

IoT Sensor portal

Software System requirements

Contents

[1 Project description 3](#_Toc494447964)

[2 Supported sensor types 3](#_Toc494447965)

[3 Public part 3](#_Toc494447966)

[3.1 Landing page 3](#_Toc494447967)

[3.2 Register 3](#_Toc494447968)

[3.3 View public sensors 3](#_Toc494447969)

[3.4 Sensor offline 4](#_Toc494447970)

[4 Private part 4](#_Toc494447971)

[4.1 Register new sensor 4](#_Toc494447972)

[4.2 View list of own sensors 4](#_Toc494447973)

[4.3 Modify existing sensor 4](#_Toc494447974)

[4.4 Share a private sensor 4](#_Toc494447975)

[5 Administration 4](#_Toc494447976)

[5.1 Edit registered users 5](#_Toc494447977)

[5.2 Add new user 5](#_Toc494447978)

[5.3 Edit sensors of registered users 5](#_Toc494447979)

[6 Sensor Data hub module 5](#_Toc494447980)

[6.1 Sensor Data Polling 5](#_Toc494447981)

[7 General requirements 5](#_Toc494447982)

[7.1 Must have requirements 5](#_Toc494447983)

[7.2 Should have requirements 6](#_Toc494447984)

[7.3 Could have requirements 6](#_Toc494447985)

# Project description

ICB wants a simple public IoT Sensor Portal to be developed. Referenced as The Application. The portal will fetch and store sensor data form various sources. It will support public sensor data access along with private sensors. The logged in users will be able to share sensors data between them.

# Supported sensor types

ICB will provide a web API’s (with set of examples how to use it) to fetch sensors data at <http://telerikacademy.icb.bg/>

The following sensor types should be supported:

* Temperature measured in °C
* Humidity sensor measure in relative humidity (in percent )
* Electric power consumption sensor in Watts
* Occupancy sensor (true or false) depending on whether there are persons or not in the measured room. True means there are persons in the room.
* Window /door sensor. Measured with true/false depending whether the door is open or closed. True means the door is closed
* Noise sensor with values in Decibels

# Public part

## Landing page

Application MUST have public landing page. The landing page MUST contain product description and links. Along with corresponding functionality:

* Login
* Register
* View public sensors

## Register

A register page MUST have at least username field, password field and email field. Email will be necessary if you decide app’s users to receive notification about the sensor, which is out of range (see Could Have Requirements 7.2 - 2)

## View public sensors

The Application MUST have a page with a list of all public sensors of all registered users. It should be represented as a list, which supports paging and filtering. The page MUST be publicly accessible.

Each sensor of the public sensor list MUST have user’s name, sensor’s name

The user MUST have a detail sensor view which shows the current value of the sensor. Use three different colors for sensor data validity (e.g you can use green/blue color for showing that value from the sensor is in acceptable range, red/orange color – value is out of acceptable range and grey color – there is no connection with the sensor. ) There should be possibility to view historical data for the sensor choosing from and to periods.

## Sensor offline

The Application MUST handle when sensor is offline and show this to the users.

# Private part

The Application MUST have a private section which is accessible only for logged users. This section MUST support the following functionality:

## Register new sensor

The newly created sensor MUST have its own:

* Name
* Description
* URL for fetching sensor data
* Polling interval which specifies the amount of time to refresh sensor data
* Measurement type which specifies the type of measurement (temperature, humidity etc. See **2 Supported sensor types**)
* Access (public or private) which specifies whether the sensor is publicly visible or only accessible for the user
* Range of acceptable values (e.g. -40 °C to +100 °C)

## View list of own sensors

The logged user MUST have a place where he/she can view his/hers registered sensors. For each sensor the list MUST include at least:

* Name
* Description
* Current value
* Access level (public or private)
* Whether the sensor is shared and with whom

Like the public sensors, each of the own sensors MUST also have a detail sensor view which shows the current value of the sensor. There MUST be three different colors for sensor data validity and also there should be possibility to view historical data for the sensor choosing from and to periods.

