Hava Durumu Sistemi

Proje B

Arda ERTAN
Kocaeli Üniversitesi
Bilişim Sistemleri Mühendisliği
Kocaeli, Türkiye
171307065@kocaeli.edu.tr

***Özet*— Web servisler ve Mikroservisler nedir? Nerelerde kullanılır? Ne işe yarar?**

Anahtar kelimeler—Mikroservis, Web servis, API, REST, SOAP, SOA, Entity, DAL, Business, Controller

I. GİRİŞ

Günümüz ihtiyaçlarına göre birden fazla yazılmış program vardır. Ve teknolojinin gelişmesi ile beraber bu programlar birleşerek birbirlerini geliştirmeye başlamıştır. Bu projede ise dünyanın birçok ülkesine hava durumu bilgisini API'ler aracılığı ile sağlayan bir siteden alınan verileri Java programlama dili ve springboot teknolojisi kullanarak işlendi ve gerekli veriler seçildi ve işlendi. İşlenen bu veriler mikroservis mimarine uygun olarak kurulan proje içerisinde RestFull servisle beraber bir endpoint oluşturuldu. Oluşturulan bu endpoint ReactJS framework'u kullanılarak oluşturulan front-end içersinde işlenerek kullanıcı ile buluşturuldu. Bu dökümantasyon içerisinde bahsettiğimiz kavramların ana ve ara kavramları hakkında bilgi verilecektir.

II. KULLANILAN TEKNOLOJİLER

Projede öncelikle back-end bölümünü Java programlama yapabilmek için Intellij Ultimate IDE'si kullanıldı. Front-end için ise Visiual Studio Code editöründe tasarım yapılmıştır.

a) SpringBoot Nedir?

Spring tabanlı uygulama geliştirmek için hazırlanmış framework'tur. Örneğin ; bir web servis yazmak için birden çok configrasyon dönüşüm işlemlerini bizim için birkaç satır kod ile yaparak yaptığımız işi hem kısaltır hem de kolaylaştırır.

b) Hibernate Nedir?

Hibernate Java geliştiriciler için geliştirilmiş bir ORM kütüphanesidir. Nesne yönelimli modellere göre veritabanı ile olan ilişkiyi sağlayarak, veritabanı üzerinde yapılan işlemleri kolaylaştırmakla birlikte kurulan yapıyı da sağlamlaştırmaktadır.

c) Maven Nedir?

Birçok yazılım geliştirmeci öğrendikleri yeni teknolojileri hemen uygulamak denemek ister fakat her uygulamanın kendi içinde dependency ve konfigrasyonları vardır . Bu işlemleri amatör bir kişinin gerçekleştirmesi bazen kolay olmayabiliyor ve birden fazla hata ile karşılaşan kullanıcı ise bu nedenle sıkılıp daha işe başlamadan bırabiliyor. İşte bu gibi durumlarda maven bize bu konfigrasyon ve dependency yönetimini kolaylıkla sağlayrak bizi birçok işe kısa sürede hazır hale getiriyor.

d) ReactJS Nedir?

React (ReactJS veya React.js olarak da bilinir) kullanıcı arayüzü oluşturmaya yarayan açık kaynak kodlu bir javascript kütüphanesidir. Facebook önderliğinde bir geliştirici grubu tarafından geliştirilmekte olan React, Model-View-Controller prensibine uygun olarak oluşturulmuştur. React ile single-page olarak adlandırılan sayfalar geliştirilebileceği gibi React-Native ile mobil uygulamalar da geliştirilebilir.

e) Mysql Nedir?

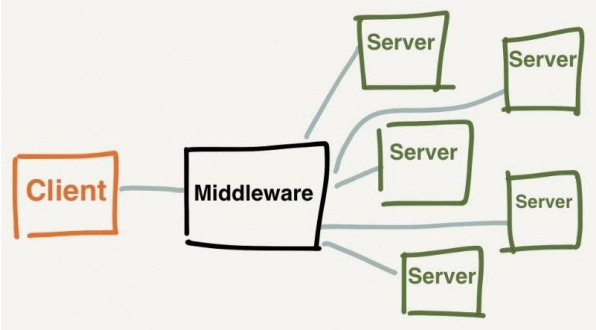
Mysql kullanıcılardan aldığımız veya kendimiz oluşturduğumuz verileri saklamak düzenlemek ve işlemek adına bir veritabanı hizmetidir.

III. YÖNTEM

Projede ilk olarak mikroservis mimarisi altında geliştirme yapacağımız için öncelikle; Mikro servis, SOA, SOAP, Rest ve RestFul kavramları hakkında kısa kısa bilgilendirmeler aşağıda bulunmaktadır.

A. SOA (SERVICE ORIENTED ARCHITECTURE)

SOA (Service Oriented Architecture) birbirinden farklı servislerin bir araya gelerek oluşturduğu yapılar ile uyumlu çalışmasıdır. Özet olarak servis içinde servistir. Bir başka deyişle birden fazla uygulamanın içerisindeki fonksiyonların farklı uygulamalar içerisinde kullanılabilecek şekilde tasarlanmasıdır.



Normalde çok katmanlı uygulamalarda her bir katman bir diğer katmanla etkileşim halinde olması gerekir böyle bir yaklaşımda hiyerarşinin sağlam olması gerekir. Basit ve klasik bir örnek vermek gerekirse projenizde bir data katmanı bir de business katmanı olduğunu varsayalım. Business katmanı data katmanını çağırarak veritabanı işlemlerini halleder, tek taraflıdır ve proje bazlı olmuş olur. Fakat data katmanını bir servis olarak oluşturursanız tek taraflı olmaktan çıkar ve istenilen her yerden kullanıma izin verir. Bu şekilde uygulamanın nereden ve/veya hangi platformda çalıştığının önemi kalmaz. SOA noktadan noktaya (point-to-point) entegrasyonlardaki bağımlılıkları ortadan kaldırır.

a) SOA Standartları

SOAP (simple object access protocol) - Servis iletişimini sağlayan mesajdır. HTTP üzerinden XML tabanlı veriler iletmemizi sağlar. Platform ve dil bağımsızdır.
WSDL (web services description language) - herhangi bir servisin arayüzünü tanımlayan (tanımlı işlemler/fonksiyonlar, giren çıkan mesajların formatları, ip ve port adresleri) xml tabanlı dildir.
UDDI (universal description, discovery and integration) - web servis hakkında bilgilerin depolandığı dizinlerdir. .Net platformu üzerine kurulmuştur. SOAP üzerinden iletişim kurar.
WS-BPEL - servislerin birleştirilmesindeki iş akışları için kullanılan bir standarttır.

b) SOA Bileşenleri

Service – SOA'nın en temel bileşenleridir.
Policies (ilkeler) - Kısaca servislerin kurallarıdır. Servisi kullanan kullanıcıları kısıtlamamıza yarar.
Contracts (sözleşmeler) - Kullanılan servisin sözleşme ve mesajlarını içerir.
Endpoints (uç noktalar) - Kullanıcıya verdiğimiz URI formunda bağlantılardır .
Messages (mesajlar) - İletişim bileşenleridir.
Service Consumer (kullanıcı/tüketici) - Servislere mesajlarala iletişim kurarlar. Örnek vermek gerekirse son kullanıcı, 3. parti yazılım veya bir servis olabilir.

c) SOA'nın Avantajları

Tekrar Kullanılabilirlik

SOA ve REST gibi mimarilerin asıl amacı tekrar kullanılabilirliktir. Bizi ekstra yükten ve karmaşadan kurtarır. Örnek vermek gerekirse bir satış sitemiz varsayalım. Bu satış sitesi yalnızca web üzerinde çalışıyorken teknolojilerin gelişmesi ile bunları Android ve IOS platformuna geçirmek istiyorsunuz. Bunu yapmak için tekrar baştan yazmak yerine bir servis katmanı ile baştan oluşturmak gibi bir zahmetten kurtuluyoruz.

