ThermAlarm

Gal K, Chen E, Sapir B

What & Why?

ThermAlarm is a smart home alarm system, which raises an alert when a warm body, without a known BT id, enters the house.

The web app vision is to make the Alarm more reliable and independent.
Those 2 properties are very significant when it comes to a real-time system, especially in the security field.

As most home alarms there are 3 states:

1. **Disarm** - The system is neutralized and will not raise alerts.
2. **Arm** - The system is attentive and will raise an alert if needed. It processes and logs data in real-time.
3. **Alarm** - The system is Alarmed. A buzzer or a led will be activated.

What makes the system smart?

- ThermAlarm relies on both a simple **PIR sensor and a Thermal Camera sensor**. This allows us to make a better assessment if a warm body entered the house. This feature will **reduce the false-positive** alerts and increase the reliability of the system.
- ThermAlarm uses a **BT chip to allow family members to enter their armed house** without raising the Alarm and waking other family members from their good night sleep.

How?

In order to accomplish the goals above, we needed to design a system with 3 main domains: HW, SW and Cloud Services.

The HW domain contains the Arduino device and sensors listed below. It is responsible for collecting and sending the sensor's measurements to the application backend, through the cloud. It is also responsible for reflecting if the alarm is Buzzing (alert has been raised during Arm state).

The Cloud Services contain database storage, application storage, web pages and an event processor web job. It allows the application to run continuously, with a stable infrastructure which is supported by Microsoft.

The SW domain focuses on the code inside the Application Web App (MVC based) and Web Job.

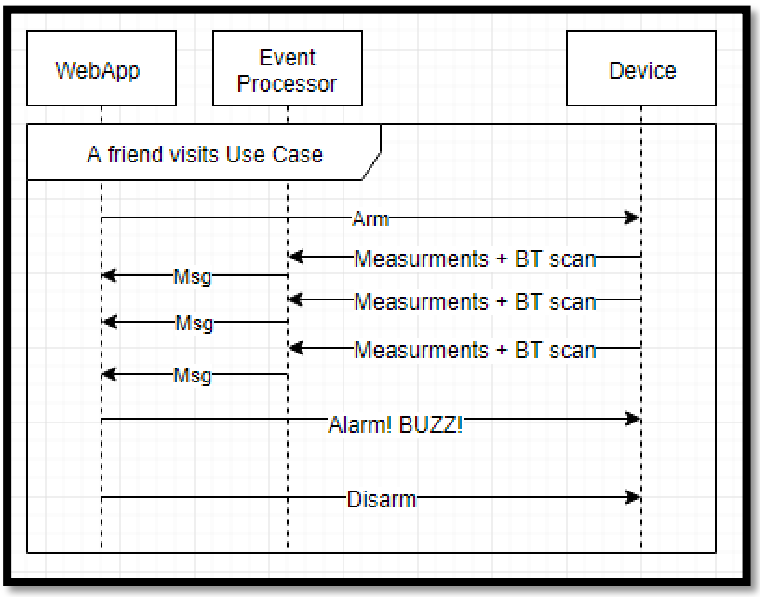
Before the first use of the application, it is required to register the family members to the DB. This can be done through the system's web app interface. Family members can be changed at any time.

System architecture

- ### Use - Cases

Application basic flows:

First Scenario



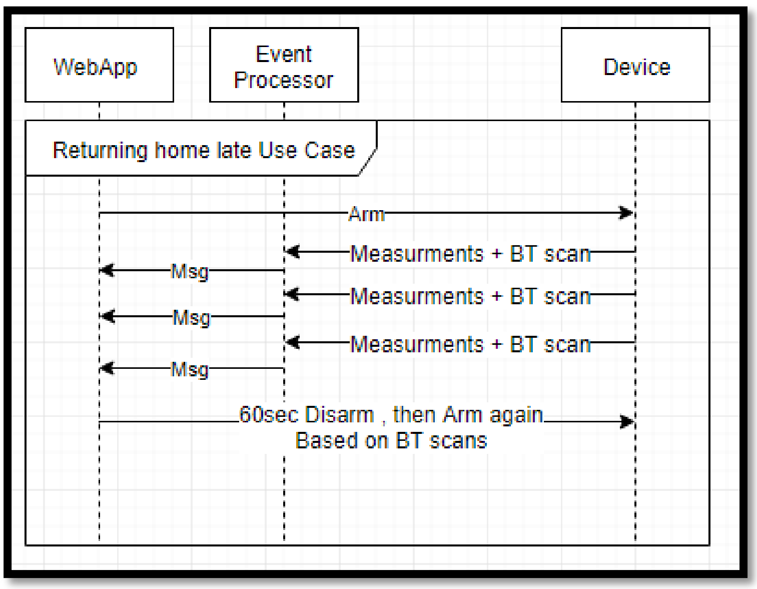
When a friend (not a registered family member, not a thief) comes to visit while the system is armed, the web app alarm logic will raise an Alarm (Buzz action), the device will notice and the blue led will be turned on.

