Hava Durumu Uygulaması

Yazılım Mühendisliği Ana Bilim Dalı

Dönem Projesi

Burak Olgun

ORCID 0000-0000-0000-0000

Proje Danışmanı: Dr. Öğretim Üyesi Serpil Yılmaz

Ocak 2024

Hava Durumu Uygulaması

Özet

Java Spring Boot ile geliştirilmiş olan bu proje, Controller üzerinden dakika başına belirli istekleri WeatherStack API'sine HTTP/GET yöntemiyle ileterek işlevsel bir hava durumu uygulaması oluşturmayı amaçlamaktadır. WeatherStack'ten elde edilen API anahtarı, Postman aracılığıyla bir REST API kullanılarak edinilir. Bu proje, kullanıcıya şehir bazlı hava durumu bilgilerini anlık olarak getirme ve görüntüleme imkanı sunar.

Anahtar Sözcükler: Java, Postman, Hava Durumu, Spring Boot, Rest API

Weather App

Abstract

This project, developed using Java Spring Boot, aims to create a functional weather application by sending specific requests to the WeatherStack API every minute through the Controller using the HTTP/GET method. The API key obtained from WeatherStack is acquired through a REST API using Postman. This project provides users with the ability to retrieve and display real-time weather information based on city.

Keywords: Java, Postman, Weather, Spring Boot, Rest API

Proje çalışmasını değerli aileme ithaf ederim.

Teşekkür

Proje çalışmasında katkılarından dolayı danışmanım sayın Dr. Öğretim Üyesi Serpil Yılmaz'a teşekkür ederim.

İçindekiler

Özet i

Abstract ii

Teşekkür iv

Şekiller Listesi vii

Kısaltmalar Listesi viii

1 Projede Kullanılan Araçlar 1

1.1 Java 1

1.1 IntelliJ IDEA 2

1.1 Spring Boot 2

1.1 Postman 3

2 Projenin Paketleri ve Bağlantıları 4

2.1 Model 4

2.2 Repository 5

2.3 Service 5

2.4 DTO 6

2.5 Exception 7

2.6 Constants 8

2.7 Config 8

2.8 Controller 9

2.8.1 Validation 10

2.9 WeatherStack API 10

3 Projenin Çalışması 11

3.1 Uygulamanın Arka Planda Çalışma Şekli 11

3.2 Uygulamanın Çalışması 11

3.2.1 Veritabanına Ekleme 12

3.3 Sonuçlar 13
Kaynaklar..... 14

Şekiller Listesi

Şekil 1.1 Java programlama dilinin alanları 9
Şekil 2.1 WeatherEntity değişkenleri 12
Şekil 2.2 Veritabanı bağlantısı için repository paketi..... 13
Şekil 3.1 Postman uygulamasında şehir bilgileri..... 20
Şekil 3.2 Veritabanına eklenen şehirler 21

Kısaltmalar Listesi

GET	API'den veri alır.
POST	API'ye yeni veri gönderir.
PUT	API'de ki verileri günceller.
API	Uygulama Programlama Arabirimi
ORM	Nesne İlişkisel Eşleme
JPA	Java Kimlik API
HTTP	Üstün Metin Transfer Protokolü
URL	Tekdüzen Kaynak Bulucu
IDE	Entegre Geliştirme Ortamı

Bölüm 1

Projede Kullanılan Araçlar

Bu bölümde, bitirme projesinde kullanılan araçlar açıklanmıştır.

1.1 Java

Java, Sun Microsystems mühendislerinden James Gosling tarafından geliştirilen açık kodlu, nesne yönelimli, yerden bağımsız, yüksek verimli, çok işlevli, üst düzey, adım adım yapılandırılmış bir dildir.

Şekil 1.1: Java programlama dilinin alanları

Java programlama dili, nesne yönelimli programlamayı savunan, saf nesne yönelimli olma hedefiyle varlığını sürdüren ve bu felsefeyi sonuna kadar uygulamaya çalışan, platformdan bağımsız yani bağımsız olma felsefesiyle ilerleyen bir dildir ve belirli bir platforma ihtiyaç duymayan "wora". Wora'nın felsefesini anlatmamız gerekirse; “Bir kere yaz, her yerde çalıştır” mantığı var.

