App controlled step for FlexCAR

Niklas Schildhauer
Hochschule der Medien
Stuttgart
Stuttgart, Germany
ns162@hdm-stuttgart.de

Prof. Dr. Ansgar Gerlicher
Hochschule der Medien
Stuttgart
Stuttgart, Germany
gerlicher@hdm-stuttgart.de

ABSTRACT
For the FlexCAR project, two steps are being developed that can be retracted and extended using an iOS app. For safety reasons, there is a requirement that the user can only extend the step facing him. In this paper, technologies were investigated that can be used to localize objects. For this reason, a concept has been developed that uses the technology of Bluetooth beacons and NFC tags to ensure the immediate proximity to the vehicle and to determine the facing side of the vehicle. A prototype has been developed which uses the Bluetooth Beacons to measure the distance to the vehicle. However, there is the problem of the technology being sensitive to interference, so the NFC tags are used as a backup strategy.

Author Keywords
iOS; Swift; MQTT; iBeacon; NFC; Raspberry Pi; IoT

1 INTRODUCTION

FlexCAR is a research project in Arena2036 that aims to develop an open vehicle platform for the mobility of the future. Part of the project is the Rolling Chassis, an electrically driven and fully X-by-Wire controlled research platform with accessible and open software and hardware. This will allow third parties to develop additional components for the platform [20].

In cooperation with the University of Applied Sciences in Esslingen, two extendable steps are being developed, which will be mounted on the two long sides of the rolling chassis. These will simulate the opening and closing of the vehicle and make it easier to enter the vehicle at the same time.

To control the steps, we are developing an iOS app that allows authorized users standing directly in front of a step to extend and retract it via a secure connection. Taking car sharing as an example, users should be able to rent a vehicle via a platform and afterwards open the vehicle via an app. This raises the question of how to ensure that the user is standing directly in front of the vehicle. For safety reasons, a car-sharing user should only be able to open the vehicle when standing directly in front of the vehicle and should only be able to extend the side of the step facing him or her. This paper examines which technologies are suitable for this purpose and develops a concept based on this for iOS Devices.

2 RELATED WORK

2.1 Established unlocking concepts

2.1.1 Share now

This car-sharing provider uses a 3-digit pin to unlock a booked vehicle. To do this, the user must enter the PIN in the app, which is shown on a display on the vehicle [13]. This ensures that the user is standing directly in front of the vehicle. To use the app, the user must first register with the platform

2.1.2 Stadtmobil

With this concept, the user receives a physical access card that can be used to open booked vehicles. With this concept, no app is required to unlock the vehicle. To rent a vehicle, the user must first register with the platform and present all the necessary documents [1].

2.2 Device capabilities to locate an object

2.2.1 Bluetooth Low Energy (BLE)

This wireless technology transmits data in the 2.4 GHz ISM frequency band and is more energy-efficient than Bluetooth Classic. It also offers the option to locate devices [14]. Interesting here are the proximity solutions described by Marcel [21]. This allows two devices to determine whether they are within range of each other. In addition, the relative distance to each other can be determined by measuring the Received signal strength (RSSI).

Bluetooth Beacons are based on this solution. They use BLE to send/broadcast advertising packets, which can be received by any BLE device in the area. This is how their position can be determined by RSSI. Bluetooth Beacons do not receive any data themselves. There are two well-known standards that use the technologies: iBeacon (Apple) and Eddystone (Google) [23]. Beacons that work with Apple's iBeacon standard can be found in iOS apps through the Core Location framework [7].

2.2.2 Near Field Communication (NFC)

With the help of NFC, it is possible to establish communication between two devices at a very short distance (approx. 10cm). NFC is based on Radio Frequency Identification (RFID) technology. It can be used for various purposes, such as making transactions, exchanging digital content, or establishing a connection between two devices. Through NFC, smartphones can

be used to make contactless payments nowadays. The 13.56 MHz frequency is used [3].

For this use case NFC tags are interesting. These work offline and store data in the NFC Data Exchange Format (NDEF) [22]. With the help of the Core NFC framework, provided by Apple, it is possible to attach small tags to objects and then read them with an iOS Device [8].

2.2.3 Camera

In 2.1.1, it is described that a PIN is used to ensure proximity to the vehicle. The principle is that the user has to be within short sight of the vehicle to be able to read the PIN. The same principle is applied when a QR code (Quick Response Code) is shown on the display instead of a PIN and the user scans this code using the camera. A QR code is a two-dimentional matrix code which, unlike 1D barcodes, is able to store a larger amount of data and can be decoded very quickly [26]. A QR code can be scanned using the AV Foundation Framework from Apple [6].

