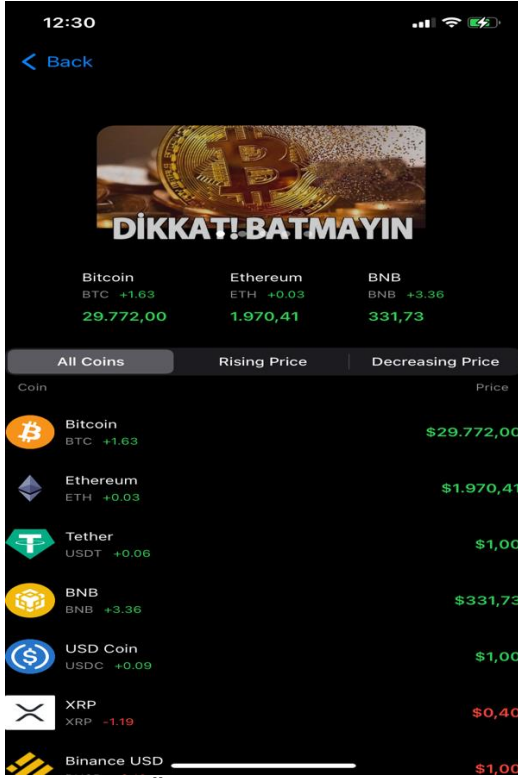
Bu ekranda kullanıcı sisteme kaydolup ardından giriş yapabilmektedir. Kullanıcı kayıtları "Firebase" adlı online veritabanında tutulmaktadır. Kayıt yaparken kullanıcıyı aynı bir kayıt doğrulama sistemine yüklerken aynı zamanda veritabanında bir kullanıcı database'i oluşturmaktadır. Oturum aç butonu ise girilen e posta ve şifrenin doğruluğunu firebase'den kontrol eder. Eğer bilgiler eşleşiyorsa sisteme giriş yapılmasına olanak sağlar. Eğer yanlış bir bilgi girildiyse sisteme erişilmesine izin vermez.



Şekil 2.1 Kayıt olma ve giriş yapma

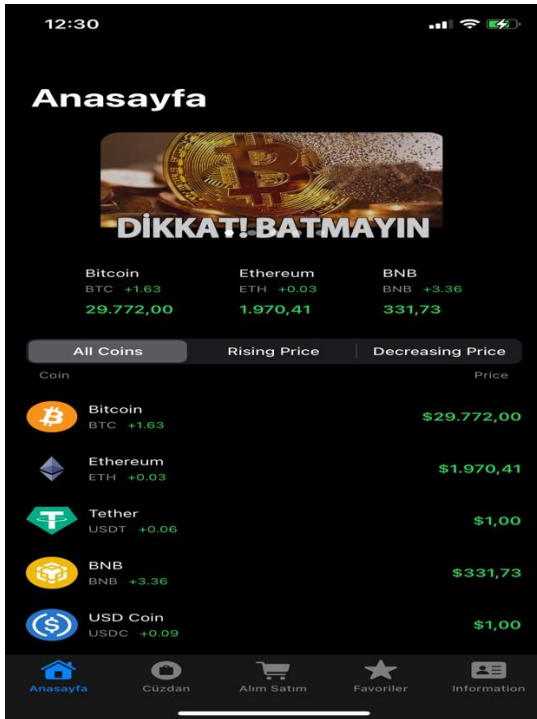
### 2.2. Ana sayfa

Burada kullanıcı uygulamaya üyelik girişi yapmadan görebileceği bazı bilgileri yer almaktadır (Şekil 2.2).



Şekil 2.2. Üye girişi olmadan sunulan bilgiler

Kullanıcı sisteme üyeliği ile girdikten sonra onu karşılayacak ekran alttaki şekildedir. Ekranda ana sayfa yazılı bir başlık, onun altında hareketli haber başlığı taşıyan fotoğraflar yer almaktadır.



Şekil 2.3. Ana sayfa

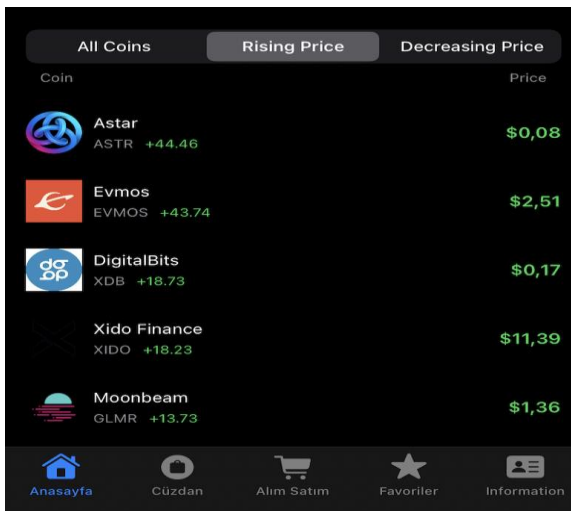
Şekil 2.3'te yer alan hareketli fotoğraflar tıklanabilir özelliktedir. Bu özellik sayesinde kullanıcı ilgi duyduğu haber fotoğrafına dokunduğunda onu haber bilgisini taşıyan ekrana yönlendirecektir. Haber tasarımı alttaki fotoğraftaki gibidir.



Şekil 2.4 Haber ekranı

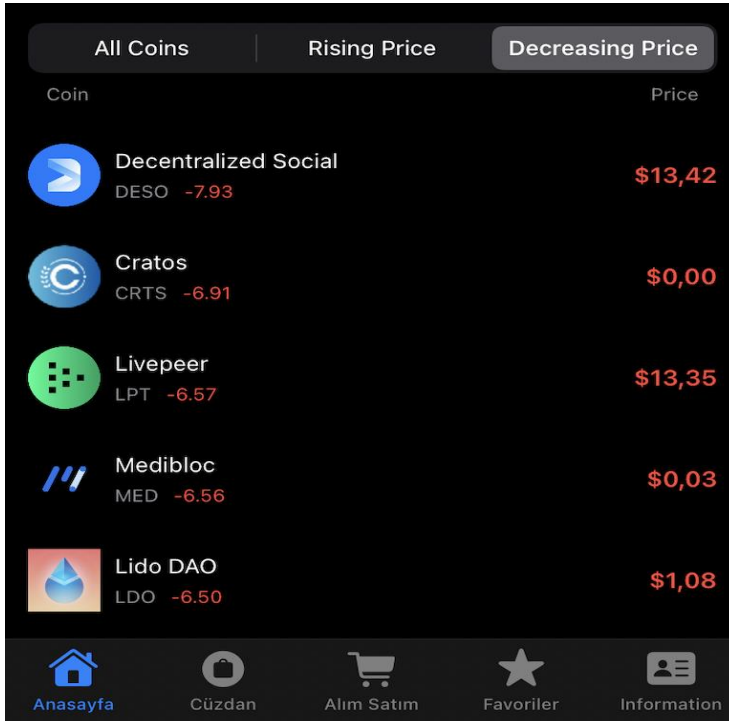
Ana sayfadaki bilgilere geri dönelim. Haber bilgisini geçtikten sonra 3 tane en popüler coinin temel bilgileri bizi karşılamaktadır. Coinin ismi, sembolü, anlık fiyatı ve fiyat değişiminin yüzdesi.

Popüler coin bilgisinin hemen altında da tüm coinler görülmektedir. Bu coinler 3 farklı şekle göre listelenmektedir. İlk liste popülerliğe göre, ikincisi fiyatı en çok yükselenler, üçüncüsü de fiyatı en çok düşenler şeklindedir.



Şekil 2.5 Fiyatı en çok yükselenler

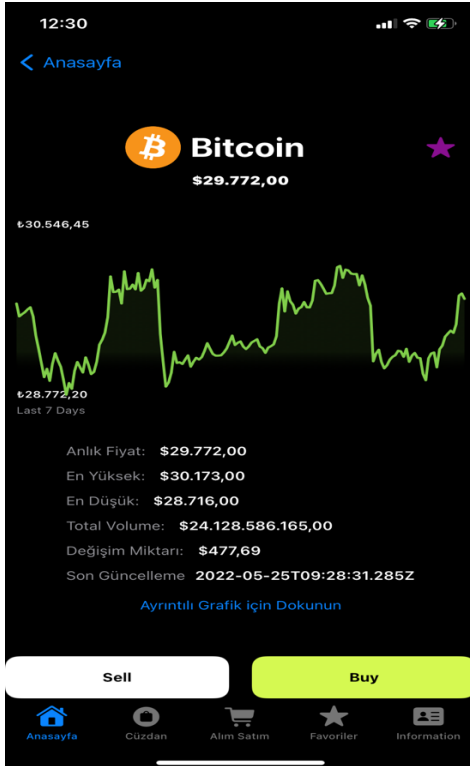
Şekil 2.6'da fiyatı en çok düşen coinler sıralanmıştır.



Şekil 2.6 Fiyat düşüş yüzdesine göre sıralanmış coinler

### 2.3. Coin Bilgi Ekranı

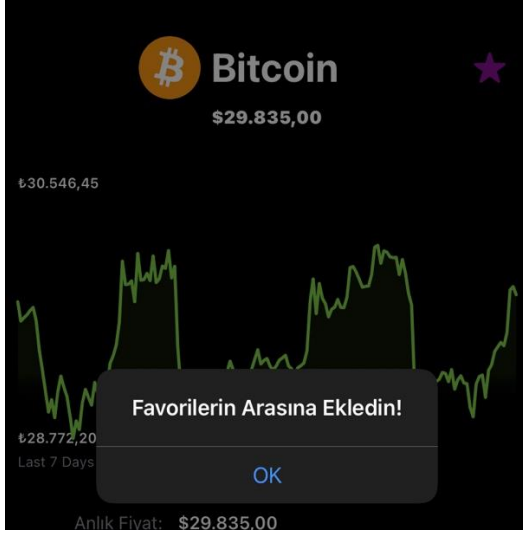
Girişte söylendiği gibi ana sayfada yer alan tüm coinlerin isimleri tıklanabilir yapılmıştır. Dokunulduğu zaman bizi coin bilgisinin yer aldığı ekrana yönlendirecektir (Şekil 2.7).



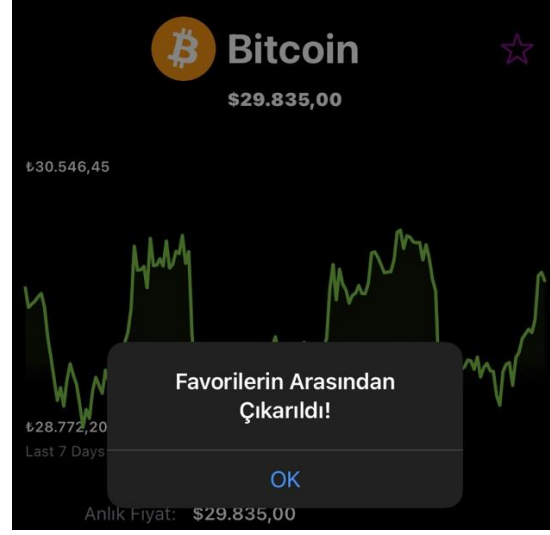
Şekil 2.7 Coin sayfası



Şekil 2.7'de ekranın en üstünde coinin fotoğrafı ve ismi yer almaktadır. Altında da dolar cinsinden anlık fiyatı bulunmaktadır. Ekranın sağ üstünde de o coinin kullanıcının favorilerine ekleyip çıkarak olan yıldız butonu yer almaktadır (Şekil 2.8, Şekil 2.9).



