

کارآموزی

نام دانشجو : سینا رزاقی کاشانی

نام استاد : دکتر مینا ملک زاده

گزارش کار نهایی (جمع بندی)

تاریخ : ۱۴۰۱ / ۴ / ۲۸

لهم إني
أنت ملائكة
أنت ربنا

گزارش کار اول:

برای درس کارآموزی یک پروژه اندروید باید تحويل داده شود. پس از کمی جست و جو متوجه شدم که فلاتر (Flutter) بروزترین و ساده ترین چارچوب واسطه موبایل (Mobile UI Framework) و یک بسته توسعه نرم افزار (Open Source) رایگان و متن باز (Software Development Kit | SDK) است.

Flutter توسط گوگل در سال ۲۰۱۷ ارائه شد. البته، فلاتر از سال ۲۰۱۴ وجود داشته و توسط گوگل معرفی شده است، اما تا قبل از اواسط ۲۰۱۷ که به صورت رسمی منتشر و روانه بازار شد، در مرحله آزمایشی قرار داشت. اولین نسخه فلاتر به نام کد «Sky» شناخته می شد. در آن زمان، Sky تنها روی سیستم عامل اندروید قابل اجرا بود.

به بیان ساده، فلاتر این امکان را برای توسعه دهنگان فراهم می کند که یک اپلیکیشن موبایل بومی (Native Application) را تنها با یک کد مبنای (پایه کد | Codebase) بسازند. یک اپلیکیشن بومی، به منظور استفاده در یک دستگاه خاص و سیستم عاملی ساخته می شود. امکان توسعه برنامه کاربردی بومی تنها با یک کد مبنای در Flutter، به این معنا است که می توان فقط با یک زبان برنامه نویسی و یک کد مبنای، دو یا چند اپلیکیشن مختلف برای سیستم عامل iOS و اندروید ساخت. در حال حاضر خروجی ویندوز و وب نیز میدهد ولی خیلی کاربردی نیست.

Dart یک زبان برنامه نویسی مبتنی بر نوع داده شئگرا (Typed Object Programming Language) است. از Dart می توان برای ساخت اپلیکیشن های موبایل استفاده کرد. دارت روی توسعه فرانکاترند متمرکز است. جهت توسعه با فلاتر، از زبان برنامه نویسی دارت استفاده می شود. گوگل دارت را در اوایل سال ۱۳۹۰ ارائه کرده و در طول سال ها آن را به میزان زیادی بهبود داده است. سینتکس (نحو | Syntax) دارت را می توان با جاوا اسکریپت مقایسه کرد.

نقاط قوت فلاتر :

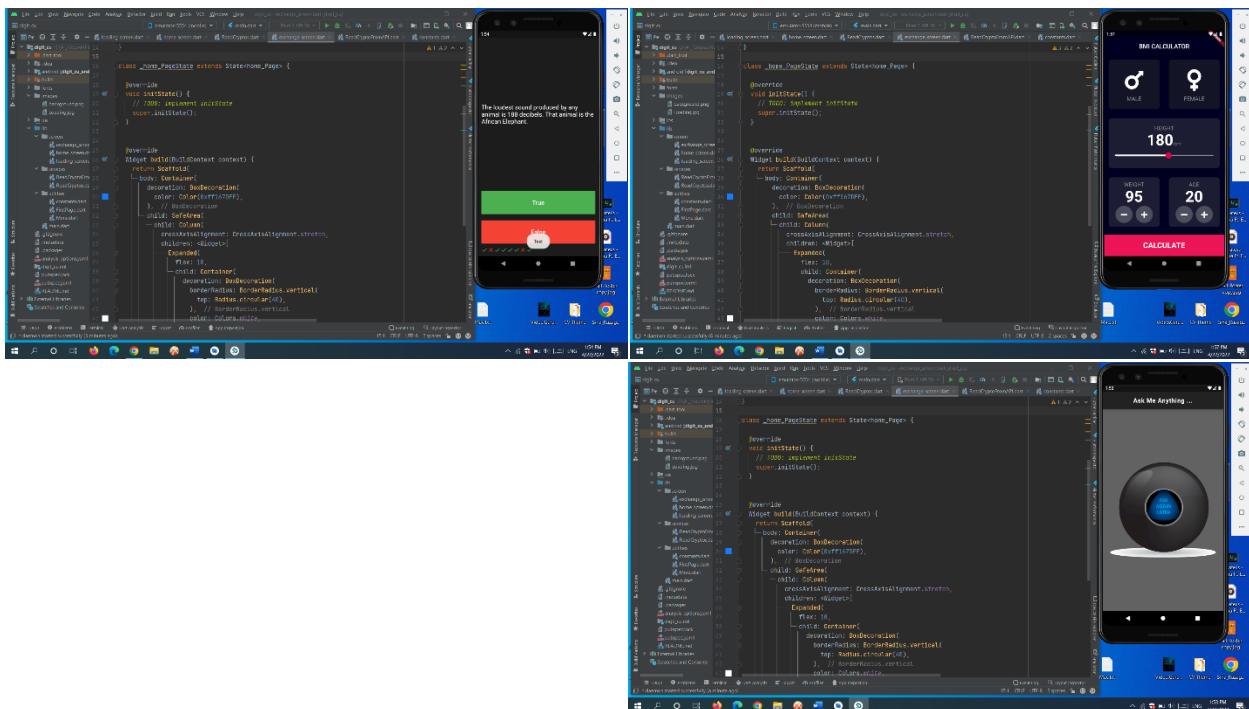
- ✓ سادگی یادگیری و استفاده از فلاتر
- ✓ سرعت بالای توسعه و بازدهی فلاتر
- ✓ سرعت اجرا و عملکرد مناسب فلاتر
- ✓ سازگاری بالا در فلاتر
- ✓ صرفه جویی در وقت و هزینه با فلاتر
- ✓ متن باز بودن فلاتر
- ✓ سازگاری فلاتر با استارتاپ های MVP
- ✓ برخورداری از مستندات کامل
- ✓ جامعه در حال رشد فلاتر
- ✓ سازگاری فلاتر با فریلنسرینگ

نقاط ضعف:

- ✓ اندازه بزرگ فایل اپلیکیشن (البته نسخه release حجم خیلی کمتری دارد)
- ✓ کمبود کتابخانه های شخص ثالث در فلاتر
- ✓ برخی مشکلات فلاتر با iOS
- ✓ مشکلات مربوط به دارت در فلاتر

از اواسط آبان ماه بخش های مقدماتی را مطالعه کردم با پروژه های کوچک.

منبع آموزش: [آموزش فلاتر مکتب خونه](#)



اولین پروژه اندروید : نرم افزار نمایش وضعیت آب و هوای

اولین فایل است که وقتی آپ ران می شود نمایش داده می شود main.dart

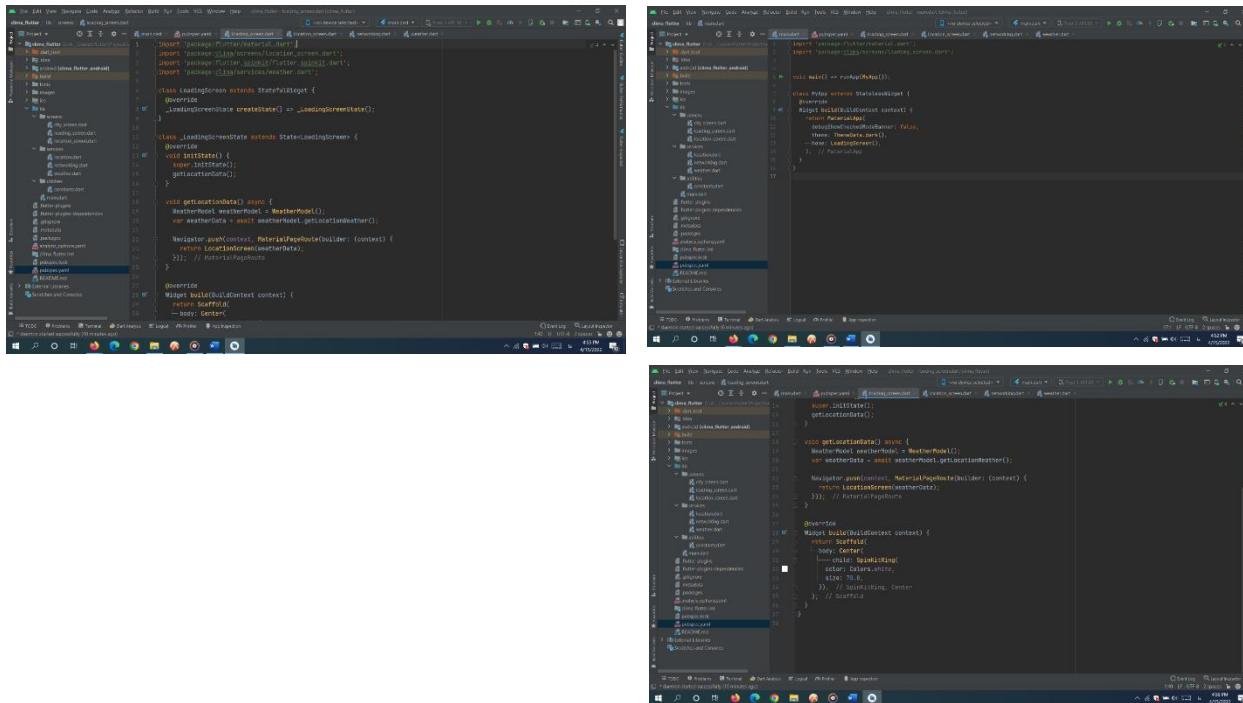
Pubspec.yaml فایلی که کتابخانه های مورد نیاز و فایل های جانبی روی آن نصب می شود.

پوشه fonts دارای فونت ها و images عکس های مورد نیاز برای پروژه است.

معماری خاصی برای آن در نظر گرفته نشده و هدف بیشتر نتیجه بوده که اولین خروجی را از flutter دریافت کنیم.

در پوشه lib یک بخش screen داریم که صفحه های طراحی شده است و پوشه services برای دریافت یکسری اطلاعات و پردازش و بعد برای نمایش در صفحه های ارسال می شود. در آخرم پوشه utilities که یک فایل به نام constants.dart شامل متغیر های ثابت برای استایل و یکسری موارد مشابه استفاده می شود.

صفحه loading را فراخوانی می کنیم (در اینجا میتوانیم از آدرس دهی برای پیج ها استفاده کنیم ولی نیازی نیست چون تعداد پیج ها زیاد نیست)



حالا وارد صفحه loading میشیم در اینجا یک کتابخانه spinkit اضافه کردیم که در فایل pubspec تعریف و نصب شده. نکته ای که در این صفحه وجود دارد این است که از مازول مکان یاب برای پیدا کردن طول و عرض جغرافیایی استفاده میکند یعنی این صفحه داده را دریافت می کند و برای صفحه بعد ارسال می کند.

وقتی صفحه باز می شود از تابع initstate آغاز می شود و تابع getlocationdata را اجرا می کند. این تابع به صورت async کار می کند و یک شی از weathermodel ایجاد می کند و تابعی برای دریافت اطلاعات فراخوانی می کند. بعد از آن اطلاعات دریافت شده و از طریق ناوبری به صفحه بعد منتقل می شود. در حین دریافت اطلاعات از کتابخانه spinkit ۱ مازول فراخوانی می کنیم تا به کاربر نمایش دهیم که در حال دریافت اطلاعات است.

```

name: clima
description: A new Flutter application.

version: 1.0.0+1

environment:
  sdk: ">=2.1.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter

  cupertino_icons: ^0.1.2
  geolocator: ^8.2.0
  http: ^0.13.4
  flutter_spinkit: ^5.1.0
  # alert: ^2.0.1

dev_dependencies:
  flutter_test:
    sdk: flutter

flutter:
  uses-material-design: true

assets:
  - images/

```

در فایل pubspec.yaml در بخش dependencies چند کتابخانه اضافه کردیم. برای استفاده از عکس‌ها و فونت‌ها آن‌ها نیز در این فایل تعریف می‌کنیم.

در فایل Loading یک شی با نام weathermodel داشتیم که در فایل weathermodel تعریف شده این بخش وظیفه دریافت اطلاعات از API را دارد.

از ارتباط async استفاده کردیم یعنی به طور مثال یک درخواست get برای API فرستاده می‌شود ولی جواب در همون لحظه باز نمی‌گردد پس روند اجرای برنامه نمی‌ایستد و ادامه بیدا می‌کند تا زمانی که دریافت شود و بازگردانده شود به صفحه loading. از آنجایی که ما داریم از طریق await اطلاعات را دریافت می‌کنیم اجرای صفحه لودینگ متوقف نمی‌شود و به صفحه بعد نیز نمی‌رود تازمانی که از API برای

ما ارسال شود یعنی همزمان هم صفحه ساخته شده و نمایش داده می‌شود هم درخواست برای API ارسال شده و جواب دریافت می‌شود بدون توقف در اجرای برنامه.

