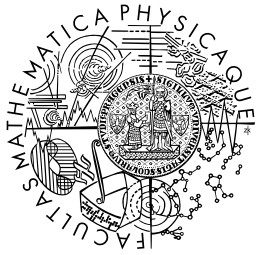

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

David Hřivna

Smart inventory control system

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis: RNDr. David Obdržálek, Ph.D.

Study programme: Computer Science

Prague 2024

I declare that I carried out this bachelor thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Dedication.

Title: Smart inventory control system

Author: David Hřivna

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. David Obdržálek, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: The goal of this thesis is to provide users with system of smart fridge that would help them track amount of groceries in their household. The thesis introduces a system consisting of server and device. The device is used as a scanner and dashboard. Server provides main application logic. The system provides adding products to the database using the bar-code, presenting products that expire soon to the user and allows user to create scheduled reminders and recipes consisting of the products in the database.

Keywords: IoT, C++, C#, storage system

Název práce: Systém pro chytré sledování inventáře

Autor: David Hřivna

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. David Obdržálek, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: Cílem této práce je poskytnout uživatelům systém chytré lednice, který by jim pomohl sledovat množství potravin v jejich domácnosti. Práce představuje systém skládající se ze serveru a zařízení. Zařízení je použito jako skener a obrazovka. Server poskytuje hlavní aplikační logiku. Systém poskytuje přidávání produktů do databáze pomocí čárových kódů, prezentace produktů, kterým brzy uplyne doba spotřeby, uživateli a umožňuje mu vytvářet vlastní upomínky a recepty z produktů v databázi.

Klíčová slova: IoT, C++, C#, skladový systém

Contents

Introduction

With raising popularity of smart devices these days, we see many devices that make our everyday life easier. Home appliances are no exception. Internet of Things (IoT) is becoming a standard not only in industrial but also personal and home use.

Manufacturers already offer smart fridges with features like tracking the inventory, connection to the home assistants, monitoring and possibility of adjustment of the temperature inside of the fridge and more. However price of these products is reflecting the invested development and uniqueness in the market. Manufacturers often don't implement these somewhat premium features to cheaper products. However our device may provide a cheaper and convenient solution to people who don't want to buy a big and expensive fridge. Instead of having a whole new fridge the users would just get the small device. Some may also find convenient that our device doesn't collect any personal data about the users and since it is open source solution, capable users can easily add new features if they wish to.

The system consists of two main parts. First of them being a small microcontroller device that would serve as a scanner and dashboard for the user. The second one is a server which communicates with the device and provides the user with a web application that helps them interact with the system in more detail. For example, seeing future expiry dates, adding own recipes, editing information in database and adding custom notifications. Outside of the sheer system we will also implement our own drivers for most of our peripherals.

The goal of this thesis is to implement smart inventory system in the form of smart fridge. This concept would provide users with tracking of their groceries, monitoring the temperature in the fridge, help track the expiry dates and therefore help users prevent wasting food. Users will have the ability to easily add and remove products and see which products are going to expire soon.

First we will introduce the conceptual overview of the problem. Then we will go through the implementation of the solution, where we will first look at the hardware parts and afterwards the software parts of the system. After that we will look into some issues that occurred during the implementation. At the end we will go through the evaluation and possible future work based on this system.

1 Concept

In this chapter we will introduce requirements for the system and features of the system. We will also go through the typical environments where the system could be used.

1.1 Features and Requirements

- The system should provide the user with ability to scan bar-codes of the products and automatically add or remove products based on the scanning.
- The system should present the user with the products expiring in the near future and the products that are below the given threshold.
- The system should let the user add custom reminders and show these reminders on the set time.
- The system should let the user add, edit and remove recipes consisting of products already registered in the system.
- The communication of individual parts of the system should be secure enough to keep integrity of the messages.
- The components of the system should authenticate each other.
- The information about the products should be persisted in system's database.
- The system should allow the user to enter products without bar-codes.

1.2 Typical use

The system is focused mainly for home use. The device would help the household keep the track of food in the storage. This would help prevent the spoiling of the food and potentially help the household create shopping lists. It would notify the members of the household to do certain activities such as take medications. The solution could be modified to function in a professional environment. One such example could be food storage in restaurants and hotels. Another example is usage in the assembly line where the assembly process consists of individual parts from the storage.

2 Analysis

In this chapter we will first look into other solutions in the market. We will then go through possibilities for individual parts of the system. Starting with hardware components of the system, continuing with software libraries and communication protocols.

2.1 Commercial solutions

Manufacturers of home appliances already introduced the concept of a smart fridge. These devices offer many features. Among these we can find: playing music, connection to home assistants, making shopping lists, mirroring smart phone, inventory tracking and more. On the other hand these solutions often come only with big fridges and the price of these products often increases in tens of thousands Czech crowns.

Examples of smart fridges in the market:

- Samsung Family Hub[1]
- Hisense RQ759N4IBU1 [2]

2.2 Problem analysis

The concept of smart fridge consists of two main parts: server and device. These parts communicate with each other over a network. Our smart fridge concept raises up a few concerns, let's introduce them. From the users point of view these are following: how will the user add and remove products from the storage system, how will the system interpret state of the system to the user? On the other hand the developer may question: how will the system cooperate, should it be real-time, should it be secure, how will the system be structured? In the following sections we will investigate these problems and find the solution that fits our needs.

