# 12.02.2024

# İş Tanımı

Bu iş kapsamında, basit bir cihaz yönetim merkezi uygulamasında bulunan domain varlıklarının (entity), bir REST API üzerinden CRUDX işlemlerine imkan sağlayan Spring Boot tabanlı bir Web Servis uygulaması geliştirilecektir.

Uygulamadan beklenen bazı temel özellikler şunlardır:

* Komut satırından çalışacak bir konsol uygulaması olacak
* Proje yapısı Apache Maven formatında olacak
* Apache Maven ile build edilecek
* Spring Boot framework ile geliştirilecek
* Grafik arayüzü (Web veya desktop) olmayacak
* Yönettiği domain varlıklarını PostgreSQL veritabanında saklayacak
* Veritabanı ilklendirme işlerini (şema oluşturma, güncelleme, veri yükleme vs.) otomatik yapacak
* REST API si üzerinden, yönetimindeki domain varlıkları üzerinde ekleme, silme, güncelleme ve listeleme işlemleri yapılabilmesine imkan sağlayacak
* Veri formatı olarak (HTTP request-reponse body) JSON kullanılacak
* API URL’leri RESTFUL olacak
* REST API üzerinden yapılan bazı işlemler için veritabanında işlem log kaydı oluşturacak
* Tüm uygulama yazılım bileşenlerinin logları bir log dosyasına yazılacak
* Kapsamı **>%50** olan birim ve entegrasyon testleri olacak

# Hedefler

Bu çalışma kapsamında hedeflenen kazanımlar şunlardır:

* Apache Maven proje yapısı ve build konfigürasyonunu tanımak
* Spring ve Spring Boot Framework tecrübesi edinmek
* REST Mimarisi vi RESTFUL Web Servisleri geliştirme deneyimi kazanmak
* Güvenlik mekanizmaları (kimlik doğrulama, TLS vs.) hakkında bilgi edinmek
* Birim ve entegrasyon testleri geliştirme pratiği yapmak

# Teknolojiler

Proje kapsamında aşağıdaki teknolojiler kullanılacaktır:

* Java JDK 17
* HTTP *(*[*https://www.tutorialspoint.com/http/index.htm*](https://www.tutorialspoint.com/http/index.htm)*)*
* TLS *(<https://www.cloudflare.com/learning/ssl/how-does-ssl-work/>)*

*(*[*https://www.geeksforgeeks.org/transport-layer-security-tls/*](https://www.geeksforgeeks.org/transport-layer-security-tls/)*)*

* REST (Representational State Transfer) *(*[*https://en.wikipedia.org/wiki/REST*](https://en.wikipedia.org/wiki/REST)*)*

*(/doc/RESTful\_Web\_Services.pdf)*

*(/doc/REST-in-Practice.pdf)*

* JSON (JavaScript Object Notation) (https://www.json.org)
* Spring Boot (v3.1.3) *(https://spring.io/projects/spring-boot)*
* Spring Web
* Spring Data
* Hibernate Validator *(*[*https://hibernate.org/validator/*](https://hibernate.org/validator/)*)*
* Jackson (JSON) *(*[*https://github.com/FasterXML/jackson*](https://github.com/FasterXML/jackson)*)*
* SL4J (<https://www.slf4j.org>)
* Logback (<https://logback.qos.ch>)
* PostgreSQL (v13)
* H2 (Embedded) Database  *(*[*https://h2database.com*](https://h2database.com)*)*
* Lombok *(*[*https://projectlombok.org*](https://projectlombok.org)*)*
* Apache Maven (v3.6.3)
* Junit 5-Mockito
* Docker *(*[*https://www.docker.com*](https://www.docker.com)*)*

# Kaynaklar

Projede kullanılacak teknolojilerin öğrenilmesi için tavsiye edilen kaynaklar şunlardır:

|  |  |
| --- | --- |
| **Teknoloji** | **Kaynak** |
| REST | /doc/RESTful\_Web\_Services.pdf  /doc/REST-in-Practice.pdf  <https://restfulapi.net>  <https://www.restapitutorial.com/> |
| JSON | <https://www.javatpoint.com/json-tutorial> |
| Spring Boot | /doc/spring-boot-reference.pdf  <https://www.youtube.com/watch?v=9SGDpanrc8U> |
| Spring Boot Maven Plugin | /doc/ spring-boot-maven-plugin-reference.pdf |
| Spring Web | <https://www.baeldung.com/rest-with-spring-series>  <https://www.javatpoint.com/restful-web-services-spring-boot> |
| Spring Data | /doc/spring-data-jpa-reference.pdf |
| Jackson (JSON) | <https://www.tutorialspoint.com/jackson/index.htm> |
| Lombok | <https://www.baeldung.com/intro-to-project-lombok> |
| Apache Maven | <https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html> |
| Junit 5-Mockito | <https://www.vogella.com/tutorials/JUnit/article.html>  <https://www.vogella.com/tutorials/Mockito/article.html>  <https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html>  <https://javadoc.io/doc/org.mockito/mockito-core/3.3.3/org/mockito/BDDMockito.html> |

# Yardımcı Araçlar

REST API denemeleri için aşağıdaki araçlar kullanılabilir:

* Postman *(https://www.postman.com)*
* Curl *(https://curl.se/)*

# Geliştirme

Tüm geliştirme **“rest-practice”** adlı Eclipse projesi içerisinde yapılacaktır. Projenin genel özellikleri şunlardır:

* Java JDK 17
* Eclipse projesi
* Spring Boot framework
* Apache Maven proje yapısı
* GIT repo

Proje dizinleri şunlardır:

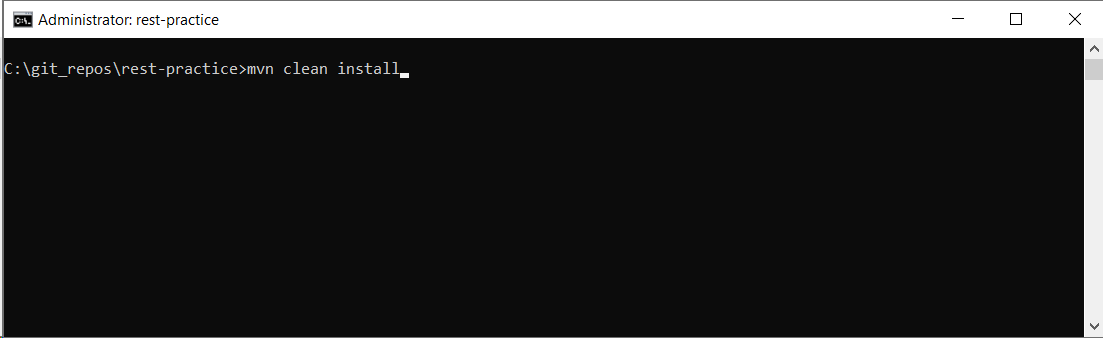
|  |  |
| --- | --- |
| **Dizin** | **Açıklama** |
| src/main/java | Kaynak kodlar |
| src/main/resources | Kod dışı kaynak dosyaları |
| src/test/java | Test kaynak kodları |
| src/test/resources | Kod dışı test kaynak dosyaları |
| log | Log dosyaları |
| doc | Dokümanlar |
| target | Build çıktıları |
| .git | GIT dizini |

