Design and analysis of IoT Software
Fall 2021

Project report

December 6th, 2021

SOFE 4610U CRN: 44430

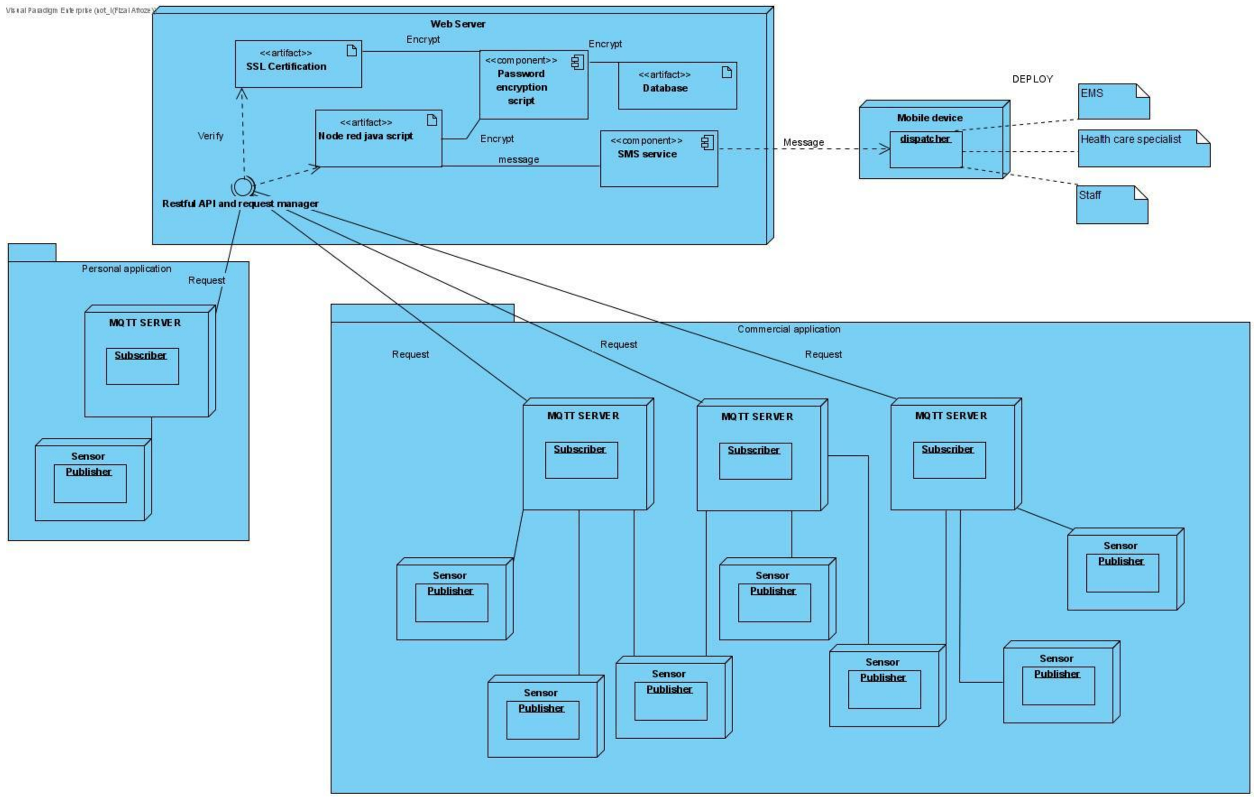
Iliya Karac	100703933
Nick Kvrgic	100613386
Pranjal Saloni	100653360

Introduction

Fall detection response is a service that allows the end user to be active without the supervision of a caretaker without the risk of life threatening injuries in the case of a fall. The service is made of 4 major devices: the sensor, the local MQTT server, the web server, and the dispatcher. The way it works is that the sensor will detect the conditions of a fall. This service can be sold to smaller organizations that are adequately staffed and can monitor their clients effectively. The system can be deployed worldwide under one organization however this would likely require an unreasonable amount of resources. The idea is to sell it to sell the service to organizations that operate within a small region where the organization can quickly deploy aid in the case of an emergency. These organizations can include hospitals, retirement homes and even health insurance companies. Hospitals and retirement homes can provide a large-scale commercial use for the application as they are the most densely populated with the target demographic for our product. While insurance companies would be able to provide a more personal service in the comfort of the user's home. To install the service a technician would install one or more MQTT servers at the users location where it would be configured with user information and wifi credentials. The user would be given the sensor in the form of a belt buckle, a clip, or a necklace. Ideally any accessory that has a definitive orientation in a 3 dimensional space and does not naturally sway with a person's natural movements like walking or sitting down. These devices would also be configured with user data and should hold a reasonable charge before needing to be recharged. The system functions by having the sensor connected to a local MQTT server. When the sensor detects a fall the sensor publishes a message to the subscriber client (which is running on the MQTT server). When the client receives the message it then sends a request to the web server that matches the request to a user from the database and updates that user. Finally the web server sends an SMS message to a dispatcher, likely the local EMS dispatcher who then in turn sends over medical professionals to the location of the patient for aid.

