

*Konzeption und Entwicklung eines
autarken Alarmsystems für
Kraftfahrzeuge auf Basis GPS
gestützter IoT-Technologie*

Projektdokumentation von:

Lucas Hanisch (577019)
Seifeddine Mhiri (577027)
Paul Krause (577157)
Monique Golnik (563075)

im Modul *M12 Projekt Netzbasierte Systeme*

Master Informations- und Kommunikationstechnik WS 2021/22

25. Februar 2022

Inhaltsverzeichnis

1 Einführung	3
2 Zielsetzung	4
2.1 Ideensammlung	5
2.1.1 Das Starkstrom Taser Auto	5
2.1.2 Die autoleere Stadt	5
2.1.3 Gesichtserkennung und Startunterbrechung	6
3 Projektmanagement	7
3.1 Umfrage	7
3.2 Persona	9
3.3 Konzept	10
4 Entwicklung	12
4.1 Arduino	12
4.1.1 Ausstattung	12
4.1.2 Software	12
4.1.3 Aufbau Code	14
4.1.4 Entwicklungsprozess	15
4.1.5 Aufbau Hardware	16
4.1.6 Probleme	17
4.2 Webserver	18
4.2.1 Einrichtung des Webservers auf einem Raspberry Pi	18
4.2.2 Probleme mit dem Raspberry Pi	20
4.2.3 Grundsätzliche Überlegungen	21
4.2.4 Testprogramme	22
4.2.5 Erstellen der PHP-Skripte zum Empfangen und Be- reitstellen von Daten	23
4.2.6 Verschlüsselung der Daten	26
4.3 Android	31
4.3.1 Mobile Apps	31
4.3.2 Android Studio	31
4.3.3 Flutter	32

4.3.4	Dart	32
4.3.5	Konzept: SchrödingersAlarm	33
4.3.6	Implementierung	34
4.3.7	Implementierung SchrödingersAlarm App	35
4.4	App - iOS	37
4.4.1	Xcode	38
4.4.2	SwiftUI	38
4.4.3	Hauptkriterien an die Funktionalität	39
4.4.4	Design	40
4.4.5	Probleme	44
5	Produkt	47
5.1	zukünftiges Produktangebot	47
5.2	Vergleich	48
6	Rück- und Ausblick	51
6.1	Lessons Learned	52
6.2	Zukünftige Entwicklung	52
6.3	Fazit	57
7	Anhänge	58
7.1	Quellcode	58
7.2	Videos	58
	Abbildungsverzeichnis	58
	Quellenverzeichnis	60

Kapitel 1

Einführung

Allein in Deutschland wurden laut dem ADAC im Jahr 2020 **10.697** kaskoversicherte **Autos gestohlen** – rechnerisch also 29 Fahrzeuge pro Tag. Die mit Abstand höchste Diebstahlrate verzeichnet dabei erneut Berlin, wo im vergangenen Jahr 2267 Pkw geklaut wurden. Wie schon im Vorjahr, findet mehr als jeder fünfte Autodiebstahl in der Bundeshauptstadt statt. [Dü21]

Im Rahmen des Gruppenabschlussprojektes im Master Informations- und Kommunikationstechnik haben wir uns daher dafür entschieden, ein System zu entwickeln, welches besonders Haltern von älteren Fahrzeugmodellen, die oft überhaupt nicht mit modernen Security Systemen ausgestattet sind, ein sichereres Gefühl vermitteln soll. Eines unserer Gruppenmitglieder ist zudem direkt von dieser Problematik betroffen, weshalb wir besonders motiviert an die Umsetzung dieses Projektes herangegangen sind und uns so ein Stück weit der Kriminalität entgegenstellen wollen.

Der Fokus des Moduls Projekt Netzbasierte Systeme liegt auf Informatik, Kommunikations- und Nachrichtentechnik. Daher basiert das hier beschriebene und realisierte Projekt auf unterschiedlichen, miteinander agierenden Technologien aus dem Hard- und Softwaresegment. Ein weiterer Schwerpunkt war, das entstehende Produkt nicht nur technisch zu entwickeln, sondern auch aus der Sichtweise des Projektmanagements. Dafür wurden neue Methoden im Laufe des Moduls wie beispielsweise Design-Thinking kennengelernt und umgesetzt.

Das nachfolgend beschriebene Projekt - gerne auch kurz Schrödingers Alarm genannt, welches von Seifeddine Mhiri, Lucas Hanisch, Paul Krause und Monique Golnik umgesetzt wird, sieht die Implementierung eines vernetzten Systems bestehend aus Arduino, Raspberry Pi und Smartphone und somit die Kommunikation dieser Module untereinander vor.

Kapitel 2

Zielsetzung

Das entstehende Produkt soll nicht nur die Problematik des Autodiebstahls verringern, sondern zusätzlich Faktoren wie Nutzerfreundlichkeit, Wartungs- und Anschaffungskosten sowie Nachhaltigkeit berücksichtigen. Des Weiteren möchten wir dank moderner Managementtechniken ein Verständnis erlangen, was es bedeutet, ein Produkt von einem Gedanken heraus zu entwerfen und zu entwickeln. Ebenfalls war es uns wichtig, dass wir uns mit Themenbereichen beschäftigen, die neu sind aber in denen wir uns gern einmal einarbeiten oder weiterentwickeln wollen - kurz: die persönlichen Interessen jedes Teammitgliedes sollten zusätzlich zu den zuvor genannten Faktoren berücksichtigt werden.

Dank der Methode Design-Thinking war es uns möglich, unsere vielen Ideen zu filtern, zu fokussieren oder zu verwerfen. Hierbei gilt es sich nacheinander an den fünf Phasen

- *Verstehen - Welche Bedürfnisse, Probleme oder Wünsche haben Kunden?*
- *Ideen generieren*
- *Konzepte entwickeln - frühzeitiges Einholen von Feedback*
- *Prototyping*
- *MVP und Markteinführung*

zu orientieren, welche auch im Laufe der Dokumentation verdeutlicht dargestellt werden. [Ful22]

2.1 Ideensammlung

Im nachfolgenden Kapitel werden wir drei unserer anfänglich favorisierten und vielleicht durchaus kuriosen Ideen vorstellen.