حال در فایل weather در تابع getlocationweather از طریق فایل نتورک دارای ۱ تابع networkhelper برای دادن درخواست به api دارد استفاده کنیم و درخواست را به صورت طول و عرض جغرافیایی و apikey و پارامترهای metric ارسال می‌کنیم و منتظر جواب می‌شویم آن جواب را برای صفحه لودینگ بازمی‌گردانیم.

فایل networking فقط برای فرستادن request با متده است و آن دیتا را با استفاده از کتابخانه json به api get می کند و متده jsondecode دیتای خام weather را برای ارسال می کند اگر ارتباط با مشکلی روبرو شد یعنی کد ۴۰۰ بازگرداند آن را در ترمینال جای می کند.



```
class BigBoss {
    String getRandomText() {
        return "I'm the big boss";
    }
}

class NetworkHelper {
    Future<String> getRandomText() async {
        final response = await http.get(Uri.parse("https://api.chucknorris.io/jokes/random"));
        var decodedData = jsonDecode(response.body);
        return decodedData['joke'];
    }
}
```

همزمان که صفحه loading در حال اجرای اطلاعات بازگردانده شده و با ناوبری به صفحه location_screen ارسال میشود. تازه وارد صفحه اصلی برنامه شدیم. این قسمت دو فایل را فراخوانی کرده، یکی constants که برای استفاده از یک سری متغیرهای ثابت برای طراحی است و دیگری weather برای دریافت اطلاعات.

هنوز اطلاعات به شکل جی سان است و استفاده نشده. در این صفحه دوباره در تابع `initstate` یعنی لحظه ای که صفحه نمایش داده میشود یک تابع به نام `update` داریم که اطلاعات را در صورتیکه وجود داشته باشد استخراج کرده و در متغیرها ذخیره می کند در غیر این صورت مقدارهای صفر و `Error` و جمله "Unable to get Weather data" را در متغیرها می ریزد.

دیزاین کلی به شکل یک عکس در بک گراند و چند تکست در زیر هم دیگه نمایش داده می شود و استایل ساده در عین حال چشم نوازی دارد. تابع `update` تابع `setstate` استفاده کردیم چون متغیرها در صفحه در حال نمایش هستند هر بار این تابع صدا زده شود متغیرها به روز می شوند چه اطلاعات باشد چه نباشد و تغییرات همان لحظه در صفحه ابحاد شده و نمایش، داده می شود.

The screenshot shows two side-by-side Java code editors. The left editor displays the `WeatherApp` class, which contains a main method and logic for reading weather data from a file and displaying it. The right editor displays the `WeatherForecast` class, which extends `StatefulWidget` and includes methods for fetching weather data and displaying it in a UI.

```
class WeatherApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return Container(
            child: Center(
                child: Text("Weather App"),
            ),
        );
    }
}

class WeatherForecast extends StatefulWidget {
    @override
    _WeatherForecastState createState() => _WeatherForecastState();
}

class _WeatherForecastState extends State<WeatherForecast> {
    String? location;
    String? condition;
    String? temperature;
    String? pressure;
    String? humidity;
    String? windSpeed;
    String? windDirection;
    String? sunrise;
    String? sunset;

    void updateDisplay() {
        setState(() {
            if (location != null) {
                condition = widget.locationConditionMap[location];
                temperature = widget.locationTemperatureMap[location];
                pressure = widget.locationPressureMap[location];
                humidity = widget.locationHumidityMap[location];
                windSpeed = widget.locationWindSpeedMap[location];
                windDirection = widget.locationWindDirectionMap[location];
                sunrise = widget.locationSunriseMap[location];
                sunset = widget.locationSunsetMap[location];
            }
        });
    }

    void fetchWeatherData() {
        Future<void> fetchWeather() async {
            final response = await http.get(Uri.parse("https://api.openweathermap.org/data/2.5/weather?q=London&appid=b1b15e8cd0a1c4f05cb26d2150d0a882"));
            if (response.statusCode == 200) {
                final data = jsonDecode(response.body);
                final weather = WeatherModel.fromJson(data);
                setState(() {
                    location = weather.name;
                    condition = weather.weather[0].main;
                    temperature = weather.main.temp.toPrecision(2);
                    pressure = weather.main.pressure;
                    humidity = weather.main.humidity;
                    windSpeed = weather.wind.speed;
                    windDirection = weather.wind.deg;
                    sunrise = DateFormat('hh:mm').format(weather.sys.sunrise);
                    sunset = DateFormat('hh:mm').format(weather.sys.sunset);
                });
            } else {
                print("Error: ${response.statusCode}");
            }
        }
        fetchWeather();
    }

    @override
    void initState() {
        super.initState();
        fetchWeatherData();
    }

    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(
                title: Text("Weather Forecast"),
            ),
            body: Container(
                padding: EdgeInsets.all(16),
                child: Column(
                    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                    children: [
                        Text("Location: $location"),
                        Text("Condition: $condition"),
                        Text("Temperature: $temperature°C"),
                        Text("Pressure: $pressure hPa"),
                        Text("Humidity: $humidity%"),
                        Text("Wind Speed: $windSpeed m/s"),
                        Text("Wind Direction: $windDirection°"),
                        Text("Sunrise: $sunrise"),
                        Text("Sunset: $sunset"),
                    ],
                ),
            ),
        );
    }
}
```

```

class Container {
    Container() {
        ...
    }
}

class Textfield {
    void onPressed() {
        Navigator.pop(context, cityData);
    }
}

class Row {
    Row() {
        ...
    }
}

class CityScreen extends StatefulWidget {
    @override
    _CityScreenState createState() => _CityScreenState();
}

class _CityScreenState extends State<CityScreen> {
    ...
}

```

آخرین بخش این نرم افزار بخش سرچ است که به صورت یک آیکون در پایین صفحه در دسترسی است ولی که متناسب با سرچ باشد را پیدا نکردم (سرچ یک حرف از اسم شهرها نتایج را به شما پیشنهاد دهد).

```

class Row {
    Row() {
        ...
    }
}

class Textfield {
    void onPressed() {
        var typeahead = await Navigator.push(
            context,
            MaterialPageRoute(
                builder: (context) {
                    return CitySearch();
                },
            ),
        );
        if (typeahead != null) {
            var weatherData = await weatherModel.getWeatherByTypeahead(
                typeahead);
            update(weatherData);
        }
    }
}

```

نکته فایل لوکیشن برای دریافت مجوز از گوشی و بعد از آن دریافت طول و عرض از مژول مکان یاب می باشد.

```

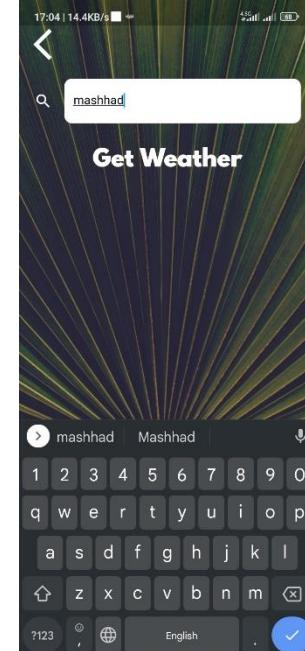
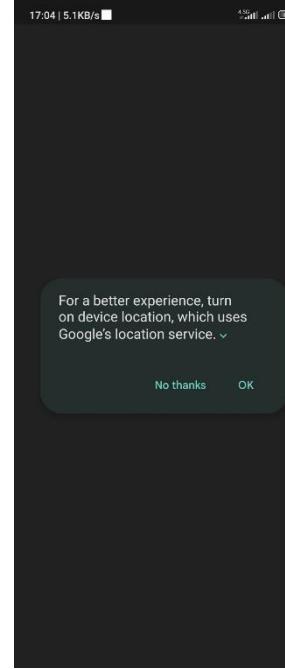
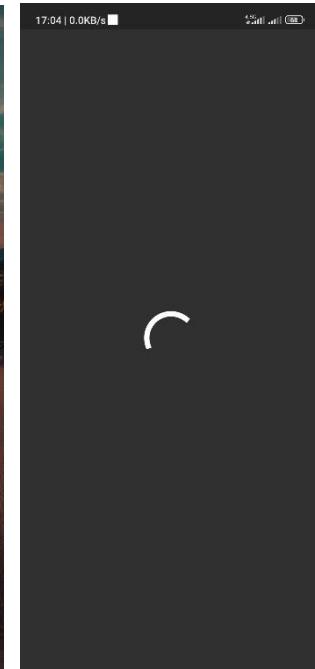
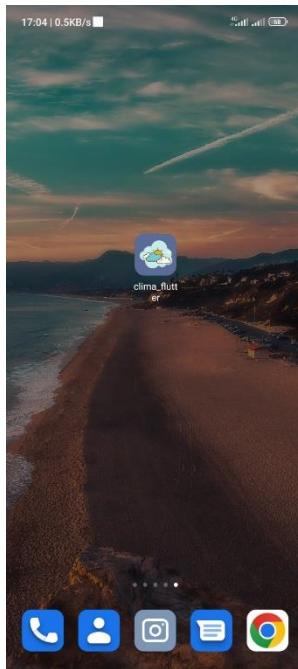
Future<void> checkPermission() async {
    permission = await Geolocator.requestPermission();
    if (permission == LocationPermission.denied) {
        ...
    }
}

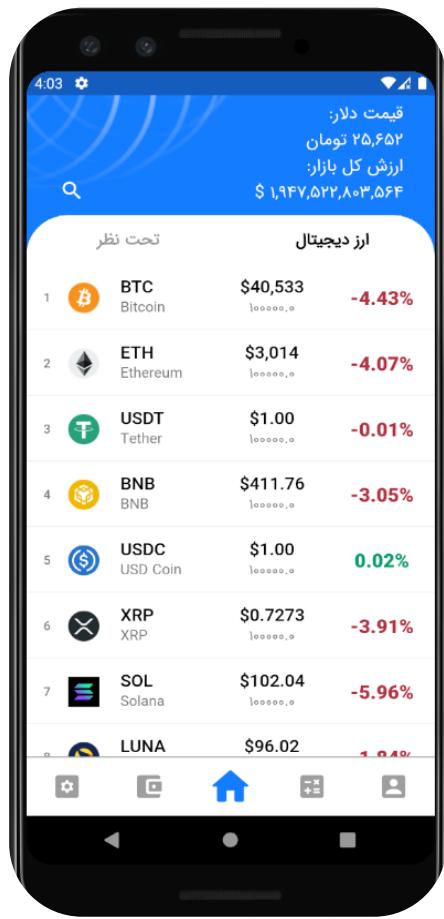
Future<void> getcurrentlocation() async {
    final Geolocator geolocator = Geolocator();
    final Location location = await geolocator.getLocation();
    ...
}

```

```
class WeatherPage extends StatelessWidget { @override void build(BuildContext context) { return Scaffold( appBar: AppBar( title: Text('Weather'), ), body: Center( child: Column( mainAxisAlignment: MainAxisAlignment.center, children: [ Text('It\'s time in'), Text('Sabzawār'), ], ), ), ); } }
```

```
class WeatherPage extends StatelessWidget { @override void build(BuildContext context) { return Scaffold( appBar: AppBar( title: Text('Weather'), ), body: Center( child: Column( mainAxisAlignment: MainAxisAlignment.center, children: [ Text('It\'s time in'), Text('Mashhad'), ], ), ), ); } }
```

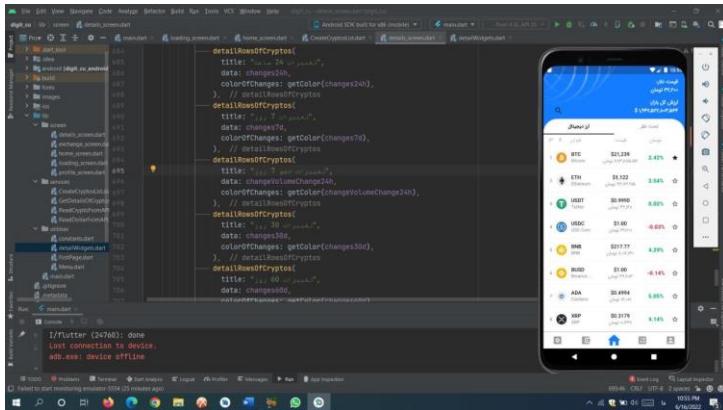




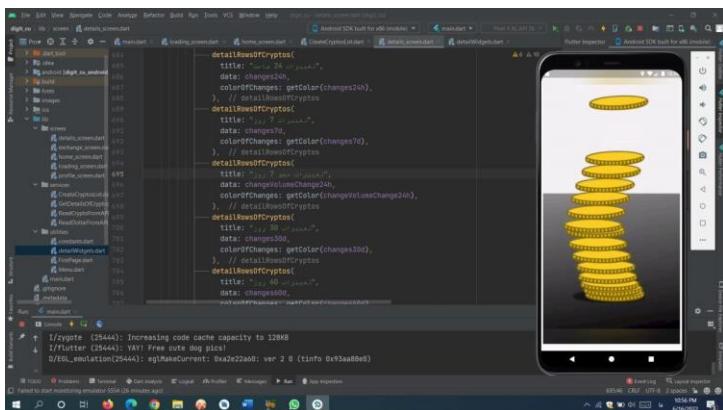
بعد از انجام این پروژه به برنامه نویسی اندروید خیلی علاقمند شدم و میخواهم ادامه دهم و همینطور سمت API نیز بروم. برای پروژه کارآموزی یک اپلیکیشن ارزدیجیتال تحويل داده خواهد شد و تحلیل نرم افزاری و پارامتر هایی که باید داشته باشد خدماتتون ارسال خواهم کرد.