2.3 Hardware

2.3.1 Server

For the server part of the system it is not necessary to introduce any special device but may be beneficial. Personal computer would be sufficient to run the server application. On the other hand it would be very inconvenient to run our server application on a personal computer. Users most of the time turn off their personal computers when they don't use it. We need a device that would provide the access to the system at any time. One option would be running the web application on user's own server. That is not really a reasonable solution since there are not many users who would run a server machine at home. Other option would be hosting the application at third party hosting platform. This option would work but user would have to pay a monthly fee. Another option that

we have is to use a local small single-board computer. Good candidate for that would be Raspberry Pi[3]. Since many Linux distributions support ARM platform, we can install Linux distribution of our choice on the device and run the server application as a service. Choosing Raspberry Pi over unused personal computer will also be more efficient since energy consumption of the board is much lower than actual personal computer.

2.3.2 Device

The selection of the hardware of the device depends on several aspects. Cost, connectivity, programming interface, availability and official libraries. Among the most popular manufacturers we find Arduino[4], Espressif[5], Raspberry Pi[3] and STMicroelectronics[6]. We compare candidates from the manufacturers in the following table. To reduce the set of the compared boards we choose the ones which has WiFi and price below 600 Czech crowns.

Board	Accessible pins	Camera	Price
Arduino Nano ESP32	+	-	-
ESP32-CAM	-	+	+
Seeed Xiao ESP32-S3	-	+	-
Raspberry Pi Zero W	+	-	+

For our purpose we choose Espressif's ESP32[7] SoC (System on Chip) for several reasons. ESP32 SoC has built-in Wi-Fi and Bluetooth support. That makes connecting the device to the network much easier since we don't need any additional hardware. The CPU has two cores. The typical use of this is using one core for the network interface and other one for the application. The SoC of the ESP32 provides us with four SPI interfaces, two I2C interfaces, three UART interfaces and much more. That creates a nice base for connecting multiple peripherals to the chip. The SoC also has hardware support for AES, SHA-2, RSA and RNG (Random Number Generator). That helps speed up communication over secured channels such as TLS (Transport Layer Security). Another thing that makes the ESP32 a good candidate is the support of the community. There are many development boards and modules that are built up on ESP32. One such example is Ai-Thinker's ESP-CAM[8]. This development board is equipped with 24-pin camera interface, 8 MB SPI RAM and MicroSD card interface. With its price being lower than most of the competitors, this is our choice for the purpose of this thesis.

2.3.3 Peripherals

Display

Since we want our device to present information to the user, the display is a nice way of doing so. When choosing the LCD we have to take into consideration communication interface of the display, its size and resolution. The most popular communication interfaces of the displays suitable for embedded devices are SPI (Serial Peripheral Interface), I2C (Inter-IC Communication also known as IIC or I^2C). For the purpose of this thesis we choose at least 3.5" 480x320 SPI display

with the touch screen. The display of this size is big enough to accommodate good preview of the camera frame and text of readable size.

Our main candidate is ILI9488[9]. The main reason for choosing ILI9488 is that it communicates over SPI interface which is already supported by the ESP32. This display is also equipped with a touch screen. The touch screen benefits us as a communication interface for the user since we can create individual buttons on the screen.

Camera

Since we will be scanning information about the product, we will need something to scan the information with. There are two possible approaches that we can take. First approach is taking an already working device and connecting it to our main device board over some communication interface such as UART, SPI or I2C. This approach limits us to the capability of the device since we are only relying on the firmware of the device. On the other hand, the second approach needs just a camera which will provide us with its frame-buffer which we can then process in our software in whichever way we need. This approach offers much more potential since it doesn't limit us to one specific function of the device.

Thermometer

It would be nice to track the temperature inside the fridge. In order to do that we will need a thermometer sensor. The thermometers can be put in two groups based on how they interpret the measured data. First group is analog thermometers. Analog thermometers send analog signal to the micro-controller. This signal then gets processed by ADC (Analog to Digital Converter) of the micro-controller itself. Second group is digital thermometers. Digital thermometers send digital data to the micro-controller. The data is usually sent using a communication interface, such as I2C or One-Wire. This thesis uses an I2C thermometer AHT10[10]. This thermometer is also accompanied by the humidity meter which is a nice benefit for our system. The accuracy of the temperature measurement is $\pm 0.3^{\circ}$C and accuracy of the humidity measurement is $\pm 2\%$.

2.4 Software

In this section we will go through the libraries and frameworks that can be used for projects of this kind.

2.4.1 Server

For the server part we want the support of major platforms. We will need not only the web application but also a service that would communicate with our device. We choose Blazor Web Application[11] in C# .NET as our platform to build on for several reasons. C# applications run on all popular server platforms (Linux, Windows). That eases the deployment of the system. The Blazor Web Application framework also incorporates Bootstrap framework. That eases the development of the web application since we are able to generate the HTML

and CSS code. Another reason why we choose C# is Entity Framework. That allows us to run database queries and transactions from within the C# code. This makes working with the database much easier. Another benefit of using this combination is that we can scaffold our web pages. Scaffolding is automated code generation for the web pages. That makes the development of the web application much smoother and the developer can therefore focus more on application logic instead of the presentation layer of the system. The reason for choosing Blazor Web Application instead of the Blazor Client Application is that we don't expect too many clients to be accessing the system. Therefore we can easily afford server side generating of the HTML. The possibility of extending the system is benefited by the architecture of the Blazor framework. Since Blazor framework uses components it feels more like developing a desktop application than a web application. Some developers may be more comfortable with this.

2.4.2 Device

ESP32 can be programmed in various ways. Frameworks that support development for ESP32 are ESP-IDF[12] (C/C++), Arduino (C subset), Micropython[13] (Python) and Nanoframework[14] (C#). We will be using ESP-IDF since its low level point of view helps us control the firmware properties in more detail. ESP-IDF also supports image processing applications with their esp-who[15] library. The library supports multiple applications such as bar-code or QR-code scanning, face detection and more. Another benefit of using the ESP-IDF is that it contains its own implementation of the FreeRTOS real-time operating system kernel. Using the ESP-IDF's FreeRTOS provides the developer with essential multitasking capabilities.

