Wstęp

Urządzenia podłączone do systemy zgłaszają się po restarcie w następującej formie:

Dla protokołu w wersji 1.6.x:
wysyłając komunikat REST na endpoint **/protocols/iot16/bootnotification/{deviceId}**

```
class BootNotificationRequest {

    String deviceVendor;

    String deviceModel;

    String deviceSerialNumber;

    String firmwareVersion;

    ...

}
```

Dla protokołu w wersji 2.0.x:
wysyłając komunikat REST na endpoint **/protocols/iot20/bootnotification/{deviceId}**

```
class BootNotificationRequest {

    Device device;

    Reason reason;

    static class Device {

        String serialNumber;

        String model;

        Modem modem;

        String vendorName;

        String firmwareVersion;

    }

    ...

}
```

W przypadku obu protokołów odpowiedzią na komunikat jest:

```
class BootNotificationResponse {

    String currentTime;

    int interval;

    Status status;

    enum Status { Accepted, Pending, Rejected }

}
```

Gdzie **interval** to ilość sekund, co którą urządzenie ma przesyłać komunikat Heartbeat.

UWAGA: Wszystkie komunikaty oraz struktura ścieżek endpointów są zestandaryzowane i nie podlegają zmianom czy rozszerzeniom.

Ćwiczenie: Modelowanie Value Objects i Policy

Zaimplementuj logikę wyliczania pola **interval** komunikatu zwrotnego **BootNotificationResponse**.

1. Implementację możesz zacząć od testów, zgodnych z opisem reguł w akapicie „Obecne reguły”. Zaimplementuj „Obecne reguły” na potrzeby testów w klasie **IntervalRulesFixture**.
2. By odizolować logikę wyliczania interwału od różnorodności protokołów komunikacyjnych, zaimplementuj Value Object (na potrzeby ćwiczenia nazwijmy go **Deviceish**), który posiada wyłącznie pola niezbędne do wyliczenia interwału.
3. Dodaj do obu klas **BootNotificationRequest** metodę, która wyprodukuje Value Object **Deviceish**. Do metody produkującej przekaż brakujące informacje jako parametry.
4. Zaimplementuj metodę **calculateInterval(Deviceish)** klasy **IntervalRules** enkapsułującą wszystkie reguły.

UWAGA w tym ćwiczeniu:
- Pomiń funkcjonalności związane z edycją / dodawaniem reguł przez administratora, skup się na wyliczaniu interwałów
- Pomiń warstwę persystgncji oraz inne technologie

Wymagania:

Heartbeat interwal może być określony przez administratora systemu dla:
- konkretnego podzbioru urządzeń wskazanych po **deviceId**,
- wszystkich urządzeń określonego modelu (**vendor** + **model**)
  gdzie model może być określony za pomocą wyrażenia regularnego
- wszystkich urządzeń komunikujących się danym protokołem
- pozostałych urządzeń (interwał domyślny).

Obecne reguły:

Interwały dla podzbioru urządzeń:

Interval	DeviceIds
600s	EVB-P4562137, ALF-9571445, CS_7155_CGC100, EVB-P9287312, ALF-2844179
2700s	t53_8264_019, EVB-P15079256, EVB-P0984003, EVB-P1515640, EVB-P1515526

Interwały dla modeli:

Interval	Vendor	Model (regex)
60s	Alfen BV	NG920-5250[6-9]
60s	ChargeStorm ABl	Chargestorm Connected
120s	EV-BOX	G3-M5320E-F2.*

Interwał dla protokołu IoT2.0 wynosi: 600s

Interwał domyślny: 1800s

Ćwiczenie: Persystencja obiektu jako dokument

Zaimplementuj i przetestuj testem integracyjnym zapis oraz pobieranie reguł wyliczania interwału z poprzedniego zadania.

1. Zaimplementuj klasę **IntervalRulesRepository**, zaimplementuj jedynie metodę:
   IntervalRules get() nie przyjmującą argumentu
2. W implementacji klasy **IntervalRulesRepository** posłuż się poniższym kodem:

```
@Data
@Entity
@Table(name = "features_configuration")
class FeaturesConfigurationEntity {
    @Id
    private String name;
    @Type(type = "jsonb")
    @Column(columnDefinition = "jsonb")
    private IntervalRules configuration;
}
```

Persystencja obiektu jako JSON w kolumnie typu binary json w bazie Postgresql

```
public interface FeaturesConfigurationRepository
        extends CrudRepository<FeaturesConfigurationEntity, String> {
    Optional<FeaturesConfigurationEntity> findByName(String name);
}
```

Interface repozytorium Spring Data

3. Posłuż się wstępnie przygotowanym testem **IntervalRulesRepositoryTest** - dokończ test.
   Test wykorzystuje bazę Postgresql uruchamianą automatycznie w kontenerze Docker-a.
   Upewnij się, że na Twoim komputerze doker jest uruchomiony.
   Za zażądanie kontenerem w trakcie testów odpowiada biblioteka **https://www.testcontainers.org**
4. Zadbaj o scenariusz w którym w bazie danych nie ma żadnej konfiguracji, w tym przypadku skonstruuj i zwróć domyślną konfigurację z domyślnym interwałem.
5. Zmień klasę **FeaturesConfigurationEntity** tak by mogła przechowywać dowolny typ w polu **configuration**, dostosuj klasę **IntervalRulesRepository** upewnij się że test nadal przechodzi.

Ćwiczenie: REST dla prostej logiki CRUD

Zaimplementuj kontroler REST-owy i przetestuj testem Spring MockMvc endpoint REST-owy pozwalający na zapis oraz odczyt aktualnych reguł wyliczania interwału z poprzednich zadania (**IntervalRules**).

1. Zaimplementuj klasę **FeaturesConfigurationController** z pozwalającą na:
   odczyt (Http GET method)
   zapis (Http PUT method)
2. Sam zaproponuj strukturę ścieżki endpointu.
3. Przetestuj przypadek z przekazaniem błędnego wyrażenia regularnego dla reguły opartej o model urządzeń, zadbaj by kod błędu zwracany w tym przypadku to BAD REQUEST (400).