Wenn wir uns alle trauen würden, kreativer zu sein, verändern wir die Gesellschaft. Weil es und dann gelingt, neue Perspektiven einzunehmen¹

2.1.1 Das Starkstrom Taser Auto

Viele Autos haben heutzutage bereits moderne Alarmsysteme und dennoch werden potentielle Diebe nicht müde, diese auszukundschaften und passende Techniken zu entwickeln, um sie auch auszuhebeln oder zumindest zu umgehen. Des Weiteren muss man ehrlich zu geben: Selbst wenn eine Alarmanlage anfängt, laut zu piepsen, schaut man dann wirklich als Passant hin und unternimmt etwas und das gerade in Berlin, wo dies mehrmals am Tag allein in einer Straße vorkommt? Wir glauben nicht! Und daher waren wir der Meinung, dass ein Alarmsystem genial wäre, dass nicht nur den Dieb daran hindert, das Auto zu stehlen, sondern gleichzeitig auch einen didaktischen Effekt aufweist. Ein wichtiges Helferlein aus unserem Ingenieurstudium könnte hier die Lösung sein. Die meisten Menschen haben schließlich auch heute noch großen Respekt vor ihm, wurde er doch auch früher bereits für Bestrafungen missbraucht - der Starkstrom. Die Idee war geboren: ein Auto, welches per App Steuerung und eigener, zusätzlich eingebauter Batterie seine Außenhülle unter Strom setzt, wenn der Besitzer das möchte. Fasst der potentielle Kriminelle das Auto an, durchzieht ihn ein Stromschlag und der Täter wird höchstwahrscheinlich das Weite suchen - so die Theorie. Jedoch - so schön der Gedanke wäre und bereits Herr Prof. Dr. Scheffler zu uns sagte "um Verzeihung zu bitten ist einfacher als um Erlaubnis", so ganz ethisch ist diese Vorgehensweise jedoch nicht. Zudem müssten viele Sicherheitsvorkehrungen vorgenommen werden, wie beispielsweise eine KI die erkennt, dass vielleicht nur ein Kind verstecken spielt oder den Schnee von der Windschutzscheibe sammeln möchte. Daher haben wir uns letztendlich gegen diese Idee ausgesprochen, da bereits beim Gedankenexperiment sehr viele Hürden auftauchten, die wir in recht kurzer Zeit hätten überwinden müssen.

2.1.2 Die autoleere Stadt

Eine weitere unserer Ideen war es, anstatt eine direkte Lösung für das Problem zu finden, das Problem selbst zu beseitigen. Hierfür haben wir uns vorgestellt, dass wir einfach Autos generell abschaffen. Die Idee ist, dass wenn es keine Autos mehr gibt, können sie auch nicht mehr geklaut werden.

¹Bod22.

Diese Idee ist jedoch an der Umsetzbarkeit gescheitert. Es gibt zwar bereits Innenstädte, wo Autos verboten sind, jedoch ist dies nicht gleichzusetzen mit dem kompletten Abschaffen von Autos, da diese zumeist nur außerhalb der Stadt geparkt sind. Die Idee wurde letztendlich verworfen, da die Umsetzung dieses Projekt unseren Projektrahmen gesprengt hätte.

2.1.3 Gesichtserkennung und Startunterbrechung

Bei dieser Lösung hatten wir uns überlegt, mittels Gesichtserkennung zu überprüfen, ob die sich auf dem Fahrersitz befindliche Person der Besitzer oder ein berechtigter Fahrer des Fahrzeugs ist. Die Zündung des Autos soll hier nur dann möglich sein, wenn der Fahrer erkannt wird. Das Problem an dieser Modifikation und der Grund, warum wir uns gegen diese Idee entschieden haben, ist die Zulassungspflicht durch den TÜV. Um den Motor am Starten zu hindern, müssten wir uns entweder auf das Steuergerät des Autos zuschalten oder direkt eine Fernsteuerung für die Motorklemme mit einbauen. Die erste Möglichkeit scheitert daran, dass sich der Zugang zu den Steuersystemen des Autos schwer gestaltet, da je nach Modell die Steuersoftware unterschiedlich ist, womit allgemeine Ansätze ausgeschlossen sind. Des Weiteren erlauben Hersteller entweder generell nicht, dass Drittanbieter ihre Software nutzen oder stimmen nur gegen hohe Lizenzgebühren einer Nutzung zu. Ein weiteres Problem ist, dass beide Lösungen eine professionelle Installation bedürfen, was unserem Konzept widerspricht. Abgesehen davon ist es, wie bereits erwähnt, notwendig, für jedes Fahrzeugmodell die entsprechende Straßenzulassung für die Modifikation zu erhalten. Entsprechend ist der Arbeitsaufwand für unseren Projektzeitraum zu groß.

Kapitel 3

Projektmanagement

Für das Projekt haben wir nach der agilen Methode SCRUM gearbeitet - daher wird man auf der Suche nach einem historischen Pflichten- und Lastenheft bei uns nicht fündig. Die in unserem Projektstrukturplan erfassten Arbeitspakete haben wir in einem sogenannten Trello-Board festgehalten. Dies ermöglichte uns dank einer selbstgewählten Kartenübersicht auf einem Blick schnell erfassen zu können, wo wir derzeit stehen und was noch unbedingt erledigt werden sollte. Ebenfalls haben wir unvorhergesehene Probleme dort festgehalten, um zum einen Stichpunkte für die Dokumentation festzuhalten und zum anderen nachvollziehen zu können, weshalb wir manche Tickets nicht so schnell fertig bearbeiten konnten, wie ursprünglich angenommen.

SCRUM – „Pläne sind nichts. Planung ist alles“¹

Die kurze zyklische Struktur von Scrum mit seinen wiederkehrenden Meetings bietet eine ständige Möglichkeit zur Überprüfung und Anpassung. Positiv ist hierbei auch, dass auch der Fortschritt zeitnah sichtbar und dementsprechend das Team motiviert wird. Anfänglich haben wir Sprints von 14 Tagen ausprobiert in der Retrospektive jedoch festgestellt, dass ein 7-tägiger Rhythmus, um schneller Probleme analysieren zu können und aufgrund der recht kurzen Gesamtbearbeitungszeit im ständigen Austausch zu sein, sinnvoller ist. Dies hat sich auch sehr bald als richtige Entscheidung herausgestellt, da das Gefühl “dass jeder nur seine Insel bearbeitet” minimiert werden konnte.

3.1 Umfrage

Im Rahmen der Projektvorbereitung sollte auch eine Umfrage unter potenziellen Kunden erfolgen. So sollen möglichst früh in der Entwicklung Wünsche und die Sicht der Kunden auf ein Produkt oder ein Problem eingeholt werden. Hierbei handelt es sich um den sogenannten Customer Insight, welcher

¹RD19.

uns neue Perspektiven eröffnen soll. Damit soll verhindert werden, dass wir in unseren Ideen möglicherweise etwas ganz Entscheidendes vergessen, was aber aus der Sicht der Kunden besonders wichtig ist. Es sollen außerdem auch möglichst viele verschiedene Anforderungen der einzelnen Personen erhalten werden. Dafür haben wir eine Umfrage mit insgesamt 19 Fragen entwickelt. Am Ende konnten wir immerhin 44 Teilnehmende aufweisen.