For the peripherals there are possibilities of using libraries. There are many libraries for SPI displays. Usage of such libraries speeds up the development but could catch us off guard with data formats or API structure. For that reason we implement our own display library. Since most of the libraries are written in C, another benefit that we get from writing our own library is that we can use C++ features such as templated code. This allows us to define compile-time constants. Example of such case would be using it for configuration structures.

The driver for the thermometer is available as ESP-IDF library: esp-idf-lib[16]. This library is written in C. Even though the library is available, the driver for the AHT10 is fairly simple so we will implement it ourselves.

2.5 Database

We will need to persist products, recipes and notifications in the database. Since our data is well structured we will be using relation database. We choose to use SQLite[17] database engine. The main reason for choosing SQLite is that it is self-contained, serverless, transactional database engine. Since it does not need a server process it is easy to deploy. Another reason for choosing SQLite is that .NET Entity Framework supports it by default and therefore it makes the development and deployment easier and smoother.

2.6 Communication

Since we will need the server and the device to communicate with each other we need some connection and communication channel. There are two main paths we could take. First option is to develop a custom communication protocol and implement it using socket connection. The other option is to use some already known protocol. When choosing such protocol in the world of IoT we take into account complexity, network cost and security. Some of the popular network communication protocols in the world of IoT are HTTP, CoAP, MQTT and AMQP. We choose MQTT[18] protocol since it is natively supported by ESP-IDF and also C# via MQTTnet[19] library.

2.6.1 MQTT protocol

Message Queueing Telemetry Transport (MQTT) is a publish-subscribe protocol used in IoT systems. Client (device) publishes to the topic and gateway (server) forwards the message to all clients subscribed to the topic. MQTT gained its popularity thanks to many factors. For example low cost of transmission compared to protocols like HTTP. The messages consists of topic and binary data. That offers versatile mean of communication. MQTT also supports communication over TLS connection. Authentication is possible in two ways. Username and password and certificate-based authentication.

The publish-subscribe mechanism allows easy extension of the system. We use topics for products, notifications, temperature and humidity but other topics could be easily added to the system. We also use the TLS property of the MQTT protocol to ensure security of our communication and authenticity of the individual parts of the system.

2.7 Security

From the security point of view there are also some decisions we have to make. On the client's side of the system the security choice is quite straight forward. Secured web applications are today's standard. Therefore we choose HTTPS to provide secured connection to the web application. HTTPS is fully supported by Blazor .NET Web Application.

On the other hand securing the communication between the device and the server may be a challenge since we are working with a small device that may not be capable of all security technologies available on standard computers.

The security of the communication between device and the server can be mitigated on different levels. We could assume that the device and server are in a local protected network without any intruders. In that case the security measures on the side of our systems are not necessary. On the other hand if our system communicates in a non-trusted environment we have to apply measures to keep our system secure. We consider non-trusted environment to be an insecure network. We assume that the user is keeping the device at home and therefore consider the importance of the device in case of physical attack negligible. The security of the communication can be improved by using secure protocol. For the

MQTT protocol there is a standardized version of MQTT communicating over TLS protocol.

Our mean of authentication will be certificates. The certificates provide us with easy authentication of the device. We can flash the certificate on the device and server will verify the certificate and allow the device to communicate only if verification was successful. The device also verifies the certificate of the server it connects to. Usage of certificates also provides more opportunities for secure extensions of the system. One example that would use such mechanism would be updating the device over the network. We would then check the firmware signature and verify that new firmware is trusted.

2.8 Libraries

Choice of libraries is an important part of development process. Libraries can save a lot of time during development. We will start with the server part. For the server we are using MQTTnet[19] library. This library maintained by .NET developers provides support for MQTT protocol in C#. Therefore it is a straight forward choice for applications written in C# that need MQTT support.

For the device we will only need libraries to handle the camera driver and code recognition. We will use the esp-who library mentioned in section 2.4.2. This library will provide us with both camera driver and code recognition. The drivers for the display and thermometer will be written by ourselves.

The esp-who library only supports Code 39, Code 128 and QR code scanning. Another possible library for bar-code scanning in C++ is ZXing ("zebra crossing")[20]. This Library supports scanning of more formats such as UPC-A, UPC-E, EAN-13, EAN-8, Code 39, Code 128, QR Code and others. This library is very memory consuming and therefore not very suitable for embedded devices.

3 Implementation

As we already discussed the system consists of two parts: server and device. These parts communicate with each other over MQTT protocol. The users are able to interact with both the parts. They interact with the device using the display and serial console interface. With the server they interact via web application over the HTTPS protocol. We will now dig deeper into the implementation details of the system.

3.1 Server

The server is a Web Application in Blazor .NET. Most of the application logic is provided via custom services/handlers: DataController, MqttHandler and NotificationHandler. The services are the main building blocks of the whole application. Data from these services are then presented to the user via the web application. The server also offers possibility of subscribing to MQTT topics outside our system.

