Задача

Создать REST API сервис, который будет принимать данные от "сенсора".

Сенсор имеет доступ в интернет, поэтому может отправлять HTTP запросы к нашему серверу.

Каждый раз, когда он будет производить измерение, он будет слать HTTP запрос с данными в формате JSON на наш сервер - для этого в реальной жизни мы бы указали устройству IP адрес того компьютера, где мы запускаем Spring REST API приложение. После этого, на нашем компьютере мы сможем принимать запросы от сенсора.

Задача

Так как реального сенсора у нас нет, в качестве сенсора будет выступать наш собственный компьютер!
То есть на нашем компьютере будет работать сервер со Spring REST API приложением, и наш же компьютер будет слать HTTP запросы к Spring приложению так, как будто он и есть "сенсор".
Для того, чтобы отправлять запросы к REST API приложению, мы будем использовать знакомый нам RestTemplate, который мы уже использовали в этом курсе для того, чтобы слать запросы к сторонним REST API сервисам.

Задача

Более формально

<u>Проект делится на 2 части:</u>

1) Создание REST API приложения с помощью Spring REST (по аналогии с тем приложением, которое мы создавали на уроках)

2) Создание Java клиента, который бы отправлял данные на REST API приложение - с помощью класса RestTemplate.

Задача (I часть)

В качестве шаблона используйте проект FirstRestApp (https://github.com/NeilAlishev/SpringCourse/tree/master/FirstRestApp)

В этот раз я не буду описывать задание строго - какие модели, контроллеры и сервисы создавать. Техническая реализация полностью за вами, я лишь опишу здесь бизнес требования к вашему REST API.

Помните, что нет единственно верного варианта решения задачи. Не бойтесь творить. Этот проект даст вам почувствовать уровень свободы принятия решений на позиции middle-разработчика. По любым вопросам пишите в чат.

После того, как вы реализуете свой вариант приложения, вы сможете сравнить его с моим, посмотрев мой видеоразбор.

ТЗ к REST API

Описывается адрес API, данные которые передаются при запросе на этот адрес и тот функционал, который должен предоставляться в результате запроса на этот адрес

- **`POST /sensors/registration`**

```
{
    "name": "Sensor name"
}
```

Регистрирует новый сенсор в системе. Другими словами, просто добавляет новый сенсор в таблицу сенсоров в БД. Как видно из JSON'a - у сенсоров есть только одно поле - название.
Вы должны помнить о правилах хорошего кода и использовать DTO для входящего объекта - сенсора.
Также, вы должны валидировать то, что сенсора с таким названием еще нет в БД (вспоминайте Spring Validator). Если сенсор с таким названием есть в БД - возвращать клиенту сообщение с ошибкой (вспоминайте урок про это).
Также, если название сенсора пустое или содержит менее 3 или более 30 символов, клиенту должно возвращаться сообщение с ошибкой.

- **POST /measurements/add**

```
{
    "value": 24.7,
    "raining": false,
    "sensor": {
        "name": "Sensor name"
    }
}
```

Добавляет новое измерение. Это тот адрес, куда настоящий сенсор посылал бы свои данные. Вещественное поле "value" содержит значение температуры воздуха, булево поле "raining" содержит значение true/false в зависимости от того, зарегистрировал ли сенсор дождь или нет. Помимо этого, в этом запросе передается сам объект сенсора, который получил и отправляет эти "измерения". Значения температуры воздуха, дождя должны сохранятся в таблице в БД. Также, в каждой строке этой таблицы должно содержаться название того сенсора, который прислал эти измерения. То есть сущность "Измерение" имеет связь с сущностью "Сенсор" (вспоминайте отношения в БД и как их выстраивать в Java классах с помощью Hibernate).
Все поля у измерения должны валидироваться.
Значение "value" должно быть не пустым и находиться в диапазоне от -100 до 100.
Значение "raining" должно быть не пустым.
Значение "sensor" должно быть не пустым. При этом, название сенсора должно валидироваться в БД.
Сенсор с таким названием **должен** быть зарегистрирован в системе (должен быть в БД).
Если такого сенсора нет в БД - выдавать ошибку. Также, не забывайте про DTO.
На сервере, у измерения должно выставляться текущее время, оно должно сохраняться в БД.

- **`GET /measurements`**

Возвращает все измерения из БД

- **`GET /measurements/rainyDaysCount`**

Возвращает количество дождливых дней из БД

Всего в приложении должно быть 4 адреса:

1) Регистрация сенсора
2) Добавление измерения от сенсора
3) Получение всех измерений
4) Получение количества дождливых дней

После того, как REST приложение будет готово, необходимо перейти ко второй части задания и реализовать клиента.

Задача (II часть)

Используйте класс RestTemplate, чтобы отправить 1000 запросов со случайными температурами и "дождями" на адрес:
`POST /measurements/add`

Не забудьте перед этим зарегистрировать новый сенсор (тоже с помощью запроса).

После этого, используйте RestTemplate, чтобы получить эти 1000 измерений с сервера, отправив **`GET`** запрос на адрес:
`/measurements`

Задание со * (необязательное):
Постройте график температур, получив 1000 температур с сервера.
Для построения графика можно использовать библиотеку xchart