Soyutlama (Abstract)

Soyutlama bizim bağımsızlaşma ve rahat kod okunması gibi birçok konuda yardımcı olur. Yani bir servis yalnızca kendi işinden sorumludur. Hadi buna küçük bir örnek verelim; Bir servisiniz yalnızca veritabanından veri çekmek için kullanılıyor ve bir diğer servisiniz ise bir veriyi kaydetmek için kullanılıyor bu da bize kod okunması ve bağımsızlık konusunda büyük bir fayda sağlıyor.

Basitleşme ve Gizleme (Facade)

Servisler birçok yazılımcı tarafından kullanılabilir. Bu nedenle verilerin oldukça basit olması gerekmektedir. Örneğin bir back-end geliştiricisi için yalnızca servisinizin cevabı önemlidir kodları değil. Fonksiyon bilgilerini tanımlamak bu kişiler için yeterli olacaktır.

Bağlılık Azaltması (Loose Coupling)

Başlıktan anlaşılacağı üzere servislerin bağlılığının en aza indirgenmesidir. Örnek vermek gerekirse birgün Mysql veritabanı ile çalışıyor iken ertesi gün MongoDB veritabanına sorunsuz geçebilmelisiniz. Servisler birbirine şema ve kontratlar ile bağlıdır. Birbirine benzeyen servisler birbirleri ile yer değiştirebilirler.

d) Soa Nerelerde Kullanılır

Birden fazla sistem varsa büyük bir ihtimalle orada bulunmaktadır. Satış sitesi örneğinden devam edelim siteniz hem mobile hem web'e hitap ediyor ve bununla birlikte 3.parti yazılımlarla çalışıyor. Her fonksiyonu kendi içerisinde katmanlara ayırıp büyük karmaşık bir uygulama yazmak yerine servislerle bunu daha dinamik ve kolay biçimde gerçekleştirebiliriz.

B. REST MİMARİSİ VE RESTFUL SERVİSLER

REST istemci-sunucu arasında hızlı ve kolay şekilde iletişim kurulmasını sağlayan bir servis yapısıdır. Açılımı Representational State Transfer olan bu ifadeyi Türkçe'ye Temsili Durum Transferi olarak çevirebiliriz.
REST, servis yönelimli mimari üzerine oluşturulan yazılımlarda kullanılan bir veri transfer yöntemidir. HTTP üzerinde çalışır ve diğer alternatiflere göre daha basittir, minimum içerikle veri alıp gönderdiği için de daha hızlıdır. İstemci ve sunucu arasında XML veya JSON verilerini taşıyarak uygulamaların haberleşmesini sağlar. REST standartlarına uygun yazılan web servislerine RESTful servisler diyoruz. Yani RESTful denildiğinde aklınıza çok farklı bir kavram veya dünya gelmesin.
REST stateless'dır, yani durum bilgisini saklamaz. Biraz daha açalım; REST standartlarında istemci-sunucu arasında taşınan verilerde ekstra başlık bilgileri saklanmaz, istemciye ait detaylar bulunmaz, bu bilgiler istemci-sunucu arasında taşınmaz. Dolayısıyla servis yönelimli uygulamalarda REST bize lightweight bir çözüm yapısı sunar.
Teorik açıdan SOAP bir protokol, REST ise bir kurallar dizisidir. SOAP servisleri RPC (Remote Process Call) çalışma yöntemini kullanır, WS-* gibi güvenlik protokollerini içerisinde barındırır, state bilgisini request ve response'larda saklar. Ancak REST'te bu durum daha farklıdır. REST servisler doğrudan bir URL çağrılarak çalışır, arada ek bir bileşen, yöntem veya protokol kullanılmaz.

SOAP bir servisi uygulamanıza dahil edebilmeniz için servisin WSDL'ına ihtiyaç duyarsınız, proxy sınıfları oluşturmanız gerekir, uzaktaki metotları tetikleyecek bileşenlere ihtiyaç vardır. DISCO, UDDI vs. derken aslında işin arka planında baya detay olduğunu görürsünüz. Kısaca istemci SOAP servisi hakkında tüm bilgilere sahip olmak ister, belirli standartları yerine getirilmeden SOAP bir servisi çağıramaz. Fakat REST servisini çağırmak için yalnızca URI'ye ihtiyacımız olacaktır. Bunun sonucunda

bize JSON veya XML formatında veri döndürecektir. Özet olarak istemci REST servisten detay istemez, yeterince esnek yapıdadır ve kural yoktur.

Bir RESTful servisi çağırmak için URI yeterli olacaktır. Örneğin; www.deneme.com/cars/1 URI'sini çağırdığımız takdirde bize id değeri 1 olan arabayı JSON veya XML formatında döndürecektir. Konumuza çok açıklayıcı bir özet olan aşağıdaki görseli incelerseniz pekiştirici olacaktır.

RESTful servisler veri iletiminde farklı HTTP metotlarını kullanmaktadır. Bunlar GET, POST, PUT, DELETE metotlarıdır. GET okuma, POST yeni kayıt ekleme(insert), PUT kayıt güncelleme(update), DELETE ise kayıt silme işlemi için kullanılır. Yapılan HTTP request'i için çağrılan URL ile beraber HTTP method bilgisi bahsi geçen 4 metottan biri olarak seçilir ve sunucu yapılan talebin kayıt üzerine nasıl etki edeceğini buna göre belirler. Kullanım kolaylığı, basit yapısı, hızlı çalışması ve esnek olması gibi özellikler RESTful servislerin avantajıdır. Fakat dezavantajları da vardır; mesela güvenlik gibi. SOAP servislerde standartlar gereği birçok güvenlik mekanizması otomatik olarak elinizin altındadır. Fakat RESTful servislerde yazılım içerisinde güvenlik sağlanmalıdır. İletişim seviyesinde güvenlik (transport level security) genellikle token aracılığıyla yapılır. İstemci kritik işlemleri çağırmadan önce bir login isteği gönderir. Bu istek sonucunda istemciye session token vb. bir değer verilir ve bundan sonra yapacağı istekler bu token değeri ile yapılır. Mesaj seviyesinde güvenlik (message level security) konusu da yine geliştirilen yazılımların içerisinde çözüm bulunması gereken bir konudur.

C. SOAP (SIMPLE OBJECT ACCESS PROTOCOL)

SOAP, HTTP üzerinden web hizmetlerine erişmek için XML tabanlı bir protokoldür. SOAP, Türkçe karşılığına bakar isek Basit Nesne Erişim Protokolü olarak bilinir. SOAP bir protokoldür veya web servislerinin birbirleriyle nasıl konuştuğunun veya onları çağıran istemci uygulamalarıyla nasıl konuştuğunun bir tanımıdır. SOAP, çeşitli programlama dilleri üzerine kurulu uygulamaların birbirleriyle kolayca konuşabilmesi ve aşırı geliştirme çabalarından kaçınabilmesi için bir ara dil olarak geliştirildi.
Günümüz dünyasında, farklı programlama dilleri üzerine kurulmuş çok sayıda uygulama bulunmaktadır. Örneğin, Java'da tasarlanmış bir web uygulaması, .Net'te başka ve PHP'de başka bir web uygulaması olabilir.

Uygulamalar arasında veri alışverişi, cihazlar arası iletişim arttığından dolayı çok önemlidir. Ancak bu karışık uygulamalar arasındaki veri alışverişi karmaşık olacaktır. Bu veri alışverişini gerçekleştirmek için kodun karmaşıklığı da öyle olacaktır.

Bu karmaşıklıkla mücadele etmek için kullanılan yöntemlerden biri, uygulamalar arasında veri alışverişi için ara dil olarak XML (Genişletilebilir İşaretleme Dili) kullanmaktır.

