HOME AND OFFICE SECURITY TRACKING VIA MOBILE DEVICES

Valentin Velikov, Dimitar Mihaylov

Angel Kanchev University of Ruse

Abstract: *This paper presents a method for combining different technologies (both hardware and software) to obtain a fully functional application that monitors multiple sensors over the Internet and displays information on them on a mobile device. The architecture, basic principles and basic components of the application are presented. The system is designed to demonstrate some advanced technologies and techniques in the subject area.*

Key words: *Computer Science, home and office security, Internet of Things, Android, Cordova, React, NodeMCU, MQTT.*

INTRODUCTION

Mobile devices are an irreplaceable part of our daily life. They perform a variety of functions. And while at first they were just mobile phones or highly specialized devices, today's smartphones are multifunctional, adding new features and new applications to them.

One possible application is remote monitoring of various security sensors in different premises, both at home and in public areas: offices, warehouses and production facilities, etc. Depending on the sensors used, fire and chemical safety, burglary and unauthorized traffic, floods, weathering, electric shocks, etc. can be monitored. Depending on the objectives set for the system, it is possible to organize and send an appropriate controlling effect to an actuator after receiving a corresponding message from the sensor.

RELATED WORK

Several other similar applications have been explored before creating this security tracking system: ***iSmartAlarm*** [3], ***Vivint Smart Home*** [1], ***Smanos W020I*** [2] and more. Each of them has its own advantages and disadvantages that have been analyzed and on this basis it has been decided what functionalities the current application should have. Most of them are commercial, they are very closed, so it is difficult to personalise their functionality and connect or integrate with others applications, to increase or change the functionality.

DETAILED DESCRIPTION

The aim of this project is to create a decentralized open source system that enables remote communication between a mobile application and a wireless network with connected hardware devices with attached sensors that monitor various atmospheric indicators - temperature, humidity, gas concentration fumes and more. A motion sensor to detect physical access to the premises where the system is installed is also present in the system.

Mobile apps and devices must be able to communicate remotely - connected to different networks. For example, hardware devices are installed at home connected to the home wireless network and the mobile application is running outside home where the device on which it is installed (phone / tablet / laptop) is connected to the network of a mobile operator or other Internet network.

The build-up of the system should be based entirely on other libraries and open source platforms to provide full access to the source code of all the components in it. This gives flexibility and the ability to easily integrate with other systems, as well as

opportunities for future build of other products to extend the quality and scope of work of this system to the specific needs of businesses and households. It is also possible to add additional data security mechanisms where needed.

The purpose of the system is to transmit real-time sensor data without storing large amounts of data for users, but only keeping the last important messages (for example, the latest reported device traffic, the latest sensor data) to prevent theft of personal data and user-sensitive information - in case of a hacker attack. If the information required by the sensors is to be stored, the system must allow for the construction of an additional module of such design that can be integrated in an easy and convenient manner without disturbing the performance of the other components.

1. Choosing technology to develop the hardware prototype

The **NodeMCU** platform, which is an open-source project used for various products that are part of the concept of **Internet of Things**, is chosen for the prototype development of the device.

ESP8266 is the microcontroller that **NodeMCU** works with, using the “Espressif systems” library, *the Espressif nonOS SDK*, which is a set of wireless networking interfaces and layers of the **TCP/IP** model. There are also interfaces for working with input pins and other hardware components. **NodeMCU** is a set of libraries for working with various components of the **ESP8266** microcontroller and its varieties. The libraries themselves are written in C, enabling developers to use *LuaScript* in their programs [7], as the language is extremely fast and convenient to describe communication in real-time systems. For a more convenient use of the microcontroller, the system uses the *NodeMCU-DEVKIT-V1.0* board, which is an extension of the *ESP* microcontroller, with the ability to connect it via a USB port that can be used as power and for uploading source files to the controller. The development of *NodeMCU-DEVKIT* is open source hardware with 4MB flash memory.

LuaScript - a light and fast scripting programming language written in ANSI C, which enables it to work on many platforms. It was developed by Roberto Jerusalemci in 1993 in Rio, Brazil as a language to extend the capabilities of managing other more sophisticated systems and allow easier and convenient script development to manage them. To date, language has developed a lot, currently using version *Lua*. It is classified as a “multi-paradigm programming language” or a multi-purpose programming language. There is no object-oriented programming in the basics of the language, but Lua is very flexible and some of the PLO's capabilities can be further developed.

LuaScript has many features of the functional languages. The concept of "functions as first class citizens" is very strong, which allows the functions to be passed as parameters, to be returned as a result of performing other functions, to be assigned to variables, and to be part of different data structures.