Java'da program doğrudan anlaşılır koda çevrilmez. JVM (Java Virtual Machine) tarafından yorumlanan bayt koduna (.class dosyası) dönüştürülür. Bu nedenle derlendiğinde her yerde çalıştırılabilen bir bytecode dosyası oluşturur. Burası onun bir kez yazıldığı ve doğada her yerde çalıştırıldığı yerdir.

Java uzun yıllardır kullanılan bir programlama dili olduğundan yıllar içerisinde birçok farklı güncelleme almış ve farklı versiyonları yayınlanmıştır. Son olarak 2023 yılın da JavaSE 21 sürümü yayınlandı. Kullanıcıların uygulamaları çalıştırmak için indirecekleri yazılım Java 8 sürümüdür. Android/IOS için mobil uygulamalar geliştirebilir, web sitelerinde kullanabilir, masaüstü uygulamalar oluşturabilir, oyunlar geliştirebilirsiniz. Kullanım alanı oldukça geniştir. IDE' ler, yazılım geliştirirken kolayca kod yazmamızı sağlayan ve geliştiriciye ilgili çerçeve veya ilgili kodlama tekniği ile destek veren programlardır. IDE kullanmadan Java programlama dilinde yazılım geliştirmek tercih edilen bir yöntem değildir. Java'da en çok tercih edilen başlıca IDE'ler: Intellij IDEA, Net Beans, Eclipse (Arnold vd., 2006).

1.2 Intellij IDEA

Intellij IDEA,geliştirici üretkenliğini en üst düzeye çıkarmak için tasarlanmış,Java ve Kotlin için Entegre Geliştirme Ortamıdır(IDE). Akıllı kod tamamlama, statik kod analizi ve yeniden düzenleme işlemleri sağlayarak rutin ve tekrarlanan görevleri sizin için yapar ve yazılım geliştirmenin parlak tarafına odaklanmanıza olanak tanıyarak bunu yalnızca üretken değil aynı zamanda keyifli bir deneyim haline getirir (Goetz vd., 2004).

1.3 Spring Boot

Spring Boot, mikro hizmetler ve web uygulamaları oluşturmak için Java tabanlı çerçevelerin kullanımını kolaylaştıran açık kaynaklı bir araçtır. Spring, Java'da uygulama oluşturmaya yönelik kolaylaştırılmış, modüler bir yaklaşım sağlayan açık kaynaklı bir projedir. Spring proje ailesi ilk olarak 2003 yılında Java gelişiminin ilk dönemlerindeki karmaşıklığa bir yanıt olarak ortaya çıktı ve Java uygulamalarının geliştirilmesine destek sağlıyor. Spring adı tek başına genellikle uygulama

çerçevesinin kendisini veya tüm proje veya modül setini ifade eder. Java Spring Boot, Spring çerçevesinin bir uzantısı olarak oluşturulan özel bir modüldür (Ünal vd., 2010).

1.4 Postman

Postman, API'lerin oluşturulmasına, test edilmesine ve değiştirilmesine yardımcı olan bir API geliştirme aracıdır. HTTP isteklerini GET, POST, PUT, gibi komutlarla kullanma, oluşturulan ortamları daha sonra kullanmak üzere kaydetme ve API'leri çeşitli programlama dilleri ile kullanılmak üzere koda dönüştürme yeteneğine sahiptir. API, yazılım uygulamalarının API çağrıları aracılığıyla birbirleriyle iletişim kurmasını sağlayan Uygulama Programlama Arayüzü anlamına gelir. Postman'de sonra Çalışma Alanı alanınızı seçmeniz gerekir. Workspace'teki Yeni seçeneğinden istediğiniz alanı oluşturabilirsiniz. Temel bir istek gönderdiğimiz için yeni bir>Koleksiyon oluşturuyoruz. Oluşturduğunuz koleksiyonla ihtiyaçlarınızı kaydedebilir ve daha sonra kullanmak üzere saklayabilirsiniz. Oluşturduğunuz koleksiyonun altındaki yeni istek alanından talebi oluşturabilirsiniz. Her koleksiyonun alt bilgisayarları çok istekli ve arzu edilebilir. Bir koleksiyon içindeki istekler çarpılarak kullanılabilir. Postman'ı bir client olarak düşünürsek, Postman üzerinden yapmak istediğiniz işlemleri sunucudan istek yoluyla iletilir (Cosmina vd., 2018).