Zuerst kamen ein paar Fragen zum Fahrzeug der Teilnehmer, z.B. von welcher Marke und wie alt es ist. Im zweiten Abschnitt stellten wir einige Fragen zur Sicherheit der Fahrzeuge der Teilnehmer, z.B. wie sie gegen Diebstahl geschützt sind und ob schon einmal etwas gestohlen wurde. Als nächstes folgten Fragen, mit denen wir direkte Bedürfnisse und Anforderungen für unser Produkt herausfinden wollten, unter anderem wurde gefragt, was ihn/sie an aktuell zu erwerbenden Autoalarmsystemen stört und wieviel er/sie maximal für ein solches System ausgeben würde. Zum Abschluss wurden noch ein paar persönliche Informationen für mögliche statistische Zusammenhänge abgefragt, wie Geschlecht, Alter und Beschäftigungsverhältnis. Konkret setzte sich die Umfrage aus folgenden Fragen zusammen:

- Von welcher Marke ist dein Auto?
- Zu welcher Fahrzeugklasse gehört dein Auto?
- Welches Baujahr ist dein Auto?
- Wie teuer war dein Auto ungefähr und wieviel ist es jetzt noch wert?
- Wie wichtig ist dir dein Auto? (Emotionale Bindung usw.)
- Wie ist dein Auto gegen Einbruch und Diebstahl geschützt?
- Wie zufrieden bist du mit deiner Sicherheitslösung?
- Wurde dir bereits ein Auto geklaut oder es versucht?
- Wurde schon einmal in dein Auto eingebrochen oder es versucht?
- Wenn ja, was wurde geklaut?
- Wie ist dein Auto versichert?
- Wo parkst du dein Auto normalerweise?
- Um was hast du Angst, wenn dein Auto unbeaufsichtigt ist?
- Was stört dich an bisher käuflich erwerblichen Alarmsystemen für Autos bzw. was würdest du an diesen ändern?

- Wie viel würdest du für ein neues Alarmsystem für dein Auto ausgeben?
- Du bist... (männlich, weiblich, divers)
- Wie alt bist du?
- Wie ist dein derzeitiges Beschäftigungsverhältnis?
- Wie hoch ist dein monatliches Nettoeinkommen?

Die Umfrage wurde mit Hilfe von Google Forms realisiert und ist noch unter folgendem Link abrufbar: <https://forms.gle/qNTWM4qkF7ykG4tt9>

3.2 Persona

Des Weiteren sollte die Frage geklärt werden: Wie kann die Fähigkeit, sich in andere Personen hineinversetzen zu können, ihre Bedürfnisse und Motivation nachzuvollziehen und Verhaltensmuster vorherzusagen, zur Zielgruppenanalyse genutzt werden? [JL] Nachfolgend stellen wir kurz unsere Persona vor, welche wir auf Basis der Auswertung der im vorherigen Abschnitt erläuterten Umfrage konzipiert und charakterisiert haben.

Hintergrund zur Person: <ul style="list-style-type: none"> - Linus ist ein stresssamer Projektmanager in der IT Branche - Er hat Elektrotechnik studiert und interessiert sich seit Kind an für Technik und arbeitet seit 10 Jahren in der Firma - Er mag das Gefühl von Sicherheit in seinem Beruf 	Demographie: <ul style="list-style-type: none"> - Männlich - 38 Jahre alt - Berlin - verheiratet 	
Identifikatoren: <ul style="list-style-type: none"> - Linus geht gerne in die Natur zusammen mit seiner Frau und ihrem Hund außerdem verbringt er auch gerne Zeit am PC - Die Urlaube werden mit und in dem eigenen VW Bus verbracht an den unterschiedlichsten Orten - Er achtet auf sein Äußeres, geht jedoch nicht mit der Mode sondern, trägt was ihm gefällt – was oftmals „nerdige“-Aufdrucke beinhaltet - Über das Internet informiert er sich gerne über die neuesten Technikinnovationen und wenn ihm etwas zusagt, nutzt er gerne die Möglichkeit des Onlineshoppens - Kaufentscheidungen werden je nach Betrag auf die schnelle allein entschieden oder gemeinsam mit seiner Ehefrau 		
Erwartungen, Ziele & Emotionen: <ul style="list-style-type: none"> - Linus kennt die Statistiken zum Thema Autodiebstahl und auch die erschreckend hohe Zahl für VW Busse - Er möchte seinen VW Bus sicherer wissen und sucht nach einem einfach zu montierenden Alarmsystem, welches nicht zu teuer ist, aber auch mit seinem Smartphone verbunden ist - Die große Auswahl und doch sehr unterschiedlichen Umsetzungen verunsichern ihn 	Herausforderungen: <ul style="list-style-type: none"> - Der Preis sollte möglichst nicht über 100€ hinausgehen - Die Stromversorgung sollte ebenfalls autark sein und nicht Strom vom VW Bus „abzapfen“ - Das System sollte nicht unsichtbar sein aber auch nicht zu sichtbar -> eine abschreckende Wirkung auf Diebe haben aber nicht zu neugierig machen - Linus achtet auf Nachhaltigkeit, also sollte das System im Notfall auch einfach repariert werden können 	
Ideale Lösung: <ul style="list-style-type: none"> - Wir wollen Linus ein sichereres Gefühl für seinen VW Bus geben, mit einem energieeffizienten GPS-gestützten autarken Alarmsystem, welches beim Auslösen direkt auch bei seinem Smartphone eine Meldung auslöst - Eine einfache und übersichtliche Bedienung erleichtern die Einrichtung und die Handhabung des Systems 	Häufige Einwände: <ul style="list-style-type: none"> - Preis-Leistungsverhältnis - Eingriff in das Interieur / Elektronik des Autos (sieht „verbastelt“ aus) - Es soll keine Werkstatt notwendig sein, das System einzubauen - Abo-kosten 	

Abbildung 3.1: Persona *Linus Schrödinger*

Unsere Persona namens Linus Schrödinger ist zusammenfassend gesagt, ein technikinteressierter naturbegeisterter Ehemann, der es liebt mit seiner Frau und seinem VW Bus zu verreisen. Er mag das Gefühl von Sicherheit und ist daher beunruhigt über die hohe Anzahl der Autodiebstähle allein für VW Busse - ein nachrüstbares Alarmsystem stellt an sich heutzutage kein Problem mehr da, jedoch ist die Auswahl groß und verunsichern ihn in seiner Kaufentscheidung. Ihm ist wichtig, dass das System möglichst nachhaltig

aufgesetzt ist, nicht in die eigentliche Fahrzeugelektronik eingreift und leicht selbst montiert werden kann.

Ziel dieser Methode ist die Entwicklung von Nutzermodellen, die Personen einer spezifischen Zielgruppe mit bestimmten Merkmalen charakterisieren. Anhand einer generierten Persona kann vorausgesagt werden, was ein Charakter in bestimmten Situationen tun würde. [JL]

Diese Persona stellt also einen fiktiven Charakter dar, der stellvertretend für einen Käufer unseres Alarmsystems steht. Ein solch imaginäres Modell half uns, sich in die Lage eines Kunden während der Entwicklung unseres Prototypen zu versetzen. Des Weiteren trägt die Auseinandersetzung dazu bei, sich mit konkreten Erwartungen, Zielen und Bedürfnissen des zukünftigen Kunden zu beschäftigen und unser System möglichst effizient und effektiv auf die von uns gewünschte Zielgruppe auszurichten.

3.3 Konzept

Nachdem wir mit Hilfe der Umfrage und der Persona schon einen groben Eindruck bekommen haben, auf welche Zielgruppe unser Produkt passen könnte und welche Eigenschaften und Funktionen es erfüllen soll, erstellten wir ein erstes Konzept. Dieses Konzept beinhaltet alle wesentlichen Komponenten unseres Produkts und ist in der folgenden Abbildung schematisch dargestellt.