Önemli proje dosyaları şunlardır:

|  |  |
| --- | --- |
| **Dizin** | **Açıklama** |
| pom.xml | Maven proje konfigürasyonu |
| .project | Eclipse proje konfigürasyonu |
| .classpath | Eclipse build path konfigürasyonu |
| lombok.config | Lombok konfigürasyonu |
| .gitignore | GIT ignore dosyası |
|  |  |
| src/main/resources/application.properties | Uygulama çalışma konfigürasyon parametreleri |
| src/test/resources/application.properties | Test çalışma konfigürasyon parametreleri |
| src/main/resources/logback.xml | Log konfigürasyonu |

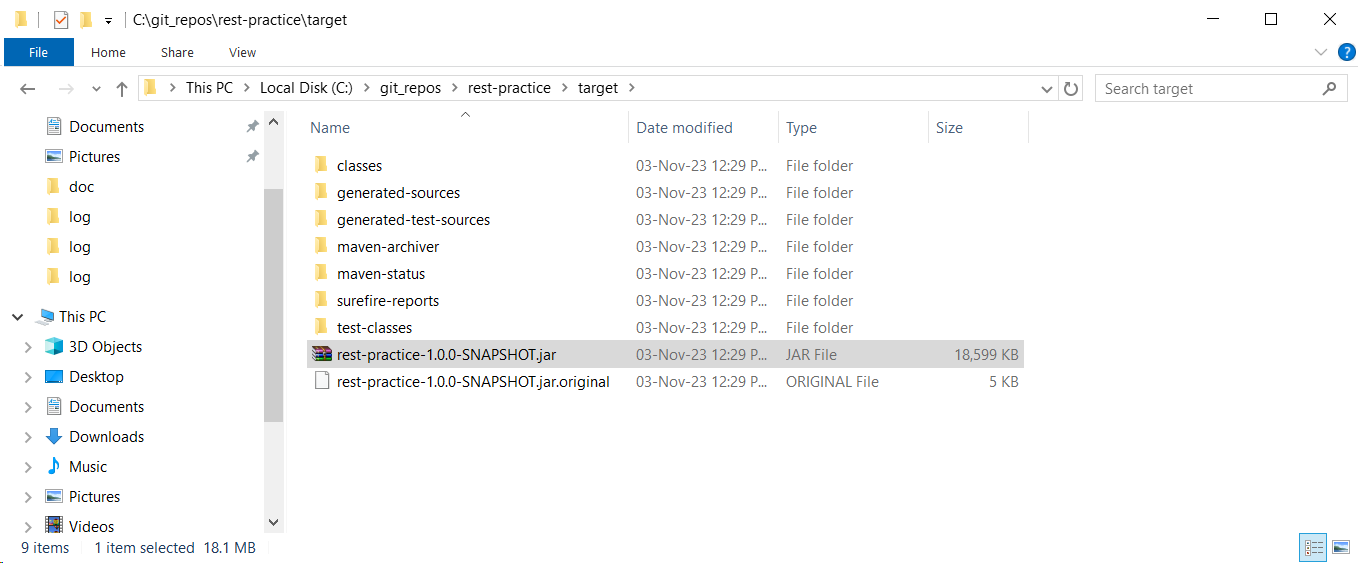
**“rest-practice”** projesi, proje dizininden aşağıdaki Maven komutunun yürütülmesi ile build edilir:

**mvn clean install**



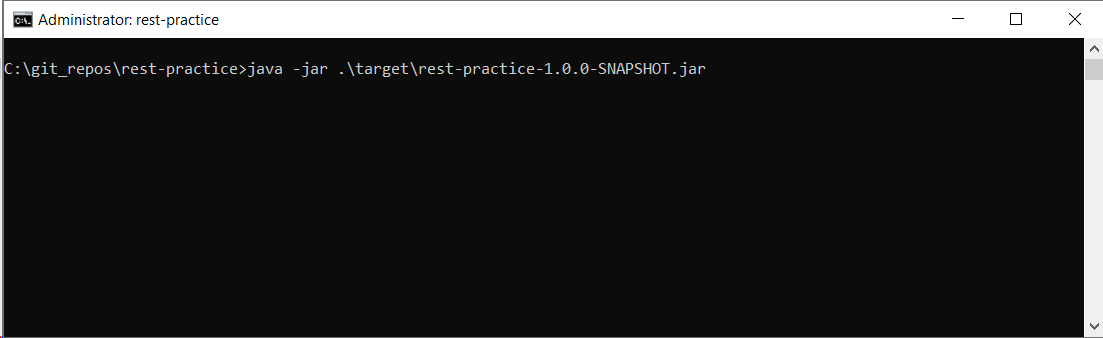
Build işlemi sonucunda aşağıdaki (executable) JAR dosyası üretilir:

**\target\rest-practice-1.0.0-SNAPSHOT.jar**



Uygulama komut satırından aşağıdaki komut ile çalıştırılır:

**java -jar .\target\rest-practice-1.0.0-SNAPSHOT.jar**



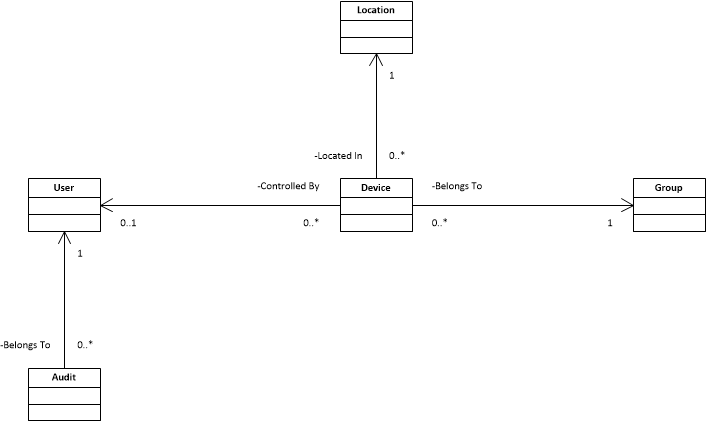
*NOT:* ***“rest-practice-1.0.0-SNAPSHOT.jar”,*** *Spring Boot Maven plugin tarafından üretilen bir executable JAR dosyasıdır.*

# Domain Varlıkları (Entity)

REST API kapsamında CRUD işlemleri yapılacak domain varlıkları (entity) şunlardır:

|  |  |  |
| --- | --- | --- |
| **Domain Varlığı** | **Java Sınıfı** | **Açıklama** |
| User | tr.gov.bilgem.restrpactice.model.**User.java** | API kullanıcısı |
| Device | tr.gov.bilgem.restrpactice.model.**Device.java** | Yönetilen cihaz |
| Group | tr.gov.bilgem.restrpactice.model.**Group.java** | Cihaz grubu |
| Location | tr.gov.bilgem.restrpactice.model.**Location.java** | Cihaz konumu |
| Audit | tr.gov.bilgem.restrpactice.model.**Audit.java** | İşlem kaydı |

Domain varlıkları arasındaki ilişkiler aşağıdaki gibidir:



Domain varlıklarının tüm alanları (attribute) ve alanlar üzerindeki kısıtlamalar (constraint) kaynak kod dosyalarında (comment olarak) belirtilmiştir. (Bknz. **tr.gov.bilgem.restrpactice.model.\***) Domain varlık sınıflarının gerçeklemesi sırasında bu bilgiler dikkate alınacaktır. (Bknz. **Görev #11, Görev #13**)

# Veritabanı

Domain varlıkları ilişkisel veritabanında tutulacaktır. Veritabanında aşağıdaki tablolar yeralacaktır: (Bknz. **Görev #11**)

* users
* locations
* groups
* devices
* audits

Uygulamanın geliştirme **(dev)** ve production **(prod)** profilllerinde **PostgreSQL** veritabanı kullanılacaktır.