Şekil 2.8 Favorilere ekleme



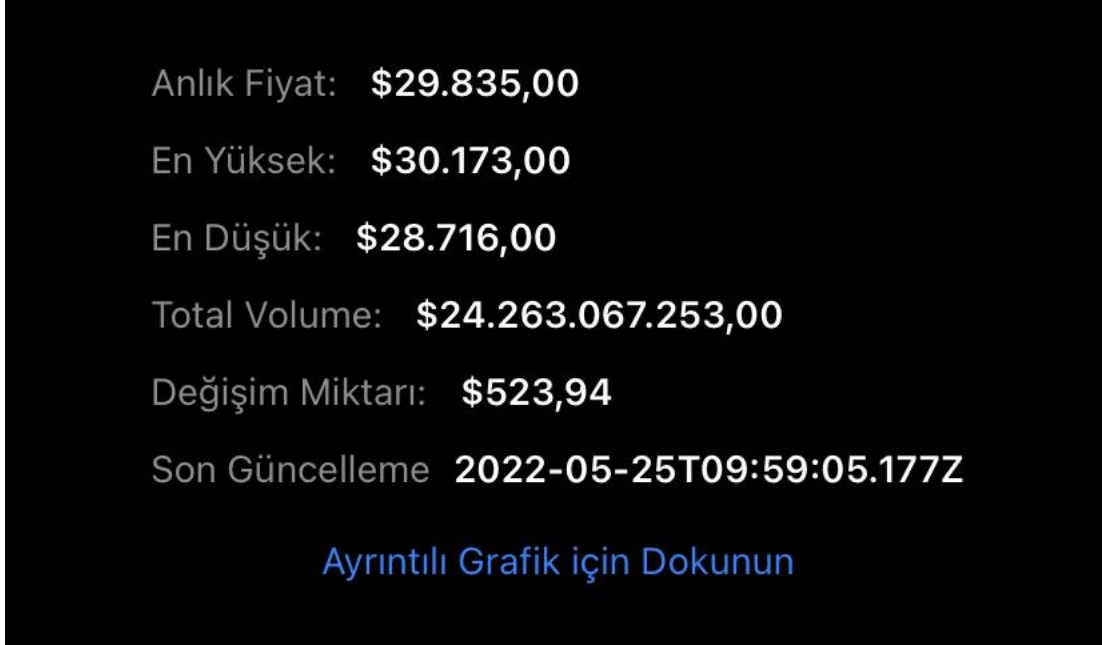
Şekil 2.9 Favorilerden çıkarma

Ekranı yer alan grafik coinin fiyatındaki değişim durumuna göre rengi yeşil ya da kırmızı olmaktadır. Aynı zamanda bu grafik üzerinde fiyatı bilgisini görebileceğiniz bir animasyon yer almaktadır (Şekil 2.10).



Şekil 2.10 Hareketli grafik animasyonu

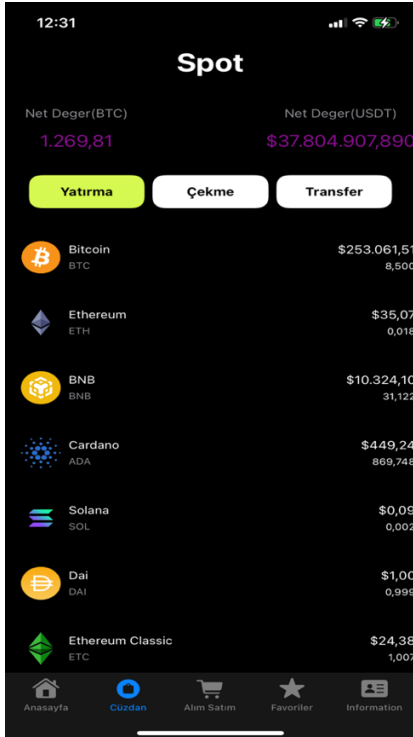
Grafiğin altında coinler ile ilgili en merak edilen bilgilerden olan anlık fiyatı, günlük en yüksek, en düşük, toplam hacim, günlük değişim miktarı ve uygulamanı bu bilgileri sunduğu tarih eklendi (Şekil 2.11).



Şekil 2.11 Ayrıntılı Bilgiler

#### 2.4. Cüzdan Ekranı

Bu ekranda kullanıcının cüzdanında yer alan coinler ve bu coinlerin miktarları gösterilmektedir (Şekil 2.12).

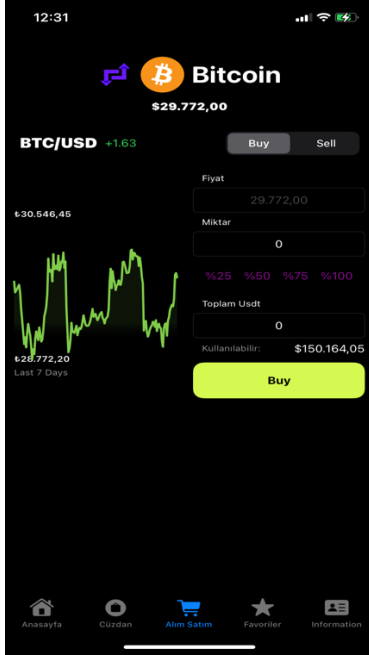


Şekil 2.12 Cüzdan Ekranı

Cüzdanda yer alan bilgiler veri tabanından alınmaktadır. Kullanıcının yaptığı her alım satım işleminden sonra da cüzdanda gösterilen bilgiler güncellenmektedir.

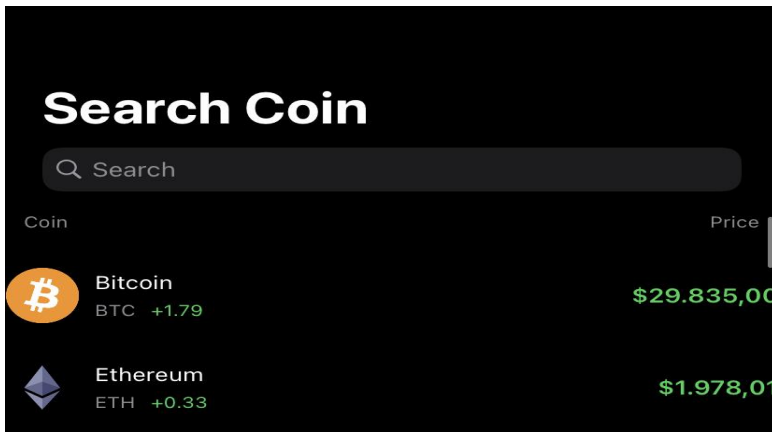
## 2.5. Alım Satım Ekranı

Bu ekranda kullanıcı al- sat işlemlerini gerçekleştirebilmektedir. Alım satımlar USDT cinsinde olmaktadır (Şekil 2.13).



Şekil 2.13 Al sat ekranı

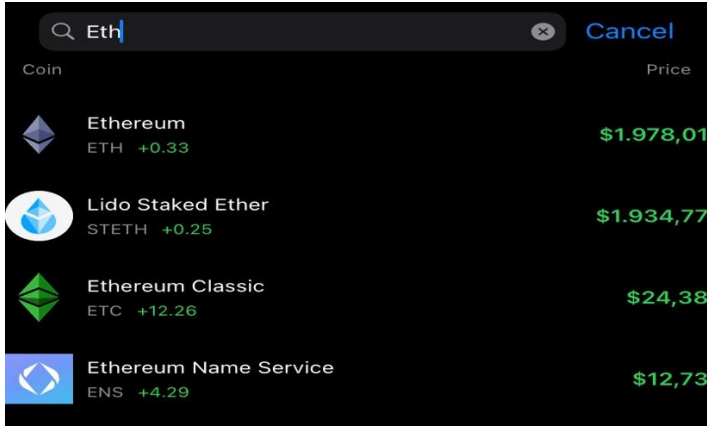
Bu ekranda tıpkı coin bilgi sayfasında olduğu gibi başlıkta seçilen coinin ismi, fotoğrafı ve anlık fiyatı yer almaktadır. Farklı olarak al - sat işleminde bulunacağımız coin'i seçmek için bir buton eklenmiştir. Bu butonun işlevi mevcutta yer alan coinleri listeleyip kullanıcının o listede al- sat yapacağı coin'i seçmesi üzerine kurulmuştur (Şekil 2.14).



Şekil 2.14 Al sat işlemi için coin seçme

Şekil 2.14'te tüm coinler listelenmektedir. Bu liste oldukça büyüktür. Kullanıcının bu listeden istediği coin'i kolayca bulabilmesi için arama yapmasını sağlayacak araç eklenmiştir.

Şekil 2.15 ve 2.16'da arama sonuçları gösterilmiştir.



The screenshot shows a search bar with 'Eth' entered. Below the search bar, there is a table with two columns: 'Coin' and 'Price'. The table lists four items: Ethereum (ETH) with a price of \$1,978.01, Lido Staked Ether (STETH) with a price of \$1,934.77, Ethereum Classic (ETC) with a price of \$24.38, and Ethereum Name Service (ENS) with a price of \$12.73. Each item also shows a small green upward arrow indicating a price increase.

Coin	Price
Ethereum ETH +0.33	\$1,978,01
Lido Staked Ether STETH +0.25	\$1,934,77
Ethereum Classic ETC +12.26	\$24,38
Ethereum Name Service ENS +4.29	\$12,73

Şekil 2.15 Arama sonuçları 1



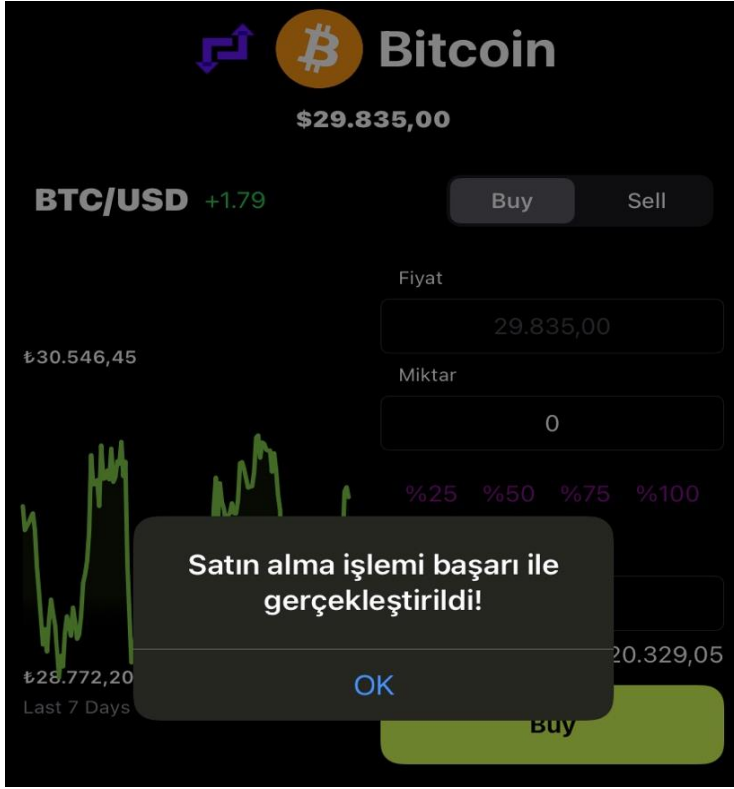
The screenshot shows a search bar with 'BNB' entered. Below the search bar, there is a table with two columns: 'Coin' and 'Price'. The table lists one item: BNB with a price of \$332.91. The item also shows a small green upward arrow indicating a price increase.

Coin	Price
BNB BNB +3.58	\$332,91

Şekil 2.16 Arama sonuçları 2

Satın alma ve satma işlemlerini gerçekleştirmek üzere butonlar ve metin girebileceğimiz ekran elemanları eklenmiştir. Buraya kullanıcı alıp satmak istediği miktarları girip (al- sat şartı gerçekleşebilecek durumdaysa) alım satım yapabilir (Şekil 2.17,2.18, 2.19). Al- sat işlemleri uygulama üzerinden gerçekleşirken aynı anda veri tabanı ile haberleşmektedir. Her al- sat işleminde sonra veri tabanında gerekli güncellemeler yapılmaktadır.