The implementation of the hardware prototype also requires input data sensors that the controller will process. They include:

- **PIR** Sensor - Passive infrared sensor, based on radiated radiation (objects with temperature above the absolute zero, emit heat in the form of radiation). The sensor has two states - 0 and 1, when the state changes, it causes an interruption of the input pin to which it is hooked.
- **MQ-2** Smoke Sensor. The MQ series of gas sensors use a built-in small heater with an electrochemical sensor. They are sensitive to different gases and are used indoors at room temperature. The output of the pins to which the sensor is attached is an analogue signal passing through the **ADC** (analog to digital converter) and converted to a value

between 0 and 1023, where 0 is the lowest gas vapor concentration, and 1023 is the highest.

- **DHT11** is a temperature and humidity sensor. **NodeMCU** has a reading library for DHT sensors [7].

For demonstration purposes, these sensors are used as they are widely available. When integrating the system, it is recommended to use other more precise sensors in cases where this is necessary.

2. Choosing mobile application development technologies

The developed application is for the *Android* operating system, but with the idea that the project code can be easily used on other platforms. To this end, the **Cordova** Library, which is an Open Source product of the *Apache* Foundation, has been selected.

Cordova enables a Web - based product written in *HTML, CSS* and *JavaScript* to be packaged and installed as a mobile application running on many operating systems. **Cordova** is a **Software Bridge** that allows you to access *JavaScript* via interfaces to work with the built-in functionality of different operating systems - *Android, iOS, Windows Phone*, etc., through open source plug-ins developed by **Cordova** developers or other community programmers around this ecosystem. In this way, it is possible to reuse a large part of the code written in *JavaScript* when transferring applications from one operating system to another [5].

This approach accelerates the workflow by allowing a significant part of the development and testing of a mobile application to be done directly in a browser on a computer without booting an emulator or using a real mobile device. An application created with **Cordova** is called "hybrid" - a combination of a web-based and mobile application [8].

JavaScript is a fast scripting language supported by modern browsers. It was created in 1995 in "Netscape", with the idea of giving dynamics to the visualization of web pages. The language has evolved over the years and is now used in the development of server programs, embedded systems and mobile devices.

For the purpose of the project the latest version of **EcmaScript6** is used, which possesses many syntax changes and features that bring it closer to some functional programming languages. One of these options allows the use of generator functions. These are functions (sub-programs) whose execution can be stopped and extended when an event occurs, and between these calls the function keeps its state. This technique is particularly suited to the development of systems that handle real-time events, for example - the generator detects that a sensor data message has been received - the data is processed and the generator function waits for the next call [4].

To implement the components of the user interface, it was decided to use the *React.js* library. It was created by "Facebook" to solve the problem of the complex data flow in one application and the change of states of the various components in it. *React* offers a declarative way of describing components in the user interface by realizing a tree structure where each component can be modified only by the data it receives from its parent.

React is a library that manages user interface components, but it can hardly be used on its own and needs integration with technology that allows data management in the application. Such technology is provided by the *Redux* library, which is very closely integrated with *React*. *Redux* takes care of the status (application data) and provides a universal way to change them - through **reducers** functions that respond to an event caused by user interactions with the interface or event caused by the server. Completing these features returns the application's new state and the components are updated.

For the processing of asynchronous events when communicating with the server, it is decided to implement a **middleware** layer. This is a set of sub-programs that intercept events and process them. This layer is implemented using the *Redux-Saga* library, which provides a set of tools for working with generators and also has good integration with *Redux*.

Implementing the application with these technologies allows you to store the data status in the temporary memory of the mobile device. To save certain user settings that are available for the next time the application is started, the local storage mechanism that is accessed through *JavaScript* is used.

3. Choosing technologies to implement communication between devices

For communication between devices in the system, the Message Queue Telemetry Transport (**MQTT**) protocol is selected. The **MQTT** is a protocol for sending and receiving messages between different devices in real time, which runs on the principle of "publish-subscribe". The protocol works on **TCP/IP** and is designed for data transfer between remote devices, most commonly used as machine to machine protocol for projects that are part of the concept of the Internet of Things. The implementation of the protocol is very small in volume and makes it easy to use on many platforms (**NodeMCU** has a library for working with **MQTT**). **MQTT** allows a secure connection to be established through **TLS/SSL** - cryptographic protocols that provide secure internet communication [6].

MQTT communication is done using a **MQTT** broker-server that distributes messages and takes care of the authorization of users who communicate and access them to different data. The **Cloudmqtt.com** cloud service is used for the purpose of the project. It allows easy configuration of communication between a group of devices and users. When integrating into a real system environment, users can choose another **MQTT** broker or integrate one on their server. **Cloudmqtt.com** uses the most massive implementation of the **MQTT** - *Mosquito* Web Server. This is an open source project and can easily be configured to run on a machine of the user's choice.