Bölüm 2

Projenin Paketleri ve Bağlantıları

Bu bölümde proje de olan paketlerden ve bağlantılardan bahsedilecektir.

2.1 Model

Model paketi dışarıya sürülmekte ve kullanımını temsil eden veri modellerini veya varlıkları saklar. Bu sınıflar genellikle veritabanı tablolarına veya harici veri kaynaklarına dönüştürülür ve uygulama verilerinin özellikleri ve özellikleri ayarlanabilir.

```
@Entity

public class WeatherEntity {

    @Id
    @GeneratedValue(generator = "UUID")
    @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")
    private String id;
    private String requestedCityName;
    private String cityName;
    private String country;
    private Integer temperature;
    private LocalDateTime updatedTime;
    private LocalDateTime responseLocalTime;
```

Şekil 2.1: WeatherEntity değişkenleri

Bu nedenle model paketinin içinde WeatherEntity class'ı oluşturuldu. Burada Entity anotasyonu ile Java class'ımızın persist class olduğunu belirtir. Id anostasyonu her kalıcı sınıf persist açıklaması gereklidir. Sınıf için birincil anahtardır. GeneratedValue ise Hibernate'in birincil anahtarımız için farklı değerler üreten üreticinin özelliklerini tanımlamaya olanak sağlayan bir açıklamadır. Hangi değerden başlanacağı, örnek gibi değerlerin tanımlanmasını mümkün kılar. GenericGenerator anotasyonu JPA anotasyonu generator alanına değer atanarak kullanılır. Daha sonra private olacak şekilde verilerimizi türleri ile birlikte yazılır. Constructorlar eklenir.

2.2 Repository

Repository, jenerik yapılarla geliştirilmiş temel operasyonel veritabanı yöntemlerini içeren bir sınıftır. Genellikle ORM araçlarıyla birlikte sorgu oluşturma ve veri eşleştirme sorumluluğunu üretir.

Şekil 2.2:Veritabanı bağlantısı için repository paketi

Spring Framework tarafından sağlanan Spring Data JPA modülünü kullanarak bir JPA repository sınıfını tanımlamaktadır. Bu kod bir WeatherEntity sınıfının veritabanı işlemlerini yönetmek için kullanılan bir WeatherRepository, arabirimini belirtir. Bir sonraki satırda WeatherEntity sınıfının bir nesnesini döndürür, ancak bu nesnenin var olup olmadığını kontrol etmek için Optional kullanılır. Bu sorgu metodu belirli bir şehir adına göre veritabanındaki WeatherEntity nesnelerini sıralar ve en son güncellenen nesneyi getirir.

2.3 Service

Service sınıfı, iş mantığını ve servis görevlerini gerçekleştiren bir sınıfı ifade eder. Bu tür sınıflar, genellikle uygulama katmanları arasında köprü görevi görerek,kullanıcı arayüzü ile veri erişim katmanında arasında işlemleri yönetir ve koordine eder. Service sınıflarında Service anotasyonları kullanılır. Spring bu anotasyonu tanıdığında,ilgili sınıfı bir Spring bean'i olarak yönetir ve uygulama bağlamında kullanılabilir hale getirir (Singh vd., 2021).

Bu sınıf hava durumu verilerini almak, önbellekte saklamak ve belirli aralıklarla önbelleği temizlemek için bir servisi temsil eder. WeatherRepository, Template ve Clock bağımlılıkları sınıfın yapıcı metodundan enjekte edilir. Bu bağımlılıklar, sınıfın veritabanı işlemleri yapabilmesi ve zamanla ilgili işlemler gerçekleştirebilmesi için

kullanılır. CacheConfig anotasyonu, sınıfın önbellekle ilgili yapılandırmalarını belirler. CachNames parametresi,bu sınıfın kullanacağı önbellek adını belirtir.Bu durumda “weathers” adındaki önbellek kullanılır. Cacheable anotasyonu getWeather metodunu önbelleklenmesini sağlar. Metot,bir şehir adı alır ve öncelikle veritabanında en gücel hava durum bilgisini kontrol eder. Eğer veritabanında güncel bir veri yoksa veya güncelliğini yitirmiş ise,dış API’den veriyi alarak önbelleğe ekler ve bu veriyi döner.