Tüm programlama dilleri XML dilini rahatlıkla çözebilir. Bu sebeple temel olarak XML kullanılmıştır. Ancak, veri alışverişi için tüm programlama dillerinde XML kullanımına ilişkin standart bir belirtim yoktur. İşte burada SOAP yazılımı devreye giriyor. SOAP, HTTP üzerinden XML ile çalışmak üzere tasarlanmıştır ve tüm uygulamalarda kullanılabilecek bir tür spesifikasyona sahiptir.

SOAP AVANTAJLARI

1-) SOAP tabanlı Web servisleri geliştirirken, istemci uygulamaları ile konuşmak için web servisleri için kullanılabilecek bir dile sahip olmanız gerekir. SOAP, bu amaca ulaşmak için geliştirilmiş mükemmel bir araçtır.

2-) SOAP, uygulamalar arasında veri alışverişi için kullanılan hafif bir protokoldür. SOAP programlama, kendisi hafif bir veri değişim dili olan XML diline dayandığından, dolayısıyla SOAP da aynı kategoriye giren bir protokoldür.

3-) SOAP, platformdan bağımsız olacak şekilde tasarlanmıştır ve ayrıca işletim sisteminden bağımsız olacak şekilde tasarlanmıştır. Böylece SOAP protokolü hem Windows hem de Linux platformunda herhangi bir programlama dili tabanlı uygulamayı çalıştırabilir.

4-) HTTP protokolü üzerinde çalışır –SOAP, tüm web uygulamaları tarafından kullanılan varsayılan protokol olan HTTP protokolü üzerinde çalışır. Bu nedenle, World Wide Web'de çalışmak üzere SOAP protokolü üzerine kurulu web servislerini çalıştırmak için gerekli herhangi bir özelleştirme yoktur.

IV. MİKROSERVİSLER

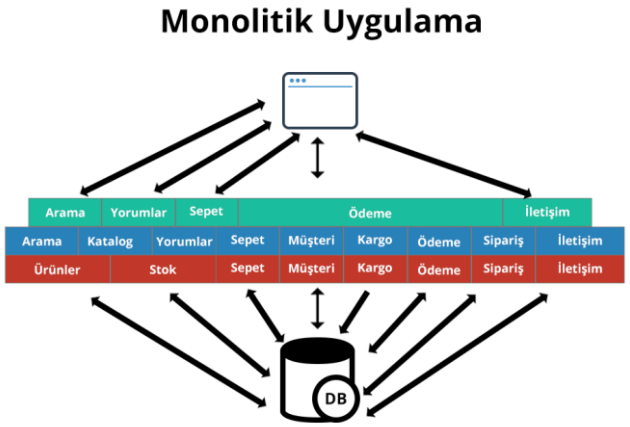
Mikroservisleri basitçe açıklayacak olursak ; Karmaşık büyük bir sistemin birbirinden bağımsız çalışan ve açık protokoller yoluyla birbiri ile iletişim kuran küçük servis parçalarına ayrılması diyebiliriz. Monolotik yapının karmaşıklığına karşı güzel bir alternatiftir.

A. Monolitik Mimari

Monolitik mimari tek bir taban üzerinde birden fazla modül içeren yapıdır. Özet vermek gerekirse tüm uygulama tek bir yerden kontrol edilir. Bu kontroller modüllerin birbiri arasında fonksiyonel veya teknik sınıflara göre ayrılması ile oluşur. Tek bir dosya üzerinden çalıştırılır ve deploy edilir.

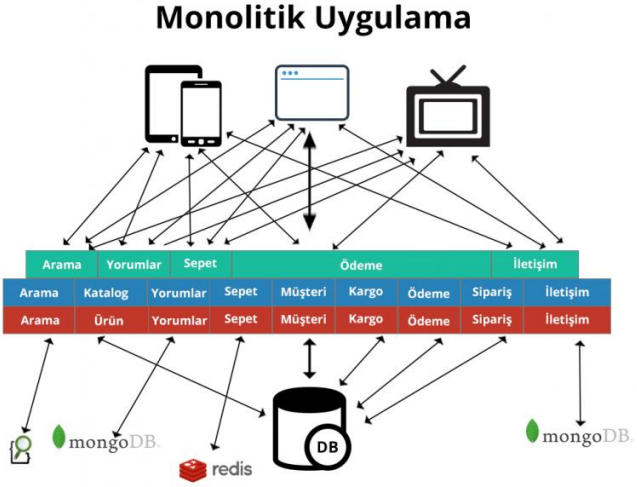
Monolitik mimariye göre tasarlanan uygulamalar üç bölümden oluşur bunlar; client yani kullanınının işlemlerini yaptığı kısım, sunucu uygulaması bölümü tüm isteklerin gerçekleştirildiği ve algoritmaların bulunğu ve son olarak verileri depolamak amaçlı veritabanı kısmıdır. Yani uygulama tek bir bütünden oluşmaktadır. Bir katman bütün

katmanları etkiler örneğin; bir katmanda sistem güncellemesi veya versiyon yükseltmesi yapacağımız takdirde bütün uygulamayı durdurmamız gerekmektedir. Aşağıdaki görsel uygulama mimarisine bir örnek olacaktır.

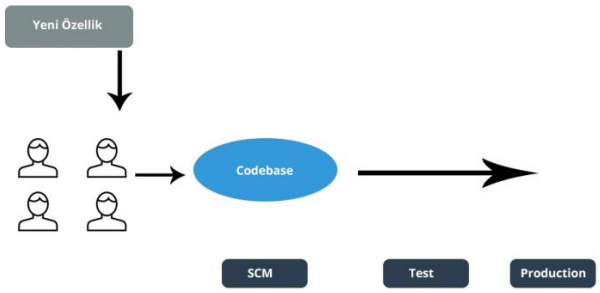


Kullanıcının işlem yaptığı katmanlara bakıcak olacak olursak. Arama, ödeme, iletişim ve sepet gibi birden fazla controller bulunmaktadır. Servis katmanında ise kullanıcının yapmak istediği işlemleri gerçekleştirmemizi sağlayan arama, katalog, müşteri vb. Modüller bulunmaktadır.
Peki neden farklı bir yapıya ihtiyacak var?

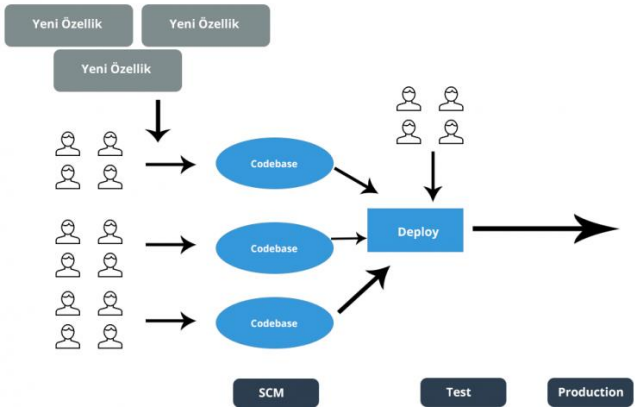
Bundan yıllar öncesinde genel olarak sadece web tabanlı uygulamalar geliştiriliyordu. Fakat günümüzde tablet, telefon, akıllı saat gibi birçok akıllı ürünün hayatımıza girmesi ile Web üzerinden yapmak istediğimiz işlemi bu cihazlardan da gerçekleştirebiliyoruz. Fakat bu erişimi sağlamak için bu mimariye ek olarak modüller eklememiz gerekiyor ve erişim sağlanacak cihaz sayısına bakarsak ve mimarimizin tek parça çalıştığını hatırlarsak bu bizim için çok fazla iş karmaşası getirecektir.