Birim ve entegrasyon testlerinde (**“test”** profili) **H2 (embedded)** veritabanı kullanılacaktır.

# Ön Hazırlık

Görevlere başlamadan önce kullanılacak bilgisayarda (Internet bilgisayarı) aşağıdaki kurulumların yapılması gerekmektedir:

* JDK17
* Apache Maven (v3.6.3)
* GIT
* PostreSQL (v13)
* Lombok+Eclipse entegrasyonu
* Postman

*NOT: PostgreSQL, kuruluma gerek kalmadan,* ***docker*** *aracı ile de çalıştırılabilir.* ***(***[***www.docker.com***](http://www.docker.com)***)***

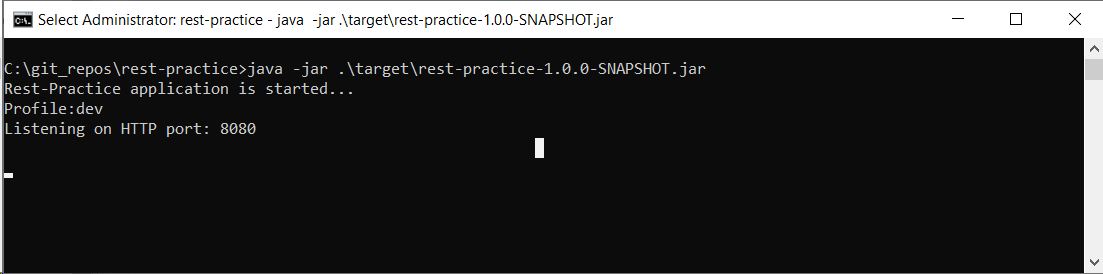
Geliştirme için kullanılacak **“rest-practice”** projesi Eclipse IDE’ye **import** edilip açılır.

# Görevler

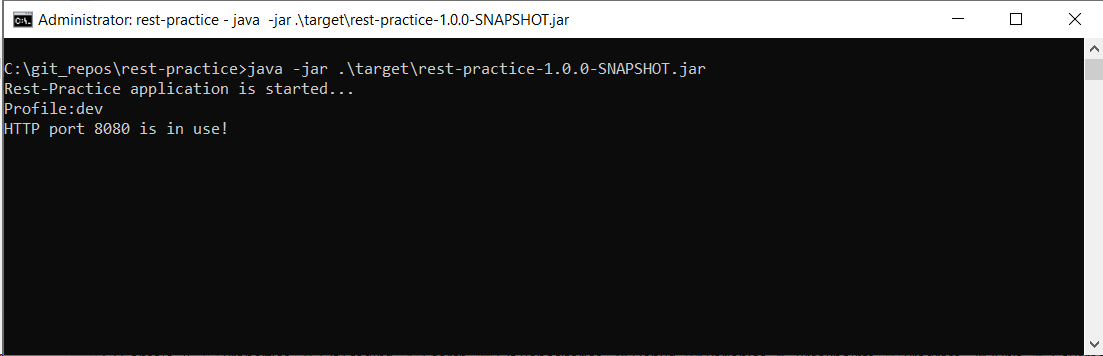
Görevler sırasıyla yapılacak ve her bir görev sonucunda, proje dosyalarında (kaynak kodları, diğer dosyalar vs.) yapılan değişiklikler **GIT commit** edilekcektir. (Commit mesajı **“görev-<no>”**) şeklinde olacaktır.

## Görev #1 Başlama Mesajı

Uygulama çalıştığında aşağıdaki mesaj görüntülenecek:



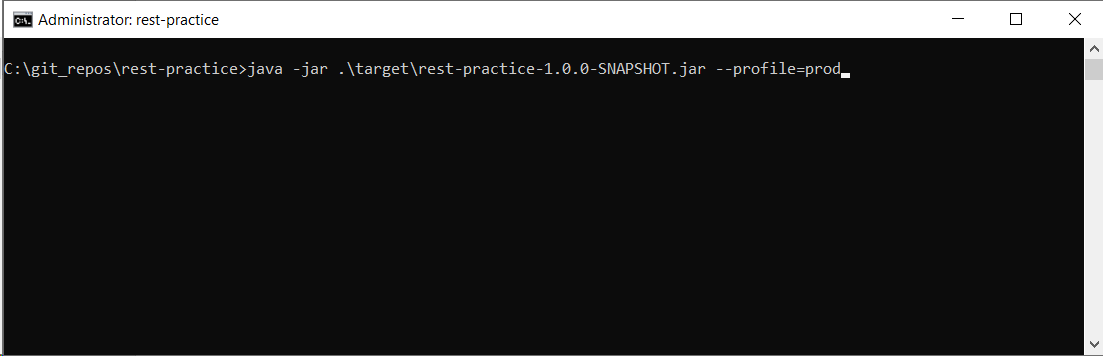
HTTP portunun kullanımda olduğu durumda ise aşağıdaki hata mesajı görüntülenecek:

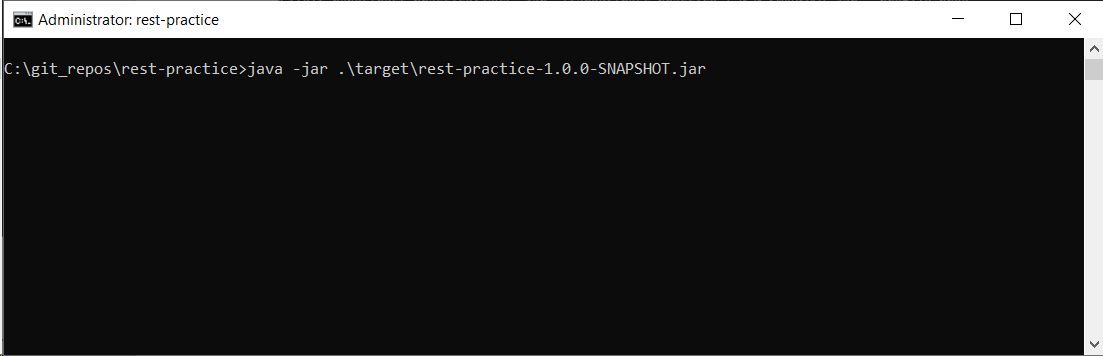


## Görev #2 Çalışma Profilleri

Uygulamanın **“dev”, “prod”** ve **“test”** olmak üzere 3 adet çalışma profili olacak:

* Profil bilgisi komut satırında **–profile** argümanı ile verilecek
* Profil bilgisi konsolda görüntülenecek (Bknz. Görev #1)
* Default profil **“dev”** olacak (**–profile** argümanı verilmediğinde)
* Tüm test kodları **“test”** profilinde çalışacak





## Görev #3 Log Konfig

**“\src\main\resources\logback.xml”** dosyasında aşağıdaki gerekleri sağlayan log konfigürasyonu eklenecek:

* Tüm uygulama logları **“\log\application.log”** dosyasına eklenecek
* **“\log\application.log”** dosyası 10 MB’i geçince arşivlenecek ve yeni log dosyası başlatılacak
* Log dosyasındaki her bir log kaydında, log mesajının yanında aşağıdaki bilgiler de yer alacak:
  + Tarih-zaman
  + Seviye (Severity)
  + Class name
  + Thread name

Uygulama açılışında, konsola yazılan mesajlar **(Görev #1)** aynı şekilde log dosyasına da yazılacak.

