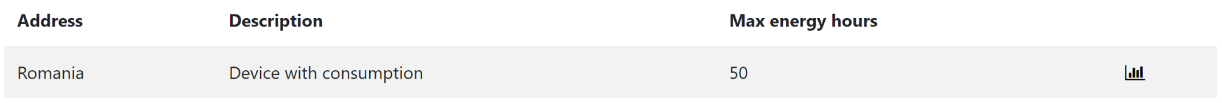
Energy utility platform implemented using .Net Core MVC and .Net Core API

The application consists of 2 main portals, one for the client that is able to see his devices and the consumption of these devices based on a selected date and a portal for the admin. The admin is able to see all the clients and modify them, delete them or create new ones. Also, the admin is able to create new devices, delete them, modify them and attribute a new owner. After the attribution, the new owner is able to see the consumption on the page. The data from a csv file is read and sent to the main app in order to be printed into the user portal. The data is sent oance 10 minutes.

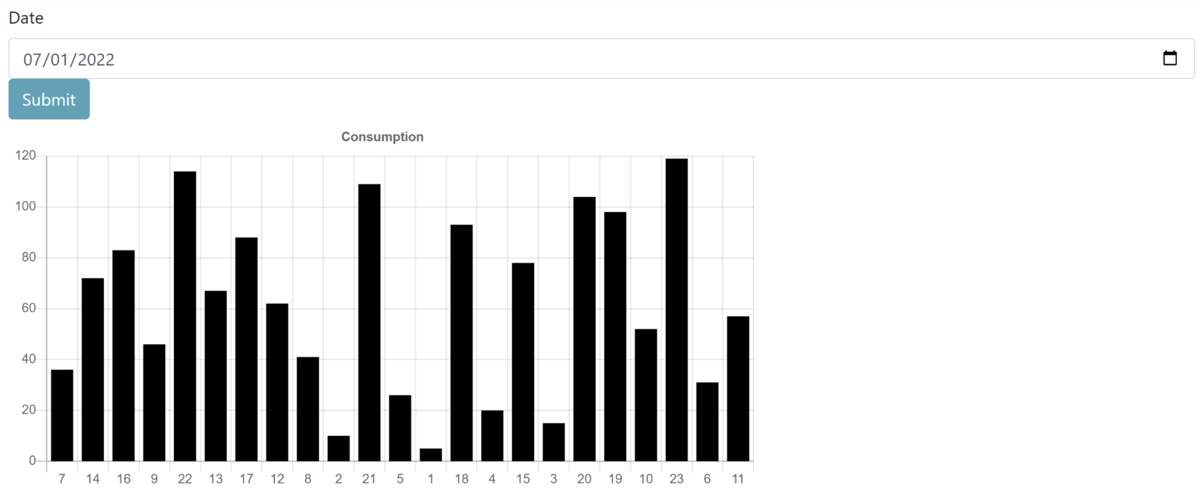
.NET Core Front End application

About the architecture:

- The application consists of 2 main pages and also one for the login. On these pages, data is presented using a combination of HTML, javascript, and CSS.
- On the user page we have the table with all its devices, and also the button to redirect him to the consumption page
- On the consumption page, we have a graphic of the consumption on a selected date, created with the help of Charts.js.



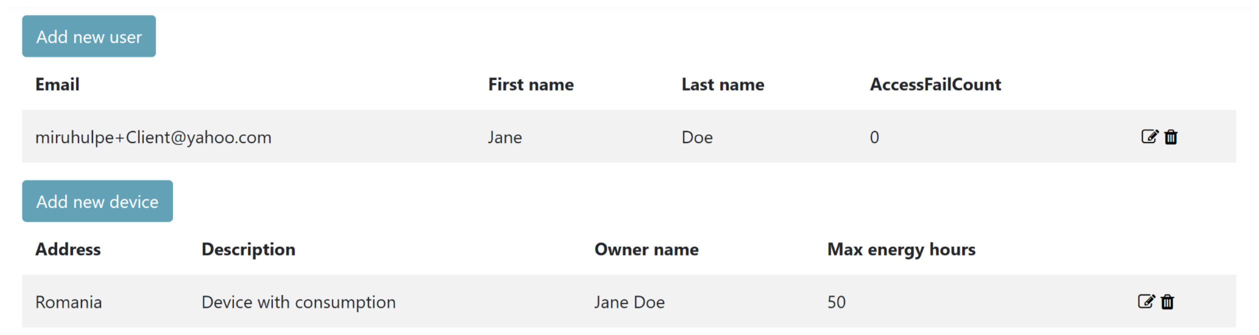
Address	Description	Max energy hours	
Romania	Device with consumption	50	📊



Date

07/01/2022

- On the admin page, we have two tables with both the users and the devices, also containing the actions buttons for creating, editing, or deleting elements.


Add new user

Email	First name	Last name	AccessFailCount	
miruhulpe+Client@yahoo.com	Jane	Doe	0	

Add new device

Address	Description	Owner name	Max energy hours	
Romania	Device with consumption	Jane Doe	50	

.NET Core Backend Application

About the architecture:

- The architecture follows the 3 layers architecture model (the presentation layer, the Business Logic layer, and the Repository layer)
- In the presentation layer, the classes are divided based on MVC (Model View Controller) principle.
- In the repository layer, there are the repositories and the DataAccess layer containing the application database context
- For the database, I used Sql Server database
- For the authorization part, in the cookies are stored claims specific to each role, that block a user with a certain role to access the pages of the other role. In this way, clients are not able to create new users or new devices and the admins cannot see a preview of their devices.
- For the authentication part, I used Identity scaffolded item in order to easily let the users, both admins and clients log in to their respective accounts.