Ayrıca back end teknolojilerinden örnek vermek gerekirse; bundan yıllar önce genellikle bütün datalar ilişkisel veritabanlarında tutuluyordu. Fakat teknolojininde gelişmesi ile özelleştirilmiş üstün performans sunun No-SQL veritabanları ortaya çıktı. Örneğin kullanıcı yorumlarını ilişkisel veritabanında tutmak veritabanına ekstra ilişki sağlamak yerine bağımsız bir biçimde bu veritabanlarında tutabiliriz. Monolitik mimariye bunu eklemek ekstra bir karmaşa yaratacaktır. Yani özet olarak ilk başta oluşturması kolay olan bu yapı uygulamamız büyüdükçe kompleks bir yapı halini alacaktır.



Bu karmaşık yapıyı yönetmek için başta birkaç kişi gerekiyor ise zaman geçtikçe bu sayı git gide artacaktır.Bu da uygulama sahibi için ekstra masraf olacaktır.



Fakat burdaki asıl büyük sorunlardan biri uygulamanın sonunda bir sorun keşfedildiği takdirde hata düzeltilene kadar güncelleme geri alınması bu da uygulamanın çalışmasının durdurulmasını gerekmektedir. Bu da çok önemli zaman ve para kaybına yol açabilir.

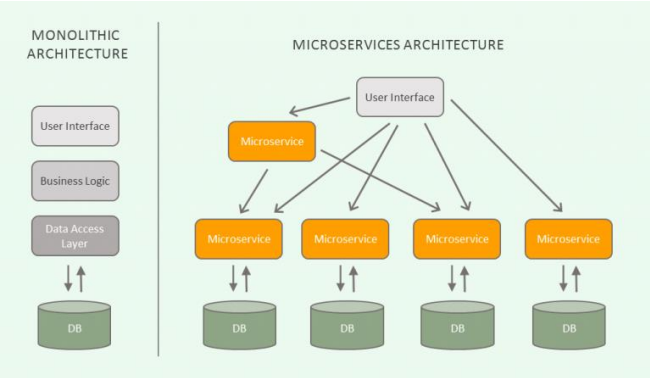
Monolitik Mimarinin Dezavantajları
• Yapının(structure) anlaşılması nispeten kolay, fakat hazmetmek zor.
• Sabit bir programlama dili içermesi. (JAVA ile başlandıysa JAVA ile devam edilmek zorunda)
• Uygulamanın modülaritesinin geliştirilen programlama diline bağlı olması.
• Uygulama büyüdükçe codebasein yönetilmesinin, bakımının ve deploy edilmesinin zorlaşması.
• Ekibe yeni bir developer katıldığı zaman uygulamanın bütün yapısını(structure) öğrenmek zorunda olması ve projeye katkı vermeye (contribute) başlama süresinin artması.

B. MİKROSERVİS MİMARİSİ

a) Mikroservis Nedir?

En kısa tabirle küçük uygulama parçalarıdır. Bir uygulamaya sonradan veya en başından dahil olup dinamik bir şekilde eklenip çıkarılabilen yalnızca kendi işine odaklı bağımsız servislerdir. Bir projede zaman geçtikçe isteğe

veya güncellemelere bağlı olarak kodlar büyüyebilir. Ve codebase ile başa çıkabilmek zorlaşır bunun gibi sorunlara müdahale etmek için kodlarımıza olabildiğince soyutlamalar ve modüller oluştururuz.



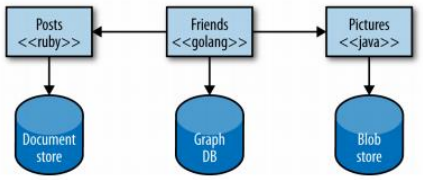
Yukarıdaki görseli incelersek monolitik mimaride uygulama düz bir çizgi halinde ilerler yani bir bütündür. Tek database ile çalışır bir değişiklik yaptığımızda bütün katmanlar etkilenir. Fakat mikroservis mimarisinde her şey parça parça haline gelmiştir. Bize uygulama yapısı konusunda ileri seviyede esneklik sağlamaktadır, yeni bir teknoloji eklemek gerekiyorsa veya farklı bir database kullanmamız gerekiyorsa diğer yapıları etkilemeden bunlar üzerinde kolaylıkla değişiklik yapabiliriz.

b) Mikroservis Mimarisinin Sağladığı Faydalar

1. TEKNOLOJİ ÇEŞİTLİLİĞİ

Her geçen gün farklı cihazlar için farklı ve yeni teknolojiler geliştiriliyor. Bizde geliştirdiğimiz uygulamalarda verimliliği arttırmak amaçlı gelişen teknolojilerden faydalanmak isteriz. Mikroservisler bize kolaylıkla bunu sağlamaktadır istediğimiz teknolojiyi programa sonradan rahatlıkla entegre edebileceğimiz yapılardır.

Örnek vermek gerekirse Java dili kullanarak bir görüntü işleme servisiniz var ve elde ettiğiniz veriyi yapay zeka ile işlemek istiyorsunuz bunun içinde Phyton programlama dili kullanmanız gerekiyor. Phyton ile küçük bir servis yazıp Javadan yolladığımız veriyi rahatlıkla çekebiliriz.



2. ESNEKLİK

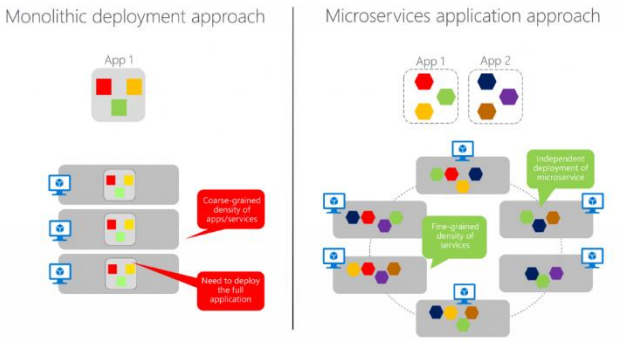
Servis içerisinde bir bileşen hata alabilir bu bileşenin hata alması durumunda başka bileşenlerde bu hatadan etkilenip programın çalışmasını tamamen bitirmesini istemeyiz. Bu olay monolitik mimaride gerçekleşir fakat mikroservislerde uygulama parçalara ayrıldığı için bir bileşenin yaşadığı sorun bir diğerini etkilemez. Bu olay bize hata çözümü ve geliştirme konusunda önemli ölçüde esneklik sağlar.

3. ÖLÇEKLEME

Büyük, Monolitik bir uygulama içerisinde her şeyi birlikte ölçeklemek zorunda kalırız. Küçük bir bölümü ölçeklendirmek amacı ile yola çıksak bile bütün proje içerisinde yapmamız gerekir. Daha küçük hizmetlerle, ölçeklendirmeye ihtiyaç duyan hizmetleri ölçeklendirebiliriz. Farklı sistemler ve platformlarda çalışabilen farklı servisler olarak geliştirildikleri için, ihtiyaça göre genişletilebilirler. Mesela yoğun "memory" kullanan bir servis ile yoğun I/O işlemi yapan bir servisi farklı şekillerde "scale" edip, kaynak kullanımını optimize etmek oldukça kolay olacaktır.

4. DEPLOYMENT KOLAYLIĞI

Monolitik bir projede küçük bir güncelleme işlemi yapmak istesek bile bütün projeyi deploy etmemiz gerekmektedir. Ancak Mikroservis mimarisi ile bu olay güncelleme yaptığımız servis üzerinde deploy işlemi uygulasak yeterli olacaktır. Servis bağımsızlığının bu avantajı bize zamandan ve işlem karmaşasına karşı büyük avantaj sağlar.