2.3 Rolling Chassis Control

The rolling chassis, for which the step is being developed, can already be controlled remotely. For this purpose, an iOS app was developed at the Hochschule der Medien (HdM) in WS20/21 [4]. This application was developed using the messaging protocol MQTT (Message Queuing Telemetry Transport). Between IoT devices, MQTT is widely used. It is lightweight and is often used for communication between non-compute intensive sensors. It communicates over a TCP connection. MQTT is a publish/subscribe push protocol, which consist of one MQTT broker and several MQTT clients. All messages are distributed via the broker. As a client, it is possible to subscribe to so-called topics at the broker and thus receive messages from other clients that publish a message on this topic [24].

3 CONCEPT

For the creation of the app concept, it was specified in advance that communication between the vehicle and the app should take place using the MQTT message protocol, as was already the case with the remote control app (see 2.3). Based on this requirement, the communication architecture described in 3.1 was created. In 3.3, it is explained how the technologies presented in 2.2 can be used to localize a vehicle with an iOS device.

3.1 Communication between car and iOS-Device

In Figure 1, the communication flow can be seen. The iOS device on which runs the app is shown at the top left. This app acts as the end user's MQTT client. At the bottom left, a Raspberry Pi is shown, which on the one hand acts as MQTT broker (thus representing the car) and on the other hand also as MQTT client for the Step-engine control. In this diagram the Python script for the Step-engine control is on the same Raspberry Pi as the MQTT broker. However, this does not have to be the case for the later implementation.

The diagram also shows the messages that are sent and received by the clients. There are a total of two topics per step. For this reason, the topic name is structured as follows:

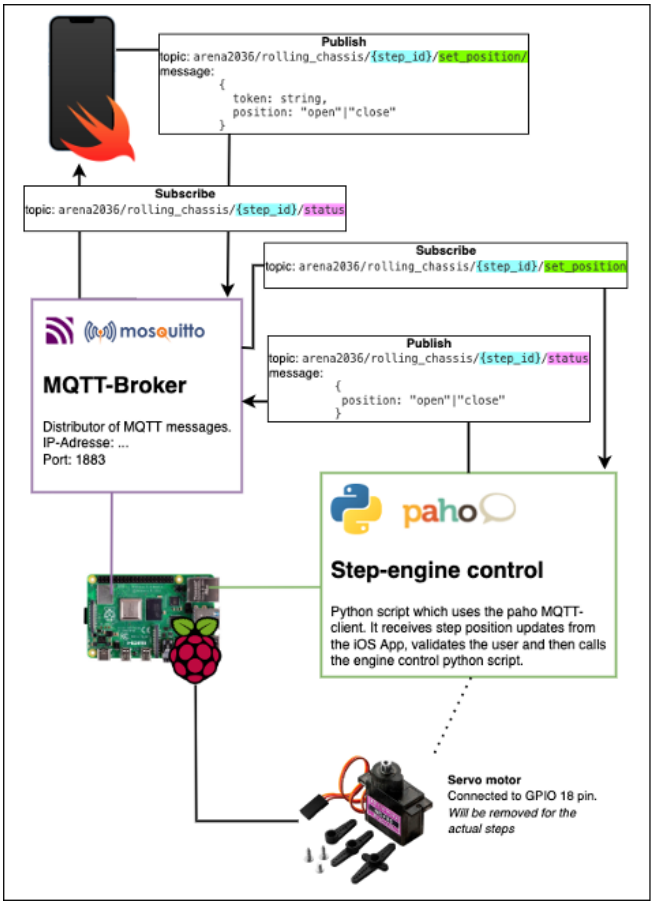

Figure 1. MQTT-Communication

"arena2036/rolling_chassis/" + step identification + "status"/ "set_position". The app publishes on the topic "set_position" to move a step in or out. The step-engine control in turn publishes on "status" the current position, so that the end user can see the current position of the respective step in the app.

The connection between the car and the iOS device should be encrypted. Since MQTT uses TCP as transport layer, TLS can be used for encryption. The article by HiveMQ [17] shows the advantages and disadvantages of a TLS-encrypted MQTT connection. A certificate authority, Mosquitto broker certificate and client certificate are required to use TLS.