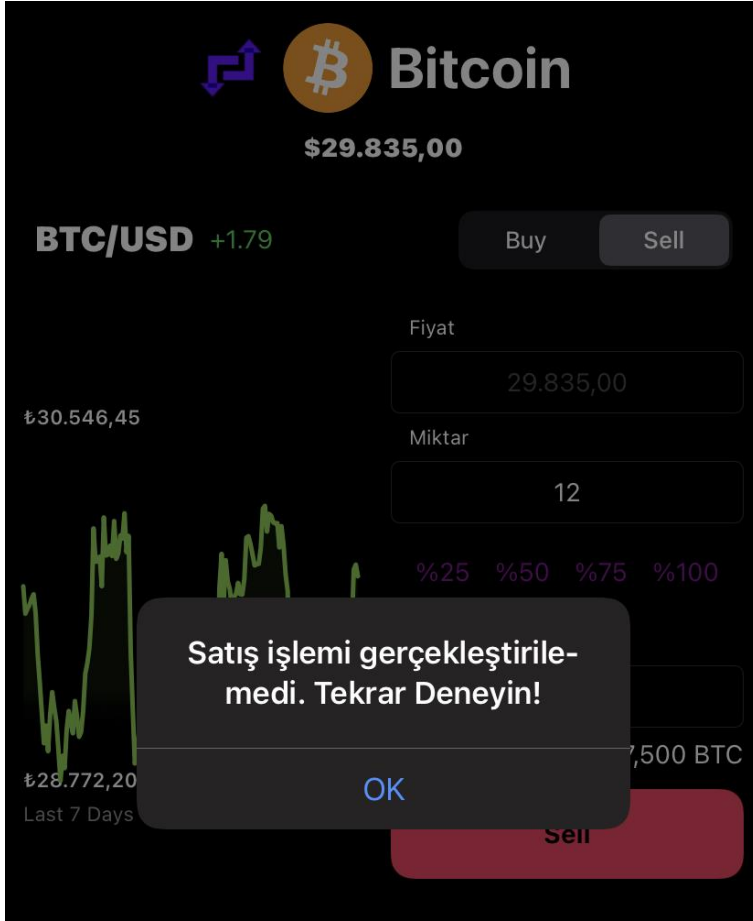
Buy ve Sell butonları alım satım yapılacağı zaman şartları kontrol edecek şekilde kodlanmıştır. Örneğin kullanıcı 2 bitcoine sahipken 3 bitcoin satmak istediğinde program bunu gerçekleştirilmesine izin vermeyecektir. Aynı durum alım yaparken de kontrol edilmektedir. 40 dolara sahip bir kullanıcı 30 bin dolar değerinde 1 adet bitcoin alımı yapmak istediğinde program buna izin vermeyecektir. Bu sayede güvenli bir alım satım işlemi gerçekleştirilmiş olacaktır.



Şekil 2.17 Satın alma işlemi



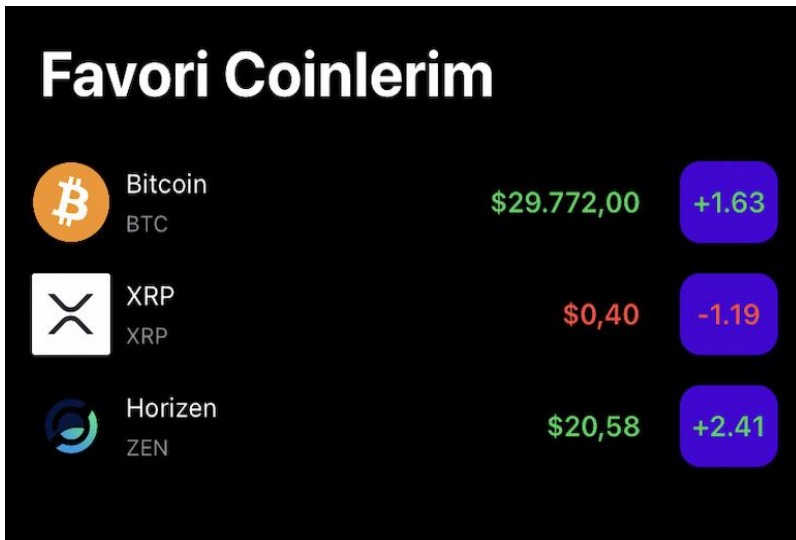
Şekil 2.18 Başarılı Satım işle



Şekil 2.19 Başarısız Satım işlemi

## 2.6. Favori Coinler Ekranı

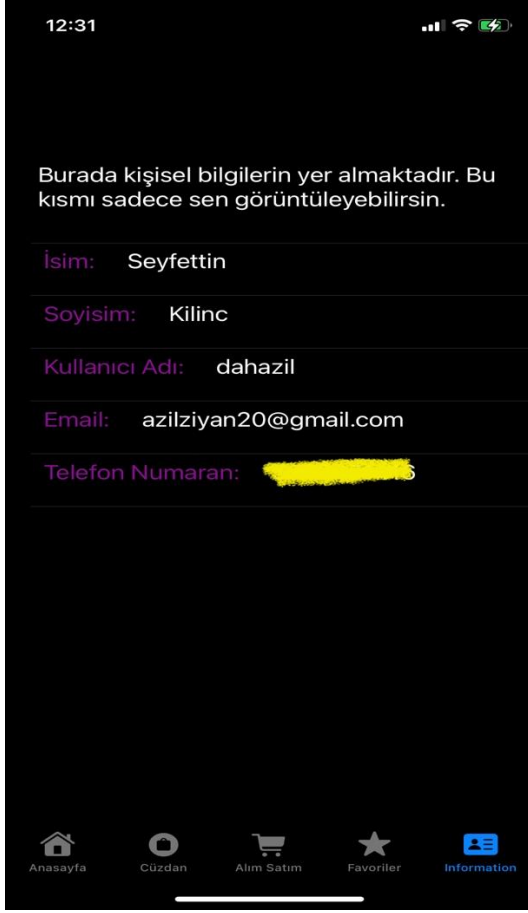
Kullanıcı sık takip ettiği coinlere daha rahat erişebilsin diye coin favorileme işlemi gerçekleştirmiştik. Bu ekranda da kullanıcı favorilerine eklediği coinleri görebilmektedir (Şekil 2.20).



Şekil 2.20 Favorilere eklenmiş coinler

## 2.7. Kullanıcı Bilgileri Ekranı

Bu ekranda kullanıcının sisteme kaydolurken verdiği bilgiler yer almaktadır (Şekil 2.21).



Şekil 2.21 Kullanıcı Bilgileri

### 3. SWIFTUI İLE UYGULANIN ANA YAPISINI OLUŞTURMA

Kodlamaya başlarken ilk adımda kripto paralar ile ilgili verileri çekeceğimiz api'in json yapısına bakmamızın gerektiğini düşünmekteyim (Şekil 3.1). Coingecko adlı web sitenin bize sunduğu api, kriptoparalar ile ilgili birçok temel bilgiyi bize verebilmektedir. Bu bilgiler kısaca anlık fiyat, id, sembol, isim, fiyat değişimi, 24 saatlik en yüksek ve en düşük gibi verileri vermektedir. Bu verileri swift dili ile uygulamamıza bütünleşmiş bir şekilde çekmeliyiz.

```
{
  "id": "bitcoin",
  "symbol": "btc",
  "name": "Bitcoin",
  "image": "https://assets.coingecko.com/coins/images/1/large/bitcoin.png",
  "current_price": 29686,
  "market_cap": 565483095564,
  "market_cap_rank": 1,
  "fully_diluted_valuation": 623401781760,
  "total_volume": 23992724669,
  "high_24h": 30173,
  "low_24h": 28716,
  "price_change_24h": 298.06,
  "price_change_percentage_24h": 1.01423,
  "market_cap_change_24h": 6066460910,
  "market_cap_change_percentage_24h": 1.08443,
  "circulating_supply": 19048943.0,
  "total_supply": 21000000.0,
  "max_supply": 21000000.0,
  "ath": 69045,
  "ath_change_percentage": -57.005,
  "ath_date": "2021-11-10T14:24:11.849Z",
  "atl": 67.81,
  "atl_change_percentage": 43678.55319,
  "atl_date": "2013-07-06T00:00:00.000Z",
  "roi": null,
  "last_updated": "2022-05-25T11:18:40.675Z",
  "sparkline_in_7d": {
    "price": [
      30027.881391536717,
      29850.866134310378,
      29881.144768342547,
      29907.88149438243,
      29946.89188314115,
      29976.28891176134,
      29851.722908349042,
      29583.133426780576,
      29402.681241556325,
      29204.99698805408,
      29014.26599807768,
      29128.983080913495,
      28963.28695458867,
      29197.82080184199,
      29344.38809720751,
      29241.649429828303,
      29157.04189112087,
      28975.01878983267,
      28772.20050112577,
      28933.53974183301,
      28853.477590685474,
      29100.110152170204,
      29222.7735574537,
      29148.070503245166,
      29078.472251350275,
      29286.656996261212,
      29185.483322189215,
      29056.226100091822,
      29086.762422998854,
      29224.199174464593,
      29462.75248548161,
      29530.45376871893,
      29627.800179670576,
      29840.233937359586,
      30401.478706403686,
      30186.035736931997,
      30189.79125483797,
      30219.79652704418,
      30028.522614298352,
      30472.01255538265,
      30226.486301814184,
      30232.007727975822,
      30289.31087717314,
      2,
      30222.997986532428,
      30448.564764580642,
      30204.782777609496,
      30239.692311994073,
      30355.59330787842,
      30335.778359754542,
      30498.4928562101,
      30247.253836113927,
      30353.09022539717,
      29610.749273550864,
      29233.558880619712,
      28883.583563813074,
      28941.11591341604,
      29078.642709409283,
      29361.983436757986,
      29209.395363568612,
      29218.857806706103,
      29194.913885389316,
      29256.813374416255,
      29313.84120108591,
      29158.69802123103,
      29227.28615150176,
      29281.41029204116,
      29255.988512722528,
      29290.06697649485,
      29377.041270304915,
      29482.022669025562,
      29403.868537604212,
      29332.91144408883,
      29310.33773441654,
      29310.68064074855,
      29315.265376915155,
      29355.49900328947,
      29408.829445605934,
      29497.347529518447,
      29586.643167160633,
      29456.156222490306,
      29488.44296610853,
      29506.42157055718,
      29435.79656041014,
      29400.69738961443,
      29438.097435799475,
      29491.507947760598,
      29512.56199034459,
      29536.76112195805,
      29437.63395643593,
      29412.806868676085,
      29348.146110206842,
      29421.371906529017,
      29440.270841599387,
      29465.85503423404,
      29577.488869025095,
      30025.810045074377,
      30236.777415095316,
      29919.258535544526,
      30195.853504758594,
      30059.79716769435,
      30009.759324959436,
      29888.987158689626,
      29958.666956261368,
      29978.384670416777,
      30049.27974428577,
      29976.10252024696,
      29974.13309415041,
      30069.910565295708,
      30341.531208439646,
      30351.050417138096,
      30300.123520864105,
      30233.304059660284,
      30162.20077793916,
      30169.673747371988,
      30172.13926006517,
      30276.76118473994,
      30528.731586363934,
      30546.4547069185,
      30381.50734327177,
      30495.217363563523,
      30483.75338516015,
      30440.18813153408,
      30433.124856438022,
      30435.747186647808,
      30278.200334421705,
      30415.521868185282,
      30292.474385454425,
      30104.347687672132,
      29998.513096702307,
      29231.832325497417,
      29339.48904783449,
      29392.00794876634,
      29219.5951408335,
      29135.362563102553,
      29179.22613289936,
      29254.361332338103,
      29294.55857934977,
      29316.325109179415,
      29378.433744223774,
      29369.613462231624,
      29294.103159102884,
      29404.765572808068,
      29225.280371935165,
      29310.803263531212,
      29348.30434667454,
      29281.64881987151,
      29275.102905605483,
      29058.5442424006,
      28971.612089682047,
      29273.849146237022,
      29341.05079474907,
      29354.60025384753,
      29149.293006524687,
      29398.13289445197,
      29449.13691004379,
      29571.14475089601,
      2,
      29614.208799995526,
      29655.02613175249,
      29624.23203844424,
      29744.344523718555,
      30136.3416441564,
      30167.0
    ]
  }
}
```

Şekil 3.1 Api'dan çekilecek verinin json yapısı

#### 3.1. Swift Programlama Dili

Swift, Apple tarafından iOS, macOS, watchOS, tvOS gibi platformlarına uygulamalar geliştirmek için oluşturulan, derlenerek çalışan güçlü ve kullanımı oldukça kolay, nesne yönelimli bir programlama dilidir. İlk olarak 2014'te duyurulmuştur. Apple, 2014 öncesinde kendi işletim sistemlerine uygulama geliştirilmesi için " Objective C" programlama dilini kullanmıştır. 2014 yılı sonrasında ise tamamen kendisinin geliştirdiği swift programlama dilini hayata geçirmiştir. Bu dil, C syntax dillerine benzemektedir. Bu projede swift programlama dili kullanılmıştır.