5. ORGANİZASYONLARIN DÜZENLENMESİ

Mikroservisleri programın küçük yapıtaşları gibidir. Bu benzetmeyi yapmamın sebebi; her servisin yapacağı iş bellidir ve bağımlılığı yoktur yani işi sabittir. Bir servis bir diğer servisi etkilemediği için monolotik mimari oluşturur gibi geniş çaplı planlama yapıp organizasyon karmaşasından kurtulmuş oluyoruz.

6. REUSEABİLİTY'İ OPTİMİZE ETME

Birçok projede geçmişten kalan ve kimsenin karışmak istemediği kritik yapılar vardır. Yıllar öncesinden kurulup güncelliğini yitirmiş makinelerde çalışıyor olabilir. Bu gerçekten büyük ve riskli bir iş olacaktır.

Mikroservislerde boyutlar küçük olduğundan dolayı geçmişte kalan bu uygulamaları silmek veya yer değiştirmenin maliyeti monolitik mimariye göre daha kolay olacaktır.

Mikroservis Mimarisine Geçilirken Karşılaşılabilecek Bazı Sıkıntılar:

- Önceki sistemde ilişkisel veritabanı kullanılıyor ise bunun geçişi sırasında biraz karmaşa olabilir. Örneğin; her servisin database'i değişeceğinden dolayı sorgularıda değişebilir.

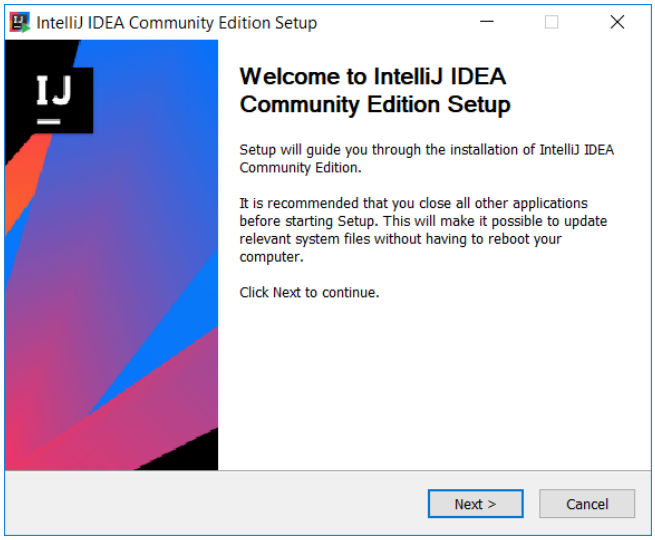
- Kullanıcı session yönetimi yapısı farklılaştırılmak zorunda kalınabilir.
- Servisler farklı platformda ve ortamlarda çalışabileceğinden, bunların yönetim ve monitoring maliyeti doğacaktır.
- Fazlaca database ve transaction yönetimi zor olabilir.

Geçiş sırasında mikroservisin getirdiği faydalar ve sıkıntılar bir teraziye konup kıyaslandığında hangi tarafın ağır basacağı görülüp ihtiyaçlara göre karar verilmesi doğru olacaktır.

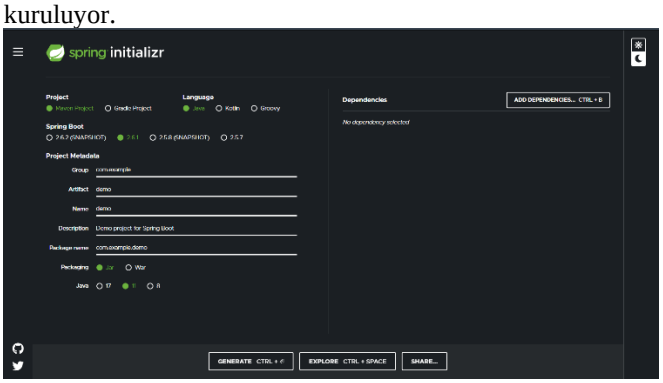
V. UYGULAMA

A. Intellij Ultimate kurulumu ve Spring Boot Projesi oluşturulması

Intellij Ultimate sürümünü kullanmak için "jetbrains.com" sitesine okul maili aktivasyonu yapıldığı takdirde ücretsiz kullanım izni verecektir. Ardından indirme linkine tıklayarak aşağıda bulunan ekrana çıkan "next" seçeneğine defalarca tıklayrak kurulumu kolayca tamamlıyoruz.

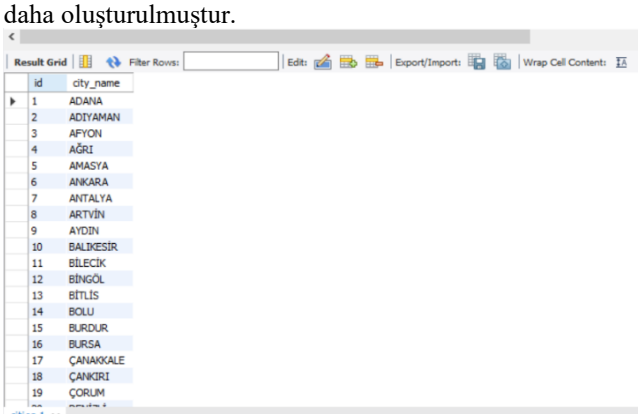


Intellij IDE kurulumunu tamamladıktan sonra "start.spring.io" sitesinden eklemek istediğiniz dependency'leri ve proje türünü seçerek "Generate" butonuna basarak dosyamızı indiriyoruz. İndirilen dosyayı Intellij IDE'si içerisinde açtıktan sonra projemiz otomatik kuruluyor.

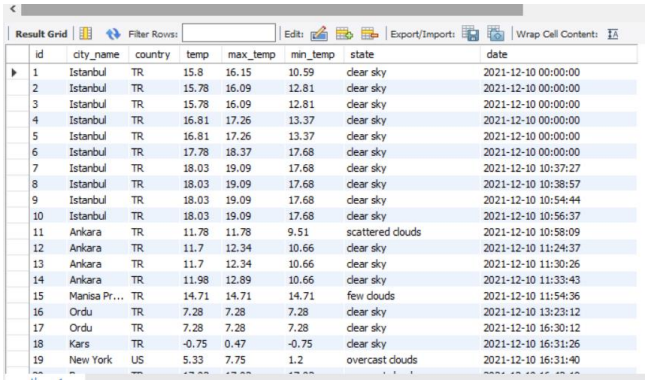


B. Mysql Veritabanı Oluşturulması ve Entity Class Tanımı

İlk olarak bilgileri çekeceğimiz API hizmeti olan "openweather.com" sitesesinden bir bölgenin hava durumu ile ilgili gerekli verileri JSON dosya formatında alındıktan sonra aralarından veritabanına yazılacaklar seçildi. Bu bilgiler veritabanında sutün isimleri tanımlanır ve oluşturulur. Hava durumu uygulamasında Türkiye içinde şehir bilgilerine göre bilgi getireceğinden dolayı veritabanı içerisinde şehirler ve plaka numaraları bulunan bir tablo daha oluşturulmuştur.