Then, one of the family members will enter the Web App and turn off the alarm with a click on the "Disarm" button.

The Device will notice and the led will be turned off.

The system will rest at "Disarm" state until it gets more commands.

Second Scenario



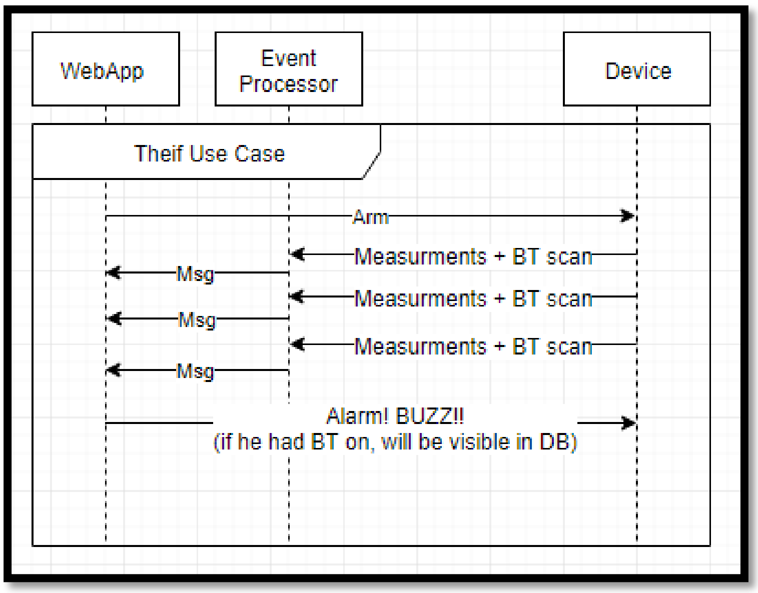
When a family member walks near the armed alarm device in the house, whether it is after a late date or just to take some milk from the kitchen, we don't want the alarm to buzz straight away.

When the web app logic sees a new known BT ID in its scans, it Disarms for 60 sec, then activates again in 'Arm' state.

Meaning, if the family member enters the house the alarm will not be raised for 60 sec.

The family member has time to go to his/her room, without waking-up any other family member and with no need to reactivate the alarm manually.

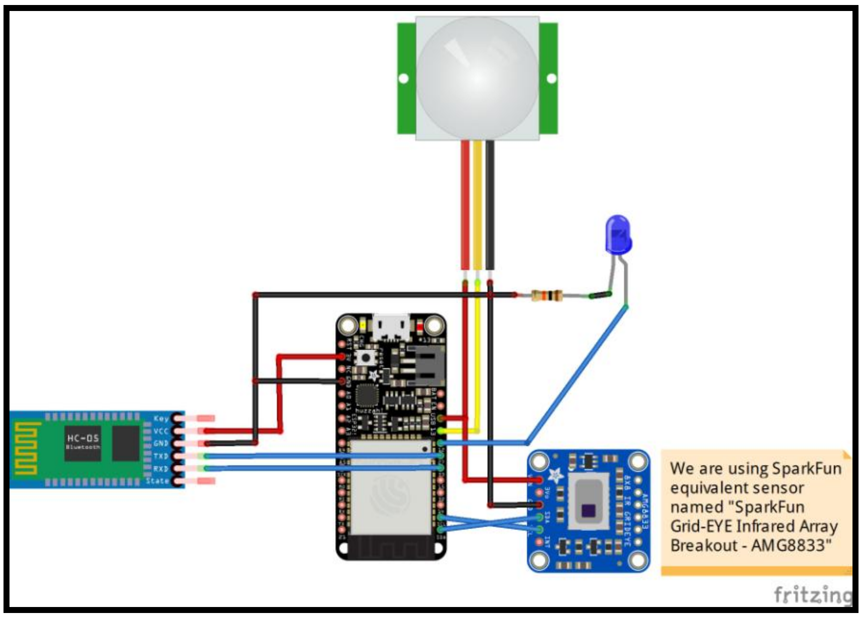
Third Scenario



When a thief enters the house while the system is armed, the web app alarm logic will raise an Alarm (Buzz action), the device will notice and the blue led will be turned on.