#### 3.2. Kripto Modeli Oluşturmak

Projenin ilk adımda çekeceğimiz veriler için uygun bir struct modeli oluşturuldu. Bu model json dosyasında gelen veriler ile eşlenecekti. Şekil 3.2'de bu model gösterilmiştir.



```

8 import SwiftUI
9
10 // MARK: Crypto Model For JSON Fetching
11 struct CryptoModel: Identifiable, Codable {
12     var id: String
13     var symbol: String
14     var name: String
15     var image: String
16     var current_price: Double
17     var last_updated: String
18     var price_change: Double
19     var last_7days_price: GraphModel
20     let totalVolume, high24H, low24H: Double?
21     let priceChange24H: Double?
22
23
24
25     enum CodingKeys: String, CodingKey {
26         case id
27         case symbol
28         case name
29         case image
30         case current_price
31         case last_updated
32         case price_change = "price_change_percentage_24h"
33         case last_7days_price = "sparkline_in_7d"
34         case totalVolume = "total_volume"
35         case high24H = "high_24h"
36         case low24H = "low_24h"
37         case priceChange24H = "price_change_24h"
38     }
39 }
40
41 struct GraphModel: Codable {
42     var price: [Double]
43     enum CodingKeys: String, CodingKey {
44         case price
45     }
46 }

```

Şekil 3.2 Kripto model

### 3.3. Modele Verileri Çekme

Modeli oluşturduktan sonra api'dan verileri rahatlıkla çekebileceğiz. Verileri api'in bize verdiği json linkinden çekeceğiz. Bu ve bundan sonraki tüm veri işlemlerini AppViewModel adında class içerisinde gerçekleştireceğiz (Şekil 3.3). Bu class'ı kısaca özetlemek gerekirse, coin bilgilerini almak, coinler üzerinden sıralama işlemlerini yapmak, uygun verilere atmak, cüzdan oluşturmak, cüzdan bilgilerini göstermek için kullanacağız. Bu class üzerinden fonksiyonlar yazacağız.

```

class AppViewModel: ObservableObject {

```

Şekil 3.3 Class tanımı

Api'dan çekeceğiz coin verilerini her bir coin için ayrı ayrı yapmak oldukça zor ve karmaşık olacağı için CryptoModel tipinde yapıları tutacak bir array oluşturarak gerçekleştirdik. Her coinin bilgisi artık bu dizide tutulacaktır (Şekil 3.4). Published ön eki swift'te class içerisindeki bu veriye her yerden ulaşılabilmesini göstermek için konur. C++'daki public ön ekine benzemektedir. Şekil 3.4'te yer alan fotoğraftaki soru işareti de swift'in sunduğu güzel bir özellikten gelmektedir. Bu soru işareti sayesinde swift'de verinin dolu ya da boş olabileceğini bildirebilmekteyiz.

```

@Published var coins: [CryptoModel]?

```

Şekil 3.4 Coinlerin tutulacağı dizi

Gerekli değişkenler oluşturulduktan sonra artık verileri çekebileceğiz (Şekil 3.5). Bunun için şekildeki fonksiyonu yazdık. Bu fonksiyon ilgili url'den verileri çekecek işlemleri göstermektedir.

```
func fetchCryptoData()async throws{
    // MARK: Using Latest Async/Await
    guard let url = URL(string:
        "https://api.coingecko
        .com/api/v3/coins/markets?vs_currency=usd&order=market_cap_desc&per_page=250&page=1&sparkline=true&price_change_percentage=\(self.denex)")
    else{return}
    let session = URLSession.shared

    let response = try await session.data(from: url)
    let jsonData = try JSONDecoder().decode([CryptoModel].self, from: response.0)

    // Alternative For DispatchQueue Main
    await MainActor.run(body: {
        self.coins = jsonData
        if let firstCoin = jsonData.first{
            self.currentCoin = firstCoin
        }
        coins?.forEach({ CryptoModel in
            if CryptoModel.name == currentCoinName{
                self.chosenCoin = CryptoModel
            }
            allsearchCoinArray.append(CryptoModel.name)
        })
    })
}
```

Şekil 3.5 Kripto verilerini çekme

2.Bölümde ana sayfa üzerinde coinleri listelediğimizi göstermiştik. Listeleme işleminde kullandığımız verileri AppViewModel class'ı üzerinde gerçekleştirdiğimizi de az önce söyledik. Bu verilerden ilki coins array'i idi. Bu array'e veriler api'dan popülerliğine göre çekilip eklenmiştir. Ek olarak fiyatı en çok yükselen ve azalan olacak şekilde sıralayan dizi de oluşturmak adına bir fonksiyon daha yazdık (Şekil 3.6). Bu fonksiyon fiyat değişimine göre verileri sıralayacaktır.

```
func riseAndDecreasingCoinListed()async throws{
    // MARK: Using Latest Async/Await
    guard let url = URL(string:
        "https://api.coingecko
        .com/api/v3/coins/markets?vs_currency=usd&order=market_cap_desc&per_page=250&page=1&sparkline=true&price_change_percentage=7d") else{return}
    let session = URLSession.shared

    let response = try await session.data(from: url)
    let jsonData = try JSONDecoder().decode([CryptoModel].self, from: response.0)

    // Alternative For DispatchQueue Main
    await MainActor.run(body: {
        self.coins = jsonData
        if let firstCoin = jsonData.first{
            self.currentCoin = firstCoin
        }
        highetsPrice = coins!.sorted(by: { x, y in
            x.price_change > y.price_change
        })

        lowestPrice = coins!.sorted(by: { x, y in
            x.price_change < y.price_change
        })
    })
}
```

Şekil 3.6 Fiyat değişimine göre coin array'ini sıralama

### 3.4. Cüzdan Modeli Oluşturma

Cüzdan modeli, coin modeline göre daha sade olacak şekilde verileri tutmak için oluşturuldu. Bu model kullanıcının cüzdanında yer coinleri tutmak için gerekmektedir. Coinin bazı bilgilerini ve miktarını içeren veriler içinde yer almaktadır (Şekil 3.7).

```

11 struct mywalletModel : Identifiable {
12     var id : String
13     var name : String
14     var symbol: String
15     var image : String
16     var currentPrice : Double
17     var coinPiece : Double
18 }

```

Şekil 3.7 Cüzdan modeli

Bu modele çekeceğimiz verileri de yine AppViewModel class'ı içerisinde gerçekleştireceğiz (Şekil 3.8).

```

@Published var myWalletCoin = [mywalletModel]()

```

```

coins?.forEach({ CryptoModel in
    vm.items.forEach { tag in
        if CryptoModel.name == tag.coinName{
            self.chosenCoin = CryptoModel
            walletCoinNameArray.append(CryptoModel.name)
            walletCoin.append(CryptoModel)
            if tag.coinPiece != 0 {
                self.myWalletCoin.append(mywalletModel(id: CryptoModel.id, name: CryptoModel.name, symbol: CryptoModel.symbol, image:
                    CryptoModel.image, currentPrice: CryptoModel.current_price, coinPiece: tag.coinPiece))
            }
        }
    }
})

```

Şekil 3.8 Cüzdan modeline verileri ekleme

### 3.5. Init Fonksiyonu

Swift'te diğer nesne yönelimli dillerde olduğu gibi constructor yazabilmekteyiz. Bunu init fonksiyonu ile gerçekleştiriyoruz. Amacımız AppViewModel class'ında nesne oluşturulduğunda gerekli olan tüm değişkenlere fonksiyonlar sayesinde veriler atanması işlemini gerçekleştirmektir.

```

init(){
    Task{
        do{
            try await fetchCryptoData()
            try await oneCoinDetail()
            try await riseAndDecreasingCoinListed()
        }catch{
            // HANDLE ERROR
            print(error.localizedDescription)
        }
    }
}

```

Şekil 3.9 Init Fonksiyonu

### 3.6. Ekranda Gösterilen Cüzdan Bilgilerini Güncelleme

Tasarım bölümünde cüzdanımızda yer alan bilgilerini gösteriyorduk. Alım satım sonrası cüzdanda yapılan değişiklikleri veritabanında yapıldığı gibi ekranda göstermek için de yapmamız gerekiyordu. UpdateWalletCoinBuySell fonksiyonu ile bu işlemi gerçekleştirdik (Şekil 3.10)

```

func updateWalletCoinBuySell(){
    self.netDegerBtc = 0.0
    self.netDegerUsdt = 0.0
    vm.listenDocument()
    myWalletCoin.removeAll()
    coins?.forEach({ CryptoModel in
        vm.items.forEach { tag in
            if CryptoModel.name == tag.coinName{
                self.chosenCoin = CryptoModel
                walletCoinNameArray.append(CryptoModel.name)
                walletCoin.append(CryptoModel)
                if tag.coinPiece != 0 {
                    self.myWalletCoin.append(mywalletModel(id: CryptoModel.id, name: CryptoModel.name, symbol: CryptoModel.symbol, image:
                        CryptoModel.image, currentPrice: CryptoModel.current_price, coinPiece: tag.coinPiece))
                }
            }
        }
    })

    coins?.forEach({ CryptoModel in
        vm.items.forEach { tag in
            if CryptoModel.name == tag.coinName && tag.coinPiece != 0 {
                self.netDegerUsdt = (tag.coinPiece * CryptoModel.current_price) + (self.netDegerUsdt)
            }
            if tag.coinName == "USDT" {
                self.netDegerUsdt = self.netDegerUsdt + tag.coinPiece
            }
        }
    })
    self.netDegerBtc = self.netDegerUsdt / (currentCoin?.current_price ?? 0.0)
}

```

Şekil 3.10 Ekranda gösterilen cüzdan verilerini güncelleme

## 4. LOGIN EKRANI

Veri ve bu verilerle gerçekleştirilen işlemleri detaylıca gösterdikten sonra artık tasarımların kodlarını gösterebiliriz. İlk önce login ekranı ile başlayabiliriz. Login ekranının tasarımı önceki bölümlerde gösterilmişti. Bu bölümde login ekranının arkasında yer alan veri tabanı işlemlerini göstermek istiyorum (Şekil 3.11).

```
FirebaseAuth.Auth.auth().createUser(withEmail: email, password: password) { result, error in
    if error == nil {
        print("Hata Yok")
        var db: Firestore!
        let settings = FirestoreSettings()
        Firestore.firestore().settings = settings

        db=Firestore.firestore()
        db.collection("AllUsers").document(email).setData([
            "name" : name,
            "soyisim" : soyisim,
            "email" : email,
            "username" : kullaniciName,
            "phoneNumber" : phoneNumber,
            "favoriCoin" : emptyFavoriCoin
        ]) { err in
            if let err = err {
                print("Error writing document: \(err)")
                alertType = .error
            } else {
                print("Document successfully written!")
                alertType = .success
                showAlert.toggle()
            }
        }
    }
}
```

Şekil 4.1 Kayıt olma

Şekil 4.1'de görüldüğü gibi ilk önce girilmiş olan e-posta ve şifre ile bir kimlik doğrulama oluşturulmaktadır. Bu oluşum bir sonuç ve hata döndürecektir. Eğer hata yoksa kimlik doğrulama oluşturulmuştur ve bundan sonraki işlemlerini gerçekleştirebiliriz. Hata varsa da kimlik doğrulama oluşturulmayacaktır.