3.1.1 Architecture

When the user interacts with the server he is presented with the HTML front-end of the application. This front-end then based on user's actions invokes the services running in the background. The communication between the individual components can be seen in the diagram below:

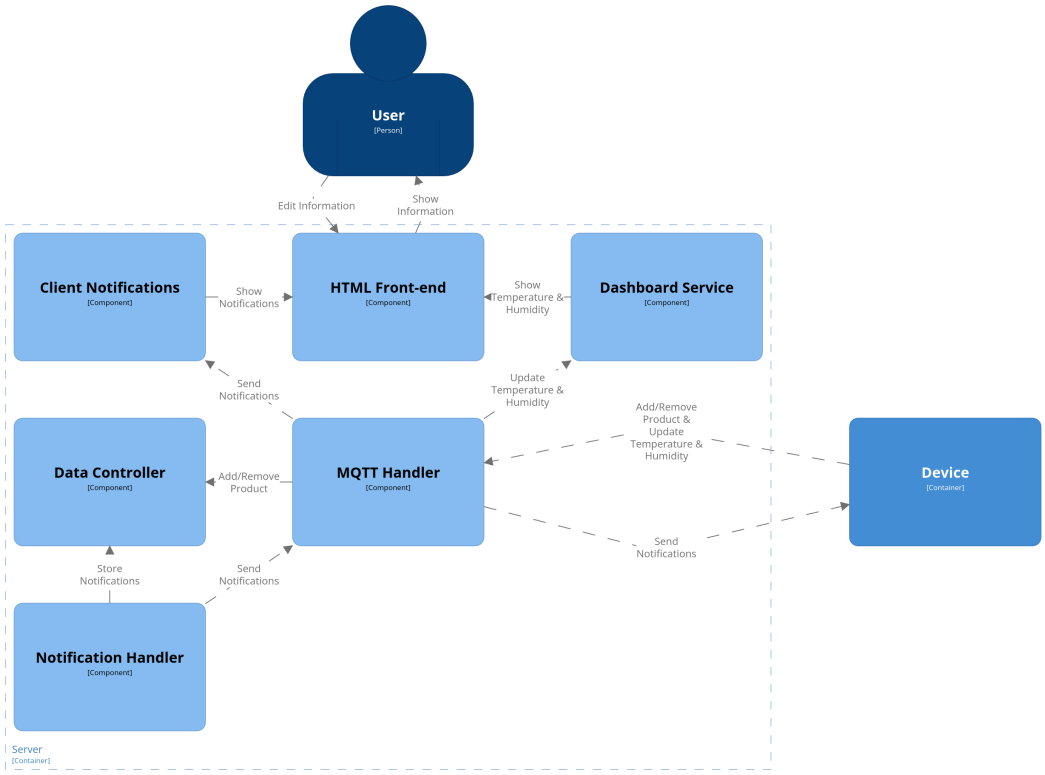

Figure 3.1 Server Architecture

The front-end interface consists of four main parts: products, notifications, recipes and dashboard. These parts are accessible from the navigation panel. The navigation panel also presents the user with current temperature and humidity in the fridge.

The dashboard presents the user with early expiring products, notifications, recipes for which there are enough products in the database and products with the amount lower than the given threshold. The products part lets user see the list of products in the database, edit them, add new or remove them. The notification part lets the user create and remove notifications (reminders). These notifications are then registered in the background notification handler. The recipes page lets the user add, edit and remove the recipes with option to link the products and their amount to the recipe.

The back-end consists of four singleton services and one scoped service. The singleton services are: Dashboard Service, Notification Handler, MQTT Handler and Data Controller. The scoped service is Client Notifications.

The Dashboard Service contains current temperature and humidity data received from the device over the MQTT. The temperature and humidity data are checked and if the values are out of the toleration, the notification is sent via the Notification Handler service. Functionality of this class can be extended to hold more information from the whole smart home system.

The Notification Handler is responsible for scheduling and triggering the notifications. It uses MQTT Handler's *SendNotif* to trigger the notifications. The notifications are persisted in the database to prevent losing them when the server application is turned off. The user has a possibility of setting the periodically repeated notifications. These periods are predefined as daily, weekly and monthly. The notifications also allow user to set priority which is then presented in the form of text color on the device display when showing the notification. There are four levels of priorities: none, low, medium and high.

MQTT Handler is responsible for communication with the device and other MQTT clients. Apart from working as standard MQTT broker, the service also checks for the authorization of the connection. The service uses this to prevent unauthorized clients from publishing to the *product* topic. Unauthorized clients are allowed to publish and subscribe to all other MQTT topics. The verification of the client happens on their connection. Verified clients are stored in hash set containing their client IDs. The client is verified if they provide the certificate issued by the same certification authority as server certificate. When the MQTT client publishes to the *product* topic, their client ID is checked and the access is allowed only when the ID is present in the hash set. When the verified client disconnects, the server removes their client ID from the hash set.

The Data Controller service is used by the other services to safely access the database. Removing product via this service ensures that when the amount of the product is less then desired, the service triggers immediate notification which is then sent via the Notification Handler.

The Client Notifications service ensures that notifications received from the MQTT Handler are presented in real time to the HTML front-end. This is achieved via the custom Blazor component which can be reused in any Blazor page. This service is a scoped service since it is used by the individual client-side pages. This ensures easy subscription mechanism to the notifications. The component

automatically subscribes to the notifications by registering itself in the MQTT Handler's collection of notification subscribers. The subscription is started on component initialization and ended on service disposal.

3.1.2 Database

Since we are using SQLite as our database engine, the communication with the database is already provided via the .NET Entity Framework. Our database structure is therefore created from our class hierarchy. The database consists of four tables: Notifications, Recipes, RecipeProducts and Products. This structure is shown in diagram below.

Figure 3.2 Database diagram

The *Notifications* table stores the information about the scheduled notifications. These notifications are loaded from the database on every start of the application.

The *Recipes* table stores the information about the recipes defined by the users. The recipes are in relation with the products via the *RecipeProduct* table which contains products and their amount needed for the recipe.

The *Products* table stores information about the individual products. This table is in relation with the *RecipeProduct* table for the reason mentioned above.

3.2 Device

We will now go through the implementation of the device. We will start with hardware components and their connection. Afterwards we will go through the architecture of the firmware.