## Modify existing sensor

The logged user MUST be able to edit his/hers own sensors. The data which MUST be editable is the same as the data entered when registering the sensor (URL, polling interval, measurement type, access, etc.).

## Share a private sensor

The logged user MUST have the ability to share his/hers private sensors with other users. Those other users MUST have Read-Only access to the original sensors.

# Administration

Application MUST have three main roles:

* Non-logged/non-registered users can see only public part.
* A Logged user (non-administrator) has the abilities described in Section 4 (Private part).
* An administrator has the same abilities like logged users and in addition he/she can:

## Edit registered users

Administrators can edit information for registered users.

## Add new user

Each administrator can add new user, including administrator.

## Edit sensors of registered users

Administrators can:

* Modify/insert all information about existing sensors (name, description, URL for fetching sensor data, polling interval, measurement type, access level and share functionality) of each registered user, no matter if sensor is private or public.
* Register new sensor for each already registered user.
* View list of all sensors (public/private) of each registered user.

# Sensor Data hub module

The Application MUST have a module which gathers data from the registered sensors and store their values for historical reasons. The module should contain also analytics.

## Sensor Data Polling

The module MUST poll data from sensors based on their pooling interval setting. Note that a sensor could have pooling interval of 1 minute and another could have pooling interval set to 5 minutes. The data MUST be stored for historical reasons. Stored data should be used when showing sensor historical data. In order to make that requirement, you will be provided with windows service and a document which include steps how to install the service, examples how to use it and steps how to uninstall it. Your task is to create controller with action, that implement the logic for data polling and provide it to the service. The provided windows service will invoke the implemented controller action on a configurable interval. It’s the controller action responsibility to decide whether and which sensors data to poll.

# General requirements

## Must have requirements

Your Web application should use the following technologies, frameworks and development techniques:

1. Use **ASP.NET MVC** and **Visual Studio 2015 (MUST)**
2. You should use **Razor** template engine for generating the UI **(MUST)**
   1. You may also use AngularJS (optional)
3. Use **MS SQL Server** as database back-end **(MUST)**
   1. Use **Entity Framework 6** to access your database **(MUST)**
   2. Using **service layer** is a must
4. Use at least **1 areas** in your project (e.g. for administration) **(MUST)**
5. Create responsive design **(SHOULD)**
   1. You may use **Bootstrap** or **Materialize**
6. Use the standard **ASP.NET Identity System** for managing users and roles **(MUST)**
7. Use **AJAX form and/or SignalR** communication in some parts of your application **(MUST)**
8. Use **caching** of data where it makes sense (e.g. starting page) **(MUST)**
9. Sensor data should change without page refreshes
10. Apply **error handling** and **data validation** to avoid crashes when invalid data is entered (both client-side and server-side) **(MUST)**
11. Prevent yourself from **security** holes (XSS, XSRF, Parameter Tampering, etc.) **(MUST)**
    1. Handle correctly the **special HTML characters** and tags like <script>, <br />, etc.
12. Use **GitHub** and take advantage of the **branches** for writing your features.
13. Create **unit tests** for your "business" functionality following the best practices for writing unit test. **(MUST)**

## Should have requirements

1. Use Angular 4 + Typescript
2. **Documentation** of the project and project architecture (as .md file, including screenshots)

## Could have requirements

1. Sensors lists data should be displayed in a different way depending on the measurement type of the selected sensor (e.g. gauges for temperature sensors, bar chart for power consumption and so on).
2. The logged user should be able to subscribe to either public or shared sensors, or to his own private sensors. This subscription should be expressed in a way that if a user is subscribed to a sensor, then he will receive a notification (via email) when that sensor’s value is out of the acceptable range of values for it.
3. Think of a way to compress the MS SQL data usage (example if a temperature is constant whole day)
4. Users can set sensor coordinates and the sensors should be visualized on a map in the public list page and the private list page