3.2 User authentication and backend communication

As with almost all Internet services, the user must authenticate to use the services, see 2.3. For a car-sharing platform, it is important so that it can be verified that the user is actually authorized to open the vehicle. For this reason, a user token must be included in the publish message of „set_position" in addition to the new position.

An additional instance (backend) is required to generate the user token and to authorize users to access specific vehicles. This backend is responsible for car rentals and user management. Since the backend development is not part of the concept, it is only shown here as a black box in Figure 2. In it, you can see that the iOS app first has to log in to the backend

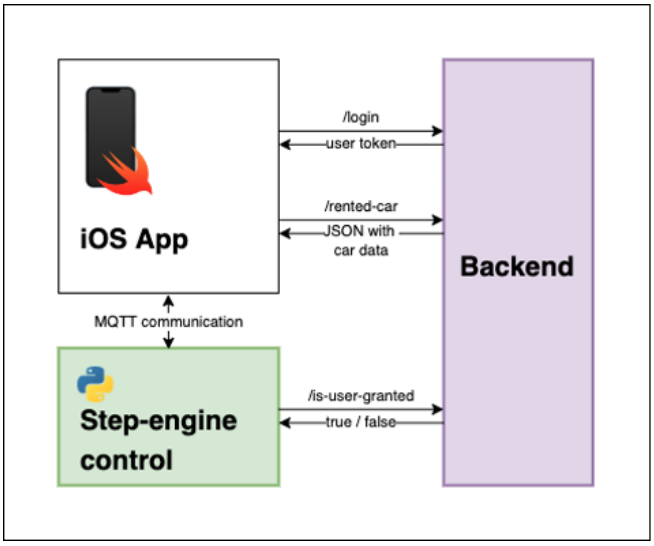

Figure 2. Backend Communication

to get a user token. This user token is later used by the step engine control to verify that the user is authorized to open the step.

In addition to user authentication, another task of the backend is to provide the app with all relevant information about the rented vehicle. Essential information includes the number and identification of the vehicle's steps, as well as the vehicle's IP address and port number, so that an MQTT connection to the vehicle can be established.

3.3 Locate the car

Based on the information collected in 2.2, concepts for localizing a vehicle presented in this section were created. All three are suitable for a reliable and accurate localization of the vehicle. Another technology not listed is GPS (Global Positioning System) [9]. This is not an option because the system is too imprecise for the application. It is not possible to localize the facing side.

Figure 3 shows in simplified form how the car and the facing side can be localized. Each side or step of the vehicle broadcasts the UUID (Universally Unique Identifier) of the vehicle and the identifier of the respective step. This is received by the app, allowing the facing step and thus the associated side to be identified. The app has previously received the UUID of the vehicle and the identifiers of the steps from the backend (see 3.2).

3.3.1 Bluetooth Beacon

A Bluetooth Beacon according to the iBeacon standard always sends out a UUID and a Major and Minor value. Major and minor have a capacity of 2 bytes each and can be used to distinguish between beacons that send the same UUID [5]. For this use case, at least two Bluetooth beacons are required, at least one per step. These send the UUID of the vehicle and via the major value the identifier of the respective step. If possible, the beacons should be attached to the outside of the step and should not be obscured by other components, because

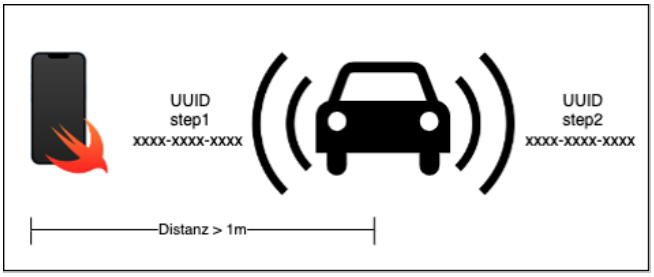

Figure 3. Car localization

physical objects such as metal or people can interfere with the signal and thus make localization impossible [5]. On the one hand, this can have a negative impact on reliability, but on the other hand, it can also be used to shield the signal of multiple treads from each other, for example by placing a metal plate in between them. Through Apple's Core Location Framework, the app can locate the Bluetooth beacons and calculate the relative position. For this, the signal strength is divided into the categories: immediate, near, far and unknown [5].

+ No user interaction necessary. The user only has to open the app, which then calculates the relative position itself.
+ Through relative position, the user can be informed of how far they are from the vehicle and receive feedback.
- The signal can be disturbed and thus the position might not be determined. A backup method is necessary because the reliability can be negatively affected by environmental influences.