CachePut anotasyonunda createCityWeather metodunun önbelleği güncellenmesini sağlar. Yeni bir şehir hava durumu bilgisi alınır ve bu bilgiyi önbelleğe ekler. CacheEvict anotasyonu,belirli bir süre aralığında tüm önbelleği temizlemek için kullanılır. ClearCache metodu PostConstruct ve Scheduled anotasyonlarını belirli aralıklarla önbelleği temizler. SaveWeatherEntity metodu dış API’den alınan hava durumu bilgilerini WeatherEntity nesnesine dönüştürür ve bu nesneyi veritabanına kaydeder. GetWeatherStackUrl metodu hava durumu API’sine yapılacak URL’sini oluşturur. GetLocalDateTimeNow metodu clock kullanılarak,anlık olarak sistem saatini döndürür.

2.4 DTO

DTO,”Data Transfer Object” kelimerinin kısaltılmasıdır ve Java programlama da bir tasarım deseni olarak kullanılır. DTO’lar, genellikle veri taşıma amaçları için kullanılan nesnelerdir. Bu nesneler, veritabanından alınan veya başka bir kaynaktan gelen verileri taşımak için kullanılır. Temel amacı,bir katmandan diğerine veri transferini kolaylaştırmak ve sadece gerekli veriyi içermek olan hafif nesneler sağlamaktır (Çiçek vd., 2009).

Current sınıfı,bir hava durumu API’sından alınan mevcut hava durumu verilerini temsil etmek üzere tasarlanmıştır. JsonProperty anotasyonları,bu sınıfın alanlarını JSON verileri eşleştirmek için kullanılır. Örneğin JsonProperty(“observation_time”) ifadesi,JSON’daki “observation_time” alanının da ki “observationTime” alanına eşleştirilmesini sağlar. Location sınıfı,bir konumu temsil eder. Yine JsonProperty anotasyonları, JSON verileri ile alanları eşleştirmek için kullanılır. Bu sınıf, belirli bir konumunun adı, ülkesi, bölgesi, enlemi,boylamı, zaman dilimi gibi bilgilerini içerir.

Request sınıfı, bir hava durumu API'sine yapılan isteği temsil etmek üzere tasarlanmıştır. WeatherDto sınıfı hava durumu verilerini taşımak ve farklı katmanlar arasında iletişimi kolaylaştırmak için oluşturuldu. WeatherResponse sınıfı,bir hava durumu yanıtının farklı bölümlerini içeren bir veri transfer nesnesidir.

2.5 Exception

Java'da exception(istisna) sınıfları, bir programın normal akışını bozan, beklenmeyen durumları temsil eden ve ele alınması gereken durumları ifade eder.Exception sınıfları, hata durumlarının daha düzenli bir şekilde ele alınabilmesini ve kodun daha sağlam hale getirilmesini sağlar. Exception'lar kodun daha geliştirilebilir olmasını sağlar. Hataların açıkça tanımlanması ve ele alınması, geliştiricilerin kodlarını daha iyi anlamalarına ve bakım yapmalarına yardımcı olur.

Error sınıfında bir hata nesnesini temsil etmek ve bu hata nesnesi üzerinde değişiklik yapılmasını önlemektedir. Bu hata nesnesi bir istisna durumu oluştuğunda veya hata durumlarını raporlamak için kullanılmaktadır. ErrorResponse sınıfı, bir servis çağrısının başarısız olduğu durumda döndürülecek hata bilgilerini içerir. GeneralExceptionAdvice sınıfı özellikle Spring Boot ile geliştirilmiş bir servis uygulamasında ortaya çıkabilecek istisna durumlarını ele almayı amaçlayan bir "global exception handleer" sınıfını tanımlar. Uygulamanın farklı hata senaryolarına karşı daha tutarlı ve anlamlı cevaplar vermesini sağlayarak,istisna durumlarının yönetimini kolaylaştırmayı amaçlar. RestServiceException genellikle bir RESTful web servisi çağrısı sırasında ortaya çıkabilecek özel hata durumlarını temsil etmek için tasarlandı. RestTemplateErrorHandler sınıfı Spring Framework tarafından sunulan ResponseErrorHandler arabirimini uygulayan bir sınıftır ve SpringRestTemplate ile yapılan HTTP isteklerinin yanıtlarını ele alır. Genellikle,bir dış servise yapılan HTTP isteği sırasında,servisten gelen yanıtın belirli bir hata durumuna işaret ettiğinde kullanılır. Hata durumda, bu sınıf, servisten gelen yanıttaki bilgileri kullanarak özel bir RestServiceException fırlatır. RestTemplateError sınıfı dış servis çağrıları sırasında ortaya çıkan hataları daha iyi işlemek ve uygulama tarafından daha anlamlı hata mesajları üretmek için yazılmıştır.