Every hardware device and person - user of the system must have an account created in the **broker** to be authorized to send or receive messages.

MQTT communication is done through the topics (titles) - the equivalent of url addresses in HTTP. Through them, the user addresses messages that they send (publish) or who they want to subscribe to. The structure of the titles is determined when designing an application. **MQTT** allows them to declare access to different data to different users. For example, if *user832* is given access to **topic** - *device292922/temperature*, he will receive all the messages sent with this **topic**. If he/she is given rights for **topic** - *device292922/#*, he/she will receive all messages sent with a title beginning with *device292922/*.

The **MQTT** protocol as a standard for communication is particularly suited for systems that involve real-time data transfer between physical devices, as a built-in notification capability for loss of Internet connectivity on a device, as well as guarantee mechanisms of delivery of a message - extremely important for security systems. Part of the **MQTT** standard is also the ability to keep an important message from a **topic**, such as the last reported motion from a sensor [3].

4. Communication architecture between devices, server and mobile clients

The diagram in Fig. 1 illustrates the communication model that is used to implement the system. A particular instance of the system is shown where a user through a mobile device remotely monitors data from sensors that are located on a single local area network. The system also allows for tracking of groups of devices that are connected to

different networks. It is also possible to monitor devices from more than one user. Each of the hardware devices is independent and can work independently.

To connect a hardware device to the server, it is necessary to have a pre-created account in the broker and its username and password stored in the device's configuration file. There are also the parameters of the **MQTT broker** - the address and port to which the **MQTT client** can connect. The advantage of this communication model is that the device cannot be accessed from the outside as it does not open a port on which the device listens but initiates a connection to a server port.