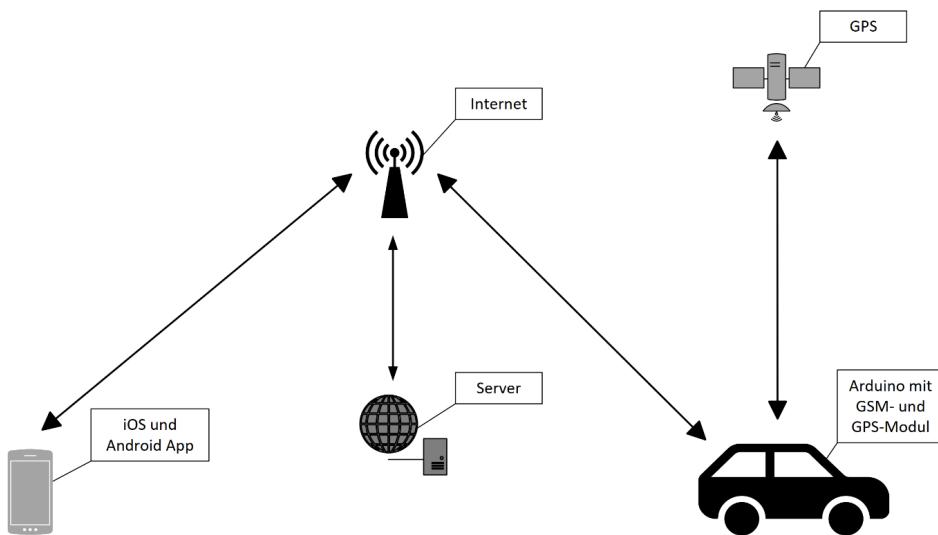


Abbildung 3.2: Systemkonzept

Im Fahrzeug wollen wir einen Arduino Uno zusammen mit einem pas-

senden GSM- und GPS-Modul platzieren. Dieser ermittelt in bestimmten Zeitintervallen die aktuelle Position des Fahrzeugs und schickt die Koordinaten über die GSM-Verbindung an unseren Webserver. Dieser verarbeitet diese Daten und stellt sie anschließend so bereit, dass die Apps beider Betriebssysteme die aktuellen Koordinaten abfragen können.

Kapitel 4

Entwicklung

4.1 Arduino

4.1.1 Ausstattung

Um den Zielen, die wir uns für unseren Prototypen gesetzt haben, gerecht zu werden, ist der Arduino mit einem Sim 808 Module sowie entsprechenden GPS und GSM Antennen ausgerüstet.

Verbunden sind das Modul und die Antennen mit Hilfe eines Shields. Es handelt sich um ein Shield des Models OAS808SIM und ist von der Firma MakerFabs.

Die Stromversorgung erfolgt mittels vier BRC 18650 4200mAh Batterien, die in Reihe geschaltet sind.

4.1.2 Software

Die Kontrolle des Shields erfolgt mittels UART und entsprechenden AT-Befehlen. Die erforderlichen Befehle sind zusammen mit einer kurzen Erklärung nachfolgend in den Abbildungen 4.1 und 4.2 aufgelistet.

AT+CPIN=<PIN>	Konfiguriert die Pin für die Sim-Karte
AT+SAPBR=3,1,"Contype","GPRS"	Definiert die Art der übertragenen Daten als GPRS Daten
AT+ SAPBR =3,1,"APN","TM"	Gibt an, welcher APN verwendet wird. Wir verwenden TM was für Things Mobile steht und dem Provider der Sim Karte entspricht
AT+ SAPBR =1,1	Startet die Verbindung mit dem Carrier mit den angegebenen Parametern
AT+ SAPBR =2,1	Fragt die Statusinformationen vom Carrier ab
AT+HTTPPARA="CID",1	Definiert, welches Profil an Trägerdiensten verwendet werden soll
AT+HTTPPARA=URL, URL	Legt fest, an welche URL gesendet werden soll
AT+HTTPPARA=CONTENT,application/x-www-form-urlencoded	Informiert den Server, um welchen Datentyp es sich bei den gesendeten Daten handelt
AT+HTTPDATA=192,5000	Definiert im ersten Parameter die Größe des gesendeten Datenpaketes und im zweiten Parameter, wie lange auf eine Antwort vom Server gewartet werden soll, bevor die Übertragung abgebrochen werden soll
param={"device":device+"", "Latitude":Feld[3], "Longitude":Feld[4]}	Sendet die gewünschten Daten im JSON Format an den Server, wie im vorhinein definiert wurde
AT+HTTPACTION=1	Gibt die Art des Http Befehls an GET=0 , POST=1 und HEAD=2
AT+HTTPREAD	Liest die Antwort vom Server
AT+HTTPTERM	Bricht laufende Http Anfrage ab
AT+ SAPBR =0,1	Beendet die Verbindung zum Carrier
AT+HTTPINIT	Startet den HTTP Service auf dem Sim 808 Modul

Abbildung 4.1: GSM und Http-Befehle

AT+CGNSPWR=1	Startet GPS Module
AT+CGNSSEQ=RMC	Definiert die Schreibweise der Zurückgegebenen GPS Informationen
AT+CGNSINF	Fragt die momentane GPS-Position ab

Abbildung 4.2: GPS-Befehle

4.1.3 Aufbau Code

Der Code ist wie ein Standard-Mikrocontroller-Code aufgebaut und besitzt zwei Unterfunktionen, die hier kurz grafisch in Abbildung 4.3 dargestellt werden.

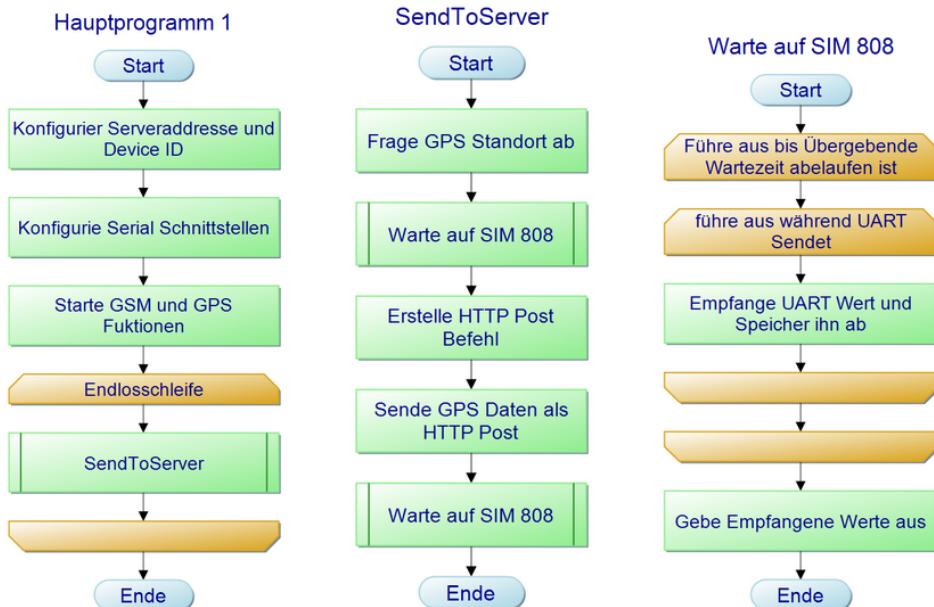


Abbildung 4.3: Programmablaufplan

Der Arduino macht in der momentanen Prototyp Version nichts anderes, als das Sim 808 Modul zu steuern. Dafür gibt es drei Funktionen:

Eine Funktion, welche die Kommunikation mit dem Sim Modul erleichtert, eine weitere Funktion für die Abfrage des momentanen Standortes und eine dritte, um die Kommunikation mit dem Server zu erlauben.