3.2.1 Architecture

Hardware

The hardware of the device consists of three components. Main ESP-CAM board equipped with OV2660 camera, USB to UART converter and ILI9488 LCD display. For the convenience of flashing we connect one button that connects GPIO 0 pin with GND when pressed. GPIO 0 is the pin designated for switching the board to the flashing mode when connected to the ground.

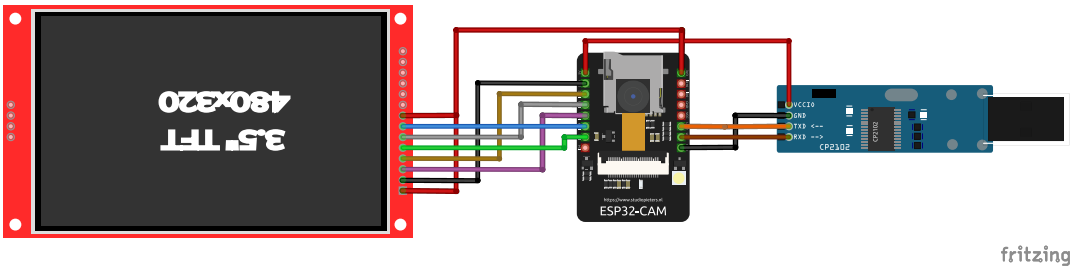

Figure 3.3 Wiring diagram

The display is connected to the HSPI bus of the ESP32. Connection of the pins are described in the table below:

Display pin	ESP-CAM pin
VCC	3.3V
GND	GND
CS	GPIO 15
RESET	GPIO 12
DC/RS	GPIO 2
MOSI	GPIO 13
SCK	GPIO 14
LED	3.3V
MISO	Not connected

The MISO pin is not connected since it is not used by the driver. The designated pin of the SPI for MISO is therefore used for the RESET pin of the display instead.

The thermometer is missing in the connection since it can't be connect it to the board due to the lack of pins. The second SPI of the display which provides touch screen communication is not connected for the same reason. Therefore we need USB to UART converter to be able to control the device.

Software

The device firmware is written in C++ operating above ESP-IDF framework. Driver for the camera is issued by ESP-IDF component esp-who. The driver for the display and thermometer is custom. The drivers are provided as C++ classes with template parameters for configuration structures. From the architectural point of view the application logic is mainly in the Console Commander class which

interconnects individual components of the device and ensures communication with other parts of the system via other components. Following diagram shows how the individual parts of the device system cooperate with other parts of the system and within the device.

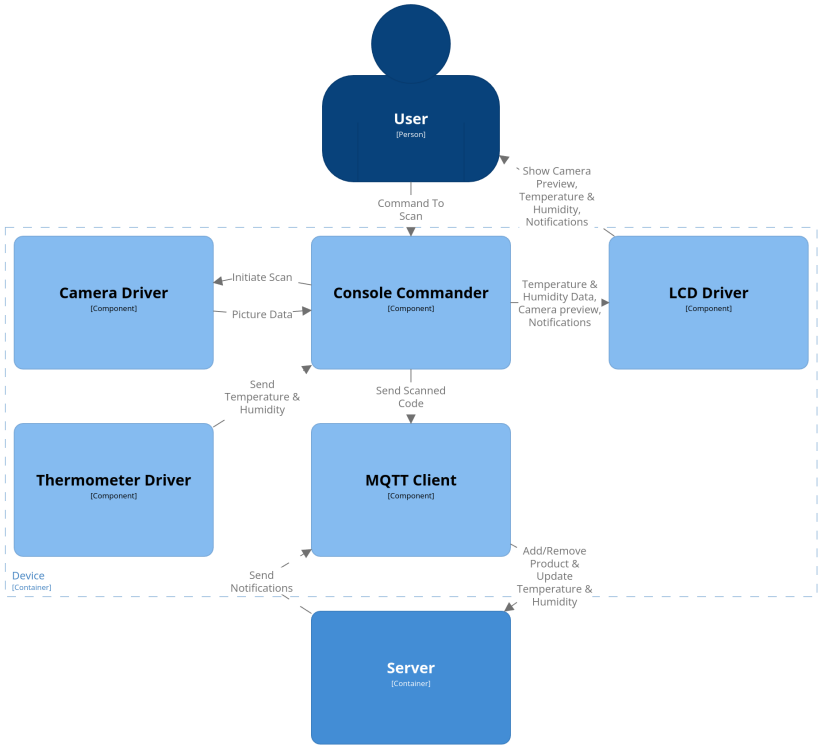

Figure 3.4 Device Architecture

As mentioned above, the core functionality of the system is provided by the Console Commander which interconnects all components. The main loop calls this component's loop function which then calls other components on demand.

The Camera Driver component is responsible for controlling the camera and code scanning. Its interface provides ability to read the frame buffer and scan for the code within the frame. The scanning of the codes uses esp-who library to get the camera frame buffer and runs the recognition for the code. This library supports recognition of Code 39, Code 128 and QR code. Even though the Code 39 and Code 128 are supported by the library, the camera is unfortunately unable to deliver the image good enough to be able to read these codes. However for big enough codes (for example 10 cm wide) the scanning works fine.

The LCD Driver component provides access to the LCD display. The interface of the driver provides functions for basic text, image and line drawing. The component holds the implementation of the SPI communication with the display together with a font and few drawing algorithms.

The Thermometer Driver component handles the communication with the thermometer device. It contains I2C communication with the device and provides the interface for getting the temperature and humidity data. The measurement is called periodically by internal timer of the device.