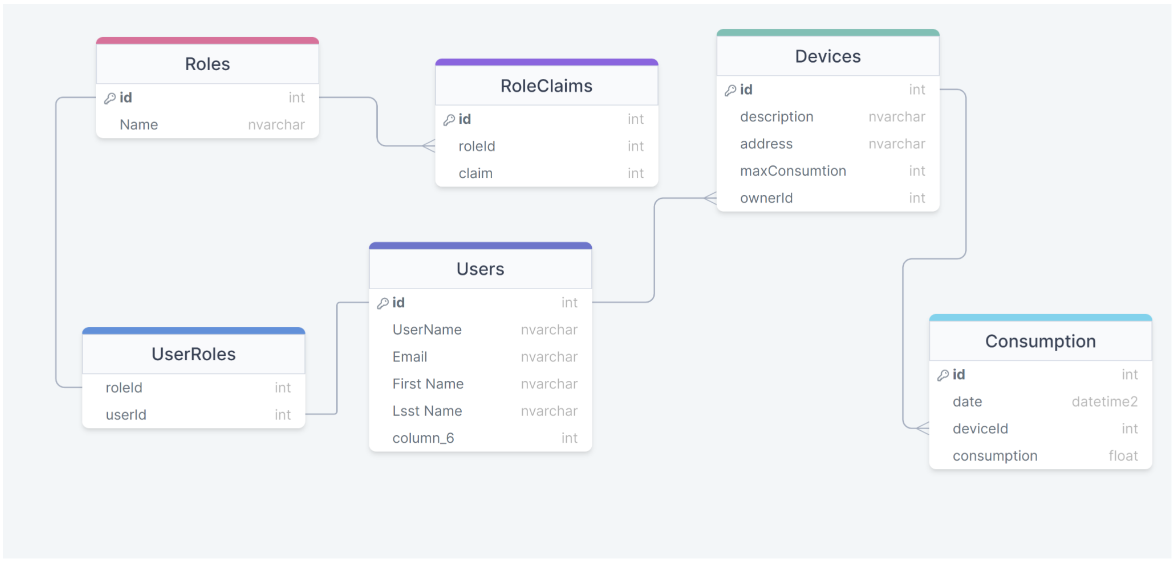
RabbitMQ:
Any application can send a message to this queue if the right credentials are known.
In this implementation 2 console applications are used to send messages to the client application,
Each console application reads a sensor id from its appsettings.json file and sends an object to the queue containing relevant information for the sensor measurement. This way on client's side the user can be identified by one of its sensor ids, and the notification according to the data measurement will be available only for his client portal page.

For sending periodically the sensor value randomly read from an excel file, a class SensorDataService is used which is registered as an IHostedService:

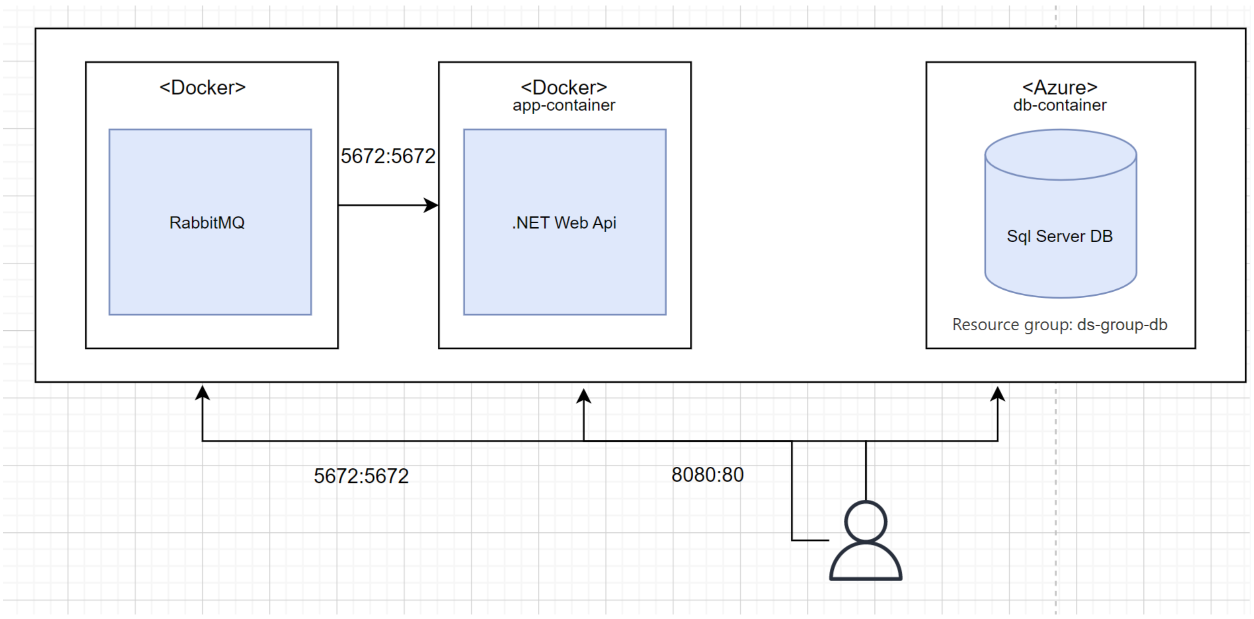
SignalR for web sockets transmission:

The notifications are sended via SignalR which is a .Net tool for websocketting. SignalR represents more than a websocket, it behaves like a framework over websocketing for .net applications. The used injected built in context is called HubContext

Database Diagram



UML Deployment diagram:


<Docker>
RabbitMQ
5672:5672
<Docker>
app-container
.NET Web Api
<Azure>
db-container
Sql Server DB
Resource group: ds-group-db
5672:5672
8080:80