نمونه طراحی اولیه صفحه اصلی:

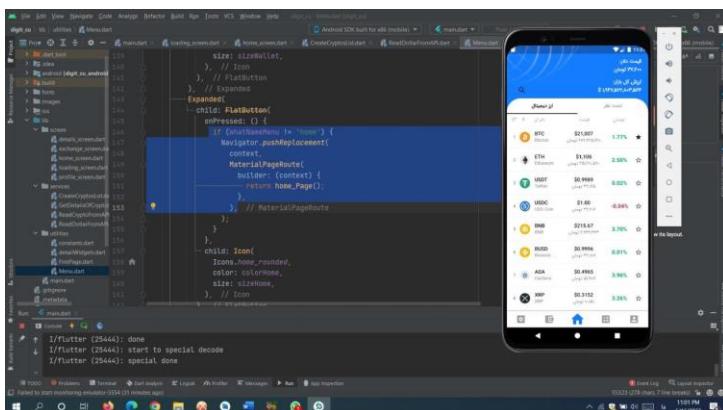
گزارش کار دوم:



اپلیکیشن ارز دیجیتال با هدف نمایش قیمت بهروز ارزهای دیجیتال و جزئیات آنها در کنار خرید و فروش مجازی شان برای ثبت سود و ضرر در کنار داشتن امکاناتی مانند اخبار نظردهی تیکت و پشتیبانی آنلاین و غیره.

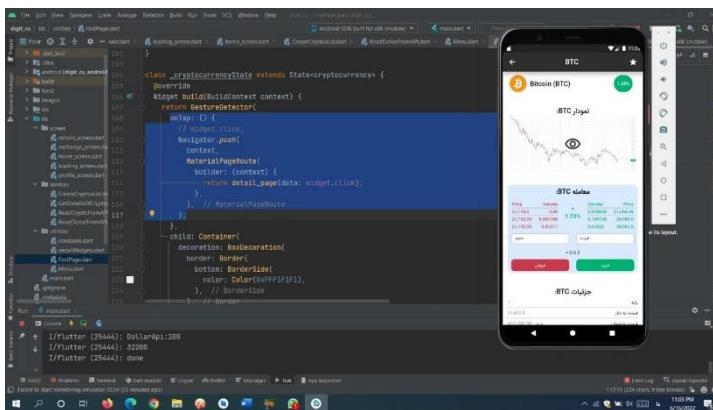
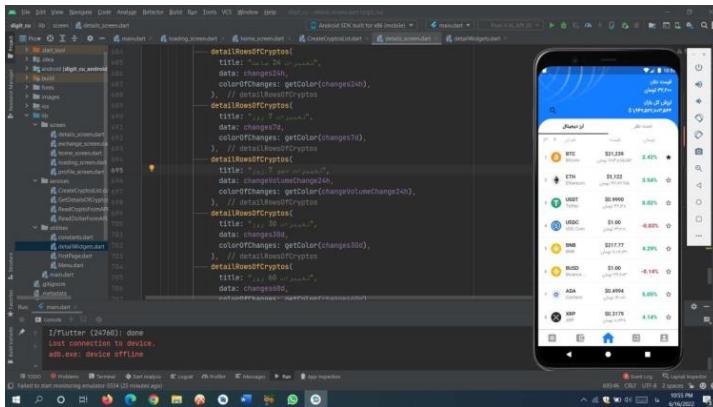
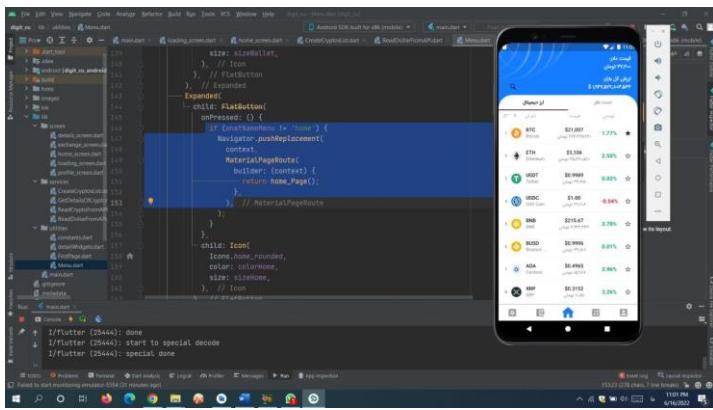
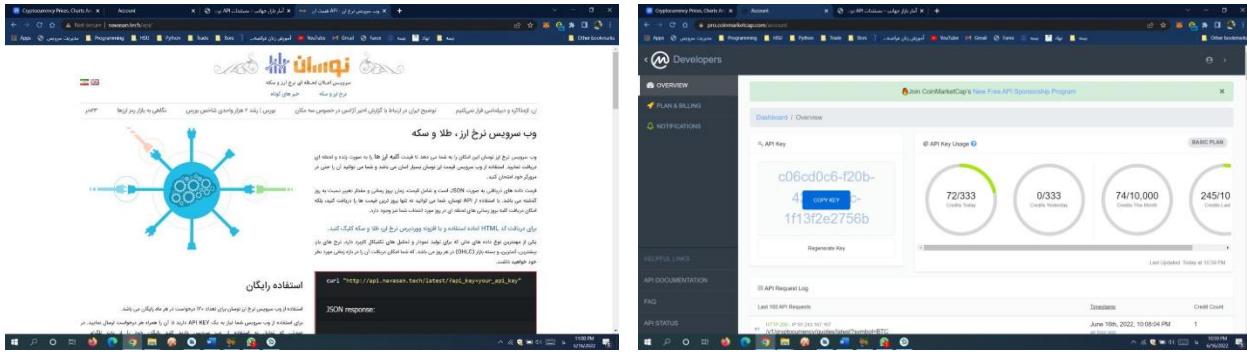


در قسمت شروع از یک صفحه استفاده شده که در آن یک گیف نمایش داده می شود تا زمانی که کانکشن اینترنت برقرار شود و بعد از آن وارد نیست تسلیح پنجاه ارز دیجیتالی اول میشود.



روش نمایش این ارزهای دیجیتال به صورت مژوالر بوده. به شکلی که یکبار ساخته می شود و چندین بار به اندازه نیاز فراخوانی میشود و لیست در صفحه اصلی نمایش داده می شود.

اطلاعات به صورت آنلاین از API دریافت میشود برای صفحه اصلی. از یک جای دیگر برای قیمت آنلاین ریال دلار استفاده میکنیم. تمامی ای پی آی های استفاده شده رایگان هستند.

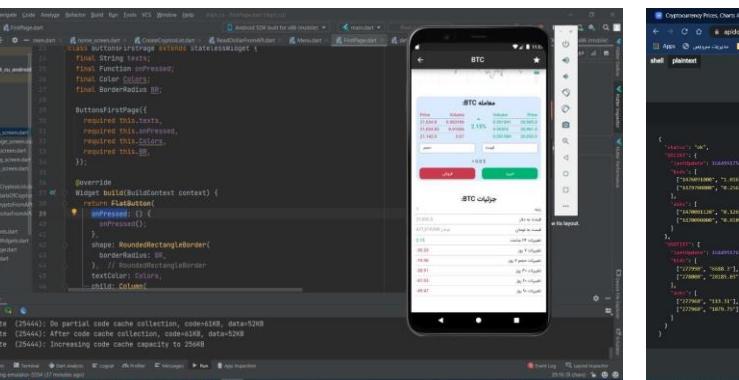
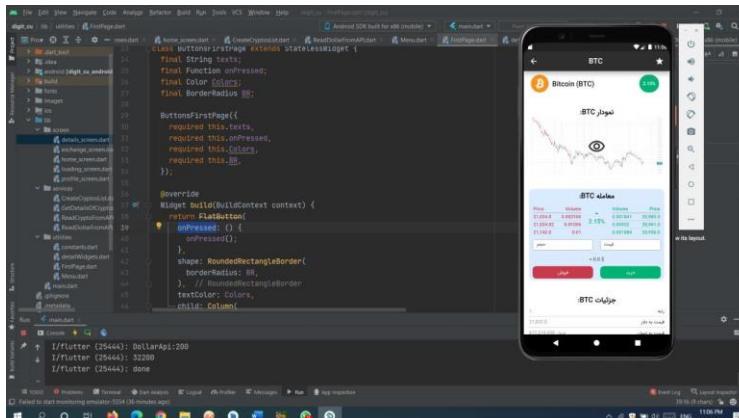
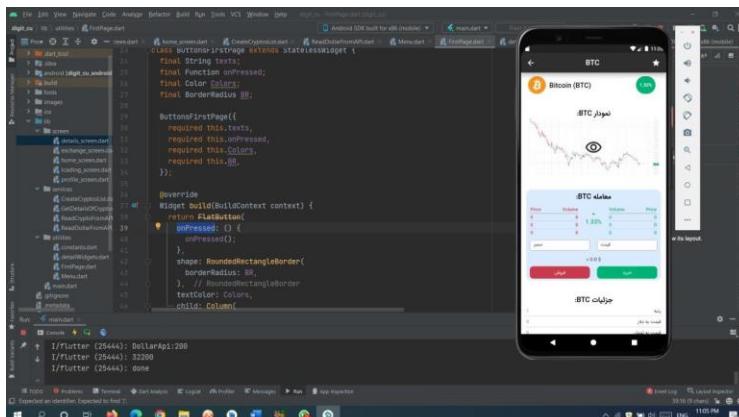


یکسری از اطلاعات به صورت local تعریف میشود یعنی اگر بخواهیم تغیر در theme به طور مثال ایجاد تاریک یا روشن ایجاد کنیم. حتی می توانیم این اطلاعات را به صورت یک فایل در داخل گوشی یک کاربر ذخیره کنیم و استفاده کنیم یعنی میشود بخش منو و جابجایی در آنها با ناوبری انجام می شود به صورتی که صفحه های اصلی که در منو عکس هاییشان مشاهده می شود از متند جایگزینی استفاده می کند ولی صفحاتی جزئیات ارزهای دیجیتال از پوش.

```

    void refreshData() async {
        // print("refreshData");
        double changeDollar = double.parse(cryptoData["change"].toString());
        if (changeDollar >= 0) {
            changeColor = Icons.arrowUpward;
            lastForChangeIcon = "up";
        } else {
            changeColor = Icons.arrowDownward;
            lastForChangeIcon = "down";
        }
        dataDetailsCrypto detailsCrypto = dataDetailsCrypto();
        MapString, String: dataDetails = await detailsCrypto
            .getDetailAtAddress(cryptoData["symbol"], toString());
        updateUI(dataDetails);
    }

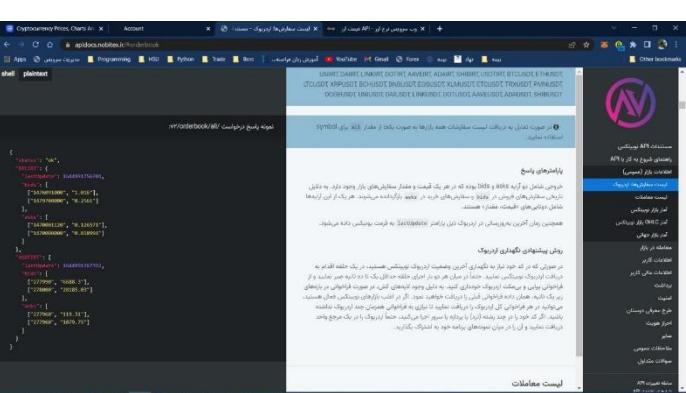
```



برنامه به صورت `async` کار میکند یعنی دارخواست API زده شده و همزمان صفحه ساخته می شود و وقتی اطلاعات دریافت شد صفحه بروزرسانی می شود.

هرکدام از این ویجت های ارز دیجیتال یک متده است که `onPressed` دارد یعنی زمانی که لمس میشود می توانیم یک تابع یا حتی چندین دستور اجرا کنیم صفحه را فراخوانی کنید و یک سری از اطلاعاتی که بر روی آن کلیک شده را برای آن ارسال می کند زمان ورود به صفحه جزئیات ارز دیجیتال یک درخواست اول کوین مارکت کپ می شود درخواست جزئیات گذشته مثل تغیرات برای دریافت جزئیات ارز دیجیتال یک ماه اخیر.