## Görev #4 Nihai JAR Dosya Adı Değiştirme

Maven konfigürasyonunda (pom.xml), nihai build çıktı (executable JAR) dosyası adı aşağıdaki şekilde olacak gerekli ayarlamalar yapılacak:

**rest-practice-1.0.0-SNAPSHOT-run.jar**

## Görev #5 Buıild-Info Dosyası üretme

Maven konfigürasyonunda (pom.mxl), Spring Boot Maven Plugininin, **“build-info”** adlı amacı (goal) devreye alınacak. Bu sayede, Maven build işlemi sırasında aşağıdaki dosya otomatik olarak oluşur hale gelecek:

**\target\classes\META-INF\build-info.properties**

## Görev #6 GIT Properties Dosyası üretme

Maven konfigürasyonuna (pom.mxl), **GIT Commit ID plugin** (<https://github.com/git-commit-id/git-commit-id-maven-plugin>) eklenecek. Bu sayede, Maven build işlemi sırasında aşağıdaki dosya otomatik olarak oluşur hale gelecek:

**\target\classes\git.properties**

## Görev #7 Maven Enforcer Konfig

Maven konfigürasyonuna (pom.mxl), **Maven Enforcer Plugin** (<https://maven.apache.org/enforcer/maven-enforcer-plugin>) eklenecek. Plugin konfigürasyonunda aşağıdaki kontroller (rule) devreye alınacak:

* Java versiyon JDK 17
* No duplicate dependency

Bu konfigürasyon neticesinde, Maven build işlemi, kullanılan JDK versiyonu 17 olmadığında veya classpath içerisinde aynı JAR’ın farklı versiyonlarının olması durumunda hata verecek.

## Görev #8 Datasource Konfig

Uygulamanın **“dev”** ve **“prod”** profilleri için **datasource** konfigürasyonu yapılacak:

* Veritabanı sunucusu olarak yereldeki (127.0.0.1:5432) PostgerSQL kullanılacak
* JDBC bağlantıları pool lanacak (max-pool-size-20)
* Veritabanı adı **“rest practice-db”** olacak

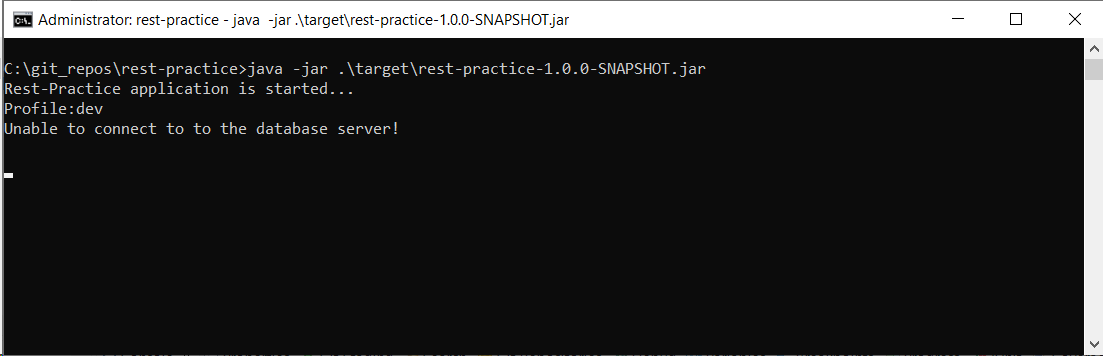
## Görev #9 Test Datasource Konfig

Uygulamanın **“test”** profili için **datasource** konfigürasyonu yapılacak:

* Veritabanı sunucusu olarak H2 (embedded database) kullanılacak
* JDBC bağlantıları pool lanacak (max-pool-size-5)
* Veritabanı adı **“rest practice-db-test”** olacak

## Görev #10 Veritabanı Bağlantı Hatası

Uygulama çalıştığında veritabanı bağlantısı yapılamazsa aşağıdaki hata mesajı görüntülenecek:



Hata detayları (Exception stacktrace) log dosyasına yazılacak

## Görev #11 Domain Varlıkları JPA Mapping

Domain varlık (entity) sınıflarına **(tr.gov.bilgem.restrpactice.model.\*)** JPA mapping anotasyonları eklenecek. (Yani domain model, ilişkisel veritabanı şemasına map edilecek.)

Bu işlem sırasında aşağıdaki isterler dikkate alınacak:

* Veritabanındaki tablo sayısı mümkün olan en az olacak
* Tablo adları ***lowercase*(class\*name)+”s”** şeklinde olacak:
  + users
  + devices
  + groups
  + locations
  + audits
* **ID (primary key)** üretme işlemi için veritabanı **SEQUENCE** mekanizması kullanılacak
* Sınıf içerisinde **NULLABLE** olmayan alanların karşılık geldiği tablo sütünlarında **NOT NULL constraint** olacak (Örneğin **Device.name, Device.serialNo**)
* Sınıf içerisinde **unique (tekil)** olarak belirlenen alanların ların ların karşılık geldiği tablo sütünlarında **UNİQUE contraint** olacak
* String tipindeki alanların üzerindeki MAX karakter sayısı kısıtlamaları veritabanı şemasına yansıtılacak
* Domain varlıkları arasındaki ilişkiler **(association)**, veritabanına yansıtılırken **FOREIGN KEY CONSTRAINT** kullanılacak
* **OPTIMISTIC LOCK (entity version)** mekanizması devreye alınacak
* Bir **User** silindiğinde, o kullanıcıya ait tüm **Audit** (log) kayıtları da (FK ilişkisi üzerinden) **otomatik** olarak silinecek (Ya JPA Provider (Hibernate) ya da doğrudan veritabanı tarafından)

## Görev #12 Veritabanı Şeması Otomatik Oluşturma

Uygulama **“dev”** ve **“test”** profillerinde” ayağa kaktığında veritabanı şemasını (tablolar, constraintler vs.) otomatik olarak **sıfırdan** oluşturacak. (**“prod”** profilinde böyle bir şey yok!!!)