Mit der Funktion `wart()` wird zum einen das Pausieren für die Bearbeitungszeiträume geliefert und zum anderen die Ausgabe des Sim Moduls an

den PC zum Debuggen geschickt. Die Funktion gpsDaten fragt den aktuellen Standort ab und die Funktion SendToServer sendet die zu übermittelnden GPS Daten an den Server. Der Code ist in C geschrieben, da diese Programmiersprache für unsere Zwecke ausreichend performant ist.

4.1.4 Entwicklungsprozess

Der Entwicklungsprozess ist in mehreren Stufen erfolgt. Zunächst wurde mittels eines einfachen Programms die Funktionalität der GPS-Antenne überprüft, da wir dafür noch keine Sim-Karte benötigten. Zur Implementierung der Standortabfrage soll zum ersten Mal die UART-Schnittstelle zum Board konfiguriert werden. Nachdem die UART-Schnittstelle konfiguriert wurde, haben wir die Funktion wart() entwickelt, um die Kommunikation mit der UART-Schnittstelle zu erleichtern. Mit Hilfe der wart() Funktion konnten wir nun auf die Antwort von der UART-Schnittstelle warten und dies auf dem Computer zum Debuggen ausgeben. Das Senden und Empfangen von Informationen über UART funktionierte also und anschließend haben wir die Abfrage der GPS-Daten realisiert. Nachdem wir die entsprechenden Befehle gefunden hatten, war dies einfach zu realisieren.

Da die ersten Tests jedoch auf dem Dachboden ausgeführt wurden und das Dach aus Metall ist, konnte zunächst kein Signal empfangen werden. Nachdem wir das Problem erkannt hatten, haben wir alle zukünftigen Test an einer besser geeigneten Position durchgeführt.

Die Standortabfrage funktionierte nun, sodass wir im nächsten Schritt zum Testen des GSM-Moduls und der Sim-Karte das Senden von SMS realisiert hatten. An dieser Stelle haben wir lediglich die bereits erstellten Funktionen genutzt und die Befehle für die SMS-Kommunikation hinzugefügt. Dabei ist aufgefallen, dass beim erstmaligen Nutzen des GSM-Moduls nach dem Einschalten immer der PIN zum Entsperren erforderlich ist.

Nachdem wir die Kommunikation mit dem GSM Modul ermöglicht hatten, haben wir die Kommunikation mit dem Server realisiert. Nachdem wir hierfür die richtigen Befehle gefunden hatten, haben wir zunächst Testdaten per HTTP POST an einen Test-Server gesendet, der einfach nur seine empfangen HTTP POSTs anzeigt. Danach haben wir das Senden von Daten im JSON-Format an unseren Server realisiert. Aus Netzwerksicht mussten wir nur die Serveradresse ändern. Damit der Server die empfangenden Daten nutzen kann, mussten wir dann jedoch noch den Daten-Sendebefehl neu konfigurieren. Da wir über die UART-Schnittstelle nur Strings schicken können, mussten wir die Formatierung für die zu sendenden Daten anders einstellen und diese dann beim versendeten String berücksichtigen.

Nachdem die Kommunikation mit dem Server erfolgreich funktioniert hatte, haben wir die einzelnen Teile zusammengefügt. Hierfür mussten wir den GPS-Antwort-String verarbeiten. Um die benötigten Werte zu erhalten, haben wir den String immer an den Kommata in kleinere Strings unterteilt und über die vordefinierte Struktur des Strings ermittelt, welche Werte der Längengrad und der Breitengrad sind. Dies Werte haben wir für zukünftige Erweiterungen beziehungsweise Funktionalitäten zwischengespeichert und binden diese in den zu sendenden String ein.

Wir haben dabei den Datentyp String der erhaltenen Daten bewusst nicht geändert, da dies sonst dazu führt, dass Nachkommastellen weggelassen werden, was die Standortbestimmung verzerrt.

4.1.5 Aufbau Hardware

Für die Stromversorgung müssen die Akkus an den Vin (*Eingangsspannung*) Pin und den GND (*Masse*) Pin angeschlossen werden. Da das Shield diese Pins einfach durch schaltet, können wir die Konstruktion trotz Shield so verwenden.

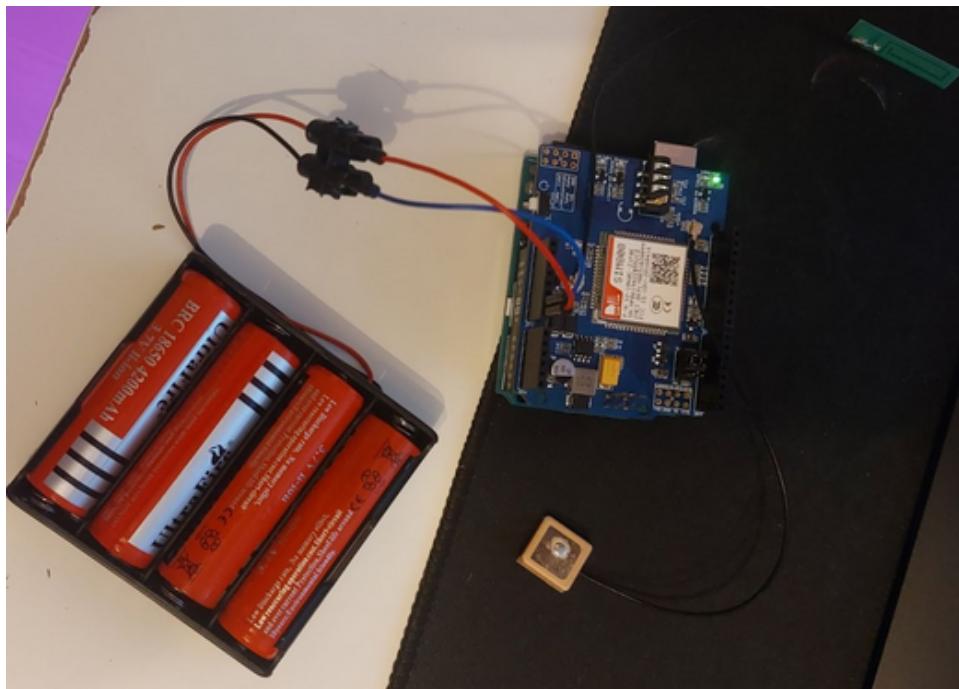


Abbildung 4.4: Systemaufbau für das KFZ

Der Vorteil dieser Konstruktion ist, dass sie sehr kompakt ist und somit leichter im Auto versteckt werden kann. Ein weiterer Vorteil dieser Bauweise

ist, dass für den Fall, dass später durch Erweiterungen die Akkulaufzeit nicht ausreicht, einfach weitere Akkus parallel hinzugefügt werden können.