دوم API هیستوگرامی یا به اصطلاح معاملاتی ارز را برای ما ارسال می کند از طریق صرافی نوبیتسس برای صفحه جزئیات ارز دیجیتال API سوم هم قیمت دلار را میدهد.



گزارش کار سوم:

API



API از نظر فنی اختصاری برای عبارت «رابط برنامه‌نویسی اپلیکیشن» (Application Programming Interface) محسوب می‌شود. در برخی موارد شرکت‌های بسیار بزرگ API-هایی برای مشتریان خود و یا کاربردهای داخلی‌شان ساخته‌اند. اما اگر بخواهیم API را به زبان کاملاً ساده توضیح دهیم، معنی بسیار گستردگی از آن چه در حوزه‌های نرم‌افزار یا کسب‌وکار استفاده می‌شود، خواهد داشت. مجموعه‌ای از تعاریف و پروتکل‌ها و ابزار‌های نرم افزاری برای ساخت نرم افزار است. به طور کلی API مجموعه‌ای از روش‌های تعریف شده میان ارتباطات بین اجزای مختلف است. یک API خوب، برنامه‌های کامپیوتری را با ارائه تمامی بلوك‌های ساختمانی که برنامه نویس آن‌ها را با هم ترکیب می‌کند، ساده تر می‌کند.

برای ساخت API میتوان از فریم ورک‌ها و میکرو فریم ورک‌های مختلف استفاده کرد، به طور مثال از میکرو فریم ورک Lumen برای برقیابی میکرو سرویس‌ها و ساخت API‌های مختلف استفاده می‌شود.

ASP.NET Core



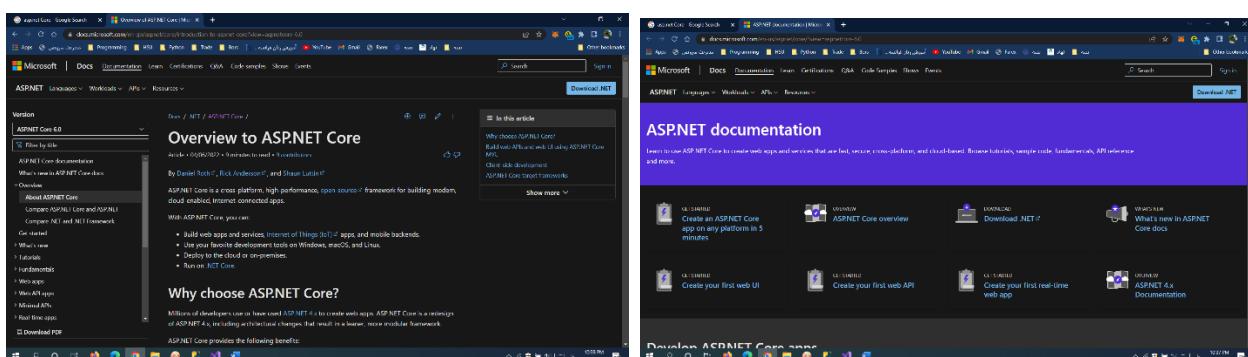
فریم ورک asp.net core یک محیط مدرن و پیشرفته است که با آن می‌توان اپلیکیشن‌ها و ابزارهای تحت وب .NET را توسعه داد. این فریم ورک توسعه وب Cross Platform است و عملکرد عالی در توسعه و اجرای اپلیکیشن‌های تحت وب دارد. با فریم ورک asp.net core شما می‌توانید ابزارهایی را روی فضای ابری توسعه دهید و امکان توسعه برای اینترنت اشیا، بک اند وب، لینوکس و ویندوز فراهم شده است. برای توضیح ASP.NET Core و کاربردهای آن بهتر است اشاره‌ای به قابلیت بی نظیر Cloud-based Cloud-based بودن آن کنیم که شما می‌توانید از این فریم ورک روی فضاهای ابری نیز استفاده کنید.

در صورت استفاده از این محیط توسعه وب، شما می‌توانید کاربردهای زیر را از آن انتظار داشته باشید.

- ✓ توسعه وب اپ‌ها و خدمات تحت وب
- ✓ توسعه برای ابزارهای اینترنت اشیا
- ✓ توسعه برای سمت بک اند موبایل
- ✓ توسعه روی پلتفرم‌های لینوکس، ویندوز و مکینتاش
- ✓ توسعه ابزارها روی فضاهای ابری و سیستم‌های On-premise

مزایای استفاده از Asp.net core :

- ✓ کراس پلتفرم
- ✓ عملکرد بالا
- ✓ پشتیبانی از چهارچوب‌های ناهم‌زمان
- ✓ پشتیبانی از MVC و وب API
- ✓ امکان توسعه جداگانه



SQL Server



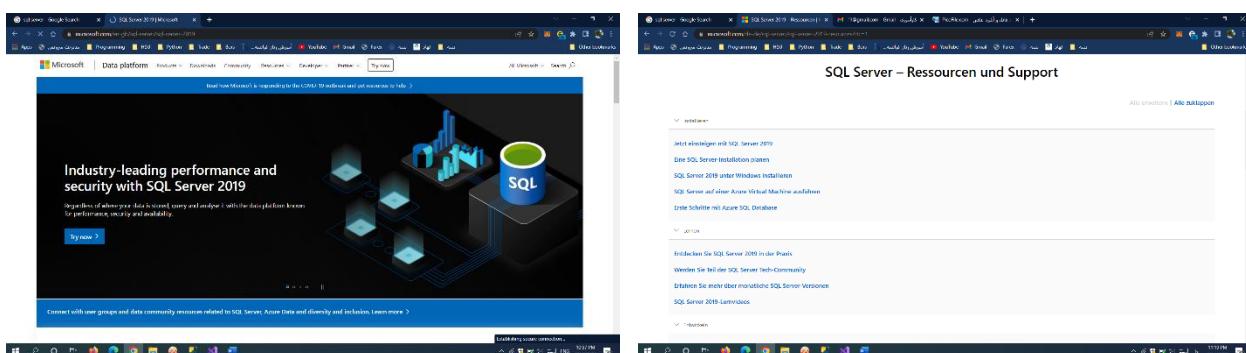
اصطلاح اختصار «جستار به زبان انگلیسی ساختاریافته» (SEQUEL) یا سیکوئل به آن اطلاق می‌شد که به دلیل انحصار تجاری این نام تحت اختیار یک شرکت هواپیمایی، به اس-کیو-ال (SQL) تغییر نام پیدا کرد. اس کیو ال یک زبان استاندارد برای دسترسی و کار با پایگاهداده (database) است. این زبان از سال ۱۹۸۷ یک زبان استاندارد بین المللی (ISO) بوده است.

SQL که به عبارت سیکوئل نیز معروف است، مخفف عبارت Structured Query Language می‌باشد که آنرا «زبان ساختاریافته جستار» یا «زبان پرس‌وجوی ساختار یافته» نیز معنی می‌کنند. در دهه ۱۹۷۰

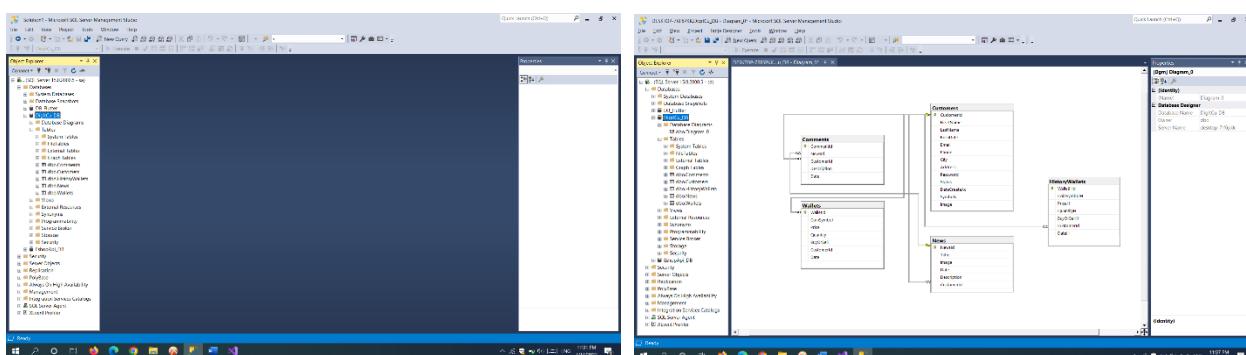
یک بانک اطلاعاتی از نوع دیتابیس‌های رابطه‌ای یا Relational Database است که توسط کمپانی Microsoft ارایه شده، و وظیفه اصلی آن ذخیره و بازیابی اطلاعات براساس درخواست نرمافزارهای دیگر می‌باشد. مهم ترین کاربرد اس کیو ال سرور ساخت بانک اطلاعاتی با حجم بالا است که تعداد زیادی کاربر بطور همزمان می‌توانند به آن دسترسی داشته باشند.

مهمنترین ویژگی‌هایی که برای SQL Server می‌توان نام برد عبارتند از:

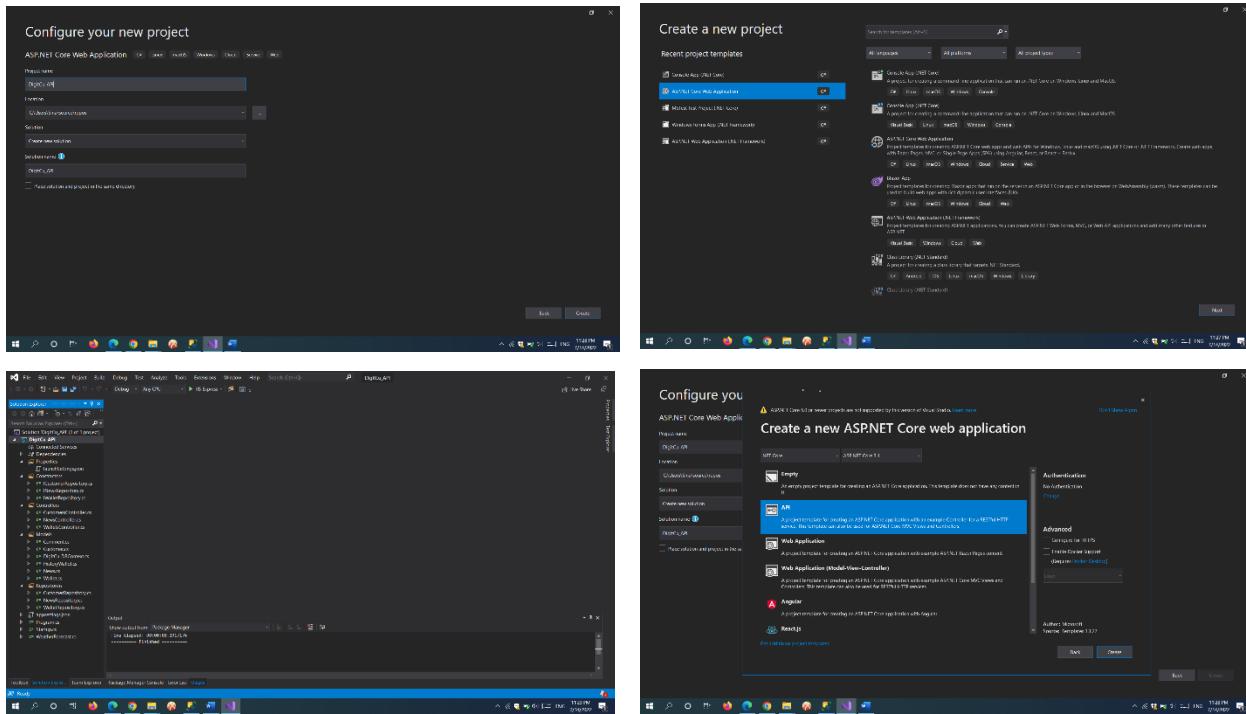
- ✓ بانک اطلاعاتی آن از نوع رابطه‌ای یا relational است.
- ✓ از فایل‌های XML پشتیبانی می‌کند.
- ✓ ویژگی OLAP را دارد.
- ✓ می‌توان اجزای stored procedure و trigger را استفاده کرد.
- ✓ از لحاظ حجم و تعداد رکورد هیچ محدودیتی ندارد و از این لحاظ بسیار قدرتمند است.
- ✓ امکان استفاده از زبان طبیعی در جستجو‌ها وجود دارد.
- ✓ برای افزایش سرعت در بازیابی اطلاعات از Full Text Search می‌توان استفاده کرد.