In addition, if the thief's phone BT was on, we will see it on the Database.

- **HW**
  - **Components:**
    - Arduino Adafruit Feather HUZZAH ESP8266
    - BreadBoard
    - BT sensor
    - PIR sensor
    - SparkFun GRID-EYE infrared array breakout - AMG8833
    - blue led
    - $10k\Omega$ resistor
    - Wires
    - USB cable
  - **Sketch:**



We are using SparkFun equivalent sensor named "SparkFun Grid-EYE Infrared Array Breakout - AMG8833"

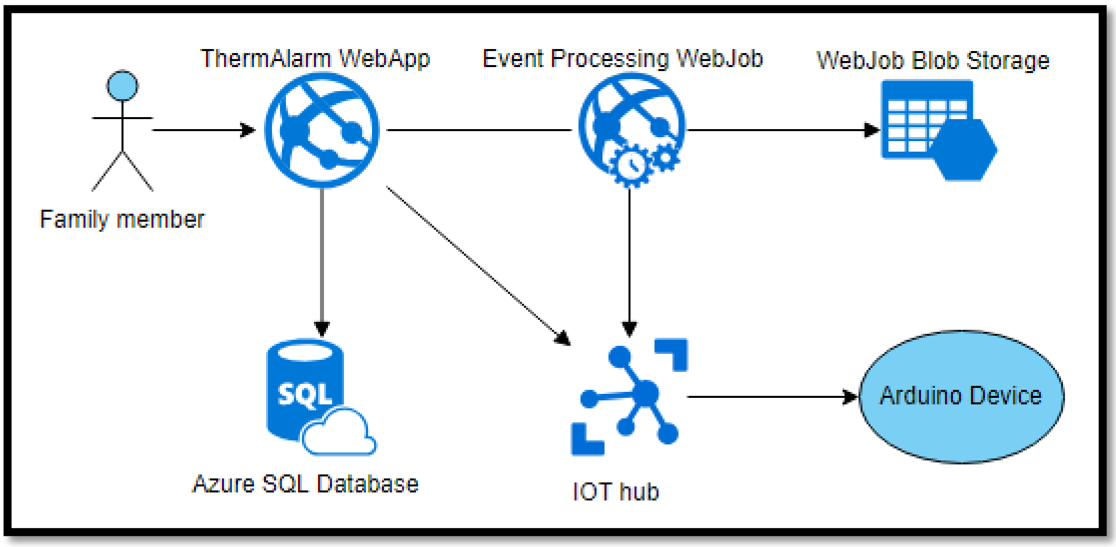
  - **More:**
    - The actual connections are made on a breadboard, this picture is for illustration only.
    - Power to the devices is supplied from a computer, through micro USB cable.
    - The blue alarming led can be replaced with a Buzzer.
  - **Code:**
    The code consists of a few modules
    - **config.h** - configuration header.
    - **sensor.ino** - sensors management (read mostly)
    - **message.ino** – build a message from the sensor's data and send to the cloud.
    - **Thermalarm.c** - communication manager (wifi & cloud). Device direct actions are defined here: Arm, Disarm, Alarm.
    - **ThermAlarm.ino** - main program: init & loop. In each loop, the sensor's data is being read and sent to the cloud.

Cloud Services (Azure)

Our Cloud Services are all stored in west Europe.

- **Components:**
  - Azure Web App - details in the SW domain.
  - Azure Web Job - details in the SW domain.
  - Azure Web Job Blobs (Storage) - holds telemetry data of the web-job.
  - Azure SQL Database - used as a service to the web app. Contains the following tables:
    - dbo._EFMigrationHistory - hold DB migration history.
    - dbo.Msg - holds all the messages sent from the device to the cloud.
    - dbo.Person - holds all family members (first name, last name, email, BT id, last seen)
    - dbo.AlarmAction - holds all alarm actions with a timestamp (Arm / Disarm / Alarm)
  - Azure IOT Hub - allows direct messages from the device to the Web App (through the Event Processing Web Job).

- **Sketch:**



- **SW**

  - **Web job:**

    This web job is a listening event processor that is based on the IOT Hub service. It defines an event processing host and configures a job that passes incoming messages to the web app as post requests (with relevant data).
    The web job is defined as a service of the main web app. (It is a triggered type and not continuous since our system is POC)

  - **Web App:**

    The web app is part of Azure Web Services.
    The model we chose to use is MVC (Model - View - Controller).