WeatherStackApiException sınıfı, Weather Stack API tarafından döndürülen hataları temsil etmek için kullanılır. Sınıf, ErrorResponse tipinde bir nesneyi içerir, bu nesne Weather Stack API'den dönen hata yanıtını içerir. Ayrıca bu sınıf equals ve hashCode metodlarını override eder,equals metodu iki WeatherStackApiException nesnesini içerdikleri ErrorResponse nesnelerinin eşit olup olmadığını kontrol eder, hashCode metodu ise nesnenin hash kodunu hesaplar ve bu da genellikle nesnenin eşitlik kontrolünde kullanılır. Bu sınıf, Weather Stack API'den gelen hata yanıtlarını daha etkili bir şekilde işlemek ve uygulama içinde bu hataları yakalamak için kullanılabilir.

2.6 Constants

Java Spring projelerinde "Constants" sınıfları,genellikle sabit değerleri,yapılandırma ayarlarını veya genel kullanılan değerleri içeren bir sınıftır. Sabitlerin ve yapılandırma değerlerinin farklı yerlerde teker teker tanımlanmasını engeller. Bu,tutarlılık sağlar ve bir değer değiştirilmek istendiğinde tüm uygulama üzerinde etki sağlar (Ayhan vd., 2006).

Bu sınıfta, Value anotasyonuyla işaretlenmiş metodlar aracılığıyla Spring uygulamasının application.properties dosyasındaki belirli özellik değerlerini alır ve bu değerleri ilgili sabitlere atar. Component anotasyonu, bu sınıfın bir Spring bileşeni olduğunu belirtir, böylece uygulama içinde kullanılabilir. Bu tür bir yapı, uygulamanın farklı yerlerinde kullanılan sabit değerleri tek bir yerden yönetmeyi sağlar.

2.7 Config

Java konfigürasyon sınıfları,özellikle Spring Framework projelerinde kullanılan bir desen ve tekniktir.Bu sınıflar,XML tabanlı konfigürasyon yerine, Java dilinde yazılmış sınıfları kullanarak uygulama yapılandırmasını gerçekleştirmek için kullanılır.

OpenApiConfiguration sınıfı, Swagger/OpenAPI belgelerini düzenlemek ve belirli bir API'nin dökümantasyonunu özelleştirmek için kullanılır. Bu belgeler genellikle API'nin nasıl kullanılacağına dair bilgileri sağlamak için kullanılır ve geliştiricilere API'yi daha iyi anlamalarına yardımcı olur.

RestTemplateConfig sınıfı Spring Framework kullanarak HTTP istekleri göndermek için kullanılan RestTemplate'in konfigürasyonunu yapar. RestTemplate'i bağlamak, yapılandırmak ve özel hata durumlarına müdahale etmek için kullanılır.RESTful servislerle etkileşimde bulunurken kullanılan RestTemplate örneklerini özelleştirmek için kullanılır. SpringCacheCustomizer sınıfı Spring Boot uygulamasında kullanılacak önbellek yöneticisinin özelleştirmesini sağlar. Bu sınıf uygulamanın önbellekleme stratejisini belirlemek ve ConcurrentMapCacheManager'ı bu stratejiyle yapılandırmaktadır. Özellikle uygulama da kullanılacak önbellek ismini WEATHER_CACHE_NAME ve null değerlerin önbelleğe eklenip eklenmeyeceğini ayarlamak için kullanılır. SpringCachingConfig sınıfı, uygulamamız da önbellekleme kullanımını etkinleştirmek ve bir ConcurrentMapCahceManager örneği oluşturarak belirli bir önbellek adı ile yapılandırmak için kullanılmıştır.