3.3.2 NFC-Tag

An NFC tag can be written with eight different data types of the NDEF [18]. Data type 2 Well-known is suitable for this use case. Thus, the UUID and the step identifier can be stored on the NFC tag in the form of text. As described in 2.2.2, the NFC tags work offline and thus can be attached to objects in any way. In this case, one tag would be attached to each step. Through Apple's Core NFC framework, the app can use the iOS device's built-in NFC chip to identify the step. Since reading only works at a short distance, the user is immediately in front of the step when reading.

+ Reliable function. Reading with an iOS device works very well.
- Writing to a tag is complex. The easiest way is to use a separate app.
- Whether the user understands the use of NFC tags is unclear and should be investigated.
- No information about the relative distance to the vehicle.

3.3.3 QR-Code

As described in 2.2.3, a QR code can store large amounts of data and can be read quickly by the camera of the iOS device. One possibility would be to create QR codes that encode the respective UUID of the vehicle and the step identification, print them out and then attach them to the side of the step so that they are clearly visible. The app then only needs to

implement a barcode scanner that scans the QR code and can then check whether the user is directly in front of a step or not.

+ Simple implementation.
+ Reliable function. Scanning a QR code with an iOS device works very well.
+ QR codes are widely used and frequently found in daily life.
- No information about the relative distance to the vehicle.

Based on the information collected, Bluetooth beacons offer the most added value for this use case due to the additional information about the relative distance. Unfortunately, this is not possible with the other technologies. However, with Bluetooth beacons, there is a risk that the signal will be interfered with and thus an opening of the vehicle will not be possible. This must be avoided, which is why a combination of two technologies is a reasonable compromise. The two technologies from 3.3.2 and 3.3.3 are very similar in their handling. In both, the localization works by the user standing close enough to the vehicle and reading either the NFC tag or the QR code with his device. This means that the two are interchangeable with each other.

4 PROTOTYPE

This section describes the implementation of the prototype. The hardware components used and their function are described in 4.1. Afterwards, the implementation of the MQTT connection (4.2), the step-engine control (4.3) and the iOS app (4.4) are described.

4.1 Hardware Components

Figure 4 shows the hardware components used for the prototype. The step hardware consists of a Raspberry Pi and a servo motor. The Raspberry Pi runs the step engine control (see 4.3) and the MQTT broker (see 4.2.1). A servo motor with low power was chosen, so that no additional power supply is required for the servo motor. The Raspberry Pi is able to supply enough power. The iOS app runs on the user end device, through which the interaction with the user takes place (see 4.4). The user end device has to be in the same WLAN network as the Raspberry Pi to be able to establish an MQTT connection. Another iPhone is used as localization hardware, which only serves as an iBeacon. But it is also possible to use other hardware, like the Bluetooth module of the Raspberry Pi, which can also be an iBeacon. NFC tags are used as a backup strategy for locating a vehicle.

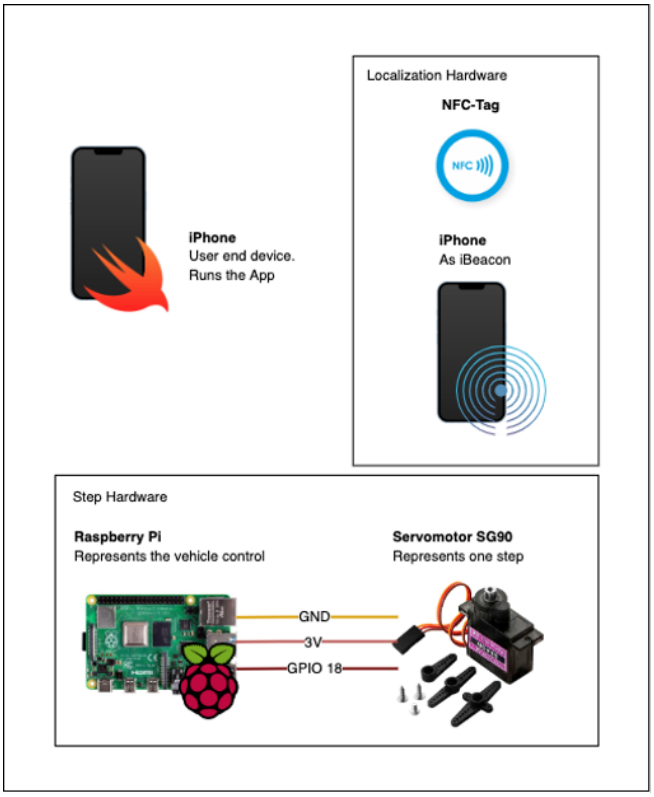

Figure 4. Hardware Components

4.2 MQTT Connection

4.2.1 MQTT Broker

Mosquitto from Eclipse [11] is used as the MQTT broker, which is operated on the Raspberry Pi. MQTT versions 5.0, 3.1.1 and 3.1 are supported. MQTT version 3.1.1 is used for communication between step-engine control and the iOS app.