    The Web app contains both client & server-side code:

    - **Client-side** - 'cshtml' files as 'Views'. There is also a use of some css and javascript (to make the website more welcoming and responsive).
      The Web app interface contains 4 pages and is the only way to pass commands to the device by a family member.
      The pages are:
      - Index - main page: contains 3 buttons: 'Arm', 'Disarm', 'Does Alarm?'.
      - About - short explanation about the product.
      - Add Person
      - Remove Person (by email address)
    - **Server-side** - This part contains the Models, Controllers and Services. The app logic is defined here.
      The connection with other parts of the system is made with GET / POST requests:
      - The event processor sends a POST request with sensors data (which arrives at the device controller).
      - The web interface sends GET / POST to activate or deactivate the alarm and add or remove family members.

      This part also contains the device manager, which calls direct actions on the device (Arm / Disarm / Alarm).

      Models

      There are 2 Models in this architecture:
      - ThermAlarmDBContext - This model represents the Web App's session with the cloud SQL database, using Entity Framework Core infrastructure.
      - ErrorViewModel - to display web errors more clearly.

      Services

      There are 2 services in the project:

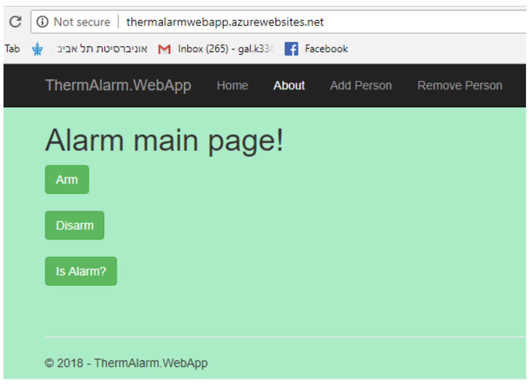
1. Alarm - a singleton class which holds the alarm's current state, family members (with lazy last seen - DB update every disarm) and a service client (allows connectivity to the device). The service allows adding or removing family members, trigger actions on the device and handles incoming messages.
2. Database Manager - a scoped service. Used by Alarm service and other controllers to log different kinds of data: family members, messages from the device, device actions and database migration history.

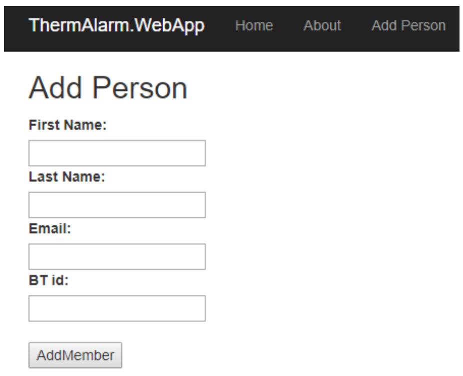
Controllers

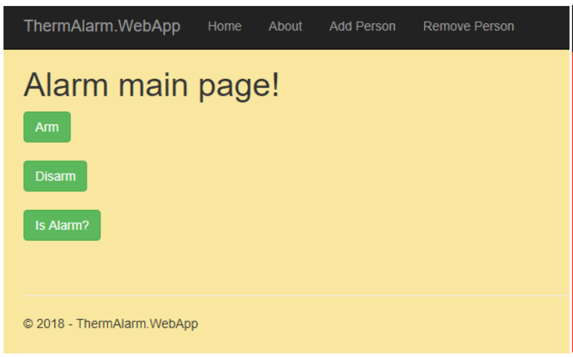
There are 3 controllers defined in this project:

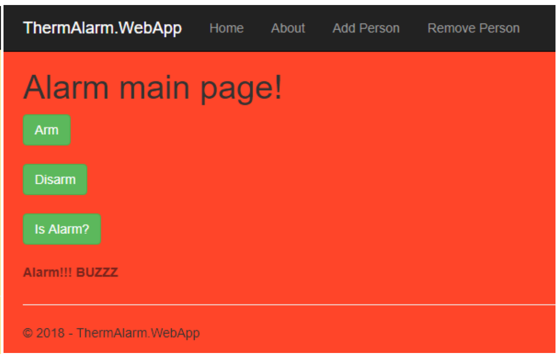
1. AlarmControllerBase (extends Controller) - All controllers need access to the DB service and the Alarm service. So that is a base class which contains those.
2. DeviceController (extends AlarmControllerBase) - Contains POST request message handler which gets messages from the event processing web job (originally from the device). This controller also contains some debugging functions for developers.
3. HomeComtroller (extends AlarmControllerBase) - main controller: GET request handlers foreach View, add/ remove persons, Arm / Disarm / Buzz handlers.