Kimlik doğrulama işlemi başarılı olduktan sonra kullanıcıyı veri tabanımıza ekleyebiliriz. Bunun için ilk önce bir firestore nesnesi oluşturuyoruz db adında. Ardından db üzerinden metodlar ile kullanıcının kayıt formuna girdiği verileri ekliyoruz. Hata olmazsa bir doğrulama mesajı, olduysa da bir hata mesajı döndürecek olan alert'ları ekliyoruz. Şekil 4.2'de alertların döndüreceği mesajların yer aldığı fonksiyonlar gösterilmiştir.

```
func getAlert() -> Alert {
    switch alertType {
        case .error:
            return Alert(title: Text("Kayıt Olunamadı!"))
        case .success:
            return Alert(title: Text("Basariyla Kayıt Olundu!"), message: nil, dismissButton: .default(Text("OK")))
        default:
            return Alert(title: Text("ERROR"))
    }
}
```

Şekil 4.2 Alert fonksiyonu

Kaydol alanında view görüntüsü şekilde 4.3'deki gibidir.

```

private var kayitOlTumTextler : some View{
    ScrollView{
        VStack(alignment: .leading, spacing: 6){
            VStack(alignment: .leading, spacing: 6){
                Text("Adınızı Giriniz: ")
                    .font(.caption)
                    .foregroundColor(.gray)
                    .padding(.leading)
                TextField(" Adınız", text: $name)
                    .background(Color.white.opacity(0.1).cornerRadius(5))
                    .textFieldStyle(.roundedBorder)
                    .autocapitalization(.none)
                    .padding(.init(top: 10, leading: 10, bottom: 0, trailing: 0))
                Text("Soyadınızı Giriniz: ")
                    .font(.caption)
                    .foregroundColor(.gray)
                    .padding(.leading)
                TextField(" Soyadınız", text: $soyisim)
                    .background(Color.white.opacity(0.1).cornerRadius(5))
                    .textFieldStyle(.roundedBorder)
                    .autocapitalization(.none)
                    .padding(.init(top: 10, leading: 10, bottom: 0, trailing: 0))
            }

            Text("Kullanıcı Adı Giriniz: ")
                .font(.caption)
                .foregroundColor(.gray)
                .padding(.leading)
            TextField(" Kullanıcı adı", text: $kullaniciName)
                .background(Color.white.opacity(0.1).cornerRadius(5))
                .textFieldStyle(.roundedBorder)
                .autocapitalization(.none)
                .padding(.init(top: 10, leading: 10, bottom: 0, trailing: 0))
            Text("E-posta Giriniz:")
                .font(.caption)
                .shadow(color: .red, radius: 10, x: 10, y: 10)
                .foregroundColor(.gray)
                .padding(.leading)
            TextField("E-posta", text: $email)
                .frame(width: .infinity, height: 40)
                .textFieldStyle(.roundedBorder)
                .autocapitalization(.none)
                .padding(.init(top: 10, leading: 10, bottom: 0, trailing: 0))
            Text("Parolayı Giriniz:")
                .font(.caption)
                .shadow(color: .red, radius: 10, x: 10, y: 10)
                .foregroundColor(.gray)
                .autocapitalization(.none)
                .padding(.leading)
            SecureField("Parola", text: $password)
                .frame(width: .infinity, height: 40)
                .textFieldStyle(.roundedBorder)

```

Şekil 4.3 Kayıt ol ekranı görünümü

Kullanıcı kaydolduktan sonra artık giriş yapabileceği sayfaya yönlendirilebilecektir. Bu sayfanın kodları şekil 4.4'teki gibidir. Tıpkı kaydol da gösterdiğimiz doğrulama oluşturma işlemi gibi burada da doğrulama ile giriş yap bölümü gösterilmiştir. Kullanıcı kayıtlı bir veri ile giriş yaptıktan sonra program içerisinde bir global email değişkenine kullanıcının giriş yaparken kullandığı email atanacaktır. Bu atama işlemi oldukça önemlidir çünkü bundan sonra gerçekleştirilecek olan veri tabanı işlemleri kullanıcının email'i ile yapılacaktır.

```

if girisYap == false {
  Profile
  passwordEntry
  VStack{
    Button {
      } label: {
        Text("Parolamı Unuttum!")
        .padding()
      } // .buttonStyle(.borderedProminent)
      // .controlSize(.mini)
    ZStack{
      RoundedRectangle(cornerRadius: 15)
        .frame(width: .infinity, height: 50)
        .foregroundColor(Color("Renk1"))
        .padding(.init(top: 0, leading: 10, bottom: 0, trailing: 10))

      Button {
        FirebaseAuth.Auth.auth().signIn(withEmail: email, password: password) { result, error in

          if error == nil {
            print("Basariyla Giris Yapildi")
            girisYapildiMi.giris=1
            globalEmail=email
          }
        }
      } label: {
        Text("Oturumu Aç")
        .foregroundColor(.white)
        .font(.title)
        // .frame(width: .infinity, height: 59)
      }
    }
  }
}

```

Şekil 4.4 Giriş yap

Aynı zamanda şekil 4.4'te girisYapildiMi tipinde bir nesne görülmektedir (Şekil 4.5). Bu nesnenin giriş adında bir değişkeni mevcuttur ve eğer giriş başarılı ise bu değişkenin değeri 1 yapılmıştır. Bu işlemin amacı çok basit ve temeldir. Bir kontrol değişkeni gibidir. Eğer giriş yapıldıysa kullanıcı artık diğer sayfaları görebilecek izne sahip olacaktır. Bu izin kontrolünü bir başka view üzerinde yapmaktayız.

```

17 class girisYapildi : ObservableObject {
18     @Published var giris = 0
19
20     @Published var girisYapildiEmail = " "
21 }

```

Şekil 4.5 GirisYapildi Class'ı

TabBarView adlı view'da kullanıcı giriş yaptıktan sonra görebileceği ekranlar yer almaktadır. Bu ekranları görüp göremeyeceği de giriş yapıp yapmamasına bağlıdır. Bu çağrışımı şekil 4.6'da yapmaktayız.

```

struct TabBarView: View {
  @ObservedObject var girisYapildiMi : girisYapildi = girisYapildi()
  var body: some View {
    VStack{
      if globalEmail == " "{
        LoginView(girisYapildiMi: girisYapildiMi)
      }
      else {
        TabView {
          HomePage()
          .tabItem {
            Image(systemName: "house")
            Text("Anasayfa")
          }
        }
      }
    }
  }
}

```

Şekil 3.16 Login Kontrolü

## 5. ANASAYFA

### 5.1. Coin List View

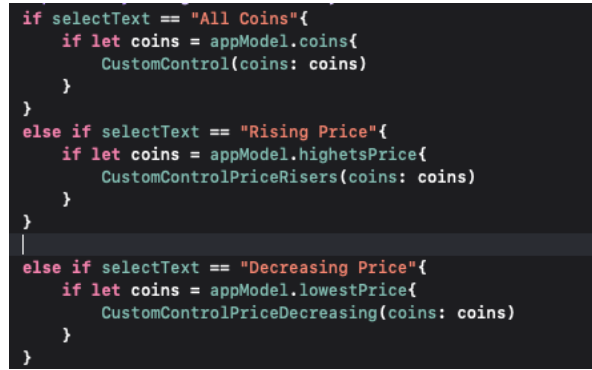
Tasarım bölümünde coinleri liste şeklinde sıralandığını göstermiştik. Bu bölümde o listeyi nasıl kodladığımızı görelim.

Sayfanın başında listeleri seçeceğimiz bir picker görünmekte. IOS'ta picker yapısı çoğu zaman aynı standartta olmaktadır (Şekil 5.1).



Şekil 5.1 Picker

Picker, seçilen text'e göre if kontrolünde olan görüntüleri ekrana getirecektir (Şekil 5.2).



Şekil 5.2 If kontrolleri

Fiyatları listeleyecek olan fonksiyonlardan biri Şekil 5.3'de CustomControl fonksiyonudur. Bu fonksiyon parametre olarak CryptoModel array'i almakta ve ekrana bir view döndürmektedir. Fonksiyon içerisinde ScrollView, VStack ve HStack yapıları görünmektedir. Bu yapılar ekranda görünen daha küçük view elemanlarını bir yapı içerisine sokmaya olanak sağlar. VStack, elemanları dikey olarak alt alt konumlandırır. HStack, yatay olarak elemanları konumlandırır. ScrollView ise yapı içindeki elemanları dikey ya da yatay olarak kaydırılabilir şekilde yerleştirir.



```

@ViewBuilder
func CustomControl(coins: [CryptoModel]) -> some View {
    ScrollView(.vertical) {
        VStack(alignment: .leading, spacing: 25) {
            HStack {
                Text("Coin")
                    .font(.caption)
                    .foregroundColor(Color.gray)
                    .padding(.leading, 10)
                Spacer()
                Text("Price")
                    .font(.caption)
                    .foregroundColor(Color.gray)
                    .padding(.trailing, 10)
            }
            ForEach(coins) { coin in
                VStack(alignment: .leading, spacing: 10) {
                    HStack {
                        AnimatedImage(url: URL(string: coin.image))
                            .resizable()
                            .aspectRatio(contentMode: .fit)
                            .frame(width: 40, height: 40)

                        VStack(alignment: .leading, spacing: 5) {
                            Button {
                                } label: {
                                    NavigationLink(destination: OneCoinGraph(currentCoin: coin.name)) {
                                        Text("\(coin.name)")
                                            .font(.callout)
                                            .foregroundColor(Color.white)
                                    }
                                }
                            HStack {
                                Text("\(coin.symbol.uppercased())")
                                    .font(.caption)
                                    .foregroundColor(Color.gray)
                                    // .foregroundColor(Color("DahaBuyuk"))
                                Text("\(coin.price_change > 0 ? "+" : "")\((String(format: "%.2f",
                                    coin.price_change)))")
                                    .foregroundColor(coin.price_change > 0 ? .green : .red)
                                    .font(.caption)
                            }
                        }
                    }
                    Spacer()
                    Text("\($")\((coin.current_price, specifier: "%.2f")")
                        .bold()
                        .foregroundColor(coin.price_change > 0 ? .green : .red)
                        // .foregroundColor(Color.white)
                }
            }
        }
    }
}

```

Şekil 5.3 Coin Bilgilerini dikey olarak ekrana yazdırma

Benzer fonksiyonlar fiyatı en çok düşen ve artan şeklinde ekranda listeleyecek şekilde yazılmıştır (Şekil 5.4).