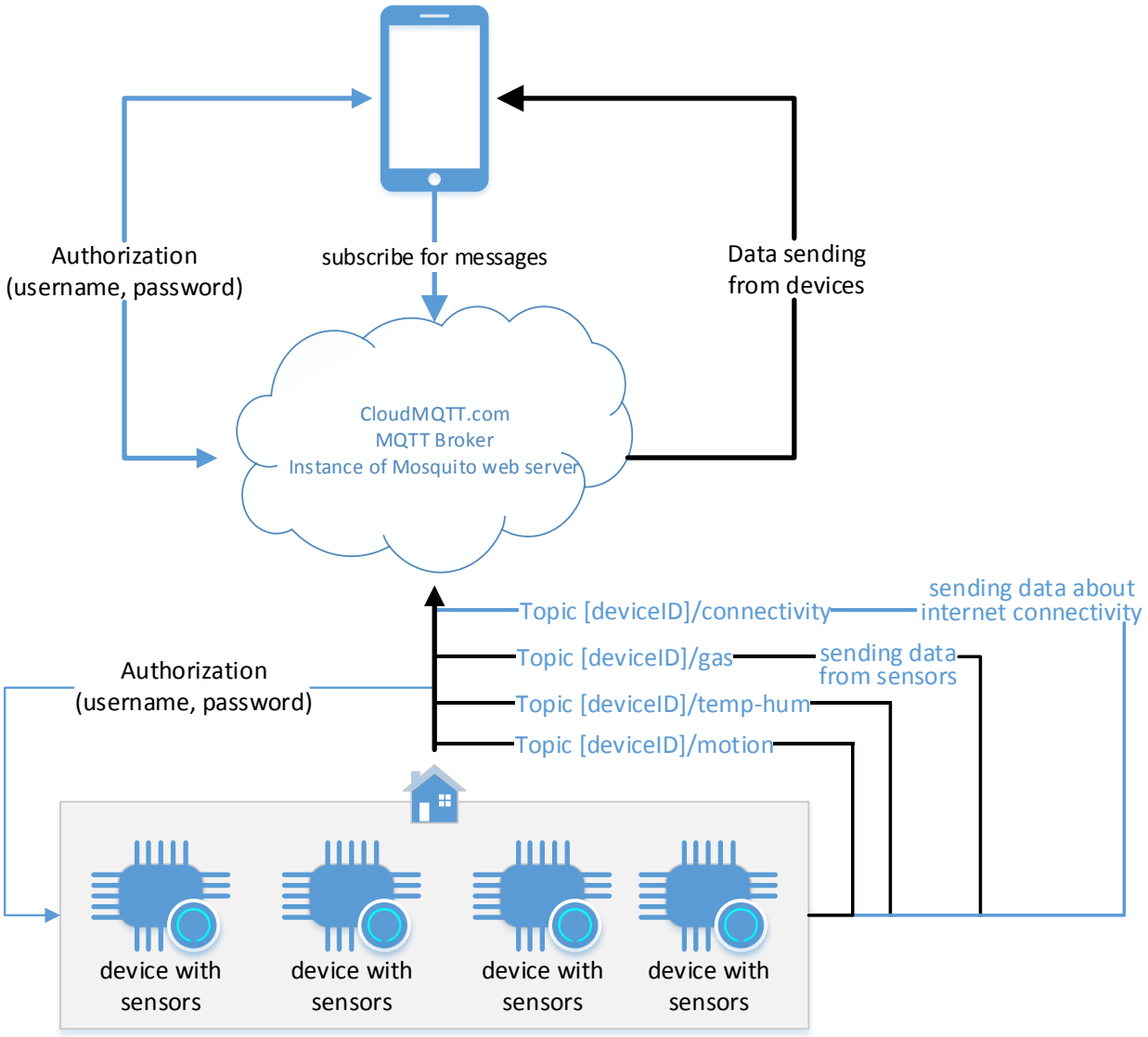

Fig. 1. Architecture of communication in the realized system

After the initial connection, the **MQTT** client that is created on the device can send messages to the server. Each message is addressed with a **topic** that has the following structure:

deviceId / message_type

• *deviceId* – device ID

• *message_type* – type of the message (connectivity, gas, temp-hum, motion)

MQTT communication allows sending messages from certain devices and subscribing and receiving these messages from other devices. Each **MQTT client** can post and receive messages, with his broker indicating the rights to which titles he has

access to. As a first parameter in the structure of each **topic** in the system, it is chosen to be the ID of the device from where the message is sent. This makes it easier to declare certain users access to the messages from their devices.

The declaration of rights happens on the broker, with two types of access to each **topic** - access for writing and for reading. For the implementation of the system, access to writing on a **topic** has hardware devices and access for reading - the users with mobile clients.

Table 1: An example of declared rights for system integration

User	Topic	Reading	Writing
Device789Kitchen	Device789Kitchen/#	False	True
Device325Office	Device325Office/#	False	True
User892	Device789Kitchen/#	True	False
User892	Device325Office/#	True	False

With these set access rights, *User892* will receive data from the *Device789Kitchen* and *Device325Office* devices. It will have access to all titles that begin with *Device789Kitchen* and *Device325Office*.

This method of declaring access rights is very flexible and allows users in the system to be limited and to receive data only from devices that are intended for them. The limit may also be for a message type, for example, *User892* can only be accessed by the *Device325Office/motion* header. Then it will receive messages only from the motion sensor on the device with ID *Device325Office*.

With each message, in addition to the title, data is also sent - a string with information. In the system implementation all messages are sent in **JSON** format, as it is extremely convenient for describing data structures when working with *LuaScript* and *JavaScript*.

JSON or **JavaScript Object Notation** is a text-based open standard designed for human-readable data exchange. It comes from the *JavaScript* language to represent simple data structures and associative arrays called objects. Despite its link to *JavaScript*, this is a language-independent specification.

The message structure in the system looks like this in **JSON** format:

{ "value": 1, "timestamp": 1494268039, "error": null }

• *value* – it contains the value the device sends (for example, 1 - detected motion from the PIR sensor)

• *timestamp* - it contains the exact time in UTC of occurrence of the events measured in seconds

• *error* – contains an error identifier for data reporting.

The **MQTT** communication standard supports a mechanism to save the last message for a title. This feature is used in the implementation of the system. A flag retained is also specified when sending a message, indicating whether this message should be retained by the broker. If the flag has a value of 0, the message will only be received by the mobile clients who are currently connected and have access to the message. If the flag has a value of 1, the **broker** will keep this message and when the mobile client is next connected, it will receive the message. The message will be kept on the server only until another message with the same topic arrives. This mechanism is very convenient as it allows the system to keep the last state of each device for each of its sensors without

realizing storage of all data from the sensors. When a mobile client connects to the server, it immediately receives information about each device that contains:

- The last recorded movement.
- Whether the device is online or offline.
- The last measured temperature and humidity.
- The last measured gas vapor concentration.

In the system implementation, the messages that are sent with flag retained = 1 are:

- Motion reporting.
- Measured data from each sensor.
- Internet connectivity messages.

The **MQTT** protocol also has provisioning mechanism to messages delivery - **QOS** (Quality of Service). The protocol has 3 levels of **QOS**, and the highest is used for the system implementation. This ensures that each of the messages will be delivered to mobile customers just once.

