

# REST API

REST API сервисы нужны практически в любой сфере деятельности.

Большинство программистов по всему миру преимущественно занимаются тем, что создают REST API сервисы различных уровней сложности.

REST API позволяет принимать данные с любого устройства, которое имеет выход в интернет (сенсор, робот, весы, конвейер, сканер штрих кодов, мобильный телефон, ... , что угодно) и как-то их обрабатывать.

Мы можем сами посылать запросы к сторонним REST API сервисам (с помощью `RestTemplate` или `WebClient` - новый класс, который позволяет совершать асинхронные запросы) либо создавать свой REST API сервис, который может принимать запросы от клиентов.

# Комментарии к заданию

- Задание специально написано человеческим языком и не содержит технических деталей реализации. Это максимально приближено к условиям настоящей работы, когда менеджер сообщает вам бизнес требования и ожидает, что вы сами переведете их в техническую реализацию.
- Ожидается, что в случае непредвиденных ошибок, вы будете их гуглить и исправлять. Эффективно гуглить - один из главных навыков программиста.
- Также ожидается, что вы будете задавать вопросы в чате и обсуждать решение с другими людьми.

# Задача

Представьте, что вы приобрели метеорологический датчик (для простоты дальше будем называть его просто "сенсор"). Этот датчик измеряет температуру окружающего воздуха и может определять, идет дождь или нет.

Обычно, эти сенсоры выглядят вот так:



**Мы хотим получать данные с сенсора и сохранять их в БД, чтобы далее их анализировать -> Надо создать REST API для этого!**

# Задача

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



Данные



**Измеряет температуру воздуха и определяет, идет дождь или нет**

**Принимает данные от "сенсора" и сохраняет их в БД**

**Примерное время выполнения проекта: 6-9 часов**

# 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**

**Проект делится на 2 части:**

- 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.

# T3 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 сохраняться 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.