The driver classes are implemented in a way so that they can be reused in other systems. The interface of the driver classes is defined as abstract class

and therefore helps the system stay modular. If the developer decides to use LCD operating on I2C instead of SPI, it can be done by just inheriting and implementing the LCD base class. Other parts of the system wouldn't notice the change. This makes the system modular in sense of both software and hardware components.

The MQTT Client component is responsible for the communication with the server. This consists of three main actions: adding or removing the product, sending temperature and humidity data and receiving notifications. When the notification is received it is enqueued to FreeRTOS queue provided by the Console Commander. This queue is then checked in the Console Commander loop function and triggers the notification.

MQTT Communication

The communication between the device and the server is ensured via MQTT. The device securely connects via MQTT over TLS. Server than verifies that the device has proper certificate. Device also verifies server certificate. These certificates must be issued by the same certification authority.

Now let's introduce the topics that are used by the system.

MQTT Topic	Payload	Purpose
notifications/0	<notification text>	Notifications with no priority
notifications/1	<notification text>	Notifications with low priority
notifications/2	<notification text>	Notifications with medium priority
notifications/3	<notification text>	Notifications with high priority
product/add	<bar-code of the product>	Increases count of product in DB
product/rm	<bar-code of the product>	Decreases count of product in DB
temperature	<temperature data>	Temperature measurement
humidity	<humidity data>	Humidity measurement

3.3 Issues

When developing computer software we can expect some issues or bugs. This expectation is even higher when prototyping or working with devices that we interconnect ourselves. Work environment is often not perfect and these issues have many sources, from loose wires to small details such as not enabling connection to the port from outside. This thesis is no exception and therefore we will go through some issues one may face during development.

3.3.1 Device

Since we are working with hardware which has tens of pins that we can connect something to, we may often find ourselves exchanging two pins that are next to each other or have some loose wire that works only in a given position. These issues can be minimized by double-checking or triple-checking the connection. The loose connections can be prevented by statically fixing the wires. For example, using electrical tape.

On the other hand, mistakes in the software can also result in long debugging sessions. For example forgetting to fully initialize a structure that holds our SPI configuration.

Another issue that we can face is debugging the firmware of the device. Since the ESP-CAM board has only limited amount of pins accessible, we can't connect J-Tag debugger. That makes the debugging much harder and more time consuming.

3.3.2 Server

The development of the server can be much smoother since we have a debugger. However using libraries may often be complicated due to unclear documentation or missing examples. This is for example a case for MQTTnet's Github Wiki page where the code examples are for older versions but not the latest one.

4 Discussion

This chapter consists of two parts. In the first part we will go through the evaluation of the work. In the second part we will go through the possible future work and extension of this project.

4.1 Evaluation

This thesis provides the reader with the overview of some technologies used in the world of IoT and smart home. The analysis of the technologies was done in chapter 2.

The device provides the user with the ability to scan the bar-codes or QR codes of the products but only the ones that are of bigger size. The bar-code types recognized by the software are Code 39 and Code 128. Usage of a different library would provide more support. We tried using the ZXing C++ library but unfortunately the memory overhead was too big and device was running out of memory. Apart from scanning, the device shows the user notifications that were received from the server. The device is able to communicate with the server in both local and global network. This communication is encrypted. The device and server verify each other based on certificates signed by the common certification authority. The server provides the user with the ability to add, remove and edit the entries in the database and create periodic notifications.

The firmware of the device is driven by the main component which then calls components for the peripherals and internet communication. The device accepts commands to initiate scanning over the UART communication. The firmware is written in C++ with ESP-IDF libraries.

The main part of server application logic is implemented as services that provide functionality for individual parts of the system such as database access, network communication and notification handling. The server is written as Blazor Web Application in C#. The server is capable of serving as a presentation platform or gateway for other devices that the user may have at home.

The system has some drawbacks. The web application does not react to the real-time events such as updating the page when new product is added by the device. On the other hand, the notifications received via the MQTT are displayed in real time. The scanning of new products is not very intuitive since user can only select add or remove the product based on the bar-code but can't edit the name or expiry date of the product. The preview of the camera on the device may not seem smooth. This issue is associated with the fact that the camera and display use different color encoding. This could be improved by using more powerful ESP32 chip such as ESP32-S2 which is equipped with vectorized instruction set. The quality of the scanning could be enhanced by using camera with less noise which would improve the ability to scan smaller bar-codes.

The system's overall usability is affected by the lack of needed pins for the display's touch screen. This could be potentially solved by using GPIO multiplexing or using another board. Even though the device lacks the real world use, the server can still be used as a standalone system or potential start point for enrolling more smart devices. Overall the market of the smart inventory tracking

systems for home use is not wide and this system could potentially ease the access to such functionality for many users or developers.

4.2 Future Work

The system may serve as a starting point for the whole smart home system. In this system the users have one common platform for all their smart devices. The system could be extended for support of more protocols and more devices. The drivers of the device peripherals can be reused on other projects. The device firmware could also be extended with OCR (Optical Character Recognition) mechanism that would scan the expiry date of the product which could be published in the MQTT message together with the bar-code. This couldn't be accomplished due to the issue with cross-compiling the OpenCV[21] library to the ESP-IDF toolchain[22].

Conclusion

The significant issue of food waste due to overlooked expiry served as a powerful motivation for the development of the smart fridge tracking system proposed in this thesis. By actively managing expiry dates and alerting users to forgotten items, this system aims to help the user utilize the groceries instead of wasting them.

Server part of the thesis can serve the users as a database of home inventory. The application lets the users add, remove and edit products, notifications and recipes. The recipes can be linked to corresponding products with the given amount. Server automatically sends notification to the device if products are less than a threshold or if the temperature and humidity are out of given range. These notifications are also displayed in the web application in real time.