4.2.2 MQTT Topics

As already described in 3.1, there are two topics for each step: "status" and "set_position". The full topic string is: "arena2036/rolling_chassis/" + step identification + "status" / "set_position". The iOS app publishes messages on "set_position" and subscribes messages from "status". For the step-engine control, the opposite is the case, which allows establishing a bidirectional communication via MQTT.

4.2.3 MQTT Messages

The message contents are already shown in Figure 1. The following JSON is sent on the topic "status" to inform the app whether the step is currently retracted or extended.

```
{
    position: "open"|"close"
}
```

The app sets a new position by sending the following JSON. It is important that the app sends the token of the user so that the step-engine control can check if the user is authorized.

```
{
    token: string
    position: "open"|"close"
}
```

4.2.4 MQTT Security

For the prototype, a TLS encrypted connection was not used and instead the "set_position" message payload was encrypted with a key agreement algorithm. Only the messages of this topic were encrypted, since the messages of the "status" topic do not contain any sensitive data. In a non-prototypical environment, TLS encryption should be used.

The key agreement algorithm is based on two parties having a

public and a private key. Both exchange their public keys for encryption. Using the public key of the other party and their own private key, a common symmetric key can be calculated, which can be used for the de- and encryption of the message [25]. The encryption was implemented in the following way. A key pair was created on the Raspberry Pi, which represents the vehicle. The public key is communicated to the iOS app along with the other information about the vehicle. The iOS app then creates a new key pair itself for each new message, which it uses to first create the shared key to encrypt the message and then sends the encrypted message with its own public key so that the Raspberry Pi can decrypt the message again.

4.3 Step-engine control

The step-engine control is written in Python and uses version 3.9. The main components are two scripts: on the one hand „MQTTClient.py“, which uses the MQTT client Paho [12] and the script „EngineControlSG90.py“, which controls the servo motor. The MQTT client connects to the MQTT broker and subscribes to the topic "set_position". When a message is received, the message is first decrypted, the user token is checked and if both were successful, the „EngineControlSG90.py“ script is called to set a new position. After that, a new status message is sent. The status message is sent with the QoS (Quality of Service) flag 1. This means that the message should be delivered at least once to all subscribers [15]. So that the Step Engine clients do not have to keep sending the current state of the step when a new iOS device connects, the messages were first sent with the retained flag activated [16]. Normally, the broker saves the retained messages and sends the last saved one to new logged in clients. Unfortunately, this is quite error-prone, which is why another variant was used. As soon as a new iOS client is logged in, it sends a “set_position" message with the position "unknown" for all steps. Each step-engine control then sends back the current status.

4.4 iOS-App

The iOS app is programmed with Swift 5.5 for iOS version 15.2. Apple’s UI Kit framework was used to develop the UI. The Coordinator pattern [19] and the Model View Presenter pattern [2], among others, were used to develop the app. In this way, the maintenance effort should be kept as low as possible and the app should be made flexibly extendable. The following section describes the individual views of the app and explains how they work.

4.4.1 Configuration View

In Figure 5, the Configuration View is shown on the left side. At the first start, this view is displayed to give the user the possibility to change the preselected configuration. This is necessary because currently no backend can provide the required data. This includes the IP address and port of the Raspberry Pi to connect to the MQTT broker, the public key for the step-engine control and the UUID which is needed for localization in 3.3. As an alternative to entering the information manually, a QR code with the information can also be scanned.