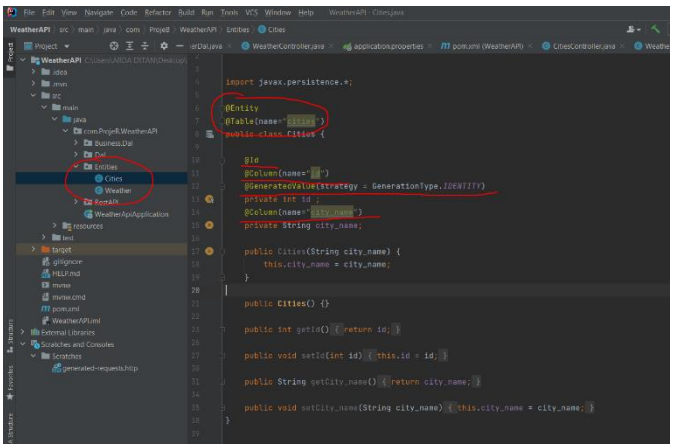


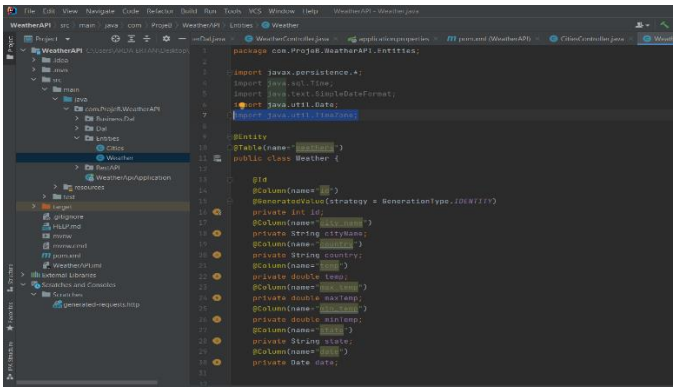
id	city_name
1	ADANA
2	ADIYAMAN
3	AFYON
4	AĞRI
5	AMASYA
6	ANKARA
7	ANTALYA
8	ARTVİN
9	AYDIN
10	BALIKESİR
11	BİLECİK
12	BİNGÖL
13	BİTLİS
14	BOLU
15	BURDUR
16	BURSA
17	ÇANAKKALE
18	ÇANKIRI
19	ÇORUM



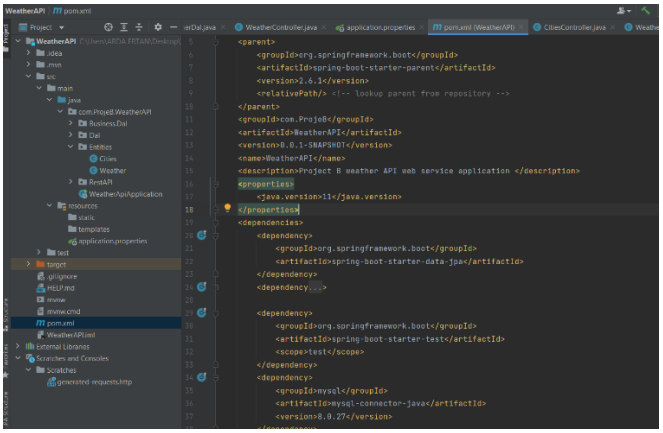
id	city_name	country	temp	max_temp	min_temp	state	date
1	Istanbul	TR	15.8	16.15	10.59	clear sky	2021-12-10 00:00:00
2	Istanbul	TR	15.78	16.09	12.81	clear sky	2021-12-10 00:00:00
3	Istanbul	TR	15.78	16.09	12.81	clear sky	2021-12-10 00:00:00
4	Istanbul	TR	16.81	17.26	13.37	clear sky	2021-12-10 00:00:00
5	Istanbul	TR	16.81	17.26	13.37	clear sky	2021-12-10 00:00:00
6	Istanbul	TR	17.78	18.37	17.68	clear sky	2021-12-10 00:00:00
7	Istanbul	TR	18.03	19.09	17.68	clear sky	2021-12-10 10:37:27
8	Istanbul	TR	18.03	19.09	17.68	clear sky	2021-12-10 10:38:57
9	Istanbul	TR	18.03	19.09	17.68	clear sky	2021-12-10 10:54:44
10	Istanbul	TR	18.03	19.09	17.68	clear sky	2021-12-10 10:56:37
11	Ankara	TR	11.78	11.78	9.51	scattered clouds	2021-12-10 10:58:09
12	Ankara	TR	11.7	12.34	10.66	clear sky	2021-12-10 11:24:37
13	Ankara	TR	11.7	12.34	10.66	clear sky	2021-12-10 11:30:26
14	Ankara	TR	11.98	12.89	10.66	clear sky	2021-12-10 11:33:43
15	Manisa Pr...	TR	14.71	14.71	14.71	few clouds	2021-12-10 11:54:36
16	Ordu	TR	7.28	7.28	7.28	clear sky	2021-12-10 13:23:12
17	Ordu	TR	7.28	7.28	7.28	clear sky	2021-12-10 16:30:12
18	Kars	TR	-0.75	0.47	-0.75	clear sky	2021-12-10 16:31:26
19	New York	US	5.33	7.75	1.2	overcast clouds	2021-12-10 16:31:40

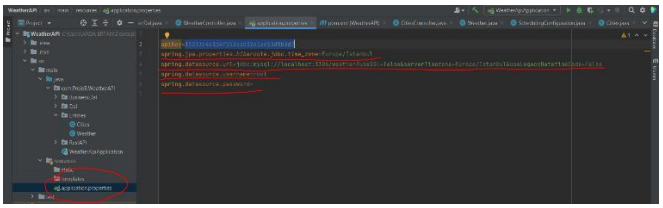
Verilerimiz ve ekleneceği tablolar oluşturuldu şimdi servisimizin bu yapı ile beraber çalışması için "@Entity" notasyonunu kullanarak private türde değişkenlerimizi tanımlıyoruz. Bu sınıflar MVC mimarisine uygun olmasına adına veritabanlarımızın tanımlandığı Entity paket katmanında bulunmaktadır.





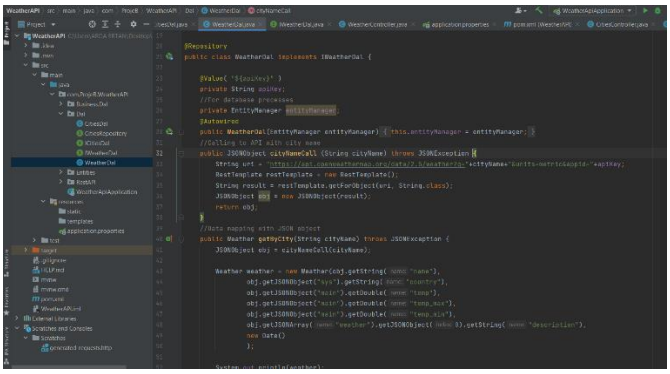
Yukarıda görsellerde belirtildiği üzere birden fazla nostasyon görüyoruz. Bunlar;
Table: Veritabanına bağladığımız tablonun ismi.
Id: Her tablonun bir primary key'i olmak zorundadır bunun için id veririz,
Column: Tablodaki eşlesen sütun ismi ,
GeneratedValue: Her veri eklediğimiz id'lerimiz otomatik atanması için bu notasyona ihtiyacımız vardır.
Geriye kalan fonksiyonlar ise yapıcı(constructor) fonksiyon, getter, setter ve Override yapılmış ToString fonskiyonudur.
Sınıfımızı tamamen oluşturduk fakat bağlantı henüz tamamlanmadı programımız MySQL'i tanımayacaktır , bunu yapmak için "pom.xml" dosyamız içinde gerekli dependency'lerimizi ekliyoruz. Ardından application.properties dosyamızda veritabanı bilgilerimizi tanımlıyoruz.