## Görev #13 Domain Varlık Alanları Validasyon

Domain varlık (entity) sınıflarına **(tr.gov.bilgem.restrpactice.model.\*)** Hibernate validasyon anotasyonları **([jakarta](eclipse-javadoc:%E2%98%82=ekc-mc-common/C:%5C/Users%5C/serdar.serpen%5C/.m2%5C/repository%5C/jakarta%5C/validation%5C/jakarta.validation-api%5C/3.0.2%5C/jakarta.validation-api-3.0.2.jar=/maven.pomderived=/true=/=/maven.groupId=/jakarta.validation=/=/maven.artifactId=/jakarta.validation-api=/=/maven.version=/3.0.2=/=/maven.scope=/compile=/=/maven.optionaldependency=/true=/=/maven.pomderived=/true=/%3Cjakarta).**[**validation**](eclipse-javadoc:%E2%98%82=ekc-mc-common/C:%5C/Users%5C/serdar.serpen%5C/.m2%5C/repository%5C/jakarta%5C/validation%5C/jakarta.validation-api%5C/3.0.2%5C/jakarta.validation-api-3.0.2.jar=/maven.pomderived=/true=/=/maven.groupId=/jakarta.validation=/=/maven.artifactId=/jakarta.validation-api=/=/maven.version=/3.0.2=/=/maven.scope=/compile=/=/maven.optionaldependency=/true=/=/maven.pomderived=/true=/%3Cjakarta.validation)**.**[**constraints**](eclipse-javadoc:%E2%98%82=ekc-mc-common/C:%5C/Users%5C/serdar.serpen%5C/.m2%5C/repository%5C/jakarta%5C/validation%5C/jakarta.validation-api%5C/3.0.2%5C/jakarta.validation-api-3.0.2.jar=/maven.pomderived=/true=/=/maven.groupId=/jakarta.validation=/=/maven.artifactId=/jakarta.validation-api=/=/maven.version=/3.0.2=/=/maven.scope=/compile=/=/maven.optionaldependency=/true=/=/maven.pomderived=/true=/%3Cjakarta.validation.constraints)**.\*,** [**org.hibernate.validator.constraints**](https://docs.jboss.org/hibernate/stable/validator/api/org/hibernate/validator/constraints/package-summary.html)**.\*)** eklenecek.

Validasyon anotasyonları ile JPA mapping için kullanılan anotasyonlar birbirileriyle uyumlu olacak!!!

Örneğin, **@Size(max = 50)** olarak tanımlanan bir alan vetitabanında **VARCHAR(50)** tipinden bir sütuna tekabül edecek.

## Görev #14 Örnek Veri Oluşturma

Uygulama **“dev”** ve **“test”** profillerinde ayağa kaktığında, aşağıdaki veriler veritabanına otomatik olarak eklenecek:

* 3 adet User (Her rol için bir adet) (Admin, Operator, Observer)
* 5 adet Group (group-0,group-1, … ,group-5)
* 5 adet Location
* 10 adet Device (dev-0, dev-1 ,…,dev-4)

Domain varlık alanları (attribute) için **random** değerler üretilecek.

## Görev #15 REST Build-Info Çekme

Uygulamaya, build bilgilerini JSON formatında dönen aşağıdaki REST end-point eklenecek:

**HTTP GET /api/build-info**

Bknz. **” tr.gov.bilgem.restrpactice.controller.BuildİnfoController”**

HTTP cevabı şu şekilde olacak:

**Content-Type: application/json**

**{**

**"time": 06.11.2023 14:35**

**"branch":"master",**

**"commitId":"87494a7825295f36200a02e1206a16717a5bda3e",**

**"version":"1.0.0-SNAPSHOT"**

**}**

NOT: Build sırasında üretilen **“build-info.properties”** ve **“git.properties”** dosyaları kullanılabilir.

## Görev #16 REST İstekleri Loglama

İstemcilerden gelen tüm http REST API istekleri **“\log\access.log”** dosyasına aşağıdaki şekilde loglanacak:

**<TIMESTAMP> <CLIENT-IP> HTTP <METHOD> <URI>?<query>**

Örneğin:

**04-09-2023 14:22:55 192.168.1.100 HTTP GET /api/devices?page=1&size=10**

## Görev #17 REST Bir Varlığın Tek Değerini Çekme

Uygulamaya, bir domain varlığının tek bir değerini (instance) dönen aşağıdaki REST end-pointleri eklenecek:

**HTTP GET /api/users/{id}**

**HTTP GET /api/devices/{id}**

**HTTP GET /api/groups/{id}**

**HTTP GET /api/locations/{id}**

Örneğin:

**HTTP GET /api/devices/1** //1 id li cihazı döner

**HTTP GET /api/groups/5** //5 id li grubu döner

Her bir REST end-pointinin gerçeklemesi, ilgili varlığın, **XXXController, XXXService ve XXXRepository** sınıfları içerisinde yapılacak.

## Görev #18 REST Bir Varlığın Değerlerini Toplu Çekme

Uygulamaya, bir domain varlığının değerlerini toplu olarak **(sayfalı-sıralamalı)** dönen aşağıdaki REST end-pointleri eklenecek:

**HTTP GET /api/users?page=<page-no>&size=<page-size>&sort=<sort-attribute>,<sort-mode>**

**HTTP GET /api/devices?page=<page-no>&size=<page-size>&sort=<sort-attribute>,<sort-mode>**

**HTTP GET /api/groups?page=<page-no>&size=<page-size>&sort=<sort-attribute>,<sort-mode>**

**HTTP GET /api/locations?page=<page-no>&size=<page-size>&sort=<sort-attribute>,<sort-mode>**

**HTTP GET /api/audits?page=<page-no>&size=<page-size>&sort=<sort-attribute>,<sort-mode>**

Örneğin:

**HTTP GET /api/users?page=0&size=10**

**HTTP GET /api/devices?page=0&size=5&sort=name,asc**

**HTTP GET /api/groups?sort=name,desc**

Sayfalama ve sıralama **(paging-sorting)** işlemlerinde aşağıdaki isterler sağlanacak:

* Sayfalama için **“page”** ve **“size”** sorgu(query) parametreleri kullanılacak
* Sayfa (page) no **default 0** olacak
* Sayfa eleman sayısı (size) **default 10** olacak
* Sıralama default **“id”** sütunu üzerinden **ASC** modda yapılacak

Her bir REST end-pointinin gerçeklemesi, ilgili varlığın, **XXXController, XXXService ve XXXRepository** sınıfları içerisinde yapılacak.

Dönen HTTP response içerinde **HTTP Link** başlığında, **“first, last, previous, last”** linkleri olacak

## Görev #19 REST Varlık Arama (Text Search)

Uygulamaya, bir domain varlığı üzerinde arama (text-search) işlemi gerçekleştirip, sonuçları toplu olarak **(sayfalı-sıralamalı)** dönen aşağıdaki REST end-pointleri eklenecek:

**HTTP GET /api/users/search?page=q=<keyword>&<page-no>&size=<page-size>&sort=<sort-attribute>,<sort-mode>**

**HTTP GET /api/devices/ search? q=<keyword>&page=<page-no>&size=<page-size>&sort=<sort-attribute>,<sort-mode>**

**HTTP GET /api/groups/ search? q=<keyword>&page=<page-no>&size=<page-size>&sort=<sort-attribute>,<sort-mode>**

**HTTP GET /api/locations/ search? q=<keyword>&page=<page-no>&size=<page-size>&sort=<sort-attribute>,<sort-mode>**

**HTTP GET /api/audits/ search? q=<keyword>&page=<page-no>&size=<page-size>&sort=<sort-attribute>,<sort-mode>&start=<epoch-seconds>&end=<epoch-seconds>**