Bu fonksiyonlarda farklı olarak fiyat bilgisi if kontrolü içerisinde yapılmıştır. AppViewModel'de zaten sıraladığımız için burada sadece sıfırdan büyükleri ve küçükleri göstermekte yeterli olmuştur.

```

@ViewBuilder
func CustomControlPriceRisers(coins : [CryptoModel]) -> some View {
    ScrollView(.vertical){
        VStack(alignment: .leading, spacing: 25){
            HStack{
                Text("Coin")
                    .font(.caption)
                    .foregroundColor(Color.gray)
                    .padding(.leading,10)
                Spacer()
                Text("Price")
                    .font(.caption)
                    .foregroundColor(Color.gray)
                    .padding(.trailing,10)
            }

            ForEach(coins){ *** }
        }
    }
}

@ViewBuilder
func CustomControlPriceDecreasing(coins : [CryptoModel]) -> some View {
    ScrollView(.vertical){
        VStack(alignment: .leading, spacing: 25){
            HStack{
                Text("Coin")
                    .font(.caption)
                    .foregroundColor(Color.gray)
                    .padding(.leading,10)
                Spacer()
                Text("Price")
                    .font(.caption)
                    .foregroundColor(Color.gray)
                    .padding(.trailing,10)
            }

            ForEach(coins){ *** }
        }
    }
}

```

Şekil 5.4 Fiyat bilgisine göre listeleme

## 5.2 Popüler Coin View

Bu ekranda Popüler olan coinler yatay bir şekilde görüntüleyecek olan view yazılmıştır (Şekil 5.5).

```

struct PopulerCoinView: View {
    let favoriCoinName : [String : String] = ["btc":"Bitcoin", "eth":"Ethereum", "bnb":"BNB"]
    @StateObject var appModel: AppViewModel = AppViewModel()

    var body: some View {
        HStack{
            if let coins = appModel.coins, let coin = appModel.currentCoin{
                CustomControl(coins: coins)
            }
        }
    }
}

@ViewBuilder
func CustomControl(coins: [CryptoModel]) -> some View{ *** }

struct PopulerCoinView_Previews: PreviewProvider {
    static var previews: some View {
        PopulerCoinView()
            .preferredColorScheme(.dark)
    }
}

```

Şekil 5.5 Popüler coinlerin görüntüsü

### 5.3 Hareketli Haber Fotoğrafları

Bu ekranda 5 tane haber fotoğrafı yer almaktadır. Fotoğraflar TabView yapısı sayesinde 6 saniyede bir değişecek şekilde yazılmıştır (Şekil 5.6).

```
55 var body: some View {
56   VStack{
57     NavigationLink(destination: NewsView(), isActive: self.$isActive_one){
58       TabView(selection: $currentIndex){
59         ForEach(0..
```

Şekil 5.6 Hareketli Haber Fotoğrafları

Bu fotoğraflar aynı zamanda tıklanabilir yapılmıştır. Her fotoğraf farklı bir haber sayfasına yönlendirecektir (Şekil 5.7).



Şekil 5.7 Haber sayfası

## 6. COIN BİLGİ EKRANI

Bu ekranı tasarım bölümünde görmüştük. Burada tasarımın arkasındaki bazı kod satırlarını göreceğiz. Göstermek istediğim ilk şey favorilere ekleme butonu olacak. Bu buton kullanıcının bilgilerinin yer aldığı veri tabanına ekleme ve çıkarmalara yapacaktır (Şekil 6.1).

```
Button {
    var db: Firestore!
    let settings = FirestoreSettings()
    Firestore.firestore().settings = settings

    db=Firestore.firestore()

    let favCoin = db.collection("AllUsers").document(globalEmail)
    if globalEmail == " "{
    }
    else {
        if viewModel.isCoinFav == false { // başlangıçtaki coin, favoriler arasında değilse eklemek için
            // Atomically add a new region to the "regions" array field.
            favCoin.updateData([
                "favoriCoin": FieldValue.arrayUnion([currentCoin]) // favori coinler dizisine coinin ismini ekleyecektir
            ])
            viewModel.favorilerArasinaEklendi() // button simgesini değiştirmek için çağrılır
            alertType = .success
            showAlert.toggle()
        }
        else { // başlangıçtaki coin, favoriler arasındaysa ve silmek istiyorsak
            favCoin.updateData([
                "favoriCoin": FieldValue.arrayRemove([currentCoin]) // favori coinler dizisinden coinin ismini silecektir
            ])
            viewModel.favorilerArasindaCikarildi() //button simgesini değiştirmek için çağrılır
            alertType = .error
            showAlert.toggle()
        }
    }
}
```

Şekil 6.1 Favorilere coin ekleme ve çıkarma

Şekil 6.2'da ekranda gösterilen coin bilgilerinin ekrana nasıl yazdırıldığı gösterilmiştir. Coin tipindeki nesne ile ilgili değişkenleri çağırıp ekrana yazdırabilmekteyiz. Ekrana yazdırma için Swift'te yer alan Text yapısını kullanıyoruz. Text yapısının birçok özelliği bulunmaktadır.

Buna göre text'in rengini, tipini, büyüklüğünü, arka planını gibi birçok özelliğini değiştirebilmekteyiz.

```

@ViewBuilder
func CoinDetailTextView(coin : CryptoModel) -> some View{
    VStack(alignment: .leading, spacing: 12){
        HStack{
            Text("Anlık Fiyat: ")
                .foregroundColor(.gray)
            Text("\("$")\$(coin.current_price, specifier: "%.2f")")
                .bold()
                //.foregroundColor(Color("Renk1"))
        }
        HStack{
            Text("En Yüksek: ")
                .foregroundColor(.gray)
            Text("\("$")\$(coin.high24H!, specifier: "%.2f")")
                .bold()
                //.foregroundColor(Color("Renk1"))
        }
        HStack{
            Text("En Düşük: ")
                .foregroundColor(.gray)
            //Text("\("$")\$(coin.currentPrice, specifier: "%.2f")")
            Text("\("$")\$(coin.low24H!, specifier: "%.2f")")
                .bold()
                //.foregroundColor(Color("Renk1"))
        }
        HStack{
            Text("Total Volume: ")
                .foregroundColor(.gray)
            Text("\("$")\$(coin.totalVolume!, specifier: "%.2f")")
                .bold()
                //.foregroundColor(Color("Renk1"))
        }
        HStack{
            Text("Değişim Miktarı: ")
                .foregroundColor(.gray)
            Text("\("$")\$(coin.priceChange24H!, specifier: "%.2f")")
                .bold()
                //.foregroundColor(Color("Renk1"))
        }
        HStack{
            Text("Son Güncelleme")
                .foregroundColor(.gray)
            Text(coin.last_updated)
                .bold()
                //.foregroundColor(Color("Renk1"))
        }
    }
}

```

Şekil 6.2 Coin Detaylarının Ekrana Yazdırılması

## 7. CÜZDAN EKRANI

Bu ekranda kullanıcının cüzdanında yer alan coinlerin bilgilerini ekranda listeledik. Ekranın başında kullanıcının sahip olduğu para miktarını BTC ve USDT değerinde gösterdik (Şekil 3.27). Hemen altına para yatırma, çekme ve transfer gibi işlemleri gerçekleştirecek butonlar eklendi (Şekil 3.28). Bu butonların altına da cüzdandaki coinler yazdırıldı. Yazdırma işlemi tıpkı önceki bölümlerde gösterildiği gibi yapıldı.

```
HStack{
  VStack{
    Text("Net Deger(BTC)")
      .font(.body)
      .foregroundColor(.gray)
    Text("\${appModel.netDegerBtc, specifier: "%.2f"}")
      .font(.system(size: 20))
      .padding(.top, 2)
      .foregroundColor(Color("Renk1"))
  }.padding(.leading, 20)
  Spacer()
  VStack{
    Text("Net Deger(USDT)")
      .font(.body)
      .foregroundColor(.gray)
    Text("\${"$"}\${appModel.netDegerUsdt, specifier: "%.3f"}")
      .font(.system(size: 20))
      .padding(.top, 2)
      .foregroundColor(Color("Renk1"))
  }.padding(.leading, 20)
}.padding(.top, 20)
```

Şekil 3.27 Net Cüzdan Değerleri

```
HStack{
  Spacer()
  Button {
    } label: {
      Text("Yatirma")
        .fontWeight(.bold)
        .foregroundColor(.black)
        .frame(maxWidth: .infinity)
        .frame(height: 8)
        .padding(.vertical)
        .background{
          RoundedRectangle(cornerRadius: 12, style: .continuous)
            .fill(Color("LightGreen"))
        }
    }
  }
  Button {
    } label: {
      Text("Çekme")
        .fontWeight(.bold)
        .foregroundColor(.black)
        .frame(maxWidth: .infinity)
        .frame(height: 8)
        .padding(.vertical)
        .background{
          RoundedRectangle(cornerRadius: 12, style: .continuous)
            .fill(Color.white)
        }
    }
  }
  Button {
    } label: {
      Text("Transfer")
        .fontWeight(.bold)
        .foregroundColor(.black)
        .frame(maxWidth: .infinity)
        .frame(height: 8)
        .padding(.vertical)
        .background{
          RoundedRectangle(cornerRadius: 12, style: .continuous)
            .fill(Color.white)
        }
    }
  }
  Spacer()
}.padding()
```

Şekil 3.28 Yatırma, Çekme ve Transfer Butonlarının Görünümü

```

VStack(spacing: 7){
  ForEach(appModel.myWalletCoin){ coin in
    HStack(spacing: 15){
      HStack(){
        AnimatedImage(url: URL(string: coin.image))
          .resizable()
          .aspectRatio(contentMode: .fit)
          .frame(width: 40, height: 40)

        VStack(alignment: .leading, spacing: 5) {
          Button {

            } label: {
              NavigationLink(destination: OneCoinGraph(currentCoin: coin.name)) {
                Text(coin.name)
                  .font(.callout)
                  .foregroundColor(Color.white)
              }
            }
          Text(coin.symbol.uppercased())
            .font(.caption)
            .foregroundColor(Color.gray)
            //foregroundColor(Color("DahaBuyuk"))
        }
      }.padding()

      Spacer()

      VStack(alignment: .trailing, spacing: 5){
        Text("\("$")\((coin.coinPiece * coin.currentPrice, specifier: "%.2f")")
        Text("\((coin.coinPiece, specifier: "%.3f")")
          .font(.caption)
      }
    }
  }
}

```

Şekil 3.29 Cüzdandaki coinlerin listelenmesi

## 8. ALIM- SATIM EKRANI

Bu ekranda kullanıcı alım- satım işlemlerini gerçekleştirebilmektedir. Diğer view'lerden yapı olarak daha karışıktır ve toplamda 650 satırdan oluşmaktadır. Bu bölümde önemli olan kısımlar gösterilecektir.

View'in yapısı iki if kontrolü temelinde oluşmaktadır. Kullanıcı picker ile al ya da sattan birini seçtiğinde sayfa ona göre değişecektir. Sayfada değişmeyen öğeler de bulunmaktadır. Coinin ismi, anlık fiyatı, resmi ve mevcut coin değıştirmek için buton yer almaktadır. İlk önce bu butona değinelim. Buton'a dokunulduğunda ekranın altından bir view açılacaktır. Bu view ekranın hepsini kapatmayacaktır. Hem daha estetik hem de kullanımı daha kolay olacaktır bu şekilde. Kullanıcı bu alttan açılan view'i dokunmatik ekran ile rahatça aşağıya kaydırarak kapatabilmektedir. Hız açısından da artık bir özellik katmaktadır bu özellik.

Alttan açılan bu ikinci ekranda coinler listene ektir. Listenin başında da arama ubuęu olacaktır (Şekil 3.30).