Project design and architecture

The planning and organization of the project was done during regular meetings usually once a week where progress on the project was discussed as well. Early on we came up with the idea for the project from the case study in the first in-class exercise. We were also inspired by the idea of a modern version of life alert. Initially we were very ambitious and wanted to add more sensors for more data, this included a temperature sensor and a heart rate sensor. Due to the amount of resources and work required to implement the extra functionality those ideas were scaled back. Instead we focused on the quality of the service. The sensor had to be accurate, the messaging system must have low latency, and the system had to be reliable and scalable. The system has a simple design that can be scaled up by adding more end devices. If an organization wanted to take our model and expand geographically they could buy more identical web servers in different locations. Consequently the architecture is also fairly simple and can be visualized as a pyramid. The foundation of the system is the sensors. They are the most numerous type of node in the system, they are in charge of sensing rwa data and reporting readings of note to the MQTT server. The middle layer are the MQTT servers, they are in range of the sensors and serve as a communication link between the sensors and the web servers. Lastly, the web server handles requests, logs data in the database and sends messages to the dispatcher when there is an emergency. It is the tip of the pyramid. Here is a diagram of the architecture and some inner components of each node:



If needed there is a larger copy of this image in the current directory available for closer examination.

Architectural design decisions

- The system must be easily scalable and flexible. For this reason instead of having a defined amount of MQTT servers there is simply an API endpoint in the server that accepts all http requests. Any new MQTT server/client must simply provide the id of the user that would have been added to the database when the signed up for the service
- The dispatcher device, while making this project we knew that there had to be some method of reaching emergency services but the specifics were unknown. When we discovered the twilio node in node red and the beauty of its simplicity we knew we could send an SMS message to mobile devices like a smartphone. The way twilio works is a user signs up for an account and they receive a phone number, a sim code, and a secret code for that number so it can only be accessed via the twilio node API. For the node to function it needs the 3 previously mentioned items and a destination number to send SMS to. In the demo we use my phone number, it also needs to be mentioned that the destination phone number has to be approved by twilio. In reality this phone would be directly in the paramedic dispatcher’s office. It should not go through the 911 operator because the format of the message is standard and the message can get through faster.
- The MQTT servers must be able to accommodate multiple sensors. If the system is installed somewhere that has more than one user i.e. sensor it should accept data from all the sensors. For that reason when the MQTT server is setup it will initiate 1 subscriber client for every active sensor at the facility
- Patient mobility, the user must be within range of an MQTT server while on the premises. For this reason there should be enough MQTT servers to cover the whole facility. All the servers must recognise every sensor (have 1 subscriber client for every user).

Deployment design decisions

- Reliability in the event of a fall the user may not get up to seek for help. We tested the reliability of the published message to be 100% within 7 meters from the MQTT server and by design if a person is lying down the sensor will publish a help message every minute.
- Latency, every minute counts in the case of a life threatening emergency. We made sure that the entire messaging process was under 1 minute from when a fall was detected.
- Accuracy, we do not want any false positives. This was a simple fix, we made the criteria for a “tumble” more rigorous so it would not activate on a strange movement.
- Server Security, any administrative pages or anything not meant for the public is blocked by a script that requires the credentials of the administrator to access. This includes the database, the node red code, and access to the web server itself.
- Client side Security, if the client is to send any http requests it should be to the correct server, therefore we have client side SSL verification of the server.
- Ideally the client should not be sending user data unencrypted but such a security measure was not implemented due to time constraints

Conclusion

Overall the project was very successful and we are proud of the final product. A little more refinement and it could have a real life application especially here in Canada with the increasing number of senior citizens. The chosen topic for the project was interesting, the functionality was there. The effort of this project was exceptional because it was the first time most of the group members experienced making their own website that had a domain name and was available for others to use. Making the sensor was fun. Having the SMS service working was something deeply satisfying. However not everything about the project was perfect. There was some trouble setting up the server due to outdated instructions that resulted in hours spent trying to resolve these issues. Furthermore, node red was our first experience with block coding interestingly enough and rather than serve its intended use of simplifying coding for beginners it had the opposite effect. The provided challenge made the final product much more rewarding. Lastly all the code can be found in the repository as well as a demo video to help better understand the project.