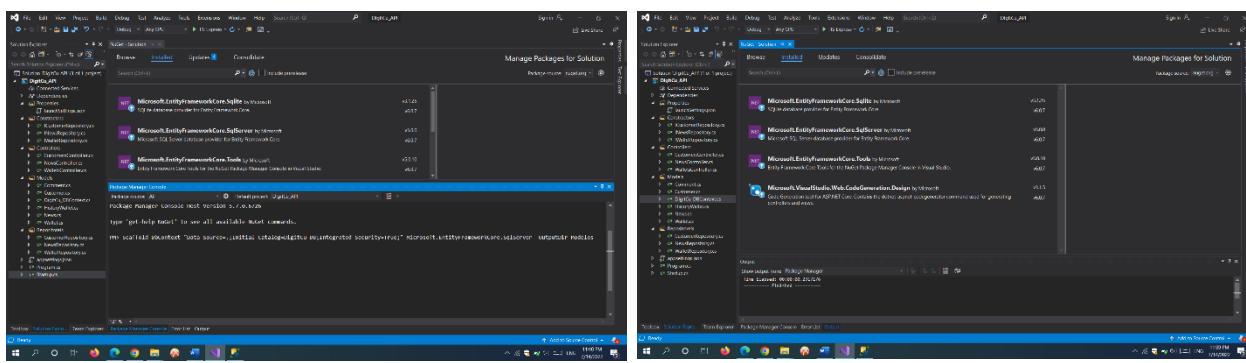
در اولین مرحله دیتابیس را می‌سازیم با پنج تیبل و ارتباط خواص آنها



در مرحله بعد پروژه API از نوع ASP.NET Core 3.1 را ایجاد می کنیم با یک اسم مشخص .NET. از نوع API بدون کانفیگ کردن HTTP و Duck Support



حال باید پکیج های زیر را نصب میکنیم دقت داشته باشید چون ورژن ویژوال استادیو بنده ۲۰۱۹ است نمیتوانم از .NET Core 3.1 استفاده کنم. بالاتر استفاده کنم. پس ورژن های خاص فقط از این پکیج ها نصب می شود. پس از نصب پکیج ها دستور Scaffold-DbContext اجرا می کنیم که مدل ها از روی دیتابیس در پوشه مدل ایجاد می شود.



برای هر Table یک کلاس در نظر می گیرد و این کلاس متدهای خاص خود را دارد به طور مثال Customer متدهایش همان ستون هاست و به صورت یک شی می توانیم از روی این کلاسها ایجاد و استفاده کنیم.

همانطور که مشاهده می کنید در فایل Startup Configure Services در قسمت ConfigureServices سرویس اضافه میکنیم به نام DbContext و فایلی که در پوشه Model ایجاد شده را در قسمت ConfigureServices ها اضافه کنیم و به این گونه می توانیم از EF Core و LINQ استفاده کنیم. در این پروژه معماری Repository استفاده شده است به شکلی که اول داخل یک Interface متد های مورد نیاز را تعریف کرده.

```

public class ApplicationDbContext : DbContext
{
    public virtual DbSet<Comments> Comments { get; set; }
    public virtual DbSet<Customer> Customers { get; set; }
    public virtual DbSet<HistoryWallet> HistoryWallets { get; set; }
    public virtual DbSet<News> News { get; set; }
    public virtual DbSet<Wallet> Wallets { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Comment>().HasRelationalCollection("Relational collection", "QQ_Latin1_General_CI_AS");
        modelBuilder.Entity<Comment>().Property(e => e.Date).HasMaxLength(50);
        modelBuilder.Entity<Comment>().Property(e => e.Description).HasMaxLength(300);
        modelBuilder.Entity<Customer>().Property(e => e.CustomerId).HasMaxLength(10);
        modelBuilder.Entity<Customer>().Property(e => e.FullName).HasMaxLength(100);
    }
}

```

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext< ApplicationDbContext >(
        options => options.UseSqlServer("Data Source=(local)\sqlexpress", 
            sqlOptions => sqlOptions.IntegratedSecurity(true)));
    services.AddTransient<CustomerRepository, CustomerRepository>();
    services.AddTransient<HistoryRepository, HistoryRepository>();
    services.AddTransient<UnitOfWork, UnitOfWork>();
    services.AddMemoryCache();
}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        app.UseHsts();
    }
}

```

بعد در فایل آن متد ها را Implement کنیم. قسمتی که باید با دیتابیس در ارتباط باشد همین فایل است.

داخل فایل Interface متد ها معرفی شدند و آنچه که باید برگردانند Task یا از نوعی متغیر Async است. در فایل Repository ما دو شی در اول کلاس تعریف کردیم اولین Context برای استفاده از محتوای Memory Cache و دومین برای استفاده از API را دو چندان می کند.

```

public class CustomerRepository : ICustomerRepository
{
    private readonly ApplicationDbContext _context;
    private readonly IMemoryCache _cache;

    public CustomerRepository(ApplicationDbContext context, IMemoryCache cache)
    {
        _context = context;
        _cache = cache;
    }

    public void AddCustomer(Customer customer)
    {
        _context.Customers.Add(customer);
        _context.SaveChanges();
    }

    public void UpdateCustomer(Customer customer)
    {
        _context.Customers.Update(customer);
        _context.SaveChanges();
    }

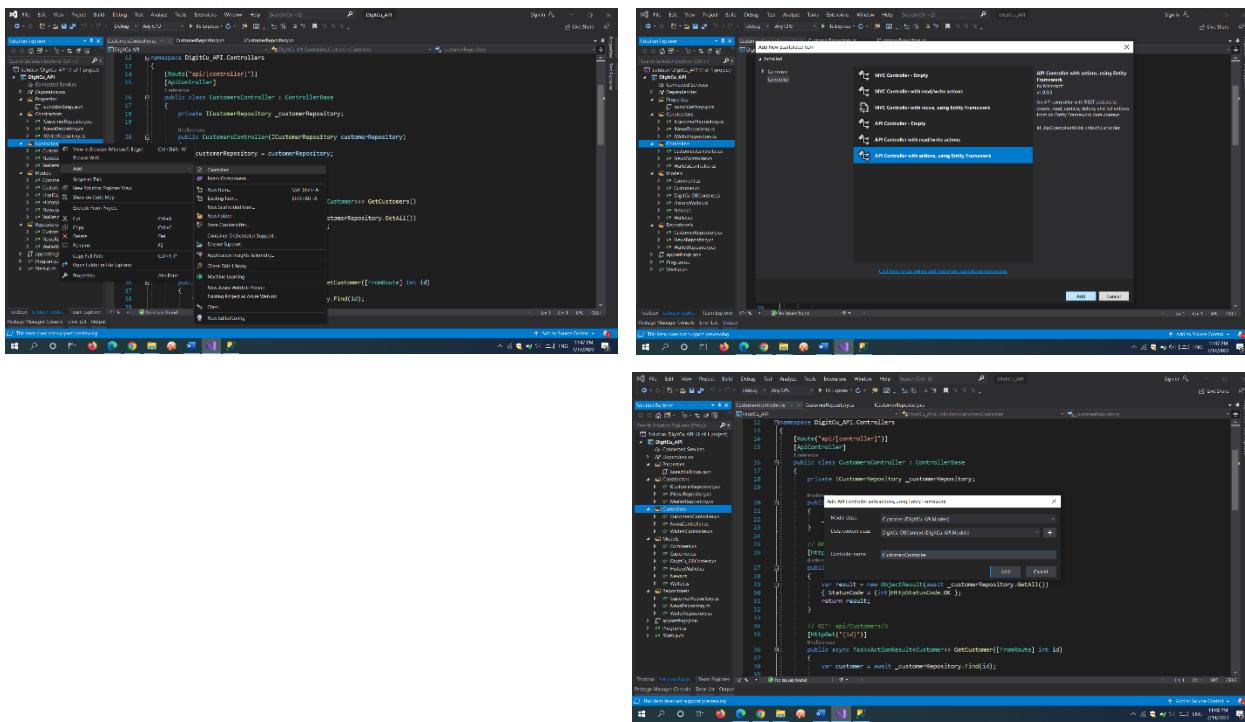
    public void DeleteCustomer(int id)
    {
        var customer = _context.Customers.SingleAsync(c => c.CustomerId == id);
        customer.Delete();
        _context.SaveChanges();
    }

    public Task<List<Customer>> GetAllCustomers()
    {
        return await _context.Customers.ToListAsync();
    }
}

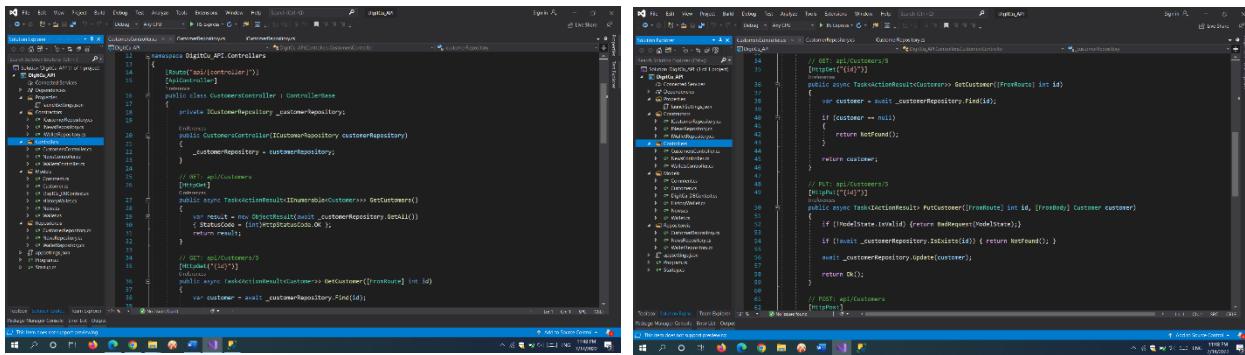
```

در Controller می توانیم بر روی پوشه کنترلر راست کلید کنیم و گزینه add Controller را بزنیم.

در اینجا گزینه API Controller with action, using Entity Framework را انتخاب می کنیم.



وقتی ایجاد شد یک شی از روی کلاس Interface ایجاد می کنیم و در هرجا این کنترلر ساخته شد نیز داده شود در ادامه تمام متدهای مورد نیاز برای ما ایجاد کرده به طوری که تمام متدها Get و Post و Put و Delete استفاده شده که مفهوم Restful API است.



در قسمت Get یک لیست برای ما باز می گرداند این لیست از دیتابیس گرفته شده به نوع `Task<List<Customer>>` تغییر کرده و `200 Status Code` به آن اضافه می شود.

در قسمت بعد می توانیم در آدرس یک متغیر تعریف کنید. ورودیتابع یک عدد از این آدرس را دریافت می کند و آن آیدی را برای ما باز گرداند از طریق متند `Find`.

```

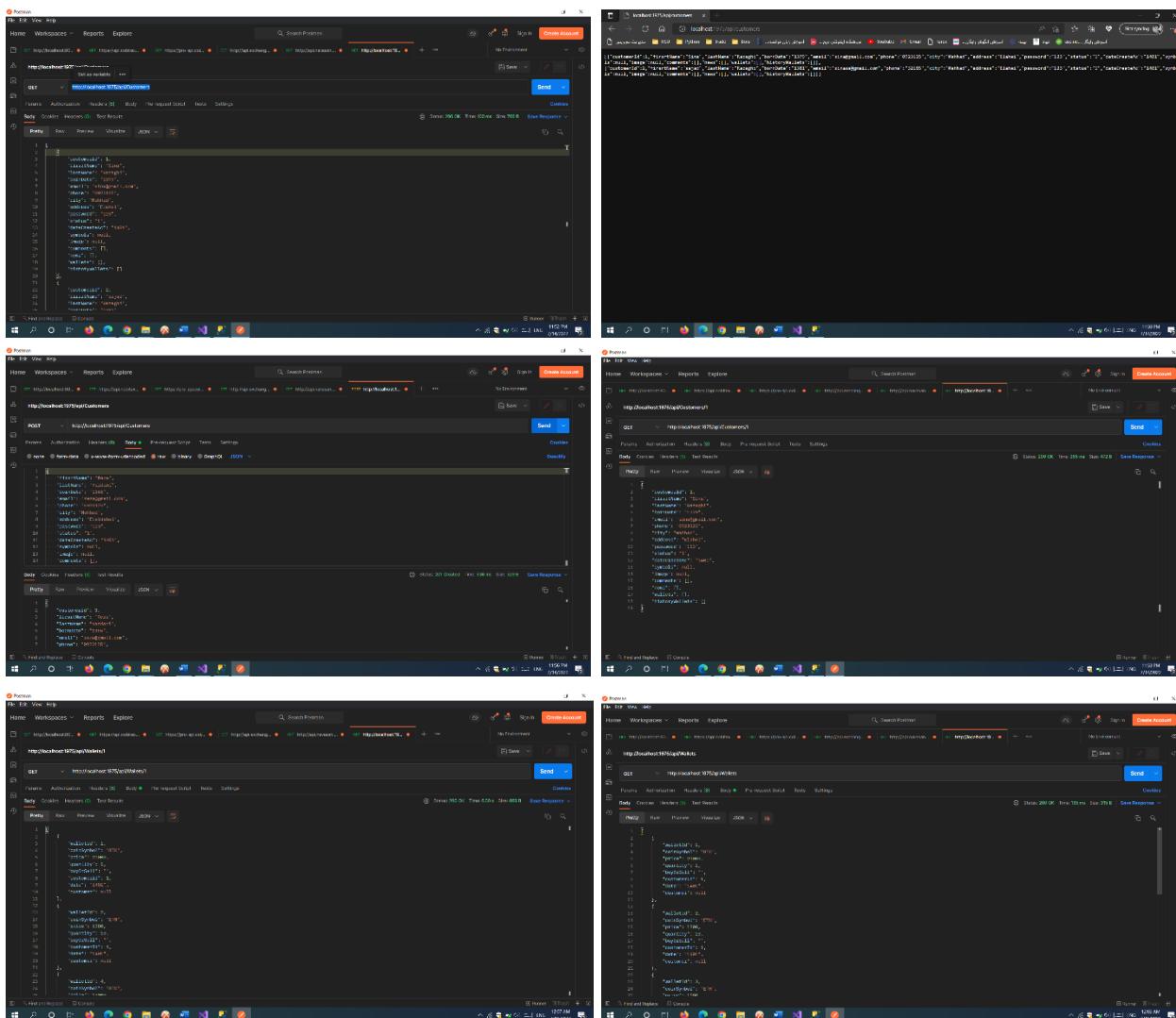
    // POST api/customers
    [HttpPost]
    public async Task<ActionResult> PostCustomer([FromBody] Customer customer)
    {
        await _customerRepository.Add(customer);
        return CreatedAtAction("GetCustomer", new { id = customer.CustomerId }, customer);
    }