Şekil 8.1 Arama ubuęu

Arama ubuęunu aşağıdaki gibi bir değışken şeklinde oluşturduk. Bu değışken CryptoModel tipindeki bir array'den oluşmaktadır. Aramalarımızı bunun üzerinden yapacağız (Şekil 8.2).

```
var searchResults : [CryptoModel] {
    if searchText.isEmpty{
        return appModel.coins!
    }
    else {
        return appModel.coins!.filter { CryptoModel in
            CryptoModel.name.contains(searchText)
        }
    }
}
```

Şekil 8.2 Arama ubuęuna Yazdırılacak olan değışken

ForEach yapısına parametre olarak üstte oluşturduğumuz arama değışkenini gönderdik (Şekil 8.3). Eğer bu değışken boş ise tüm coinler listelenecek değilse yazılan isme göre listeleme yapılacaktır.



```

ForEach(searchResults){coin in
    VStack(alignment: .leading, spacing: 10){
        HStack(){
            AnimatedImage(url: URL(string: coin.image))
                .resizable()
                .aspectRatio(contentMode: .fit)
                .frame(width: 40, height: 40)

            VStack(alignment: .leading, spacing: 5) {
                Button {
                    appModel.chosenCoin = coin
                    viewModel.coinName = appModel.chosenCoin?.name ?? " "
                    viewModel.SellBuyCoinsData()
                    presentationMode.wrappedValue.dismiss()

                } label: {
                    Text("\(coin.name)")
                        .font(.callout)
                        .foregroundColor(Color.white)
                }
                HStack{
                    Text("\(coin.symbol.uppercased())")
                        .font(.caption)
                        .foregroundColor(Color.gray)
                        //.foregroundColor(Color("DahaBuyuk"))
                    Text("\(coin.price_change > 0 ? "+" : "")\((String(format:
                        "%.2f", coin.price_change)))"
                        .foregroundColor(coin.price_change > 0 ? .green : .red)
                        .font(.caption)
                }
            }
        }
        Spacer()
        Text("\(("$")\((coin.current_price, specifier: "%.2f")")"
            .bold()
            .foregroundColor(coin.price_change > 0 ? .green : .red)
            //.foregroundColor(Color.white)
        )
    }
}

```

Şekil 8.3 Arama Ekranı Listesi

Şekil 8.4'te coin isimleri tıklanabilir bir butona dönüştürülmüştür. Tıklandığında da al-sat yapılacak coin değiştirilecektir. Aynı zamanda alttan açılan bu ekran da tıklanma sonrası kapatılacaktır bu buton sayesinde (Şekil 8.4).

```

Button {
    appModel.chosenCoin = coin
    viewModel.coinName = appModel.chosenCoin?.name ?? " "
    viewModel.SellBuyCoinsData()
    presentationMode.wrappedValue.dismiss()

} label: {
    Text("\(coin.name)")
        .font(.callout)
        .foregroundColor(Color.white)
}

```

Şekil 8.4 Al sat yapılacak coinin değiştirilmesi

```

HStack{
  Button {
    showSheet.toggle()
  } label: {
    Image("icons8-change-48 (1)")
      .resizable()
      .scaledToFit()
      .frame(width: 40, height: 40)
      .foregroundColor(.white)
      //.background(Color.gray)
  }.sheet(isPresented: $showSheet, content: {
    // DO NOT ADD CONDITIONAL LOGIC
    SecondScreen(coinName: appModel.chosenCoin?.name ?? " ", appModel: appModel, viewModel:
      viewModel)
  })
}

```

Şekil 8.5 Alttan açılacak olan ekranın buton ile çağırılması

Alım satım işlemlerinde kullanıcının miktar gireceği alanlar şekilde 8.6'de gösterilmiştir. Textfield'lar bir dikey bir stack içerisinde tutulacak şekilde tasarımı hazırlanmıştır. Bu tasarımın içerisinde text girilecek alan ve üzerinde o text alanın açıklama etiketi yer almaktadır. Bu noktada asıl önemli olan text alanlarının beraber dinamik biçimde çalışmasıdır. Özetle, kullanıcı 1 bitcoin almak için text alanına 1 sayısını girdiğinde toplam usdt text alanında 1 bitcoinin güncel fiyatını göstermesidir.

```

VStack{
  VStack(alignment: .leading, spacing:6){
    Text("Fiyat")
      .font(.caption)
      .padding(.leading,10)
    TextField("\${appModel.chosenCoin?.current_price as? Double ?? 0.0}, specifier:
      "%.2f\"", text: $price)
      .keyboardType(.numberPad)
      .background(Color.white.opacity(0.1).cornerRadius(5))
      .textFieldStyle(.roundedBorder)
      .autocapitalization(.none)
      .multilineTextAlignment(.center)
  }
  VStack(alignment: .leading, spacing:6){
    Text("Miktar")
      .font(.caption)
      .padding(.leading,10)
    TextField("Miktar \("${(coinName)}"): ", value: $miktarCoin, format:.number)
      .onChange(of: miktarCoin) {
        self.toplamUsdt = self.miktarCoin * (appModel.chosenCoin?.current_price ?? 1)

        print($0) // You can do anything due to the change here.
        // self.autocomplete($0) // like this
      }
      .keyboardType(.numberPad)
      .background(Color.white.opacity(0.1).cornerRadius(5))
      .textFieldStyle(.roundedBorder)
      .autocapitalization(.none)
      .multilineTextAlignment(.center)
  }
}

```

Şekil 8.6 Text Alanlarının hazırlanması

Kullanıcılar alım satım yaparken satın almak istediği miktarı değiştirmek istediği zaman genelde girdiği miktarı orantılı bir şekilde değiştirmek ister. Bir miktar parasının yüzde ellisi ile alım yapar ya da yüzde 25'ini satmak ister. Bunu kolaylaştırmak adına 4 tane yüzdellik ifade içeren butonlar eklendi. Bu butonlar girilen miktarın yüzdesini alıp text alanlarını değiştirir (Şekil 8.7).

```

HStack{
  Spacer()
  Button {
    self.miktarCoin = (toplamUsdt / appModel.chosenCoin!.current_price as?
      Double ?? 0.0) / 4
  } label: {
    Text("%25")
    .font(.body)
  }
  .foregroundColor(Color("Renk1"))
  Spacer()
  Button {
    self.miktarCoin = (toplamUsdt / appModel.chosenCoin!.current_price as?
      Double ?? 0.0) / 2
  } label: {
    Text("%50")
    .font(.body)
  }.foregroundColor(Color("Renk1"))
  Spacer()
  Button {
    self.miktarCoin = (((toplamUsdt / appModel.chosenCoin!.current_price as?
      Double ?? 0.0) * 3) / 4)
  } label: {
    Text("%75")
    .font(.body)
  }.foregroundColor(Color("Renk1"))
  Spacer()
  Button {
    self.miktarCoin = (toplamUsdt / appModel.chosenCoin!.current_price as?
      Double ?? 0.0) / 1
  } label: {
    Text("%100")
    .font(.body)
  }.foregroundColor(Color("Renk1"))
}

```

Şekil 8.7 Yüzdelik Butonlar

Gerekli tüm hazırlıklar yapıldıktan sonra al ve sat butonlarına geçebiliriz. Bu iki buton temelde şu işleri gerçekleştirmek üzere hazırlandı; miktarı kontrol et, miktar gerekli şartları sağlıyorsa (kullanıcının cüzdanında yeteri kadar mevcutsa) sat ya al işlemini gerçekleştir. Bu işlemi ara yüzde gerçekleştirdiği gibi veri tabanında da gerçekleştirecektir (Şekil 8.8).

Eğer alım ya da satım işlemi gerçekleşmeyecekse bir durum varsa o zaman da bir hata mesajı döndürecek kullanıcıya. Kullanıcı bu konuda bilgilendirecektir.

Veri tabanı işlemleri için önceki bölümlerde firebase kullandığımızı söylemiştik. Firebase birçok açıdan bize kolaylık sağlamaktadır. Metotlar sade ve net bir şekilde dökümente edilmiştir.

```

Button {
    if viewModel.usdtPieces >= toplamUsdt && miktarCoin != 0 {
        if viewModel.usdtPieces >= miktarCoin{
            viewModel.oneWalletPieces = miktarCoin + viewModel.oneWalletPieces
            viewModel.usdtPieces = viewModel.usdtPieces - toplamUsdt

            toplamUsdt = 0
            miktarCoin = 0

            alertType = .success
            showAlert.toggle()
            let washingtonRef = db.collection("AllUsersWallet").document(globalEmail)

            // Set the "capital" field of the city 'DC'
            washingtonRef.updateData([
                appModel.chosenCoin?.name ?? " ": viewModel.oneWalletPieces,
                "USDT" : viewModel.usdtPieces
            ]) { err in
                if let err = err {
                    print("Error updating document: \(err)")
                } else {
                    print("Document successfully updated")
                }
            }
        }
    }
    else if toplamUsdt > viewModel.usdtPieces {
        alertType = .error
        showAlert.toggle()
    }
    else if miktarCoin == 0 && toplamUsdt == 0 {
        alertType = .error
        showAlert.toggle()
    }
} label: {
    Text("Buy")
        .fontWeight(.bold)
        .foregroundColor(.black)
        .frame(maxWidth: .infinity)
        .padding(.vertical)
        .background{

```

Şekil 8.8 Satın al butonu

## 9. FAVORİ COİNLER EKRANI

Kullanıcının yakında takip etmesine kolaylık sağlamak adına favori coinlerini oluşturabileceğimizi göstermiştik önceki bölümlerde. Şekil 3.40'ta da kullanıcının favorilerine aldığı coinleri görebileceği view'in kodları gösterilmiştir (Şekil 3.40).

```
36     var body: some View {
37         NavigationView{
38             VStack{
39                 if selectButton == "Favori Coinlerim" {
40                     if let coins = appModel.coins, let coin = appModel.currentCoin{
41                         CustomControl(coins: coins, favCoin: viewModel.coinFav)
42                             .padding()
43                     }
44                 }
45                 Spacer()
46             }.navigationTitle("Favori Coinlerim")
47             .onAppear{
48                 self.viewModel.veriYakalama()
49             }
50         }
51     }
```

Şekil 9.1 Favori coinlerin view'inin ana yapısı

CustomControl fonksiyonun içeriği şekil 9.2'de gösterilmiştir. Bu bölümde farklı olarak tüm coinleri ekrana basmak yerine seçilmiş olan yani favorileri ekrana basmaktadır. Favori coinlerin isimleri veri tabanından çekilmektedir.

```
86 @ViewBuilder
87 func CustomControl(coins: [CryptoModel], favCoin : [String])->some View{
88     ScrollView{
89         VStack(spacing: 15){
90             ForEach(coins){coin in
91                 ForEach(favCoin, id: \.self){ tag in
92                     if tag == coin.name{
93                         HStack(spacing: 20){
94                             HStack(){
95                                 AnimatedImage(url: URL(string: coin.image))
96                                     .resizable()
97                                     .aspectRatio(contentMode: .fit)
98                                     .frame(width: 40, height: 40)
99                             }
100                             VStack(alignment: .leading, spacing: 5) {
101                                 Button {
102                                     } label: {
103                                         NavigationLink(destination: OneCoinGraph(currentCoin: coin.name)) {
104                                             Text(coin.name)
105                                                 .font(.callout)
106                                                 .foregroundColor(Color.white)
107                                         }
108                                     }
109                                 Text(coin.symbol.uppercased())
110                                     .font(.caption)
111                                     .foregroundColor(Color.gray)
112                                     //foregroundColor(Color("DahaBuyuk"))
113                             }
114                         }
115                     }
116                 }
117             }
118             Spacer()
119             Text("\($")\((coin.current_price, specifier: "%.2f")")
120                 .bold()
121                 .foregroundColor(coin.price_change > 0 ? .green : .red)
122                 //foregroundColor(Color.white)
123             ZStack{
124                 RoundedRectangle(cornerRadius: 10)
125                     .fill(Color("AltBaslik"))
126                 Text("\((coin.price_change > 0 ? "+" : "")\((String(format: "%.2f",
127                     coin.price_change)))")
128                     .foregroundColor(coin.price_change > 0 ? .green : .red)
129                     .bold()
130                     .background(Color("AltBaslik"))
131                 }.frame(width: 50, height: 40)
132             }
133         }
134     }
135 }
```