Örneğin:

**HTTP GET /api/users/search?q=admin&page=0&size=10**

**HTTP GET /api/devices/search?q=dev1&page=0&size=5&sort=name,asc**

**HTTP GET /api/groups?q=g2&sort=name,desc**

**HTTP GET /api/audits?q=abcd&start=1699275235&end=1699276235**

Arama (text-search) işlemi ile ilgili şu isterler sağlanacak:

* Aranacak kelime (keyword) için **“q”** sorgu (query) paramertesi kullanılacak
* Arama, ilgili varlığın tüm String(text) tipi alanları üzerinden **%keyword%** (containing) yöntemi ile yapılacak
* Varlığın her bir alanı üzerinde dönen arama sonuçları **LOGICAL OR** lanacak
* Arama **case-insensitive** olacak
* **Audit** varlığının aramasında **“start”** ve **“end”** sorgu (query) parametreleri ile zaman aralığı verilebilecek. Bu parametreler değer olarak EPOCH saniye alacak.

Sayfalama ve sıralama **(paging-sorting)** işlemlerinde aşağıdaki isterler sağlanacak:

* Sayfalama için **“page”** ve **“size”** sorgu(query) parametreleri kullanılacak
* Sayfa (page) no **default 0** olacak
* Sayfa eleman sayısı (size) **default 10** olacak
* Sıralama default **“id”** sütunu üzerinden **ASC** modda yapılacak

Her bir REST end-pointinin gerçeklemesi, ilgili varlığın, **XXXController, XXXService ve XXXRepository** sınıfları içerisinde yapılacak.

## Görev #20 REST Varlık Değeri Silme

Uygulamaya, bir domain varlığının tek bir değerini (instance) silen aşağıdaki REST end-pointleri eklenecek:

**HTTP DELETE /api/users/{id}**

**HTTP DELETE /api/devices/{id}**

**HTTP DELETE /api/groups/{id}**

**HTTP DELETE /api/locations/{id}**

Örneğin:

**HTTP DELETE /api/devices/1** //1 id li cihazı siler

**HTTP DELETE /api/groups/5** //5 id li grubu siler

Her bir REST end-pointinin gerçeklemesi, ilgili varlığın, **XXXController, XXXService ve XXXRepository** sınıfları içerisinde yapılacak.

Varlık silme işlemleri ile ilgili aşağıdaki isterler sağlanacaktır:

* İlgili veritabanı kaydı fiziksel olarak silinecek (DELETION MARK yöntemi değil)
* İçerisinde en az bir cihaz olan bir grup silinemez
* En az bir cihaz ile ilişkili olan bir lokasyon silinemez
* Bir kullanıcı (User) silindiğinde, o kullanıcı ile ilgili tüm **Audit** varlıkları (işlem logları) otomatik olarak silinir. *(Silme işlemi, foreign key ilişkileri üzerinden, JPA\Hibernate veya doğrudan veritabanı tarafından otomatik yapılacak)*

## Görev #21 REST Varlığın Bir Alanını (Attribute) Değiştirme

Uygulamaya, bir cihazın (yalnızca )adını değiştiren aşağıdaki REST end-point eklenecek:

**HTTP PATCH /api/devices/{id}/name**

HTTP isteği şu şekilde olacak:

**Content-Type: application/json**

**{**

**"name": "dev-1"**

**}**

Örneğin:

**HTTP PATCH /api/devices/1/name**

**Content-Type: application/json**

**{**

**"name": "dev-2"**

**}**

Yeni cihaz adına, **Device** varlığının **“name”** adlı alanı üzerindeki tüm validasyonlar **(not-blank, max-size, unique vs.)** uygulanacak.

## Görev #22 REST Varlık Güncelleme

Uygulamaya, bir cihazın değişebilir tüm alanlarını (“**id”** dışında) bir kerede değiştiren aşağıdaki REST end-point eklenecek:

**HTTP PUT /api/devices/{id}**

HTTP isteği şu şekilde olacak:

**Content-Type: application/json**

**{**

**"name": "dev-3",**

**"serialNo": "1234",**

**"ipaddres": "192.168.1.100"**

**…**

**}**

Gelen veri içerisindeki her bir alana**, Device** varlığı içerisinde tüm validasyonlar **(not-blank, max-size, unique vs.)** uygulanacak.

## Görev #23 REST Eski İşlem Loglarını Silme

Uygulamaya, belirli bir zamandan eski tüm Audit (işlem logu) kayıtlarını toplu silen aşağıdaki REST end-point eklenecek:

**HTTP DELETE /api/audits?time=<time>**

Eşik zamanı, **“time”** adlı sorgu (query) parametresi **EPOCH saniye** cinsinden değer alacak.

Örneğin:

**HTTP DELETE /api/audits?time=** **1699276235**

## Görev #24 REST Cihaza PING Atma

Uygulamaya, bir cihaza, sunucudan ping (ICMP echo) atılmasını sağlayan aşağıdaki REST end-point eklenecek:

**HTTP POST /api/device/{id}/ping**

HTTP isteği şu şekilde olacak:

**Content-Type: application/json**

**{**

**"ttl": <ttl>,**

**"timeout": <timeout>,**

**"size": <size>**

**"count": <count>**

**}**

Sunucuda, Windows **ping.exe** prosesi çalıştırılacak.

Windows **ping.exe** prosesinin çıktısı olduğu gibi HTTP cevabı olarak dönülecek:

**Content-Type: text/plain**

**Pinging 10.40.36.1 with 32 bytes of data:**

**Reply from 10.40.36.1: bytes=32 time<1ms TTL=255**

**Reply from 10.40.36.1: bytes=32 time<1ms TTL=255**

**Reply from 10.40.36.1: bytes=32 time<1ms TTL=255**

**Reply from 10.40.36.1: bytes=32 time<1ms TTL=255**

**Ping statistics for 10.40.36.1:**

**Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),**

**Approximate round trip times in milli-seconds:**

**Minimum = 0ms, Maximum = 0ms, Average = 0ms**

## Görev #25 REST Varlık Ekleme

Uygulamaya, bir domain varlığının yeni bir değerini (instance) ekleyen aşağıdaki REST end-pointleri eklenecek:

**HTTP POST /api/users**

**HTTP POST /api/devices**

**HTTP POST /api/groups**

**HTTP POST /api/locations**

Her bir REST end-pointinin gerçeklemesi, ilgili varlığın, **XXXController, XXXService ve XXXRepository** sınıfları içerisinde yapılacak.

HTTP isteği içerisinde, ilgili varlığı, girilmesi gereken tüm alanlarının değerleri taşıyan bir JSON verisi gönderilecek.