    // DELETE api/customers/{id}
    [HttpDelete("{id}")]
    public async Task<ActionResult> DeleteCustomer([FromBody] int id)
    {
        if (_customerRepository.Exists(id)) { return NotFound(); }
        await _customerRepository.Remove(id);
        return Ok();
    }

```

برای Put ویرایش دیتا را در نظر گرفته به طوری که اگر ورودی ها درست بود از مدل استیت می‌گذرد و ریکوئست و بد ریکوئست باز نمی‌گردد و بررسی می‌کند که آیا آن متغیر وجود دارد یا نه در صورت وجود داشتن اطلاعات جدید را برای آنچه آپدیت می‌کند. برای Post اطلاعات را مستقیم اضافه می‌کنیم و ساخته شده آن را به صورت کامل نمایش می‌دهیم. از Delete هم برای پاک کردن دیتا از دیتابیس استفاده می‌کنیم.

همانطور که مشاهده می‌کنید آدرس مورد استفاده [api/[contoller]] است یعنی اسم controller باید جای این کلمه کلیدی قرار بگیرد و با Postman مورد استفاده قرار می‌گیرد.



در انتها یک مشکلی داریم که با توضیح یک مثال برایتان شرح میدهد.

در قسمت کیف پول های ارز دیجیتال این به شکل عمل می شود که تمام موجودی فرد بعد از خرید و فروش هر ارز نمایش داده شده و میزان سود و ضرر آن بر اساس نقطه ورود وی مشخص می شود. فرض کنید ۱ کاربر یک ارز دیجیتال را دوستان در روی ۱۰۰ دلار خریداری می کند این کاربر بعد از گذشته چند روز ۱۳ همدیگر در قیمته ۸۰ دلار خریداری می کند میانگین وزن دار این اعداد می شود. نقطه ورود کاربر به مامله یعنی حدود ۹۳ کاربر الان در ضرر است زیرا او سهم در قیمت ۹۳ دلار خرید ولی الان قیمت آن عرض ۸۰ دلار است. کاربر تصمیم می گیرد که یک سهم را بفروشد به جای ۳۳ وی دو سه هم برایش باقی می ماند موجودی کاربر از ۳۳ که در حال حاضر قیمت ۸۰ دلار است به ۲۳ یعنی ۱۶ دلار کاهش یافته آیا نقطه ورود به معامله فرقی میکند؟

```
SELECT t1.CoinSymbol, t1.Quantity, (t2.[Price] / t2.[SumQuantity]) AS 'Price'
FROM
    (SELECT CoinSymbol,
            SUM(Quantity) AS 'Quantity'
     FROM Wallets
     WHERE CustomerId = 1
     GROUP BY CoinSymbol) t1
LEFT JOIN
    (SELECT CoinSymbol,
            SUM([Price] * Quantity) AS 'Price' ,
            SUM(Quantity) AS 'SumQuantity'
     FROM Wallets
     WHERE CustomerId = 1 AND Quantity >= 0
     GROUP BY CoinSymbol) t2
ON (t1.CoinSymbol = t2.CoinSymbol);
```

پس برای نمایش موجودی کاربر فقط یک جمع و تفریق بین حجم های خرید و فروش کاربر قرار می دهیم و با گرفتن قیمت روز ارز میتوانیم موجودی کاربر را نمایش دهیم ولی برای محاسبه سود و ضرر آن کاربر باید فروش ها را حذف کرده و خرید ها را در نظر گرفت و میانگین وزن دار بین آنها قرار داد.

داخل دیتابیس باید یک Group by انجام شود از روی سمبل های رکورد برای فلان کاربر که این شرایط در کوئری روبرو قرار داده شده حال باید این دوتا متغیر را بدست آوریم.

اولی با یک مجموع خیلی ساده از حجم های مثبت یعنی خرید آن سهم و حجم های منفی یعنی فروش ان سهم حاصل کنیم و نمایش دهیم. در بخش دوم یک میانگین وزن دار از خرید هایی که کاربر انجام داده بر اساس حجم های مختلف ارائه کنیم.

در این قسمت ما اول قیمت و حجم را در هم ضرب کرده و تمامی را با هم جمع می کنیم و جمع آن حجم خرید ها را نیز قرار میدهیم برای میانگین گیری. حال خیلی راحت این دو را تقسیم می کنیم بر یکدیگر و خروجی میشود نقطه ورود کاربر برای خرید آن سهم. (برای محاسبه سود و ضرر)

تا اینجا کویری که از دو سلکت استفاده می کند درست است و مشکل ندارد ولی از آنجایی که مشکل نمایان می شود که ۱ کاربر تمام سهم خود را از یک ارز می فروشد و موجودی وی در آن ارز صفر می شود. پس در معاملات بعدی نباید رکوردهای خرید و فروش قبل تاثیری داشته باشد.

برای این کار رکوردهایی که دیگر نباید در محاسبات بررسی شوند را در یک Table اضافه میکنیم و از Table History Table اصلی حذف میکنیم.

از طرفی حجم محاسبات کم شده و این محاسبات دوباره از اول تکرار و بررسی نمی شود برای نمایش موجودی آنلاین کاربر خیلی راحت تر و سریع تر از قبل عمل میکند.

```
--when we have a 'full sell'
--First save them on another tb
INSERT INTO HistoryWallets
SELECT * FROM Wallets
FROM Wallets
WHERE CustomerId = 1 AND CoinSymbol = 'BTC';