2.8 Controller

Java da controller sınıfı, uygulamamızdaki HTTP isteklerini işleyen ve ilgili iş mantığını yürüten bir Java sınıfıdır. Controller sınıfları, kullanıcıların uygulamamızla etkileşimde bulunduğu ve, HTTP isteklerini karşıladığı ve HTTP yanıtlarını oluşturduğu yerlerdir.

WeatherController sınıfında RequestMapping("/v1/api/open-weather) anotasyonu,bu controller sınıfının temel URL path'ini belirler. Bu controller'a gelen gelen tüm HTTP istekleri /v1/api/open-weather altında ele alınır. Validated anotasyonu, sınıfın methodlarında bean validation kullanılacağını belirtir. CityNameConstraint ve NotBlank anotasyonları ile city parametresinin belirli kısıtlamalara tabi olduğunu belirtir. Örneğin boş olmamalı ve özel bir şehir kuralına uymalıdır. RateLimiter(name="basic") anotasyonu resilience4j kütüphanesinden gelir ve bu API'nin belirli bir süre içinde alabileceği istek sayısını sınırlar. Burada basic adlı bir rate limiter kullanılmıştır ve dakika da en fazla 10 isteği kabul eder. GetMapping("/{city}") anotasyonu HTTP GET isteklerini "v1/api/open-weather/{city}" yönlendirme olur. City path değişkeni, istemcinin sorguladığı şehir adını temsil eder. Metodun dönüş tipi ResponseEntity<WeatherDto> olarak belirlenmiştir. Bu hem HTTP yanıtının içeriğini hem de yanıtın HTTP durum kodunu içerir.

2.8.1 Validation

Java validation sınıfları, gelen verilerin belirli bir kriterlere uygun olup olmadığını kontrol etmek için kullanılan sınıflardır. Bu sınıflar genellikle DTO veya form nesneleri olarak adlandırılan sınıflarla ilişkilidir. Validation sınıfları, kullanıcı girişi veya dış kaynaklardan alınan verilerin doğruluğunu sağlamak için kullanılır.

CityNameConstaint sınıfı, özel bir java anotasyonu olan CityNameConstraint'i tanımlar. Bu anotasyon,bir şehir adının geçerliliğini kontrol etmek için kullanılır. İlgili kuralları uygulamak ve şehir adının geçerli olup olmadığını kontrol etmek için CityParameterValidator adlı bir sınıf tarafından uygulanır. CityParameterValidator sınıfı CityNameConstraint adlı bir Jakarta.validation.ConstraintValidator'u uygular. Bu sınıf,bir şehir adının geçerliliğini kontrol etmek için kullanılır.

2.9 WeatherStack

2015'in başlarında, hava durumu verilerini gerçek zamanlı olarak sunan ilk self-servis tabanlı hava durumu bilgileri web hizmetlerinden birini sağlamak için weatherstack farklı bir ad altında piyasaya sürülmüştür. Yıllar geçtikçe API, dünya çapındaki geliştiriciler ve şirketler arasında giderek daha popüler hale geldi ve müşterilerin geçmiş hava durumu verilerini ve hava tahminlerini basit ve hafif bir şekilde almasını sağlayacak şekilde genişletildi. Bugün hava durumu API'si dünya çapında 75.000'den fazla kişi, serbest çalışan, küçük ölçekli şirketler ve büyük şirketler tarafından dünyadaki hemen hemen her konum için en güncel hava durumu bilgilerini almanın güvenilir ve tutarlı bir yolu olarak kullanılıyor. API'yi kullanmanın ilk adımı, kayıttan sonra hesap kontrol panelinizde bulabileceğiniz weatherstack hesabınızın benzersiz API erişim anahtarıyla kimlik doğrulaması yapmaktır. API ile kimlik doğrulaması yapmak için aşağıdaki temel URL'yi kullanmanız ve API erişim anahtarınızı API'nin erişim_key parametresine aktarmanız yeterlidir.

Bölüm 3

Bu bölümde projenin çalışması anlatılmaktadır.