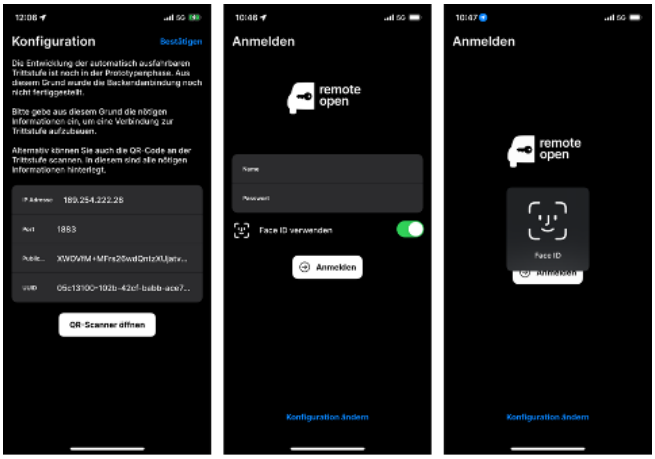

Figure 5. Configuration View (left), Login View (middle) and Face ID (right)

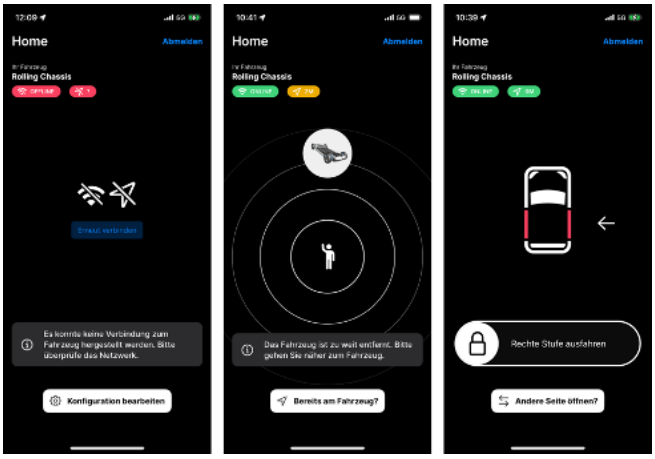

Figure 6. Home Views

4.4.2 Login View

Next to it on the right, you can see the login view in Figure 5. Before the app can be used, the user must first authorize himself. After entering the login data for the first time, the user can then unlock the app using Face ID or Touch ID. Like the data in 4.2.1, the authorized users should also be managed by the backend. For the prototype they are also preconfigured.

4.4.3 Home View

Figure 6 shows the Home View and its changes depending on the vehicle status. On the far left, you can see that no connection to the vehicle could be established. The reason for this could be that the IP address is incorrect or the Raspberry Pi is connected to a different Wi-Fi network. The user then has the option to adjust the configuration or to try to establish a connection again. With the connection to the car is meant the connection to the MQTT broker, which is established with the help of the CocoaMQTT framework from Emqx [10].

In Figure 6 a connection to the vehicle has been established, in the middle screenshot. The user is in the process of locating the vehicle. Here, it is ensured that the user is directly in front of the vehicle before he can move the step in or out. The

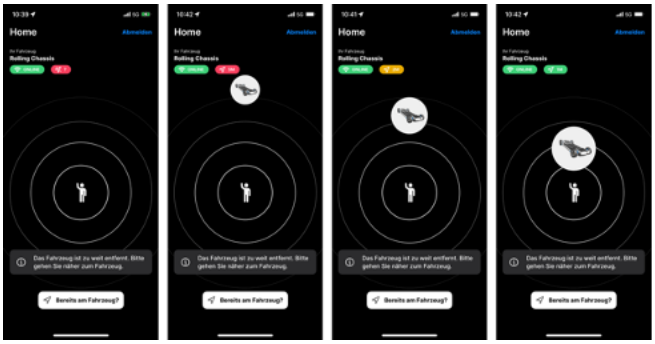

Figure 7. Localization status

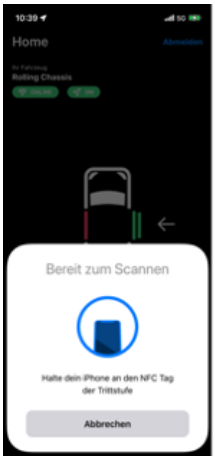

Figure 8. NFC Scan

concept from 3.3.1 is implemented for this purpose. Using Apple's Core Location Framework [7], Bluetooth beacons are searched around, and their relative position is determined. Based on this position, the distance to the vehicle is displayed in the view. For this purpose, a tag with a navigation icon is displayed in the upper area and beside it, the current distance in meters is displayed. In the middle of the view is an icon of a person, which represents the user and the vehicle, which is searched for. The vehicle gets larger and closer to the user the closer the user is to the vehicle. Figure 7 shows the four possible states. As soon as the vehicle is directly in front of you (described as immediate in 3.3.1), the view changes to the screenshot on the far right in Figure 6.