--Second delete them records
DELETE FROM Wallets WHERE CustomerId = 1 AND CoinSymbol = 'BTC';
```

گزارش کار چهارم:

بعد از کلی آزمون و خطا و جستجو های فراوان راحل اشتراک گذاشتن localhost را پیدا کردم که با استفاده از ngrok یک port forwarding از یک دامین در اینترنت به localhost بروی سیستم میزند و سیستم تبدیل به سرور آنلاین می شود. مطمئن و امن نیست ولی کار راه اندازی برای نسخه های اولیه پروژه.



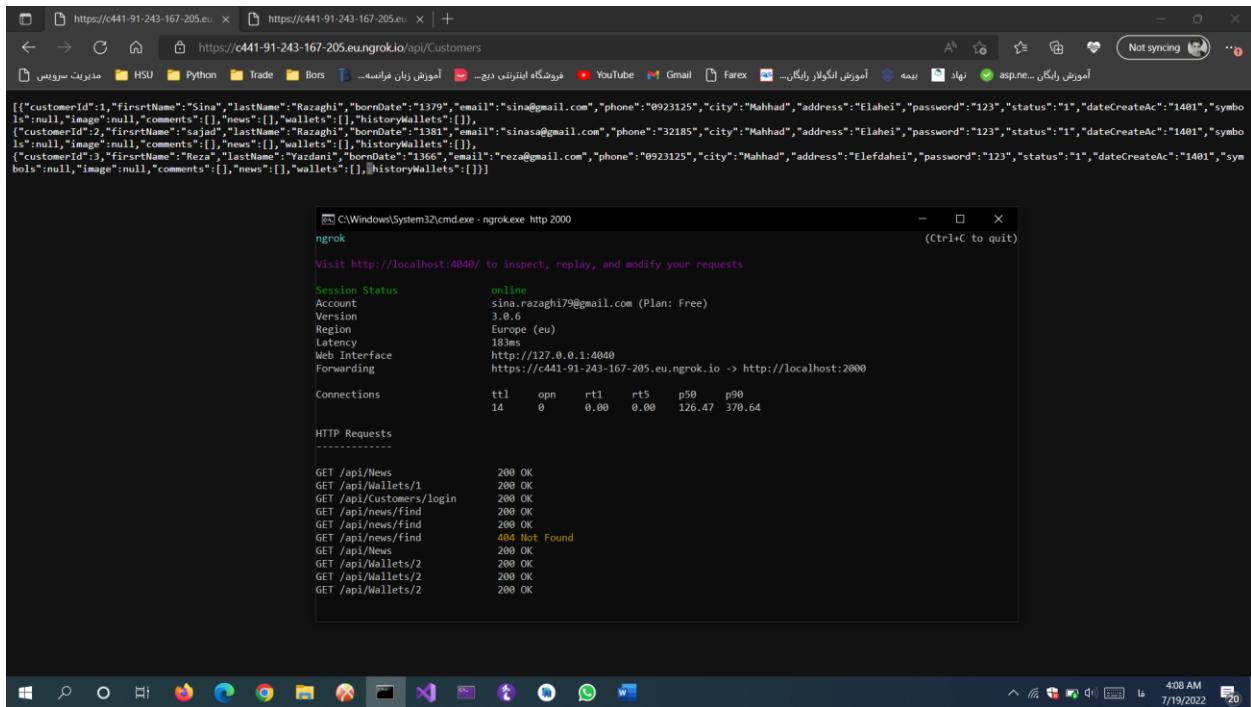
Ngrok سرویسی برای ارایه سرویس های محلی (localhost) شما مانند ssh, webserver,ftpserver ... در اینترنت است. به عبارت دیگر این ابزار یک آدرس عمومی به سرویس محلی شما اختصاص می دهد که میتوانید این آدرس را در اختیار دیگران قرار دهید تا به سرویس محلی شما دسترسی داشته باشند. این سرویس کاملا Open Source می باشد و میتوانید سورس آن را از آدرس گیت هاب آن دریافت نمایید. البته لازم به ذکر است که این سرویس به وسیله زبان Go توسعه یافته است.

قطعاً تبدیل شدن رایانه شما به یک وب سرور کار عاقلانه ای نیست. اما شاید این کار برای دمو یک پروژه یا تست آن بتواند برای شما مفید باشد. همچنین دانلود فایل ها به صورت مستقیم از رایانه شما، میتواند شما را از آپلود آن فایل ها بی نیاز کند و به این وسیله در زمان و مصرف اینترنت شما صرفه جویی کند.

حال پروژه را روی <http://localhost:2000> اجرا می کنیم و دستور زیر را در فولدری که ngrok قرار دارد اجرا می کنیم در cmd

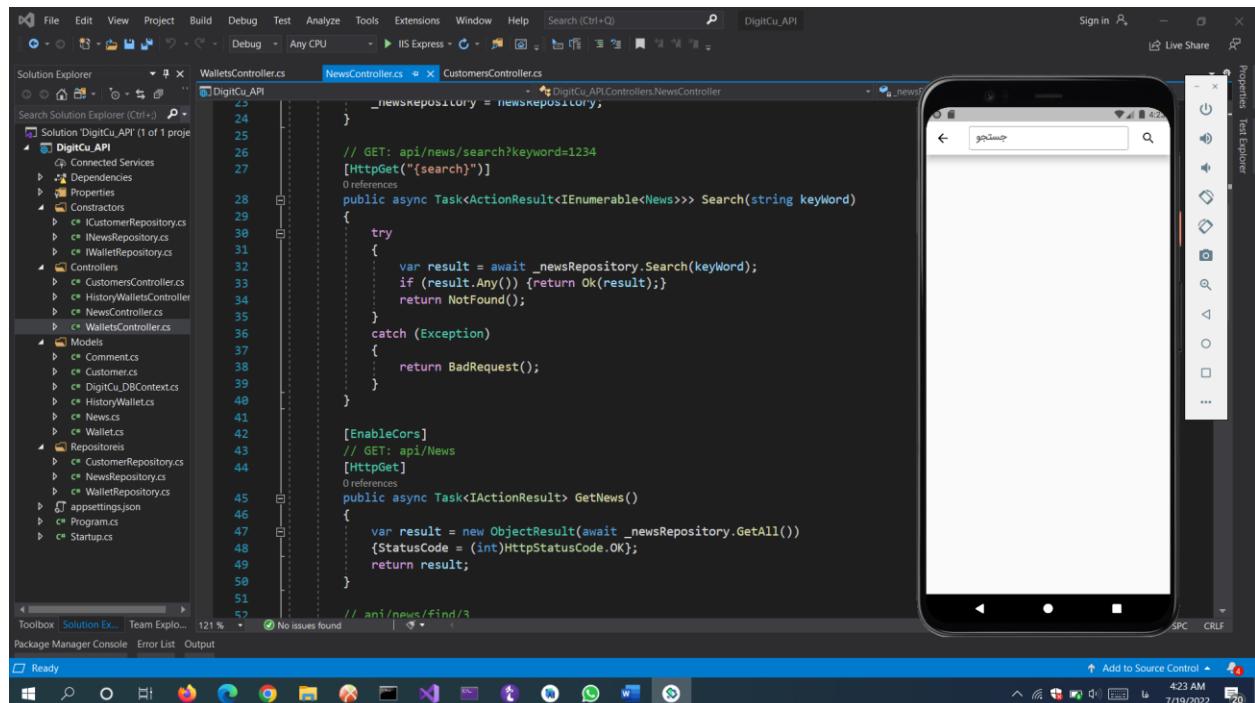
```
>> ngrok.exe http 2000
```

این پنجره را نمایش میدهد:

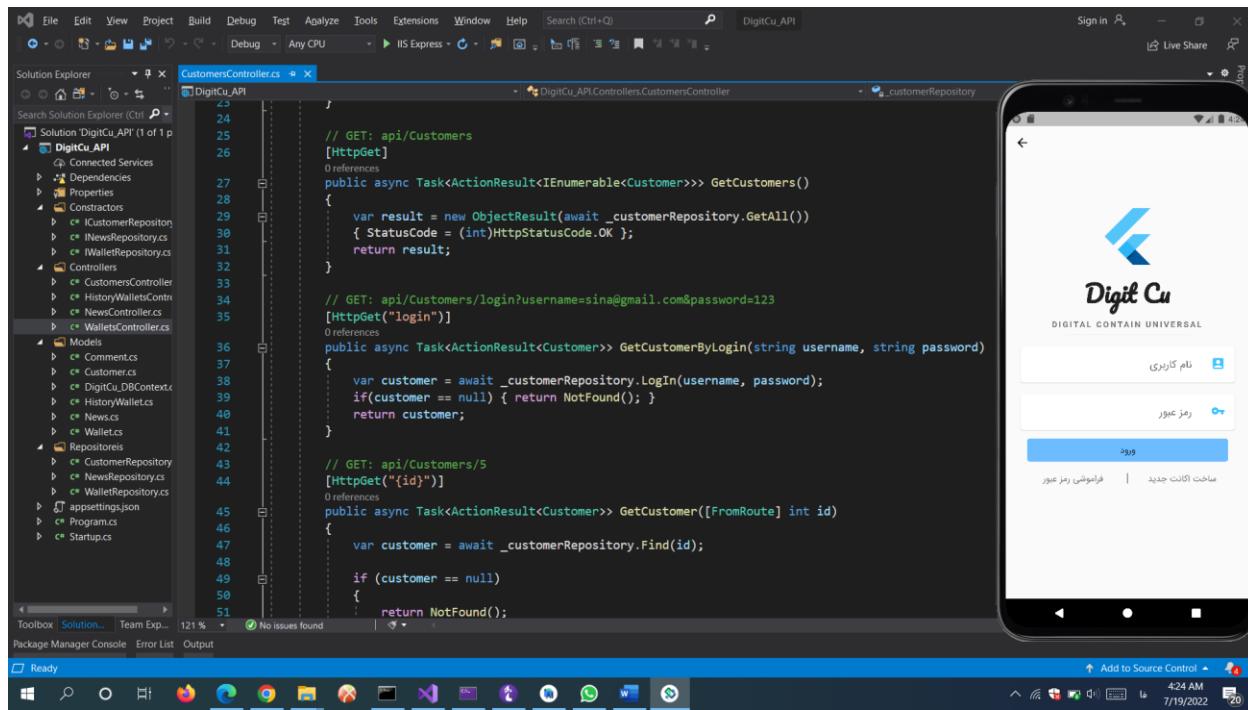


علت استفاده پشتیبانی نکردن درخواست های android به API از طریق یک port خاص. پس باید یک سرور داشته باشیم برای اشتراک گذاشتن API ولی این یک راحل ارزان و سریع است.

دو مورد به API اضافه شده که اول سرچ در اخبار است. یک کلید از جنس string دریافت میکند که در title های اخبار میگردد و هر کدام که آن کلید را داخلش داشته باشد باز میگرداند.

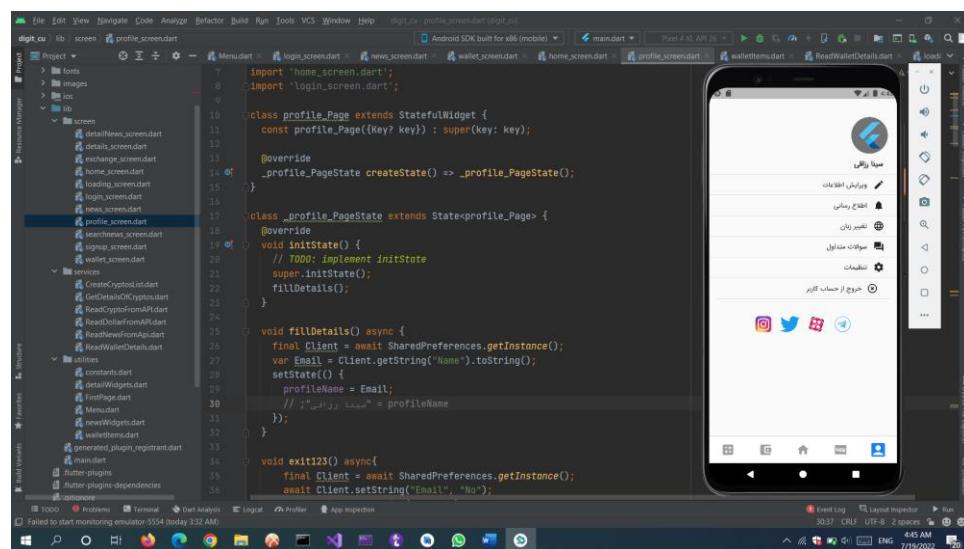


مورد دوم Login کاربر است که با گرفتن دو پارامتر username و password و بررسی آنها در دیتابیس می‌گوید آیا شخصی با این مشخصات ثبت شده است یا نه؟ اگر بود باز میگرداند و اگر نه Not Found

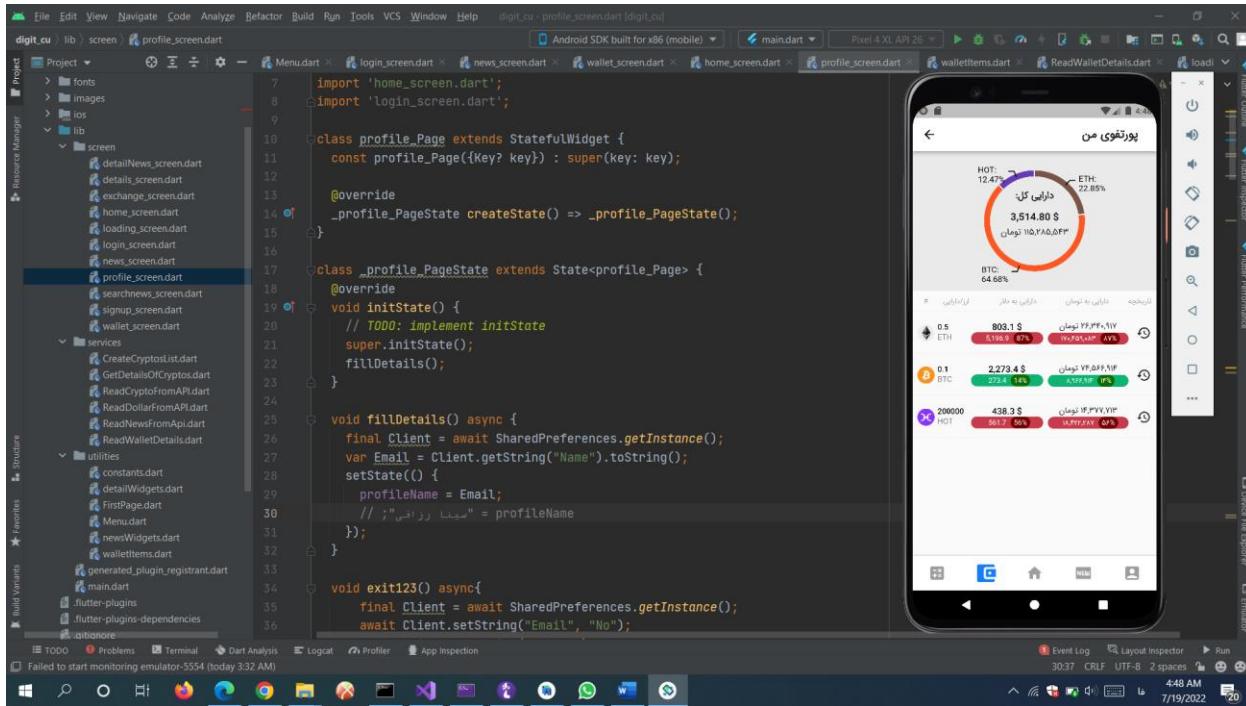


متاسفانه امشب ۲۸ تیر یک آپدیت داشته و کلی تغییر ایجاد کرده از محاسباتی و یا view های مختلف. اضافه کرد بخش جستجوی اخبار و ثبت کاربر جدید و ثبت سفارش جدید انجام نشد. (باید doc مربوط به ۳ را دوباره بررسی شود) ولی در کل آماده است و کمی ریزه کاری دارد.

برای login کاربر خطاهای مختلف قرار داده شده. اگر ایمیل و رمز را وارد نکند خطا میگیرد و اگر اشتباه وارد کند خطای دیگری میگیرد. در لحظه ورودی درست اطلاعات وی وارد صفحه پروفایل شده و اطلاعات او در shared preference داخل گوشی (جزء کتابخانه های flutter) ذخیره می شود برای انجام کارهای مختلف.

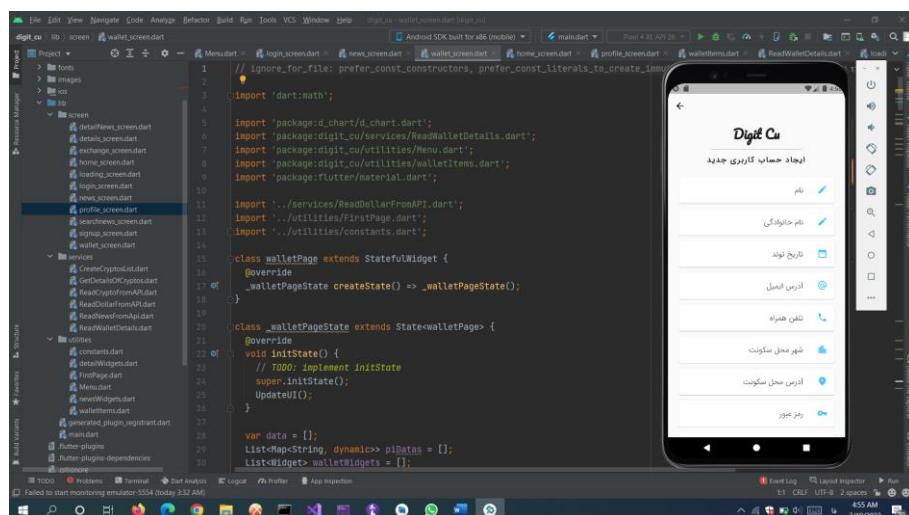


بعد از ورود کاربر حال اجازه دسترسی او به کیف پول یا همان پورتفوی داده می‌شود. اگر کاربر سفارشی ثبت نکرده باشد برای او اطلاعاتی نمایشی نمیدهد. ولی اگر اطلاعات خرید و فروشی ثبت کرده باشد، با استفاده از کوئری نوشته شده (که قبل توضیح دادم) میتوانیم برآیند ها را دریافت کنیم و حتی با داشتن قیمت‌های به روز ارزها سود و ضرر وی را نیز مشخص کنیم.

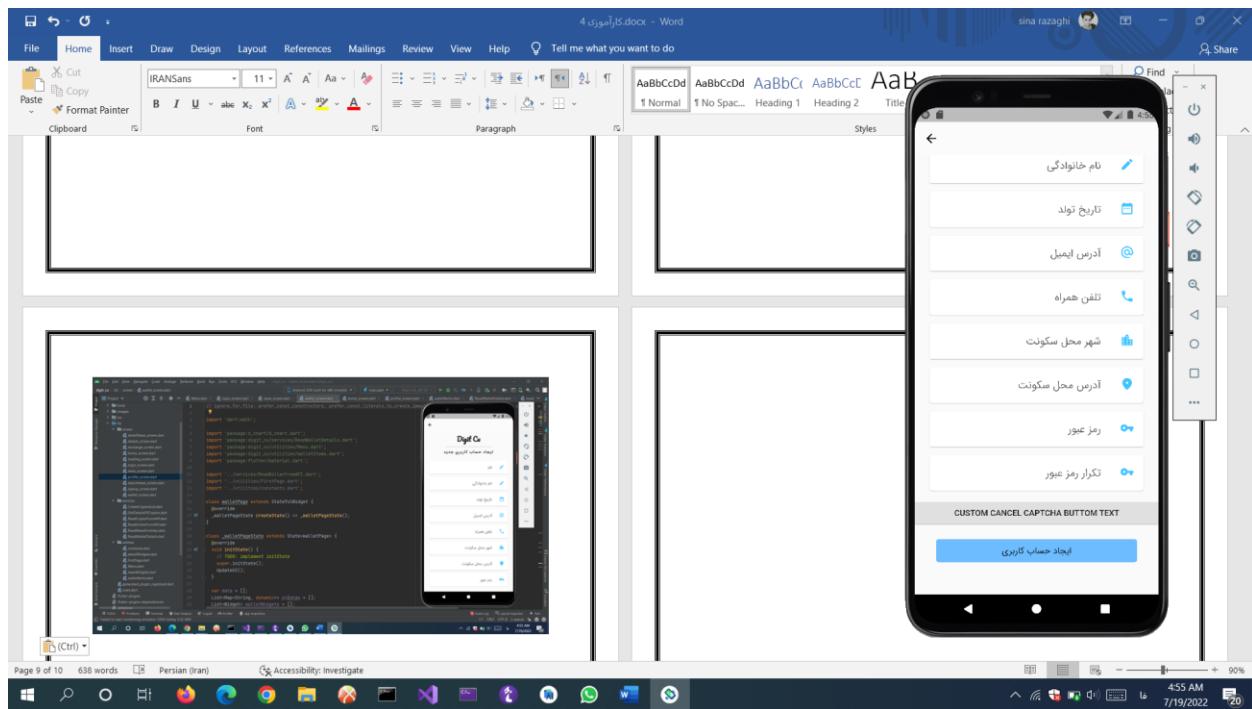