4.1.6 Probleme

Die Probleme, die bei der Entwicklung des Controllers aufgetreten sind, lassen sich in zwei Kategorien unterteilen. Zum einen gab es technische Probleme und zum anderen logistische Schwierigkeiten. Das Problem bei der Beschaffung der GSM-Module war, dass diese meist aus China importiert werden mussten, was dazu geführt hat, dass wir unter langen Wartezeiten gelitten haben. Von den vier bestellten Modulen wurden zwei nicht geliefert (*die Bestellung wurde vom Lieferanten gestrichen*). Des Weiteren konnte eines der Module nur über eine bereitgestellte Software verwendet werden.

Die technischen Probleme sind zum Teil auf die schlechte Dokumentation des Arduino Shields zurückzuführen. Zum einen fehlt die Information für die richtige Baudrate für die UART Schnittstelle, welche über strukturiertes Testen herausgefunden werden musste. Das Problem hierbei war, dass das Modul entsprechend keine Fehlermeldung zurückgeben konnte, da die Kommunikation zwischen Modul und Arduino selbst gestört war und nichts zurückgeben wurde:

```
Test
:::::::::::;

Test
AT+CMEE=1
OK
AT+CPIN=<1503>
+CME ERROR: 3
AT+CGNSPWR=1
OK
AT+CGNSSEQ=RMC
OK
AT+CGNSINF
+CGNSINF: 1;0;19800106000003.000;;;;0.00;0.0;0;;;;0;0;;;;AT+CGNSINF
+CGNSINF: 1,0,19800106000003.000,,,0.00,0.0,0,,,,,,0,0,,,
```

Abbildung 4.5: fehlerhafte und korrekte Ausgabe

In Abbildung 4.5 sieht man zunächst, wie die fehlerhafte Ausgabe aussieht. Unter dieser ist zum Vergleich die richtige Ausgabe abgebildet.

Auch die AT-Befehle, welche zur Kommunikation mittels HTTP dienen soll-

ten, haben nicht entsprechend funktioniert. Gelöst werden konnte das Problem, indem die Befehlskette verwendet wird, mit der jedes Datenpaket einzeln und manuell konfiguriert werden kann. Entsprechend mussten die Pakete als IOT-HTTP-Paket definiert werden und die Netzbetreiber-Daten sowie Paketinformationen für die Verbindung eingestellt werden.

Des weiteren hat das GSM-Modul die Kommunikation mit einer normalen Vodafone Sim-Karte nicht aufbauen können. Die Verzögerung, eine geeignete Prepaid Karte zu bekommen, hat den Zeitrahmen für diesen Teil erheblich nach hinten verschoben. Das Problem an diesem Fehlern war, dass das Modul nur ERROR 3 zurückgegeben hat, was als „*Operation nicht erlaubt*“ definiert ist. Es wird dabei aber nicht genauer spezifiziert, warum diese Operation nicht erlaubt ist. Dies hat das Troubleshooting weiter erschwert.

Für die GPS-Abfrage haben sich zwei wesentliche Probleme ergeben. Das am schwierigsten zu lösende Problem war, dass die benötigten Befehle nicht in der richtigen Dokumentation zu finden sind, sondern nur in einigen Beispiel-Programmen vom Shield Hersteller. Das andere Problem ist, dass das GPS-Signal beim ersten Konfigurieren (*Einstellen der internen Uhr*) guten Empfang braucht.

4.2 Webserver

Damit die vom Arduino ermittelten Standortdaten des Fahrzeugs an die Apps weitergeleitet werden können, benötigt es eine Zwischeninstanz. Diese soll die empfangenen Daten aufbereiten und sie anschließend so bereitstellen, dass sie einfach von den Apps abgefragt werden können. In einem zweiten Entwicklungsschritt soll dann der Sicherheitsaspekt der Daten berücksichtigt werden. Hierfür sollen Verschlüsselungsalgorithmen implementiert werden, sodass die Daten nur von den Apps entschlüsselt und ausgewertet werden können.

4.2.1 Einrichtung des Webservers auf einem Raspberry Pi

Der HTTP-Webserver soll auf einem Raspberry Pi mit Hilfe des „lighttpd“-Webservers erstellt werden. Die Installation und Einrichtung desselbigen gestaltete sich relativ einfach. Zunächst musste das entsprechende „lighttpd“-Paket auf dem Raspberry Pi installiert werden. Im nächsten Schritt mussten noch ein paar Berechtigungen erteilt werden. Nach einem Neustart von lighttpd war die Einrichtung soweit fertig und der Webserver war aktiv. Alle Dateien und Skripte, die der Webserver bereitstellt und bearbeitet liegen im Verzeichnis „/var/www/html“.

Damit der Server später auch von außerhalb erreichbar ist, waren noch einige

Arbeitsschritte zu erledigen. Zuerst musste der Raspberry Pi eine statische IPv4-Adresse erhalten, wir setzten diese auf 192.168.178.215. Die .215 wurde deshalb ausgewählt, weil der Bereich der automatischen Adresszuweisung von den meisten DHCP-Servern nur von .0 bis .200 geht und somit eine doppelte Vergabe sehr unwahrscheinlich ist.

Bei DYNDNS handelt es sich um einen Service, der es ermöglicht, Domains mit dynamisch wechselnden IP-Adressen im Internet zu verwalten. Dieser Service ist notwendig, da der heimische Router vom Provider in regelmäßigen Abständen eine neue, externe IP-Adresse zugeordnet bekommt und somit der Raspberry Pi bzw. der darauf aktive Webserver nach einem „IP-Wechsel“ nicht mehr über das Internet erreichbar wäre. Bei Nutzung eines DYNDNS Anbieters, bleibt der Raspberry Pi bzw. der Webserver weiterhin unter seinem vergebenen Hostnamen erreichbar. Damit ein solcher Service genutzt werden kann, ist es notwendig, sich bei einem DYNDNS-Anbieter einen Account anzulegen. Für diese Arbeit wurde sich aufgrund einer persönlichen Empfehlung für Securepoint DYNDNS entschieden. Auf deren Webseite konnte nach der Erstellung eines Benutzerkontos ein IPv4-Host hinzugefügt werden. Dafür wurde neben der IP-Adresse des Raspberry Pi der Hostname eingetragen, welcher dann automatisch eine Domainerweiterung des DYNDNS Anbieters erhalten hat.

Im letzten Schritt musste nun der heimische Router so konfiguriert werden, dass ein Zugriff auf den Webserver von außerhalb des Heimnetzwerks erlaubt ist. In unserem Fall handelt es sich beim Router um eine FRITZ!Box, bei welcher diese Einstellungen einfach vorgenommen werden konnten. Dafür musste zunächst eine neue Freigabe für das Gerät „Raspberry Pi“ hinzugefügt werden. Anschließend wird als Anwendung „HTTP-Server“ und damit Port 80 ausgewählt. Zusätzlich musste dann noch das DYNDNS in der FRITZ!Box konfiguriert werden. Dafür wird eine Update-URL des DYNDNS-Anbieters sowie der Domainname und die Benutzerinformationen benötigt. Damit waren alle notwendigen Einstellungen vorgenommen und der Webserver ist nun von überall erreichbar.