Örneğin:

**HTTP POST /api/locations**

**Content-Type: application/json**

**{**

**"city": “IST”**

**"country":"TR",**

**…**

**}**

**“id”** alanı veritabanı (SEQUENCE) tarafından otomatik olarak üretileceği için verilmeyecek.

Varlık ekleme işlemleri ile ilgili aşağıdaki isterler sağlanacaktır:

* Gelen veri içerisindeki tüm alanların validasyonu otomatik olarak yapılacaktır
* Tekil(unique) alanlar ile ilgili kontroller programatik olarak yapılacaktır

## Görev #26 REST Cihaz Toplu Silme

Uygulamaya, seçilen bir veya birden fazla cihazın toplu silinmesini sağlayan aşağıdaki REST end-point eklenecek:

**HTTP DELETE /api/devices**

HTTP isteği içerisinde, silinecek cihaz varlıklarının ID leri bir dizi (array) şeklinde taşıyan bir JSON verisi gönderilecek.

Örneğin:

**HTTP DELETE /api/devices**

**Content-Type: application/json**

**[**

**1,2,3**

**]**

Toplu silme işlemi, veritabanı tarafında tek bir **BULK DELETE** işlemi ile gerçeklenecektir.

## Görev #27 REST Cevap HTTP Status Kodları

Tüm REST API end-point lerinde, HTTP cevapları (response) ve dönüş kodları için (HTTP STATUS) için aşağıdaki isterler sağlanacaktır:

* İsteğin başarılı olduğu durumlarda, HTTP cevabı boş değil ise **HTTP 200 SUCCESS** dönülecek
* İsteğin başarılı olduğu durumlarda, HTTP cevabı boş ise **HTTP 204 NO CONTENT** dönülecek
* Varlık ekleme **(HTTP POST /api/devices)** isteklerinde, işlem başarılı ise **HTTP 201 CREATED** dönülecek. HTTP cevap alanı boş olacak. Cevap içerisinde **HTTP Location** başlığı (header), eklenen varlığın HTTP GET URL sini taşıyacak **(HTTP GET /api/devices/{id})**
* Bir domain varlığının tek bir değerini hedefleyen isteklerde, (**HTTP GET-PUT-DELETE /api/../{id}/…),** ilgili varlık veritabanında mevcut değilse **HTTP 404 NOT FOUND** dönülecek
* İstek URL’si hiçbir end-point’e karışlık gelmiyorsa **HTTP 404 NOT FOUND** dönülecek
* Gövdesinde veri taşıyan HTTP isteklerde (varlık ekleme, varlık güncelleme vs.), gelen veriden kaynaklı ilgili tüm hata durumlarında (validation, conversion, format error vs.) **HTTP 400 BAD VALUE** dönülecek
* İsteğin işlenmesi sırasında **OPTIMISTIC LOCK** hata durumu oluştuğunda **HTTP 409 CONFLICT** dönülecek
* İsteğin yürütülmesi sırasında bilinmeyen bir hata (Örneğin NullPinterException) oluşması durumunda **HTTP 500 INTERNAL SERVER ERROR** dönülecek ve exception, uygulama log dosyasına ERROR seviyesinde loglanacak (Exception stacktrace dahil)

## Görev #28 REST İstek Sayaçları

Uygulamaya,tüm REST API istekleri kapsamında, başarılı ve başarısız istekleri sayan aşağıdaki sayaçlar eklenecektir:

* Başarılı istek sayısı (HTTP 20x)
* İstemci kaynaklı başarısız istek sayısı (HTTP 40x)
* Sunucu kaynaklı başarısız istek sayısı (HTTP 50x)
* Ort. İstek İşleme süresi //milisaniye

|  |  |  |
| --- | --- | --- |
| **Sayaç Adı** | **Açıklama** | **HTTP Status** |
| req-success-count | Başarılı istek sayısı | HTTP 20x |
| req-client-fail-count | İstemci kaynaklı başarısız istek sayısı | HTTP 40x |
| req-server-fail-count | Sunucu kaynaklı başarısız istek sayısı | HTTP 50x |
| req-avg-duration | Ort. İstek İşleme süresi |  |

Uygulamaya, tüm metriklerin anlık yansımasını (snapshot) dönen aşağıdaki REST end-point eklenecek:

**HTTP GET /api/metrics**

HTTP cevabı şu şekilde olacak:

**Content-Type: application/json**

**{**

**"req-success-count": 100,**

**"req-client-fail-count": 10**

**"req-server-fail-count": 5**

**"req-avg-duration": 10**

**}**

Sayaçlar ile ilgili şu isterler sağlanacaktır:

* Sayaçlar bellekte tutulacaktır (Uygulama kapanınca yok olacak)
* **“metrics.enabled”** adlı uygulama parametresi ile devreye (dışına) alınabilecektir
* **“metrics.enabled=false”** durumumda **HTTP GET /api/metrics** isteğine **HTTP 404 NOT FOUND** dönülecektir.
* Sayaçlar, REST istek işlemine olan etkileri minimal olacak şekilde gerçeklenecektir.
* Metrik değerleri anlık snapshot dönme işlemi (**HTTP GET /api/metrics**) ile metrik değeri güncelleme işlemi SENKRONİZE edilecektir. (Yani REST API isteği geldikten, HTTP cevabı dönülene kadar geçen süre zarfında, metrik değeri arttırma istekleri bekletilecek)

## Görev #29 REST Kimlik Doğrulama

Uygulamaya, tüm REST API isteklerini kapsayacak, kimlik doğrulama (authentication) mekanizması eklenecektir.

Kimlik doğrulama kapsamında aşağıdaki isterler sağlanacaktır:

* Yöntem olarak HTTP BASIC AUTHENTICATION kullanılacaktır (<https://datatracker.ietf.org/doc/html/rfc7617>) (<https://en.wikipedia.org/wiki/Basic_access_authentication>)
* Kullanıcı hesabı olarak veritabanındaki **“users”** tablosundaki kayıtlar kullanılacaktır
* Bir REST isteği geldiğinde, kullanıcı adı ve\veya parola hatalı ise **HTTP 401 UNAUTHORIZED** durum kodu dönülecektir.
* **HTTP GET /api/build-info** istekleri **(Görev #15)** kimlik doğrulamadan muaf olacaktır

## Görev #30 Transport Layer Security (TLS) Konfigürasyonu

Uygulamaya, **Transport Layer Security (TLS)** özelliği eklenecektir. Bu kapsamda tüm REST API URL’leri **HTTPS** ile başlayacaktır:

<https://127.0.0.1:8080/api/devices>

<https://127.0.0.1:8080/api/locations/100>

TLS konfigürasyonu ile ilgili aşağıdaki isterler sağlanacaktır:

* **TLS v1.3** protokolü kullanılacaktır
* Sertifika anahtar deposu **PKCS#12** formatında olacaktır **(\*.p12)**
* İstemci kimlik doğrulaması **(Client Authentication) (mTLS)** olmayacaktır

## Görev #31 Web Sunucu Konfig

Uygulama Web sunucu konfigürasyonu, aşağıdaki isterleri sağlayacaktır:

* Azami 5 **paralel (concurrent)** istek kabul edilecek
* Yalnızca **localhost (127.0.01)** den gelen istekler kabul edilecek

## Görev #32 Audit (İşlem Logu) Oluşturma

Uygulama aşağıdaki işlemler neticesinde işlem log kaydı üretecektir.