Projenin Çalışması

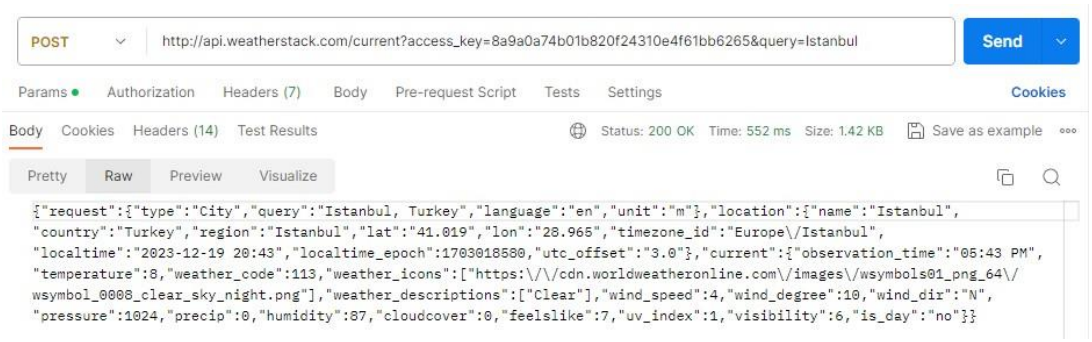
3.1 Uygulamanın Arka Planda Çalışma Şekli

Uygulama Başlangıcı itibariyle Spring Boot uygulamamız başladığında, belirttiğimiz port (varsayılan olarak 8080) üzerinde HTTP sunucu başlar. Projemizin ana sınıfındaki main metodu çalıştırılır. WeatherController sınıfındaki getWeather metoduna yapılan bir HTTP GET isteği alındığında, bu metot çalıştırılır. Yapılan isteğe verilen cevap (ResponseEntity<WeatherDto>) HTTP yanıtına dönüştürülerek gönderilir. WeatherController sınıfı, WeatherService'i kullanarak hava durumu bilgilerini çeker. WeatherService'in getWeather metodu çağrılarak ilgili şehir için hava durumu bilgileri alınır. WeatherService sınıfı, önbellek kullanarak daha önce alınmış hava durumu bilgilerini kontrol eder. Eğer varsa, önbellekteki bilgileri kullanır ve API çağrısına gerek duymaz. Eğer önbellekte bilgi yoksa veya bilgiler güncel değilse, WeatherService hava durumu API'sine bir istek yapar ve yeni bilgileri alır. Alınan bilgiler, WeatherEntity sınıfına dönüştürülerek veritabanına kaydedilir. Son olarak, alınan bilgiler WeatherDto'ya dönüştürülerek WeatherController'a geri döner. WeatherController sınıfı, alınan hava durumu bilgilerini bir HTTP yanıtına dönüştürerek istemciye gönderir.

3.2 Uygulamanın Çalışması

Postman'da belirtilen URL, WeatherController sınıfındaki getWeather metodu tarafından karşılanır. Bu metodun üzerinde GetMapping("/{city}") bulunuyor, ancak bir POST isteği gönderilir. PathVariable("city") anotasyonu ile isteğin URL'sinden "city" parametresi alınır. Ancak WeatherController sınıfında bu parametrenin validasyonu için kodlama da gösterildiği üzere CityNameConstraint anotasyonu kullanılmıştır.

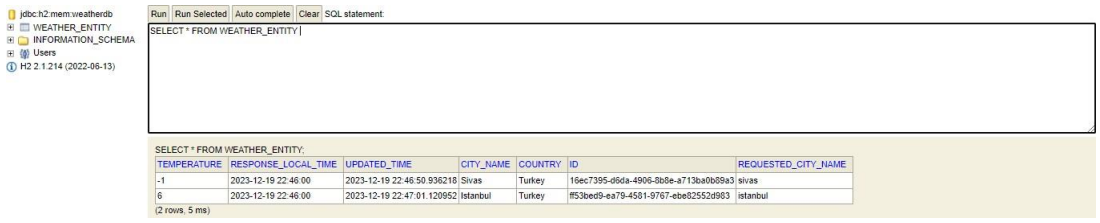
Bu, CityParameterValidator sınıfındaki kurallara dayanarak gelen şehir isminin geçerliliğini kontrol eder. WeatherController sınıfında, WeatherService'i kullanarak hava durumu bilgilerini çeker. WeatherService.getWeather(city) metodunu çağırarak ilgili şehir için hava durumu bilgilerini alır. Alınan hava durumu bilgileri, ResponseEntity<WeatherDto> şeklinde HTTP yanıtına dönüştürülerek Postman'a geri gönderilir.