Zusätzlich haben wir die Software „VNC Server“ auf dem Raspberry Pi installiert. Mit Hilfe dieser Software und dem entsprechenden Client „VNC Viewer“ auf einem PC oder Laptop kann eine Remotedesktopverbindung zum Raspberry Pi hergestellt werden. Dies ist besonders in Zeiten von Lockdowns und digitalen Remotearbeiten eine wichtige Funktion, da sich so jedes Gruppenmitglied auf dem Raspberry Pi schalten kann.

4.2.2 Probleme mit dem Raspberry Pi

Während der ersten Wochen der Projektarbeit mussten wir leider alle paar Tage feststellen, dass keine Verbindung per VNC zum Raspberry Pi hergestellt werden konnte. Wir vermuteten zunächst keinen dramatischen Fehler, denn nach einem Neustart funktionierte wieder alles. Dieser Fehler trat allerdings doch mit einer gewissen Regelmäßigkeit auf und nachdem ein erstes PHP-Skript auf unserem Webserver lief, konnten wir feststellen, dass auch dieses regelmäßig nicht erreichbar war. Der Fehler lag also eindeutig nicht an der VNC-Software, sondern es scheint generelle Internet Verbindungsprobleme des Raspberry Pi zu geben.

Deshalb versuchten wir doch genauer herauszufinden, wo der Ursprung des Verbindungsverlustes liegt. Nach einigem Recherchieren und dem Sichten diverser Log-Files auf dem Raspberry Pi, konnten wir die Fehlermeldung „*brcmf_cfg80211_scan: scan error (-110)*“ für die Verbindungsabbrüche ausfindig machen. Bei diesem Fehler handelt es sich um ein Verbindungsproblem des WiFis des Raspberry Pi's. Dieser Fehler ist nicht unbekannt und mögliche Lösungen werden in diversen Internetforen diskutiert. Allerdings scheint keine der vorgeschlagenen Workarounds generell zu funktionieren, außer ein wie auch von uns praktiziertes, regelmäßiges Neustarten. Bevor wir diese alle ausprobieren und im ungünstigsten Fall eine Menge Zeit ohne Ergebnis verschwenden würden, entschlossen wir uns einfach dazu, den Raspberry Pi per LAN-Kabel direkt mit der Fritz!Box zu verbinden. Nachdem dann auch nach über zwei Wochen keine Verbindungsabbrüche registriert wurden, konnten wir dieses Problem als gelöst ansehen und uns wieder anderen Aufgaben des Projekts widmen.

Im Verlauf der weiteren Programmierung der PHP-Skripte stellte sich heraus, dass die PHP-Version 7.3 auf dem Raspberry Pi veraltet war und einige nützliche Funktionen nicht unterstützt. Deshalb wurde die Version 7.3 deinstalliert und anschließend mit PHP 8.0 eine der aktuelleren Versionen installiert. Allerdings lief entweder die Deinstallation der alten oder die Installation der neuen Version nicht fehlerfrei. Nach einigem Suchen in verschiedenen Log-Files stellte sich heraus, dass fälschlicherweise die Socket-Datei „*php7.3-fpm.sock*“ der Version 7.3 vom lighttpd-Server versucht wurde zu öffnen. Diese existierte aber natürlich nach der Deinstallation nicht mehr. Mit PHP 8.0 wurde eine entsprechende, neue Datei „*php8.0-fpm.sock*“ im gleichen Dateipfad „/var/run/php“ bereits erzeugt. In der Konfigurations-Datei „15-fastcgi-php.conf“ im Verzeichnis „/etc/lighttpd/conf-enabled“ ist der Pfad der vom lighttpd-Server zu nutzenden Socket-Datei hinterlegt. Allerdings ist es nicht möglich, diese Datei einfach mit Hilfe eines Texteditors auf dem Raspberry Pi zu verändern. Dafür benötigt es höhere Benutzerrechte. Die Datei lässt sich auch über die Kommandozeile öffnen und bearbeiten.

Dabei ist es wichtig den Zusatz „sudo“ vor jeden Befehl zu schreiben, da somit für den folgenden Befehl die notwendigen höheren Rechte temporär gewährt werden. So war es möglich, in der Datei „15-fastcgi-php.conf“ den Pfad der Socket-Datei auf die aktuelle Datei von PHP-Version 8.0 zu ändern. Im Anschluss musste der lighttpd-Server noch einmal neu gestartet werden, sodass alle Änderungen übernommen wurden.

4.2.3 Grundsätzliche Überlegungen

Bevor es ans konkrete Programmieren verschiedener HTML- oder PHP-Skripte geht, wollte wir uns zunächst überlegen, was denn grundsätzlich auf dem Server passieren soll. Auf jeden Fall benötigen wir ein Skript, das Daten nutzen und weiterverarbeiten kann, die an seine URL¹ übermittelt wurden. Darüber hinaus ist es sicherlich sinnvoll und möglicherweise sogar notwendig, das Bereitstellen von Daten für die Apps in einem separaten Skript zu realisieren. Zusätzliche Skripte für zum Beispiel eine Verschlüsselung der Daten sind gegebenenfalls zu einem späteren Zeitpunkt noch interessant. Auch wenn der GPS-Sensor am Arduino neben den Längen- und Breitengraden noch viele andere Informationen erfasst, reicht es für unser primäres Projektziel aus, nur die Koordinaten an den Webserver zu senden. Um die Koordinaten-Paare bei der Weiterverarbeitung in den Apps unterscheiden zu können, soll der Webserver die Daten noch mit dem aktuellen Zeitstempel verknüpfen.

Sowohl die Daten, die der Arduino an den Webserver übermittelt als auch die Daten, die der Webserver für die Apps wieder bereitstellt, sollen im JSON-Format² übertragen werden. Dieses Format ist einerseits leicht lesbar und andererseits ist es unabhängig von Programmiersprachen und somit universell nutzbar.

Über mögliche Sicherheitsaspekte unserer Datenübertragung dachten wir anfangs nicht wirklich nach, da eine an sich funktionierende Kommunikation zwischen den verschiedenen Teilnehmern erst einmal im Vordergrund stand. Lediglich die Überlegung, wie man verhindern kann, dass nicht jeder beliebige Internetnutzer irgendwelche Daten an unseren Webserver schickt, beschäftigte uns am Anfang des Projekts. Es lässt sich natürlich nicht verhindern, dass andere Geräte versehentlich oder auch absichtlich irgendwelche Daten an unseren Webserver beziehungsweise an ein dort laufendes PHP-Skript senden. Aber wir können dieses Skript so programmieren, dass nur bestimmte empfangene Daten auch weiterverarbeitet werden. Um dies zu realisieren, muss jedes Gerät eine von uns manuell vergebenen Geräte-Identifikationscode³ zu-

¹Uniform Resource Locator

²JSON = JavaScript Object Notation

³nachfolgend Geräte-ID

sammen mit den Nutzdaten verschicken. Diese Geräte-IDs sind 20-stellige, alphanumerische Werte, die wir einmal zufällig generiert haben und die in der txt-Datei „*allowed_devices.txt*“ lokal auf dem Raspberry Pi gespeichert sind. So kann später darauf zugegriffen werden und ein Abgleich erfolgen, ob die mitgesendete Geräte-ID bei uns bekannt ist. Wenn dies nicht der Fall ist, sollen diese Daten nicht weiterbearbeitet werden.