As described in 3.3.1, a backup strategy was implemented. For this purpose, the NFC tag technology described in 3.3.2 was used. Clicking on the button "Already on the vehicle?" opens the NFC scanner from Apple (see Figure 8). This is provided by the Core NFC Framework [8].

In Figure 6, the view that is displayed when the vehicle has been localized and the user is directly in front of the vehicle is shown on the right. It can be seen that the localized step can now be extended by activating the slider. In Figure 9 the two states (extended and shrinked) are shown. The localized step is indicated by the arrow. The MQTT topics and messages (described in 4.2) are used to retrieve the current status (topic: "status") of the steps and to extend or retract the selected step

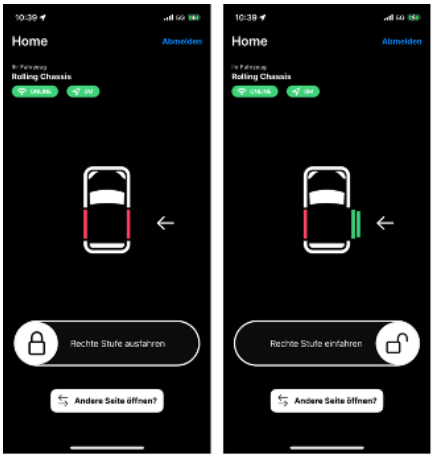

Figure 9. MQTT-Communication

via the topic "set_position". For this purpose, a JSON object with user token and new position is created, then encrypted and sent to the "set_position" topic of the selected step.

5 DISCUSSION

This paper investigated how to ensure that a user is directly in front of a specific vehicle. Based on the technical conditions of an iOS device, three different possibilities were worked out and a concept for this was developed, which was then implemented in a prototype. As already described in 3.3.1, the Bluetooth signal could be interfered with. This was confirmed during testing of the prototype. This makes a backup strategy, as already implemented, mandatory. As long as there is a clear view towards the beacon, the localization works reliably. There are no problems with the back-up strategy of the NFC tags. Reading works reliably with the iPhone.

However, all three localization paths have the common characteristic that the UUID that is transmitted is static in all cases. This should be taken into consideration, since any device that supports the technology can read the data. For security reasons, UUIDs should change at a regular time interval. It is true that a third party who reads the data cannot open the car if he is not authorized from the backend to open it. But it would be possible for an authorized user to copy the UUIDs and broadcasted them to any other position, which would allow them to extend the step even if they were not standing directly in front of the car. This should be considered in a possible future implementation of the concept, but is not considered in the implementation in 4. In the case of 3.3.3, instead of printed QR codes, a display could be attached that has an additional control unit. Similarly, in 3.3.1 and 3.3.2, the Bluetooth beacon and NFC tag could be connected also to an additional control unit that dynamically exchanges the UUIDs.

As already described in 3.1, another problem with the implementation is that encryption of the MQTT connection by TLS is missing. This means that currently anyone can read the messages, subscribe to the topics and also can publish messages. This poses many security risks, such as Man in the Middle attacks [16]. For example, an opening message could be listened to and sent repeatedly later.

6 FURTHER WORK

In the further development of the remote open app, the next step is to conduct usability tests and try out the concept described here. It is interesting to find out whether the user would prefer one of the three types (localization by Bluetooth, by NFC tags or QR codes), or which of the three is most intuitive.

GPS was not considered for this use case because it is too inaccurate and alone is not sufficient to be sure that the user is directly in front of the vehicle, nor is it possible to determine the facing side of the vehicle. Since a backup concept is also needed in the presented concept, and the Bluetooth beacon technology is sensitive to interference, another experiment could be to use GPS to approximately locate the vehicle and then use one of the technologies described in this concept to identify the facing side. This extension would have the advantage of navigating car-sharing users to the car.

MQTT was used to connect to the car in this concept. The MQTT messages for this purpose are simple and contain little data. It would be interesting to investigate if there is another way to connect to the car, exchanging the same messages, but using a different technology. NFC offers many possibilities for exchanging data over short distances, as well as Bluetooth. However, with iOS devices, communication is very limited here. Perhaps with Android Devices it would be possible.

A backend is also required to realise the whole idea. For this, a concept must be developed that describes the functionalities and architecture of a backend system, which is then used for authentication and authorization. Furthermore, the booking process for a vehicle is also missing.

REFERENCES

[1] Stadtmobil Carsharing AG. n.d. Homepage – Stadtmobil. https://www.stadtmobil.de. (n.d.). Retrieved on: 2022-06-01.