- **Web app interface**



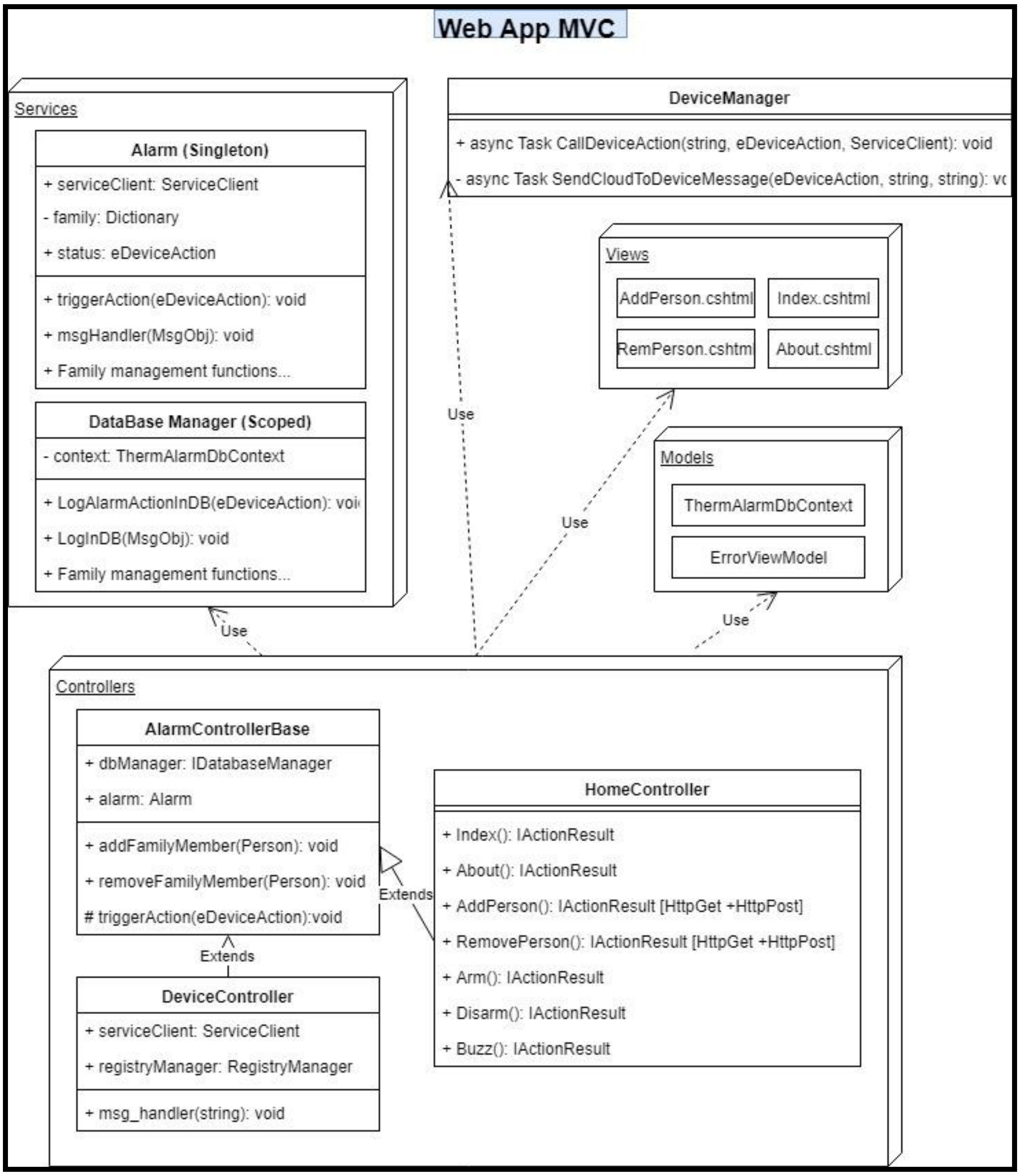




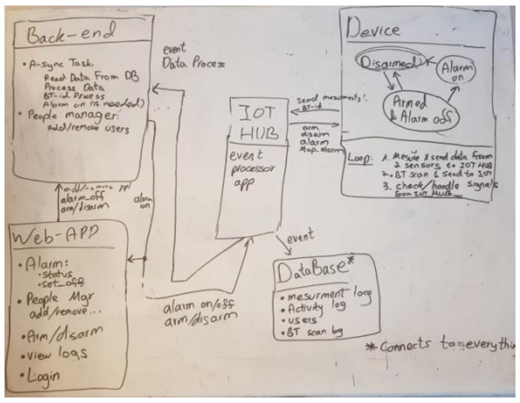


Alarm!!! BUZZZ

- **Sketch:**


Web App MVC
Services
Alarm (Singleton)
+ serviceClient: ServiceClient
- family: Dictionary
+ status: eDeviceAction
+ triggerAction(eDeviceAction): void
+ msgHandler(MsgObj): void
+ Family management functions...
DataBase Manager (Scoped)
- context: ThermAlarmDbContext
+ LogAlarmActionInDB(eDeviceAction): voi
+ LogInDB(MsgObj): void
+ Family management functions...
DeviceManager
+ async Task CallDeviceAction(string, eDeviceAction, ServiceClient): void
- async Task SendCloudToDeviceMessage(eDeviceAction, string, string): vc
Views
AddPerson.cshtml
Index.cshtml
RemPerson.cshtml
About.cshtml
Models
ThermAlarmDbContext
ErrorViewModel
Use
Use
Use
Use
Controllers
AlarmControllerBase
+ dbManager: IDatabaseManager
+ alarm: Alarm
+ addFamilyMember(Person): void
+ removeFamilyMember(Person): void
# triggerAction(eDeviceAction):void
Extends
Extends
DeviceController
+ serviceClient: ServiceClient
+ registryManager: RegistryManager
+ msg_handler(string): void
HomeController
+ Index(): IActionResult
+ About(): IActionResult
+ AddPerson(): IActionResult [HttpGet +HttpPost]
+ RemovePerson(): IActionResult [HttpGet +HttpPost]
+ Arm(): IActionResult
+ Disarm(): IActionResult
+ Buzz(): IActionResult

Problems, obstacles and solutions



As we learned about the project's components, our design changed.
This was a challenging yet educating process.
This is a picture of our first design for example:

- **Problems we have solved**
  - HW obstacle - The PIR chip we received required Vcc of 7.5[V], but our Arduino only supplies 5[V] or 3.3[V].
    <u>Solution</u> - Chen managed to borrow a PIR chip from his workplace, which requires 5[V] Vcc.
  - HW obstacle - No BT chips were available through our workshop**.**
    <u>Solution</u> - Gal's colleague let us borrow his BT chip.
  - Knowledge -obstacle - We didn't know how to use most components of the project**.** Starting from Arduino connectivity and sensors, to cloud services with Azure, using Azure database, building MVC application, web jobs etc.