The device can scan QR codes of bigger size and send them to the server as the products to add or remove. Communication is secured via TLS. Device shows the notification received by the server and update's the server with virtual temperature and humidity.

The systems of this type are not widely available for home use. Therefore this system is an accessible and affordable solution for home use. The system offers cost-efficient open-source solution which can be easily extended. Even though the system is mainly designed for home use, it could be modified to fit in the professional environment.

Bibliography

1. Samsung. *Samsung Family Hub*. 2024. https://www.samsung.com/us/explore/family-hub-refrigerator/overview/ [Accessed: 23.6.2024].
2. Inc., Hisense. *Hisense RQ759N4IBU1*. 2024. https://www.shophisense.com/product/refrigerator/smart-refrigerator-rq759n4ibu1-black-color-with-touch-screen-water-dispenserice-maker/ [Accessed: 23.6.2024].
3. Pi, Raspberry. *Raspberry Pi*. 2024. https://www.raspberrypi.com/ [Accessed: 22.6.2024].
4. Arduino. *Arduino*. 2024. https://www.arduino.cc/ [Accessed: 22.6.2024].
5. Systems, Espressif. *Espressif*. 2024. https://www.espressif.com/ [Accessed: 22.6.2024].
6. STMicroelectronics. *STMicroelectronics*. 2024. https://www.st.com/ [Accessed: 22.6.2024].
7. Espressif Systems (Shanghai) Co., Ltd. *ESP32 Series Datasheet v4.5*. 2024. Available at https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf [Accessed: 10.6.2024].
8. Technology, Ai-Thinker. *ESP32-CAM camera development board*. 2024. https://docs.ai-thinker.com/en/esp32-cam [Accessed 12.6.2024].
9. CORP., ILI TECHNOLOGY. *ILI9488 Datasheet*. 2012. Available at https://www.hpinfotech.ro/ILI9488.pdf [Accessed: 5.6.2024].
10. Guangzhou Aosong Electronics Co., Ltd. *AHT10 Datasheet*. 2021. Available at https://cdn.sparkfun.com/assets/8/a/1/5/0/DHT20.pdf [Accessed: 7.6.2024].
11. Microsoft. *Blazor*. 2024. https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor [Accessed: 15.6.2024].
12. Espressif Systems (Shanghai) Co., Ltd. *ESP-IDF Programming Guide*. 2024. https://docs.espressif.com/projects/esp-idf/en/v5.2.2/esp32/index.html [Accessed 12.6.2024].
13. Limited, George Robotics. *Micropython*. 2024. https://micropython.org/ [Accessed: 27.6.2024].
14. *.NET nanoframework*. 2024. https://www.nanoframework.net/ [Accessed: 27.6.2024].
15. Ltd., Espressif Systems (Shanghai) Co. *ESP-WHO Github*. 2024. https://github.com/espressif/esp-who [Accessed: 13.6.2024].
16. Uss, Ruslan V. *esp-idf-lib Github*. 2024. https://github.com/UncleRus/esp-idf-lib [Accessed: 27.6.2024].
17. *SQLite*. 2024. https://www.sqlite.org/ [Accessed: 15.6.2024].
18. MQTT.org. *MQTT*. 2024. https://mqtt.org/ [Accessed: 15.6.2024].
19. dotnet. *MQTTnet*. 2024. https://github.com/dotnet/MQTTnet [Accessed: 13.6.2024].

20. *zxing-cpp Github*. 2024. https://github.com/zxing-cpp/zxing-cpp [Accessed: 28.6.2024].

21. *OpenCV*. 2024. https://opencv.org/ [Accessed 22.6.2024].

22. *OpenCV Github Issue*. 2024. https://github.com/opencv/opencv/issues/22949 [Accessed 22.6.2024].

A Attachments

A.1 Source code

Gitlab repository of the project: Repository

B User Documentation

This section will guide the user through the deployment and usage of the system.

B.1 Deployment

B.1.1 Certificate generation

In order to have MQTT working over TLS we need to create the certificates first. We will generate them using the *openssl* utility.

1. Generate key and certificate for the certification authority: *openssl req -x509 -newkey rsa:4096 -keyout ca.key -out ca.crt -nodes -sha256 -days 60*

   Remember to fill country, company, and common name.

2. Create configuration file *san.cnf* for the certificate request (example can be found in the repository of the project)

3. Generate request for the server certificate: *openssl req -new -nodes -out server.csr -newkey rsa:4096 -keyout server.key -config san.cnf*

4. Sign the certificate by CA: *openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 60 -sha256 -extensions req_ext -extfile san.cnf*

5. Convert certificate to format accepted by C#: *openssl pkcs12 -export -out server.pfx -inkey server.key -in server.crt*

6. Generate key and certificate request for the device: *openssl req -new -nodes -out device.csr -newkey rsa:2048 -keyout device.key*

7. Sign the device certificate by CA: *openssl x509 -req -in device.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out device.crt -days 60 -sha256*

B.1.2 Server

Prerequisites

1. Raspberry Pi or computer running Linux

2. Install .NET SDK 8: `https://dotnet.microsoft.com/en-us/download/dotnet/8.0` - note that for arm64 it can't be installed via package manager (This is not needed if compiling with `--sc`)

Compilation

1. Clone the code to the server:

   git clone ***https://gitlab.mff.cuni.cz/hrivnad/smart-fridge.git***

2. Navigate to the *server/Server* directory
3. Run

   dotnet build -o bin -c Release

   In case of Raspberry Pi you can build the code on the personal computer using (This will ship the .NET Runtime with the application):

   dotnet build -c Release -o bin --sc -a arm64 --os linux
4. Move *ca.crt* and *server.pfx* to the *bin* directory
5. Create database:

   dotnet ef migrations add Init

   dotnet database update

Then run the server using *./Server*

B.1.3 Device

Prerequisites

1. Install ESP-IDF toolchain and add *idf.py* to your system path.

1. Navigate to the code directory
2. Copy your certificate files (*ca.crt*, *device.key*, *device.crt*) to the *main* directory
3. Build the project: *idf.py build*
4. Flash the project on the device
5. After boot the device should automatically connect to the WiFi and server (SSID, password and MQTT URL has to be defined in *main.cpp*)

B.2 Usage

B.2.1 Server

To access the server web application, connect to the server via the web browser. For navigating to individual pages use the navigation menu on the left. Products, notifications and recipes show list of the entries in the database. These entries can be edited using the edit button.