[2] Iyad Agha. 2016. A dumb UI is a good UI: Using MVP in iOS with swift. http://iyadagha.com/using-mvp-ios-swift/. (2016). Retrieved on: 2022-06-06.

[3] Kevin Curran, Amanda Millar, and Conor Garvey. 2012. Near Field Communication. *International Journal of Electrical and Computer Engineering (IJECE)* 2 (04 2012). DOI:http://dx.doi.org/10.11591/ijece.v2i3.234

[4] Robin Deeg, Johannes Goos, and Tobias Müller. 2020. Remote Control Rolling Chassis. https://www.hdm-stuttgart.de/mediathek/projectpage/3501/details. (2020). Retrieved on: 2022-06-02.

[5] Apple Developer. 2014. Getting Started with iBeacon. https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf. (2014). Retrieved on: 2022-06-01.

[6] Apple Developer. n.d.a. Cameras and Media Capture. https://developer.apple.com/documentation/avfoundation/cameras_and_media_capture. (n.d.). Retrieved on: 2022-06-02.

[7] Apple Developer. n.d.b. Core Location. https://developer.apple.com/documentation/corelocation. (n.d.). Retrieved on: 2022-06-01.

[8] Apple Developer. n.d.c. Core NFC. https://developer.apple.com/documentation/corenfc. (n.d.). Retrieved on: 2022-06-02.

[9] Ahmed El-Rabbany. 2002. *Introduction to GPS - The Global Positioning System*. Artech House, Norwood.

[10] EMQX. n.d. CocoaMQTT. https://github.com/emqx/CocoaMQTT. (n.d.). Retrieved on: 2022-06-06.

[11] Eclipse Foundation. n.d.a. Mosquitto. https://mosquitto.org. (n.d.). Retrieved on: 2022-06-01.

[12] Eclipse Foundation. n.d.b. Paho Python MQTT Client. https://www.eclipse.org/paho/index.php?page=clients/python/index.php. (n.d.). Retrieved on: 2022-06-04.

[13] Share Now GmbH. n.d. Homepage – Sharenow. https://www.share-now.com/de. (n.d.). Retrieved on: 2022-06-01.

[14] Robin Heydon. 2012. *Bluetooth Low Energy - The Developer's Handbook*. Prentice Hall, London.

[15] HiveMQ. Quality of Service (QoS) 0,1, 2: MQTT Essentials. https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/. (????). Retrieved on: 2022-06-05.

[16] HiveMQ. Retained Messages: MQTT Essentials: Part 8. https://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages/. (????). Retrieved on: 2022-06-06.

[17] HiveMQ. TLS/SSL: MQTT Security Fundamentals. https://www.hivemq.com/blog/mqtt-security-fundamentals-tls-ssl/. (????). Retrieved on: 2022-06-06.

[18] Tim Igoe, Don Coleman, and Brian Jepson. 2014. *NFC mit Android und Arduino*. O'Reilly Germany, Köln.

[19] Soroush Khanlou. 2015. Coordinators Redux. https://khanlou.com/2015/10/coordinators-redux/. (2015). Retrieved on: 2022-06-06.

[20] Michael Lahres. n.d. FlexCAR: Entwicklung einer offenen Fahrzeugplattform für die Mobilität der Zukunft. https://www.arena2036.de/de/flexcar. (n.d.). Retrieved on: 2022-06-01.

[21] Jason Marcel. 2019. Bluetooth Technology Is Moving Proximity Solutions In the Right Direction. https://www.bluetooth.com/de/blog/bluetooth-proximity-solutions/. (2019). Retrieved on: 2022-06-01.

[22] M.Q. Saeed and C.D. Walter. 2012. Off-line NFC tag authentication. *2012 International Conference for Internet Technology and Secured Transactions, ICITST 2012* (01 2012), 730–735.

[23] Bluetooth SIG. n.d. Einführung in Bluetooth Beacons. https://www.bluetooth.com/de/bluetooth-resources/intro-to-bluetooth-beacons/. (n.d.). Retrieved on: 2022-06-01.

[24] Dipa Soni and Ashwin Makwana. 2017. A survey on MQTT: a Protocol of Internet of Things (IoT).

[25] Audrey Tam. 2020. Introducing CryptoKit. https://www.raywenderlich.com/10846296-introducing-cryptokit#toc-anchor-013. (2020). Retrieved on: 2022-06-04.

[26] Sumit Tiwari. 2016. An Introduction to QR Code Technology. 39–44. DOI: http://dx.doi.org/10.1109/ICIT.2016.021