    <u>Solution</u> - We spent a lot of time learning with "Pluralsight", Microsoft documentation and other sites.
  - The Alarm service requires an active connection to the database but using a scoped service (the DB context) within a singleton (Alarm) is problematic.
    <u>Solution</u> - We extracted the DB calls out to wrapper functions in the calling controller.
  - At the time of development, .NET Framework 4.7.2 was not available for Azure app services.
    <u>Solution</u> - Deployed our Web App as a self-contained application.

- **Obstacles we haven't solved yet**
  - In order for the BT chip to recognize the phone's BT, the phone needs to open the BT and stay in the BT page (meaning the phone is actively scanning).
    Another problem with this BT chip - in order for it to enter scanning mode, we need to physically push a button every time we reset the device. Not only is it uncomfortable, it also prevents us from doing a software reset from afar.
    <u>Possible Solution</u> - use a BT chip which is more adjusted to our needs.
  - We haven't found a way to update the Web - App interface dynamically from afar whenever the alarm goes on (without reloading the page).
    <u>Possible Solution</u> - Learn more about web design solutions.
  - We cannot distinguish between a person and a pet.
    <u>Possible Solution</u> - Using ML algorithm and better resolution camera.
  - HW obstacle – The thermal camera has a shorter range than required
    <u>Possible Solution</u> - Use better camera chip.

- **Future improvements**
  - As of now, the decision if an alarm should be raised is hard-coded with some formula**.**
    A better way to do it will be using some Machine Learning Algorithm based on labelled data.
  - Better interface design.
  - Support SW updates from afar, at run-time.
  - Use a better resolution camera.
  - Add timestamp to telemetry data and ignore irrelevant data for better handling and analysis

Links

The application:

http://thermalarmwebapp.azurewebsites.net/

Video:

https://youtu.be/5KyaFBrHXXM