Şekil 9.2 Favori coinleri yazdıracak olan fonksiyon

## 10. KULLANICI BİLGİLERİ EKRANI

Üyelik sistemi ile giriş yapılan bir uygulamada kullanıcı kendi bilgilerini görmek isteyecektir. Bunu sağlamak amacıyla bu view oluşturuldu. Bu view'de kullanıcının kaydolurken girdiği bilgiler ekranda gösterilecektir. Bilgiler veri tabanından çekilip bir kullanıcı modeline atanacaktır (Şekil 10.1).

```
14 struct myUserModel : Identifiable{
15     var id : String = UUID().uuidString
16     var email : String
17     var name : String
18     var phoneNumber : String
19     var soyisim : String
20     var usurname : String?
21     var favoriCoin : [String] = []
22 }
```

Şekil 10.1 Kullanıcı Modeli

Kullanıcı modelini oluşturduktan sonra veri tabanında verileri çekebiliriz. Bunun için firebase'nin sunduğu metotları kullandık. Bu metotları kullanarak şekil 10.2'deki gibi bir fonksiyon yazdık. Fonksiyonda görüldüğü üzere bir sorgu ve bu sorgunun döndürdüğü anlık bir görüntü ve hata mesajı bulunmaktadır. Verileri bu anlık görüntüden alacağız biz.

```
81 func veriYakalama() {
82     getArray()
83     print(myFavoriCoin)
84     db.collection("AllUsers").whereField("email", isEqualTo: globalEmail)
85     .addSnapshotListener { querySnapshot, error in
86         guard let documents = querySnapshot?.documents else {
87             print("Error fetching documents: \(error!)")
88             return
89         }
90         self.myUser = documents.map { queryDocumentSnapshot -> myUserModel in
91             let data = queryDocumentSnapshot.data()
92             let email = data["email"] as? String ?? ""
93             let name = data["name"] as? String ?? ""
94             let phoneNumber = data["phoneNumber"] as? String ?? ""
95             let soyisim = data["soyisim"] as? String ?? ""
96             let usurname = data["usurname"] as? String ?? ""
97             self.coinFav = data["favoriCoin"] as? Array ?? []
98
99             /*let coin = data["favoriCoin"].map { indis in
100                 self.favoriCoin.append(indis as! String)
101             } */
102             return myUserModel(id: .init(), email: email, name: name, phoneNumber: phoneNumber, soyisim: soyisim, usurname: usurname, favoriCoin: self.coinFav)
103         }
104     }
105 }
106 }
```

Şekil 10.2 Kullanıcı verilerini veritabanından çekme

Her bir veri için farklı bir değişken oluşturup veri tabanındaki key'lerin değerlerini onlara atadık. Atama işlemi bittikten sonra bu verileri oluşturduğumuz kullanıcı modeli nesnesine gönderdik. Böylece bir kullanıcı modelimiz ve onun değerlerini atayabilmiş olduk.

Kullanıcı modeli oluşturma ve buna veri çekme işleminden sonra geriye kalan bu verileri kullanıcıya gösterecek view'i oluşturmak kaldı. Bunu da şekil 10.3'te yaptık. Hem güzel bir tasarım oluşturduk hem de verileri uygun bir şekilde gösterebildik (Şekil 10.4).

```

15 struct UserInformation: View {
16     @ObservedObject var viewModel = kullanıcıModeliniAlma()
17
18     var body: some View {
19         NavigationView{
20             VStack(alignment: .center, spacing: 15){
21                 Text("Burada kişisel bilgilerin yer almaktadır. Bu kısmı sadece sen görüntüleyebilirsin. ")
22                     .font(.system(size: 20))
23                     .padding()
24                     .clipShape(RoundedRectangle(cornerRadius: 5))
25
26                 List(viewModel.myUser){ alluser in
27                     HStack(spacing: 20){
28                         Text("İsim: ")
29                             .font(.system(size: 20))
30                             .padding(.leading,10)
31                             .foregroundColor(Color("Renk1"))
32
33                         Text(alluser.name)
34                             .font(.system(size: 20))
35                     }.padding(.bottom)
36                     HStack(spacing: 20){
37                         Text("Soyisim: ")
38                             .font(.system(size: 20))
39                             .padding(.leading,10)
40                             .foregroundColor(Color("Renk1"))
41                         Text(alluser.soyisim)
42                             .font(.system(size: 20))
43                     }.padding(.bottom)
44                     HStack(spacing: 20){
45                         Text("Kullanıcı Adı: ")
46                             .font(.system(size: 20))
47                             .padding(.leading,10)
48                             .foregroundColor(Color("Renk1"))
49                         Text(alluser.usurname ?? "Bosluk")
50                             .font(.system(size: 20))
51                     }.padding(.bottom)
52                     HStack(spacing: 20){
53                         Text("Email: ")
54                             .font(.system(size: 20))
55                             .padding(.leading,10)
56                             .foregroundColor(Color("Renk1"))
57                         Text(alluser.email)
58                             .font(.system(size: 20))
59                     }.padding(.bottom)
60
61                     HStack(spacing: 20){
62                         Text("Telefon Numaran: ")
63                             .font(.system(size: 20))
64                             .padding(.leading,10)
65                             .foregroundColor(Color("Renk1"))
66                         Text(alluser.phoneNumber)
67                             .font(.system(size: 20))

```

Şekil 10.4 Kullanıcı bilgilerini view'de göstermek

## 11. VERİ TABANI

Firebase, mobil ve web uygulamaları oluşturmak için geliştirilmiş ücretsiz bir platformdur. Firebase, bağımsız bir şirket olarak 2011 yılında kurulmuştur. 2014 yılında da Google tarafından satın alınmıştır.

Firebase; uygulama yönetimi, kullanım takip, depolama, bildirim iletme gibi temel işlemleri sunucu tarafı kod yazmaya ihtiyaç duymadan halleder. Realtime Database, Notification, Remote Config gibi özelliklerle birlikte her uygulama için ayrı ayrı ulaşım imkânı sağlıyor.

Firebase'i bu proje içerisinde kullanıcı verilerini tutma ve doğrulama sistemi için kullandık.

### 11.1 Doğrulama

Firebase authentication, datalarınıza erişim izinlerini kolayca kontrol edebilmenizi sağlamak amacıyla gerçek zamanlı veri tabanı ve depolama ile sorunsuz bir şekilde çalışır.

Şekil 11.1'de sisteme kayıtlı kullanıcılar görülmektedir.

Search by email address, phone number, or user UID					Add user	↺	⋮
Identifier	Providers	Created ↓	Signed In	User UID			
seyfettinkln@gmail.com	✉	May 8, 2022	May 8, 2022	EgbG53v72BR1Gm6HWllb8Sh5uW...			
seyfettinkilinc3@gmail.com	✉	May 8, 2022	May 8, 2022	GSeCXP5IWEM9caBlbXS0dwHbuc...			
zeynepdurmaz@gmail.com	✉	May 1, 2022	May 1, 2022	1zy4en8lmqRPION0LW3FxCNc0aTB2			
cansuozyildiz@gmail.com	✉	Apr 22, 2022	Apr 29, 2022	7QLGbQsS3of9mEfYi1MShiMD1T...			
azilziyan20@gmail.com	✉	Apr 22, 2022	May 27, 2022	ub34LHUdlwOphAx7iX29bkqqN3y1			
mehmetcan@gmail.com	✉	Apr 22, 2022	Apr 22, 2022	0c4WI3Z8UhgPzMSSfqgkZkhe0oB2			
busecaliskan@gmail.com	✉	Apr 22, 2022	Apr 26, 2022	YSxZeWbU5dZFFhJHGx26EQWwff...			
Rows per page: 50					1 – 7 of 7	⏪	⏩

Şekil 11.1 Firebase authentication ile sisteme kayıtlı kullanıcıların tablosu

### 11.2 Kullanıcı Verileri

Kullanıcı bilgileri sisteme kayıt olurken veritabanında kullanıcı e-postası ile oluşturulur. Her bir kullanıcının verilerine o kullanıcının e-postası ile ulaşılabilir. Şekil 6.2'de kullanıcıların tutulduğu veritabanı gösterilmiştir. Burada kullanıcının e-postası, gerçek ismi, soyismi, telefonu, kullanıcı adı ve son olarak favori coinlerinin isimleri yer almaktadır (Şekil 11.2).



mycrypto-8dc87	AllUsers	azilziyan20@gmail.com
<a href="#">+ Start collection</a> AllUrl <b>AllUsers</b> AllUsersWallet	<a href="#">+ Add document</a> 0EawT095Ci0X8AV6uSe1 <b>azilziyan20@gmail.com</b> busecaliskan@gmail.com cansuozyildiz@gmail.com mehmetcan@gmail.com seyfettinkilinc3@gmail.com seyfettinkiln@gmail.com zeynepdurmaz@gmail.com	<a href="#">+ Start collection</a> <a href="#">+ Add field</a> email: "azilziyan20@gmail.com" favoriCoin 0 "XRP" 1 "Render Token" 2 "Horizen" 3 "Bitcoin" name: "Seyfettin" phoneNumber: "5393251416" soyisim: "Kilinc" usurname: "dahazil"

Şekil 11.2 Kullanıcı bilgileri

### 11.3 Cüzdan Verileri

Cüzdan verileri için ayrı bir veritabanı oluşturduk. Bu veritabanında her kullanıcının cüzdanında yer alan coinler ve miktarları tutulmaktadır. Bu cüzdana kullanıcının e-postası ile ulaşabilmekteyiz (Şekil 6.3).

mycrypto-8dc87	AllUsersWallet	azilziyan20@gmail.com
<a href="#">+ Start collection</a> AllUrl AllUsers <b>AllUsersWallet</b>	<a href="#">+ Add document</a> azilziyan20@gmail.com cansuozyildiz@gmail.com deneme	<a href="#">+ Start collection</a> <a href="#">+ Add field</a> BNB: 31.12198980185767 Bitcoin: 8.499983581256362 Bonded Luna: 1.0018785222291797 Cardano: 869.7475521354984 Cosmos Hub: 0 Dai: 0.9990009990009991 Ethereum: 0.017801360554571488 Ethereum Classic: 1.00683207479324 Solana: 0.0018016694804443567 USDT: 150164.05000003224 XRP: 0

Şekil 11.3 Cüzdan veritabanı

#### **11.4 SONUÇ**

Kriptoparalar günümüzün en popüler yatırım araçlarından biri olarak var olmaktadır ve insanların kolay yoldan para kazanma, yatırım yapma gibi isteklerinden ötürü uzunca bir süre daha var olmaya devam edecektir. Mobil uygulamaların da gelişmesiyle birlikte kripto paralar hakkındaki bilgilere telefonumuz ile kolaylıkla ulaşabilmekteyiz. Bu raporda yer alan projede bu konudaki ihtiyacı gidermek adına hazırlanmıştır.

Proje, swift programlama dili yazıldı ve ios işletim sistemi olan cep telefonlarında çalışabilmektedir. Proje, özgün bir tasarıma sahiptir. Kullanıcı için önemli olan bilgiler, kullanıcının kolay bir şekilde ona ulaşabilmesi için ana sayfaya koyulmuştur. Ekstra bilgiler için de coin sayfası oluşturuldu. Bu sayfada daha çok bilgi ve grafik mevcuttur.

Uygulama içerisinde deneme amaçlı al-sat işlemleri gerçekleştirilebilmektedir. Her kullanıcının bir cüzdanı vardır. Bu cüzdandaki miktara alım satım stratejisi gerçekleştirebilecektir. Bu strateji ile de gerçek bir alım- satım işlemine hazırlanabilmektedir.

#### 11.4 **KAYNAKÇA**

<https://developer.apple.com/swift/>

<https://www.hackingwithswift.com/>

<https://www.youtube.com/c/SwiftfulThinking>