Web application preview:

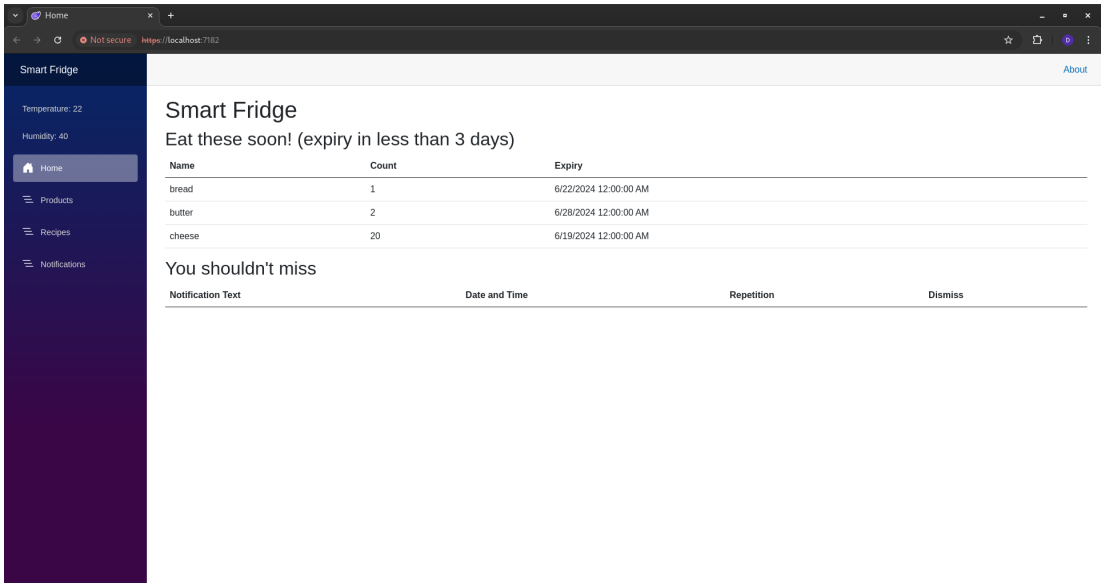

Figure B.1 Web application Dashboard

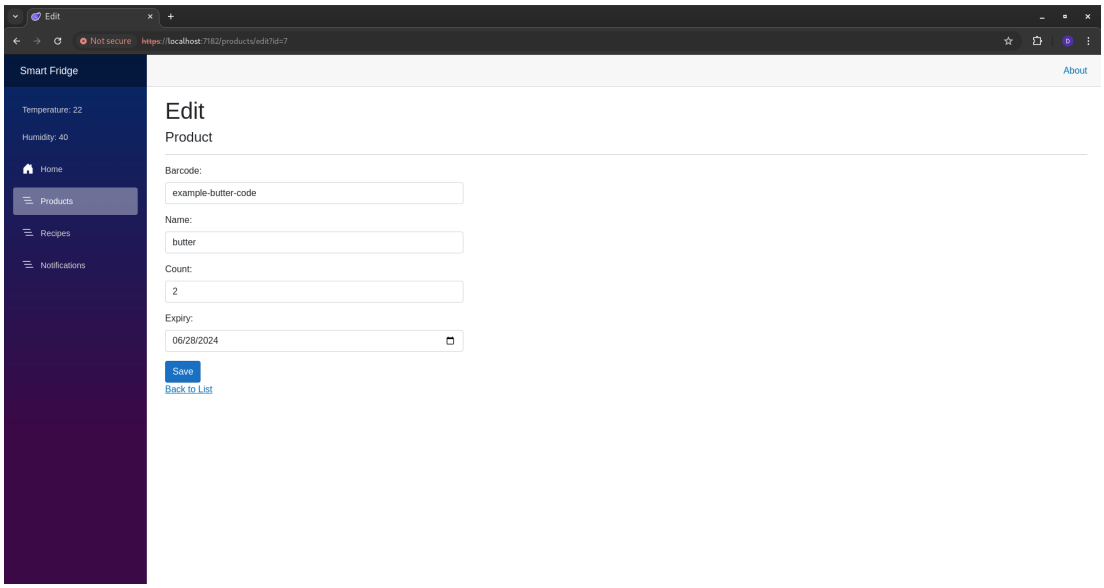

Figure B.2 Web application Product Edit

The walk-through videos on basic application usage are available through the link.

B.2.2 Device

The device is operated via the console interface. To access the console interface connect the device to the computer and to the appropriate serial interface. Select baudrate *115200*. After successful connection you can enter the following commands:

1. ***add product*** - starts the scan of the camera. Point the camera at the code to scan it. The scanned code will then be automatically sent to the server as a product to add to the database.

2. ***rm product*** - starts the scan of the camera. Point the camera at the code to scan it. The scanned code will then be automatically sent to the server as a product to remove/decrease the number of products.

3. ***stop scan*** - stops the scanning of the codes