از کتابخانه `d_chart` استفاده شده در بخش اول که یک متده است `chartPi` دارد که یک لیست دو بعدی از اطلاعات را میگیرد و نمایش می‌دهد. برای رنگ رنگ های رندم در نظر گرفتم و چیزی پیش بینی نکرده ام. در وسط نیز مجموع ارزش فعلی هر ارز که خریداری کرده است به دلار و ریال.

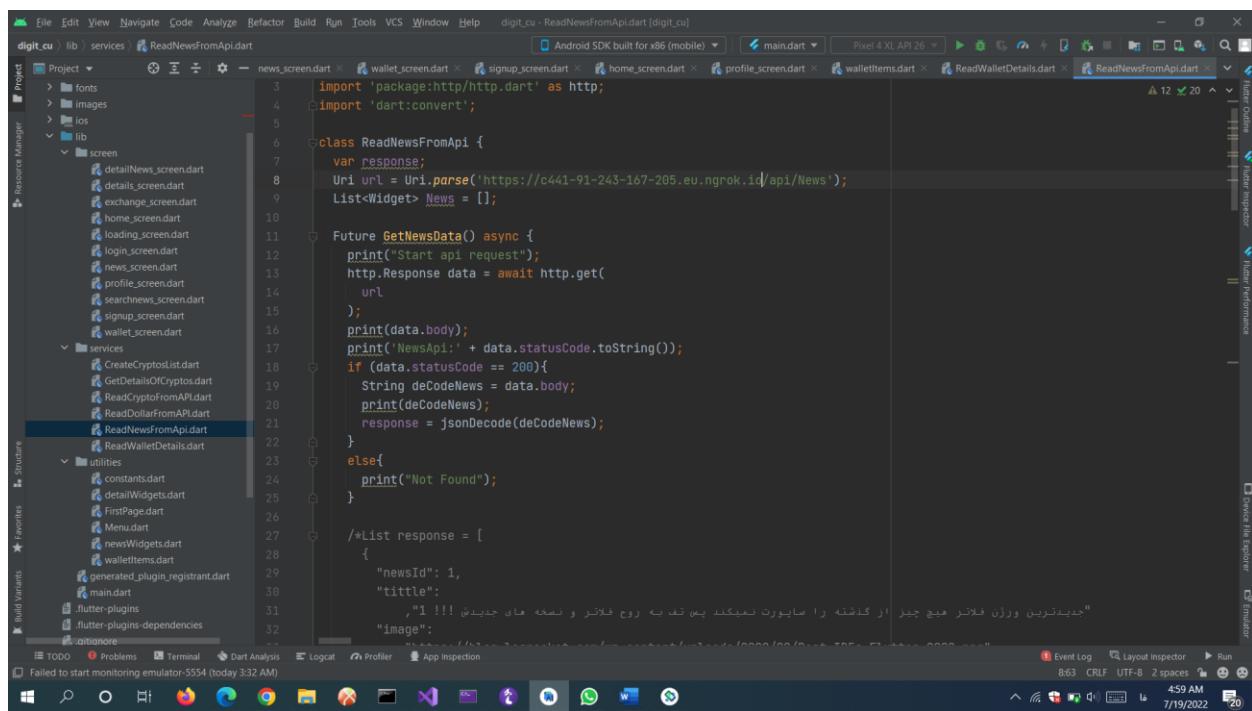
برای نمایش هر ارز نیز از یک کلاس استفاده که به شکل یک ویجت از روی آن ساخته شده با پارامتر های خواسته شده ایجاد میشود برای هر رکورد.



کتابخانه `flutter_recaptcha_v3` که وظیفه همان انسان بودن کاربر را بررسی می‌کند.



در کل روش خواندن اطلاعات برای فلاتر از طریق هر API به این شکل است:

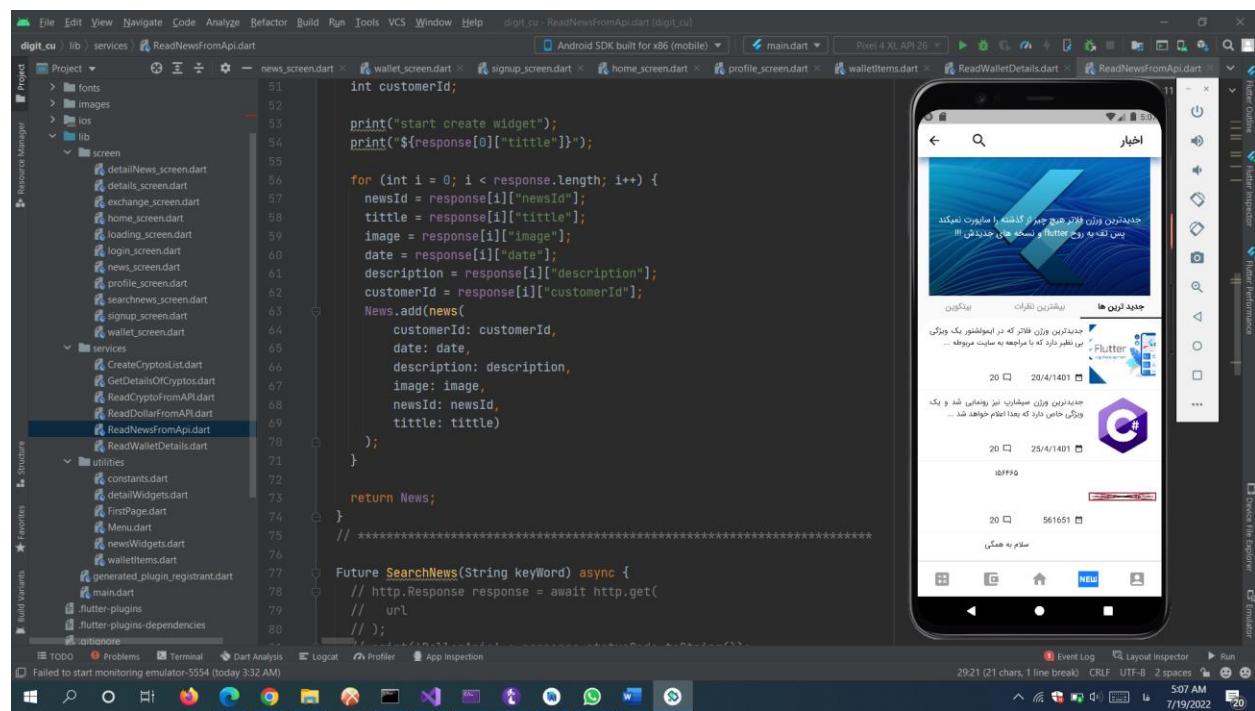


اول باید کتابخانه http به پروژه اضافه شود. از آنجایی این نوع درخواست باید async پس به یک تابع async نیز احتیاج داریم و یک درخواست به طور مثال get میسازی. نوع لینک نمیتواند یک string یا url معمولی باشد، بلکه باید از نوع Uri باید که با تبدیل () Uri.parse() به راحتی انجام می شود.

نکته: آدرس حتما باید string باشد.

بعد از دریافت اطلاعات status کد جواب بررسی می شود و اگر ۲۰۰ بود یعنی اطلاعات سالم و بدون مشکل دریافت شده و اگر هر عدد دیگری بود یعنی یک مشکل وجود دارد.

حال بدن جواب را باید deCode کنیم، چرا که جواب API از نوع Json بوده و باید تبدیل شود. بعد از تبدیل یا بصورت یک لیست دو بعدی خواهد بود که چندین رکورد به ما برمیگرداند و یا یک لیست دو پارامتری که یک رکورد.



روش ساخت لیست اخبار نیز خیلی ساده است به طوری که جواب دریافت شده بررسی می شود و اطلاعات از داخل آن استخراج می شود و داخل شی از کلاس نوع ویجت اضافه میشود و در لیست مورد نظر که از نوع ویجت است ذخیره مشود که برای نمایش داده آمده است.

نکته: محاسبات باید قبل از ساخت ویجت ها باشد و گرنه دیگر نمیشود از اطلاعات استفاده کرد و دوباره باید درخواست به API ارسال شود.

منابع:

<https://ngrok.com/download>
<https://linuxlearn.org/what-is-ngrok/>
<https://stackoverflow.com/questions/56367108/adb-exited-with-exit-code-1>
<https://github.com/londonappbrewery/Flutter-Course-Resources>
<https://git.ir/flutter/>
<https://i.stack.imgur.com/ZAggD.png>
https://www.youtube.com/results?search_query=update+android+sdk+flutter
https://pub.dev/packages/flutter_recaptcha_v3/install
https://pub.dev/packages/pie_chart
https://maktabkhooneh.org/course/%D8%A2%D9%85%D9%88%D8%B2%D8%B4-%D8%AC%D8%A7%D9%85%D8%B9-%D9%81%D9%84%D8%A7%D8%AA%D8%B1-%D8%A8%D8%A7-%D8%AF%D8%A7%D8%B1%D8%AA-%mk1289/?code=Ads.fluter&gclid=CjwKCAjwrNmWBhA4EiwAHbjEQAIJVIWbOd3N_3NX2DqTLkDwrAubW98zmdqe9ldRdrVSEDxGg_NL2BoCuBUQAvD_BwE
https://pub.dev/packages/d_chart/example
<https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0>
<https://docs.microsoft.com/en-us/ef/core/cli/powershell>
<https://www.entityframeworktutorial.net/efcore/create-model-for-existing-database-in-ef-core.aspx>
<https://toplearn.com/courses/71/%D8%A2%D9%85%D9%88%D8%B2%D8%B4-web-api-%D8%AF%D8%B1-asp-netcore>
<https://www.appcoda.com/restful-api-tutorial-how-to-upload-files-to-server/>
<https://www.tutorialsteacher.com/linq/linq-query-syntax>
https://www.w3schools.com/sql/sql_distinct.asp
<https://stackoverflow.com/questions/1136380/sql-where-in-clause-multiple-columns>

مطالب آموزشی که در طی ۱۰ ماه اخیر توسط این پروژه آموختم:

Flutter

Work with ready API in Flutter

ASP.NET Core Web API

EF Core

LINQ

OOP

SOLID

Restful API

Repository Design

SQL Server

Query writing

Ngrok

Postman