Şekil 3.1: Postman uygulamasında şehir bilgileri

3.2.1 Veritabanına Ekleme

Postman'da bir GET isteği yaparak localhost:8080/v1/api/weather/(yazmak istenilen şehir) adresine gidilmek istendiğinde programda sırasıyla belirtilen yollar izlenir. WeatherController sınıfındaki getWeather metoduna yönlendirme yapılır. Bu metodun parametresi olan PathVariable("city") ile “yazmak istenilen şehir” şehri alınır. CityNameConstraint anotasyonu, " yazmak istenilen şehir " şehrinin geçerli olup olmadığını kontrol eder. Bu doğrulama, CityParameterValidator sınıfındaki kurallara dayanarak gerçekleşir.



Şekil 3.2: Veritabanına Eklenen Şehirler

Doğrulama başarılı ise, WeatherService sınıfındaki getWeather metoduna yönlendirme yapılır. Bu metod, "yazmak istenilen şehir" şehri için hava durumu bilgilerini getirir. WeatherService içinde, WeatherRepository paketinin aracılığıyla veritabanından bu şehre ait hava durumu bilgileri çekilir. Eğer "yazmak istenilen şehir" için kayıt yoksa, belki de ilk kez istendiği içindir ve Weather API'ye gidilerek bilgiler çekilir ve veritabanına kaydedilir. Alınan hava durumu bilgileri, ResponseEntity<WeatherDto> şeklinde HTTP yanıtına dönüştürülerek Postman'a geri gönderilir.

Spring Boot uygulamanın ayarlarına bağlı olarak, H2-Console genellikle localhost:8080/h2-console adresinde bulunur. Ancak, farklı bir adres istenildiği taktirde bu adresi değiştirebiliriz. Postman'da localhost:8080/v1/api/weather/(yazmak istenilen şehir) adresine yapıldığında GET isteği veritabanına bir kayıt ekliyorsa, H2-Console'u açarak bu veritabanındaki kayıtları gözlemlenebilir. Tarayıcıdan H2-Console adresine gidilir (localhost:8080/h2-console veya uygulamanızın ayarlarına bağlı olarak değişebilir) ve gerekli bağlantı bilgilerini girilir. H2-Console içinde SQL sorguları yazabilir veya tabloları gözlemlenir. weather_entity veya benzer bir tablo adı kullanılarak kayıtları görüntülenir.

3.3 Sonuçlar

Bu çalışma, geliştirilen Java Spring Boot tabanlı hava durumu uygulaması üzerinde gerçekleştirilmiş ve kullanıcılara anlık hava durumu bilgilerini sağlama amacını taşımaktadır. Uygulama, kullanıcının istediği konum için hava durumu bilgilerini başarıyla çekmektedir. API aracılığıyla alınan veriler, kullanıcıya anlık sıcaklık, nem oranı, rüzgar hızı ve hava durumu gibi önemli parametreleri göstermektedir. Geliştirilen uygulama, performans odaklı bir şekilde tasarlanmıştır. Verilerin hızlı bir şekilde çekilmesi ve kullanıcının beklentilerine hızlı bir yanıt verilmesi amaçlanmıştır.

Kaynaklar

Arnold, K., Gosling, J., and Holmes, D. (2006): The Java Programming Language, Addison-Wesley

Goetz, B. (2004). Java theory and practice: The exceptions debate. from http://www.ibm.com/developerworks/java/library/j-jtp05254.html

Ünal, C., & Bay, Ö. (2010). Java Programlama Dili'nin Bilgisayar Destekli Öğretimi. Bilişim Teknolojileri Dergisi, 2(1).

Cosmina, I. (2018). Java for Absolute Beginners: Learn to Program the Fundamentals the Java 9+ Way. Apress.

Singh, V., & Misra, U. (2021). Training in Software Development With Java

Çiçek, A. N, 2009 Restful web servisleri ile e-sağlık sistemleri gerçekleştirimi. TOBB Ekonomi ve Teknoloji Üniversitesi Fen Bilimleri Enstitüsü

Ayhan, K. Java Programlama Dilinin Web Üzerinden Sunumu, Yüksek Lisans Tezi, Sakarya Üniversitesi, Eylül 2006