The mechanism for Internet connection losing notification, the **LWT** (Last Wave and Testament), is used. Each device configures such a message and in the event of loss of connectivity between the server and the device, this message is sent to mobile customers. This allows monitoring the connectivity of each device in real time.

TEST RESULTS

Functional tests have been performed for all hardware and software modules. The identified inaccuracies were removed. All subsystems work correctly and according to the specifications.

After the tests it is possible to conclude that the prototype of the system is reliable, it works well and it can be successfully integrated for use in a real environment. The architecture and implementation make it easy to maintain and use the system after the initial installation. It allows integration of other modules and programs to extend the scope and quality of the system. Flexibility is allowed when selecting infrastructure - servers and devices to operate.

CONCLUSION

Many security monitoring systems are available on the market through sensors and other devices that have a wide range of capabilities and functions. The cost of such products is quite high and their successful integration and use is highly dependent on the user's needs. All systems are closed and do not provide good customization capabilities to suit the specific needs of the user. This stops other companies from developing products that are compatible with these systems to expand their scope and quality of work.

Security systems that are offered do not allow users to choose which server their data should pass through. Information from such security monitoring devices is extremely important. Most users would not want to give their data freely to large companies and organizations, as this has the risks of leakage of large amounts of information in a hacker attack on a large system. When the information is not concentrated in one place, and each user can choose where his data goes to, the chance to run a large amount of sensitive information is significantly smaller.

When using commercial systems, companies and households that integrate them do not have a good opportunity to choose the system infrastructure themselves and customize it according to their needs.

All this provides opportunities for the implementation of decentralized systems based on open source libraries. With a host of different hardware components, devices for tracking and managing mechanisms across the Internet can be successfully synthesized.

REFERENCES

[1] Aaron Tilley, Vivint Smart Home Security System, https://www.vivint.com/ https://www.forbes.com/sites/aarontilley/2016/07/06/vivint-smart-home/#53e76808525e 14.03.2018.

[2] Don Reisinger, Smanos W020 WiFi Alarm System review, https://www.techhive.com/article/3153495/connected-home/smanos-w020-wifi-alarm-system-review-buy-just-the-parts-you-need.html, http://www.smanos.com/w020i, 2018.

[3] iSmartAlarm Home Security System, https://www.cnet.com/products/ismartalarm/ https://www.ismartalarm.com/, 15.03.2018 .

[4] Michael Fogus, O'Reilly Media, Functional JavaScript, 2013.

[5] Nizamettin Gok, Nitin Khanna, O'Reilly Media, Building Hybrid Android Apps with Java and JavaScript, 2013.

[6] Valerie Lampkin, Weng Tat Leong, Leonardo Olivera, IBM, Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry, 09.2012.

[7] Documentation of libraries from the NodeMCU platform NodeMCU, Nodemcu SDK https://nodemcu.readthedocs.io/en/master/, 20.05.2017.

[8] Documentation for using the Cordova platform for Android apps, Cordova Android development, https://cordova.apache.org/docs/en/latest/guide/platforms/android/index.html 20.03.2018.

CONTACT ADDRESSES

Pr. Assist. Valentin Velikov,PhD
Department of Informatics and Information Technologies
Faculty of Natural Sciences and Education
Angel Kanchev University of Ruse
8 Studentska Str., 7017 Ruse,Bulgaria
Phone: (++359 82) 888 326,
Cell Phone: (++359) 886 011 544,
E-mail: val@ami.uni-ruse.bg

Dimitar Mihaylov, BSc student
Department of Informatics and Information Technologies
Faculty of Natural Sciences and Education
Angel Kanchev University of Ruse
8 Studentska Str., 7017 Ruse,Bulgaria
Cell Phone: (++359) 886 868 316
E-mail: dmihaylov93@gmail.com

СЛЕДЕНЕ НА СИГУРНОСТТА В ДОМА И ОФИСА ПРЕЗ МОБИЛНИ УСТРОЙСТВА

Валентин Великов, Димитър Михайлов

Русенски университет „Ангел Кънчев”

Резюме: *Статията представя метод за обединяване на различни технологии (както хардуерни, така и софтуерни) за получаване на функционално пълно приложение, което следи през Интернет множество датчици и показва информацията от тях на мобилно устройство. Представени са архитектурата, основни принципи и основни компоненти на приложението. Системата е създадена за демонстрация на някои техники, съвременни технологии и прийоми в предметната област.*

Ключови думи*: информатика, домашна и офис защита, Интернет на нещата, Android, Cordova, React, NodeMCU, MQTT.*

ISSN 1314-3077