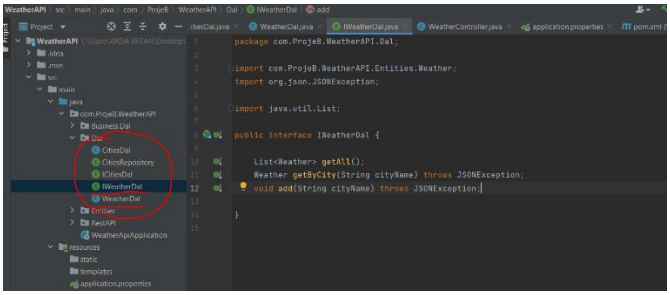




C. Data Access Layer (Veri Erişim Katmanı)

Veritabanımız ile bağlantıları sağladık verilerimiz projemiz ile eşleşti. Şimdi verilere erişip bunlar üzerinde işlem yapmamız gerekiyor. Bunun için ise Dal (Data Access Layer-Veri Erişim Katmanı)' nında işlem yapacağız bunun için sınıfımızın en başına "Repository" nostasyonunu ekliyoruz. Projemizde ilk olarak daha önce de bahsettiğimiz openweather.com sitesinden aldığımız API verilerini çeken bir fonksiyon tanımlıyoruz. Bu datayı öncelikle bir JSONObject türünde bir nesneye atıyoruz. Bu objeden alınan verileri JSONObject sınıfı içerisinde bulunan getter metodları ile çekerek seçtiğimiz bu verileri Entity paketimizde bulunan Weather sınıfında bulunan yapıcı fonskiyona atıyoruz ve Weather sınıf türünde bir nesnemiz oluşuyor. En önemli kısım tamamlanmış bulunuyor. Kalan kısımda ise bu nesnemizi veri tabanına ekleme , çıkarma , güncelleme ve getirme işlemlerini yapcağız , bu nedenle EntityManager sınıfından bir nesne türetiyoruz. Bu sınıf bizim veritabanına yapmak istediğimiz işlemlerin fonsksiyonlarını içeriyor.
İlk olarak veri tabanına ekleme fonksiyonu olan "add" fonksiyonunu tanımlıyoruz. Bu fonksiyon kullanıcıdan aldığımız "CityName" parametresi içeriyor. Bu parametre ile API'den şehir adına göre verileri çekip eşleştiriyor ve ardından "Weather" objemize atıyoruz. Hibernate'ten aldığımız Session sınıfından bir obje içerisine EntityManager nesnemizi "unwrap" fonksiyonu ile atıyoruz. Sonrasında session nesnemizle "SaveOrUpdate" fonskiyonu içerisine weather objemizi attığımızda verimizi veritabanına kaydetmiş oluyoruz.
Son olarak servis mimarisini tanımlamak için sınıfımızın içindeki fonksiyonları bir "Interface" sınıftan implement edildikten sonra çağırmamız gerekiyor , bunun için "IWeatherDal" isminde bir interface tanımlıyoruz ve veri erişim katmanında yapmak istediğimiz fonskiyonları bu sınıf içinde tanımlıyoruz ardından tekrar veri erişim katmanımıza dönüyoruz ve interface'imizi implemente ediyoruz ve fonksiyonlarımız "Override" nostasyonu alıyor bu da bize program içerisinde bir dinamiklik sağlıyor. Aynı işlemleri "City" tablomuz içinde tekrarlıyoruz.



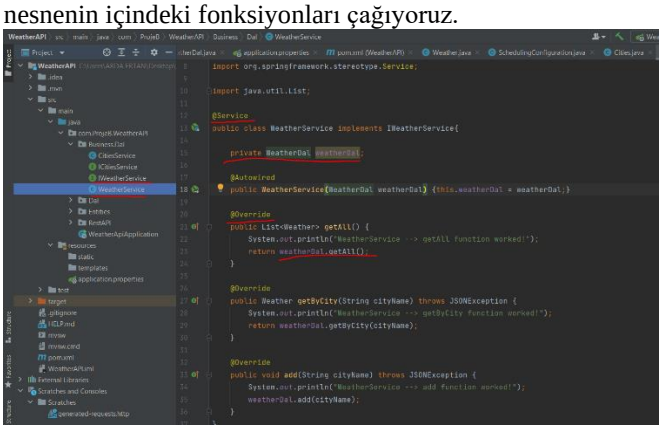


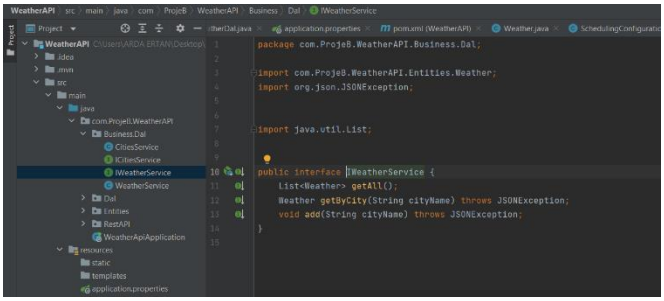
D. Business – Service Layer (İş - Servis Katmanı)

Mikroservis mimarisine göre çalışmak istediğimiz için bizim herhangi bir güncellemede bütün kodları baştan aşağı değiştirmek yerine yalnızca iş yaptığımız katmanda değiştirmemiz daha mantıklı olacaktır. Bunun için veri erişim katmanının üstüne birde business(iş) katmanını ekliyoruz. Bu projede ekstra olarak bir yapmamış olsakta

ana maksadımız katman içerisinde değişiklik yapıp göstermek istiyorsak burada tanımlarız. Kafamızda olaşabilecek soru işaretlerini kaldırmak adına bir örnek verelim; Dal'dan weather objemiz içerisinda hava durumu derece bilgisi 278 Kelvin olsun. Biz servis kullanıcısına direk veriyi bu şekilde göndermek yerine (derecebilgisi-273) gibi bir işlemi burada aynı fonksiyon içerisinde yaptığımız takdirde daha temiz bir veri elde edebiliriz. Ve veri erişim katmanında değişiklik yapmadığımız için her katmanı değiştirmemize gerek kalmayacaktır.

İşin teoresini yukarıda açıkladık şimdi ise bu olayı nasıl oluşturacağımızı anlatalım. İlk olarak servis sınıfımızı "Service" notasyonu ile tanımlıyoruz ardından oluşturduğumuz veri erişim katmanındaki fonskiyonlarına erişim için bir nesnesini tanımlıyoruz. Veri erişim katmanındaki yapının aynısı olarak bir interface'de fonskiyonlarımızı tanımlayıp sınıfımızda implemente ediyoruz. Fonksiyonlar içerisinde Dal'dan oluşturduğumuz nesnenin içindeki fonksiyonları çağıyoruz.

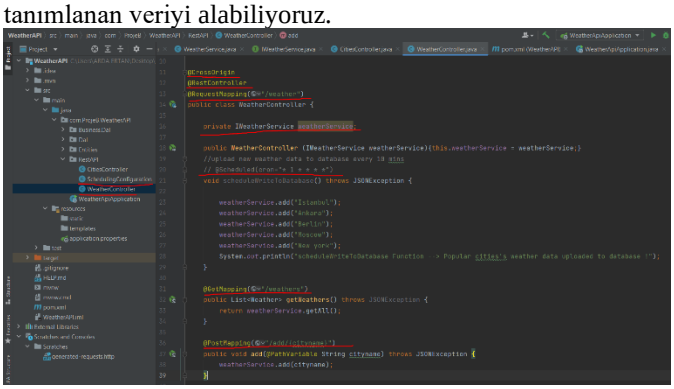


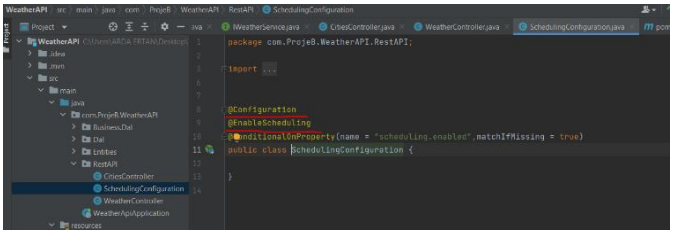


E. Rest API – Controller (Kullanıcı Arayüzü Katmanı)

Servisimizin işlemleri neredeyse tamamlandı artık verilerimizi kullanıcı ile buluşturabilecek bir endpoint oluşturmamız yeterli olacaktır. Bu endpoint sayesinde front-end içerisinde serviste oluşturduğumuz veriyi alıp kullanıcıya ulaştıracağız.