**(Bknz. tr.gov.bilgem.restrpactice.model.Audit.java)**

|  |  |  |
| --- | --- | --- |
| **İşlem** | **HTTP İstek** | **Log Mesajı** |
| Cihaz Ekleme | HTTP POST /api/devices | “{cihaz-adı}” adlı cihaz sisteme eklendi |
| Cihaz Silme | HTTP DELETE /api/devices/{id} | “{cihaz-adı}” adlı cihaz sistemden silindi |

İşlem logu üretme (Audit) işlemi ile ilgili olarak aşağıdaki isterler sağlanacaktır:

* İşlem loğları veritabanında **“audits”** tablosuna yazılacak
* İşlem logu yalnızca işlemin başarılı olduğu durumda üretilecek
* İşlem logu oluşturma işi, asıl işlem (cihaz ekleme-silme) ile aynı veritabanı transaction içerisinde gerçekleştirilecek
* İşlem log kaydında tarih-zaman ve kullanıcı bilgisi yer alacak (Bknz. **Audit.java**)

## Görev #33 Audit (İşlem Log) Kayıtlarını Otomatik Temizleme

Uygulama 1 yıldan eski işlem log kayıtlarını otomatik olarak silecektir. Bu özelliğin gerçeklenmesi kapsamında sağlanması gereken isterler şunlardır:

* Uygulama her açılışta, bu iş için bir zamanlanmış (scheduled) görev başlatacak
* Zamanlanmış görev (default) **her Pazar 02:00 de** çalışacak (CRON schedule)
* Görev CRON ifadesi **“audits.clean.job.schedule”** adlı konfigürasyon parametresi ile ayarlanabilecek
* Görev çalıştığında, veritabanında **“audits”** tablosunda (default) **365 günden** eski kayıtları silecek
* 365 gün değeri **“audits.expire.days”** adlı konfigürasyon parametresi ile ayarlanabilecek
* Görev çalışması ve çalışma sonucunda kaç kayıt silindiği uygulama loglarında görülebilecek

## Görev #34 Cihaz Bağlantı Durumu Gözleme

Uygulamaya, sisteme ekli her bir cihazın bağlantı durumunu PING (ICMP ECHO) mesajları ile gözleyen bir cihaz bağlantı gözlem modülü eklenecek. Cihaz bağlantı durumları veritabanında tutulacak. Bu modülün gerçeklenmesi kapsamında aşağıdaki isterler sağlanacak:

* Gerçekleme **“tr.gov.bilgem.restrpactice.device.watcher”** paketinde yapılacak
* Her bir cihaz için **ayrı (1 adet)** bağlantı gözleme görevi olacak **(DeviceWatcherTask)**
* Bir cihaz eklendiğinde bağlantı gözleme görevi başlatılacak
* Bir cihaz silindiğinde o cihaza ait bağlantı gözleme görevi durdurulacak
* Bağlantı gözleme görevi **(default) 60 saniyede bir (FIXED-DELAY)** çalışacak
* 60 saniye değeri **“device.watcher.period”** adlı konfigürasyon parametresi ile ayarlanacak
* Görev çalıştığında cihaz PING (ICMP ECHO) mesajı gönderilecek ve cevap durumuna göre, veritabanında **“devices”** tablosundaki **“accessible”** alanı **(Device.accessible)** **true** veya **false** set edilecek
* Görev zamanlama işleri **Spring Task Scheduler** ile gerçeklenecek

## Görev #35 REST Entegrasyon Testleri

Uygulamaya, görevler kapsamında geliştirilen tüm REST controller metodları için entegrasyon testleri eklenecek.

Entegrasyon testleri geliştirilirken aşağıdaki isterler sağlanacak:

* Her bir REST controller, **“src/test/java”** kod dizinininde, aynı Java paketinde bir test sınıfı olacak
* Test sınıfı adı **{Controller Class name}+Test** şeklinde olacak

Örneğin:

src/main/java/tr/gov/bilgem/restrpactice/device/**DeviceController**

src/test/java/tr/gov/bilgem/restrpactice/device/**DeviceControllerTest**

* Komut satırından, **“mvn test”** komutu ile tüm testler çalıştırılabilecek
* Spring Boot entegrasyon test mekanizmaları kullanılacak
  + @SpringBootTest
  + MockMvc mvc
* Testler **Mock Web MVC** ortamında çalışacak (Web Server gerçekten ayağa kalkmayacak!)

## Görev #36 REST Test Kapsam (Coverage) Analizi

Uygulamanın test kapsamı (coverage) analizini yapıp raporlayan bir Maven plugin bulunup, proje konfigürasyonuna entegre edilecek.

İlgili konfigürasyon kapsamında sağlanması gereken isterler şunlardır:

* Plugin, Maven build işlemi sırasında otomatik olarak devreye girecek
* Analizi yerelde (lokalde) gerçekleştirecek, external bir sunucu gerektirmeyecek
* Analiz sonuçları için HTML, PDF vs. formatında rapor üretecek
* Lisans koşulları uygun olacak (Crack olmayacak)

## Görev #37 Docker İmajı Uretme

Uygulamanın, bir software container içerisinde çalıştırılabilmesi için gerekli **docker imajı** oluşturulacak.

## Görev #38 Sürüm (Release) Çıkarma

Uygulamaya **Maven Release Plugin** (<https://maven.apache.org/maven-release/maven-release-plugin/>) entegre edilecek.

Maven Release Plugin vasıtasıyla aşağıdaki sürüm işlemleri gerçekleştirilecek:

* Geliştirme sürümü **(v1.0.0-SNAPSHOT)** sürümü kapatılıp, **release** sürümü **(v1.0.0)** oluşturulacak
* Sürüm tag **(v1.0.0)** oluşturulacak
* Sürüm artiffact **(rest-practice-v1.0.0.jar)** proje içerisinde **“/release”** dizinine konulacak
* Yeni geliştirme sürümü **(v1.0.1-SNAPSHOT)** başlatılacak

## Görev #39 Access Token (OAUTH) Bazlı Kimlik Doğrulama

Görev#29’daki HTTP Basic Authentication ile gerçeklenen kimlik doğrulama mekanizması, OAUTH **Access Token (JWT)** formatına dönüştürülecek.

İsterler:

* Access token **JWT (Json Web Token)** olarak gerçeklenecek
* Access token simetrik anahtarla **(HMAC SHA-512)** imzalanacak
* Anahtar **PKCS#12** formatında bir dosyada **(.p12)** tutulacak
* Mümkünse gereklemede **Spring Security** kullanılacak
* Kullanıcı(İstemci) **login ve logout** işlemleri için aşağıdaki REST end-pointler olacak:

**HTTP POST /api/login**

Authorization: Basic <username:password>

**HTTP POST /api/logout**

Authorization: Bearer <access-token>

* Login end-point aşağıdaki formatta dönüş yapacak:

**{**

**"access\_token":asdwersdf…,**

**"token\_type": "bearer",**

**"expires\_in": 3600**

**}**

*NOT: Standart OAUTH token response*

* Logout sonrası ilgili **Access token** bir daha kullanılamayacak

*NOT:* ***Token revocation list*** *gerçeklenecek*

* Her API çağrısında, gelen request içerisinde geçerli bir **access-token** olup olmadığı kontrol edilecek *(İmza doğrulama ve zaman aşımı kontrolü)*

*NOT:* ***HTTP GET /api/build-info*** *end-point muaf olacak (Kimlik doğrulamasız erişilebilecek)*