Genel maksadımızı açıkladık uygulamaya gelelim; İlk olarak servisimizin port'u farklı bir port'ta çalışabilmesi için "CrossOrigin" notasyonunu kullanılıyoruz. HTTP istek işlemleri olan ; GET, POST, PUT, DELETE işlemlerini gerçekleştirebilmek için ise "RestController" nostasyonunu kullanıyoruz. Bunun yanında istek atıcağımız end point'i isimlendirmek adına "RequestMapping" notasyonunu kullandık. Oluşturduğumuz servis katmanına bağlantı kurmak için ise daha önceden de yaptığımız gibi bir nesnesini oluşturarak sağlarız. Sisteme farklı bir dinamik etmek adına bazı özel şehirlerin hava durumlarını belirli zaman aralıklarla veritabanına kaydetmek için "Schedule" configrasyonu kullarak çalıştırıyoruz. Ardından yapmak istediğimiz metoda göre (metod ismi +Mapping) notasyonu ile belirterek servisimi çağırıyoruz. Kullanıcıdan şehir ismini alacağımız için "citymame" değişkenin yanına "PathVariable" notasyonunu tanımlayarak Uri'den tanımlanan veriyi alabiliyoruz.





F. Front-End projesi Oluşturulması (ReactJS)

Servisimiz tamamlandı sıra kullanıcıya bir görsellik sunmakta bunun için ReacJS framework'unu kullanarak basit bir arayüz oluşturalım. Bunun için editör olarak Visiual Studio Code kullandık, basit kurulumunun ardından projemizi oluşturalım. React projemizi "npx create-react-app uygulama-ismi" komutu ile içinde bulunduğumuz dosya dizini içerisine oluşturduktan sonra VSCode terminalini açıp "npm start" komutu ile kendi portumuzda projemizi çalıştırıyoruz.

Projemizi derinden anlamak adına en dıştan en içe doğru açıklama yapmak daha doğru olacaktır, bu nedenle ilk olarak klasör yapımızdan başlayalım.

- **My-app:** Projemizi çalıştırdığımız tüm klasörleri içinde bulunduran klasördür.
- **Node_modules:** Npm paket yöneticisi kullanarak programımız için gerekli kütüphaneleri import ederiz. Bu modülleri içeren klasördür.
- **Public:** Html, css, json gibi genel dosyaların bulunduğu klasördür.
- **Src:** React projemizi oluşturduğumuz projemizin neredeyse tüm parçalarının bulunduğu klasördür. Programımızın yapısını daha iyi anlamak adına bu klasörü biraz daha detaylandıracağız

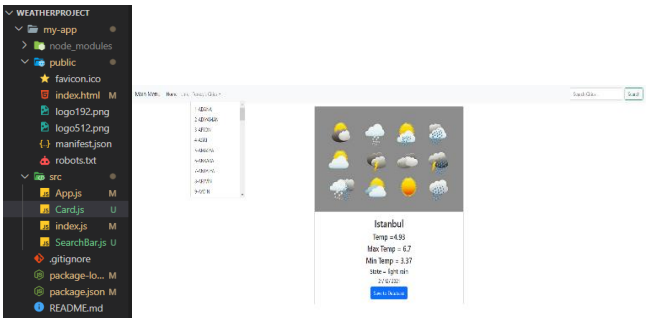
I) İndex.JS

Uygulamamızın temel çercevesidir bu dosya içerisinden bir root (kök) oluşturduktan sonra index.html dosyamıza yolluyoruz bu sayede kullanıcıya bu framework aracılığı ile görsellik sağlıyoruz. Bu dosya içerisinde daha kolay görsellik katmak adına bootstrap kütüphanesinin importlarını ekliyoruz.

II) App.js

İsminden de anlaşılacağı üzere uygulamanın parçalarını birleştirdiğimiz bölümdür. Parçaları detaylı olarak bir sonraki bölümlerde açıklayacağız. Burada daha çok içerdiği fonskiyon ve state'leri açıklayacağız. İlk olarak veritabanımıza kaydedeceğimiz formatta React'ın hook'larından biri olan "useState"'i kullanarak bir state tanımlıyoruz. Servisimiz'den verileri çekmek için "axios" kütüphanesinden yardım alan asenkron bir fonksiyon ile verileri çektik ve elde ettiğimiz veriyi "setState" fonskiyonu ile güncelliyoruz. Artık elimizde API'den çektiğimiz veriler var bunları "Card" nesnemize prop olarak yolluyoruz.

Bir diğer fonksiyonumuz ise servisimizin bağlandığı Türkiye şehirlerini ve plakaları tuttuğumuz servis bu serviside "fetchCity" fonksiyonunda verilerini alıyor ve "setCities" ile dolduruyoruz. Bunlara ek olarak bir de "useEffect" hook'umuz bulunuyor burada ise uygulama ilk başlatıldığı zaman ve şehir bilgisi her güncelledeği takdirde bizim "fetch" fonksiyonlarımızı çağıyor.
Son olarak ise "SearchBar" ve "Card" bileşenlerinden state bilgisi çeken fonksiyonlar tanımlanmıştır.



III) SearchBar.js

Bu bileşenimizde daha çok bootstrap'ten alarak oluşturduğumuz bir arama menüsüdür. Burada App.js üzerinden çağırdığımız şehir servisimizdeki verileri set ediyoruz. Ek olarak yalnızca şehir ismine göre arama yapmak için bir input bulunmakta buraya girdiğimiz şehir ismini App.js'e gönderiyor ve fetch fonksiyonlarımızı güncelliyor ve giriş yaptığımız şehirin bilgisini Card bileşenimize atıyor.

IV) Card.js

Kullanıcıya çektiğimiz verileri güzelleştirerek sunduğumuz bir tasarım bileşenidir. App.js'ten aldığımız hava durumu bilgilerini set ediyoruz farklı olarak ise içinde bulunduğu veritabanına kaydetme butonu bulunuyor bu buton sayesinde servisimizde yazdığımı veritabanına kaydetme fonksiyonunu çağıyoruz. Card bileşeni içerisinde bulunan şehir ismine göre veritabanımıza güncel saate göre kayıt işlemi yapıyor.

G. Karşılaşılan hatalar

1. Gerekli dependencylerin eklenmemesi
2. NPM package güncellemesi yapılmaması
3. Veri tabanı server'ı çalıştırılmadan veri çekilmesi
4. Classbase compenent içerisinde react hook kullanımı
5. Arrow fonksiyonu asenkron çalıştıramamız
6. UseEffect içinde state kontrolü ile sonsuz döngüye girmesi
7. Date sınıfı kullanımı
8. Application.properties bölümünde timezone set edilmemesi
9. Notasyonların eklenmemesi
10. Versiyon hataları
11. CORS hataları
12. JSONException eklememek

KAYNAKÇA

[1] https://spring.io/quickstart
[2] https://stackoverflow.com/
[3] https://www.youtube.com/watch?v=jig4GPO7OTo&list=PL-Hkw4CrSVq_eyixSZ4sVI1x6d7akLpsy
[4] https://www.youtube.com/playlist?list=PLZlA0Gpn_vH8EtggFGERCwMY5u5hOjf-h
[5] https://reactjs.org/docs/getting-started.html
[6] https://www.javatpoint.com/spring-boot-tutorial
[7] https://www.udemy.com/course/programcilik-kursu/learn/lecture/19341662?start=285#overview
[8] https://mvnrepository.com
[9] https://getbootstrap.com
[10] https://www.npmjs.com
[11] https://medium.com/@furkanbegen/mikroservis-mimarisi-nedir-ve-avantajları-nelerdir-1369175cc4e6
[12] https://medium.com/architectural-patterns/soa-service-orietented-architecture-nedir-75092cd90d61
[13] https://www.javatpoint.com/restful-web-services-tutorial