Es ist klar, dass auch diese Sicherheitsvorkehrung nicht besonders stark ist, solange die Geräte-IDs unverschlüsselt und quasi frei zugänglich auf dem Server gespeichert sind. Aber es ist ein erstes Mittel, um zumindest einfache Störversuche zu unterbinden und auch uns selbst vor möglichen Übertragungsfehlern zu schützen.

Es ist sehr wahrscheinlich, dass nicht alle Projektteile mit der gleichen Geschwindigkeit bearbeitet werden und somit einzelne Teile unterschiedlich schnell in einem testbaren Stadium sind. In diesem Fall betrifft das die GSM-Sendeeinheit des Arduinos und die einzelnen Skripte des Webservers. Damit grundsätzliche Funktionalitäten der Skripte möglichst schon während der Entwicklung getestet werden können, ohne dass erst auf den Projektteil des Arduinos gewartet werden muss, wollten wir eine einfache Anwendung für einen Windows-PC entwickeln. Die Entscheidung fiel hier relativ schnell auf die Programmiersprache Python, da wir diese auch schon zuvor in einigen Studienprojekten benutzt hatten und sie für den hier gedachten Einsatzzweck eine einfache und schnelle Umsetzung verspricht.

4.2.4 Testprogramme

Im Verlauf der Programmierung der PHP-Skripte auf dem Webserver wurden insgesamt drei Python-Skripte geschrieben, um die Funktionen auf dem Webserver zu testen.

Das erste Python-Skript heißt „Senden.py“. Mit ihm soll getestet werden, ob gesendete Daten beim Webserver ankommen und auch weiterverarbeitet werden können. Dafür werden neben einer gültigen Geräte-ID auch je eine Variable für den Breitengrad (Latitude) und den Längengrad (Longitude) mit frei gewählten Startwerten deklariert. Ebenso wird die URL des empfangenen Skripts auf dem Webserver („http://retrorbit.spdns.de/get_Data.php“) als Variable gespeichert. Der restliche Code ist eine endlose while-Schleife, in der die aktuellen Daten ins JSON-Format gebracht und dann mittels HTTP-POST an die gespeicherte URL gesendet werden. Danach folgt noch eine kurze Wartezeit von 5 Sekunden sowie eine Inkrementierung der Latitude und eine Dekrementierung der Longitude jeweils um den Wert 1.

Im zweiten Python-Programm mit dem Namen „Abfragen.py“ werden inner-

halb einer endlosen while-Schleife die aktuellen Werte vom Webserver bzw. von der Webseite „http://retrorbit.spdns.de/provide_Data.php“ per HTTP-GET abgefragt und die Daten im JSON-Format ausgegeben.

Das dritte Python-Skript heißt „Störer.py“ und gleicht dem „Senden.py“-Skript. Mit diesem sollten zeitgleich entweder beliebige oder auch vom Format her gleiche Werte an das „get_Data.php“-Skript gesendet werden. So wollten wir überprüfen, ob die Sicherheitseinrichtung mit den Geräte-IDs tatsächlich funktioniert.

4.2.5 Erstellen der PHP-Skripte zum Empfangen und Bereitstellen von Daten

Erläuterungen zum Code

Bei allen drei übermittelten Parametern, also Geräte-ID, Latitude und Longitude, wird immer eine Prüfung durchgeführt, bevor deren Werte tatsächlich in Variablen des PHP-Skripts geschrieben werden. Die Struktur dieser Prüfung ist bei allen drei Parametern identisch, bis auf den entsprechenden Bezeichner und den Variablennamen. Die Variable übernimmt nur dann den Wert der \$POST Daten, wenn diese einerseits von „null“ verschieden sind und andererseits nicht leer sind. Dies wird realisiert, indem in einer if-Abfrage das Ergebnis der „`isset`“-Funktion mit der Bedingung, dass der \$POST Wert ungleich leer ist, über ein logisches UND verknüpft sind. Die „`isset`“-Funktion prüft dabei, ob die Variable existiert und sich von „null“ unterscheidet. Wenn diese Bedingungen allerdings nicht erfüllt sind, wird im else-Zweig mittels goto-Anweisung zum Ende des Skriptes gesprungen und eine Fehlermeldung ausgegeben. Die aktuell empfangenen Daten werden dann nicht weiterverarbeitet, sodass immer noch der vorherige Standort von der Seite „http://retrorbit.spdns.de/provide_Data.php“ abgefragt werden kann.

Mit der Funktion `str_contains` kann überprüft werden, ob ein String in einem anderen String enthalten ist. In unserem Fall wird so gecheckt, ob die Geräte-ID (`$device_id`) im String `$all_devices` enthalten ist, welcher alle Geräte-IDs aus der Datei „`allowed_devices.txt`“ beinhaltet.

Die aus den \$POST Variablen übernommenen Werte für Latitude und Longitude werden dann zusammen mit den aktuellen Datumsinformationen in ein Array mit der Bezeichnung `$koor` kopiert. Anschließend wird dieses Array noch in JSON codiert und als `$json_koor` gespeichert. Wichtig ist an dieser Stelle der zweite Parameter des `json_encode` Befehls. Wenn hier nichts angegeben wird, werden alle enthaltenen Werte als String interpretiert. Für das Datum und die Uhrzeit ist das sinnvoll, für die Koordinaten hingegen eher nicht. Damit deren Werte auch als Zahlenwerte interpretiert werden,

ist der Parameter „JSON_NUMERIC_CHECK“ zu setzen, welcher automatisch Zahlenformate erkennt und ins JSON Format übernimmt. Im letzten Schritt des „get_Data.php“ Skripts wird der aktuelle Inhalt der Datei „Koordinaten.txt“ mit den Daten der JSON-codierten Variable \$json_koor überschrieben.

Die Trennung für Datenempfang und Datenbereitstellung hat folgenden wichtigen, technischen Hintergrund. In dem Moment, in dem ein http-Request an das „get_Data.php“ Skript gestellt wird – egal ob Daten mittels POST dorthin übermittelt werden oder nur die bereitgestellten Daten mittels GET abgefragt werden sollen – erst einmal ausgeführt. Da bei einer reinen GET-Abfrage aber natürlich keine Daten mitgeschickt werden, wird der entsprechende PHP-Code zur Datenverarbeitung übersprungen und die Koordinaten mit „null“ gefüllt. Genau dies wird danach auch ausgegeben und von dem http-GET-Request ausgelesen. Somit ist es nicht möglich innerhalb eines PHP-Skripts Daten zu empfangen und bereitzustellen.

In der nachfolgenden Abbildung 4.6 ist der grundsätzliche Programmablauf der beiden PHP-Skripte auf dem Webserver dargestellt, die die Datenverarbeitung sowie die Datenbereitstellung realisieren.

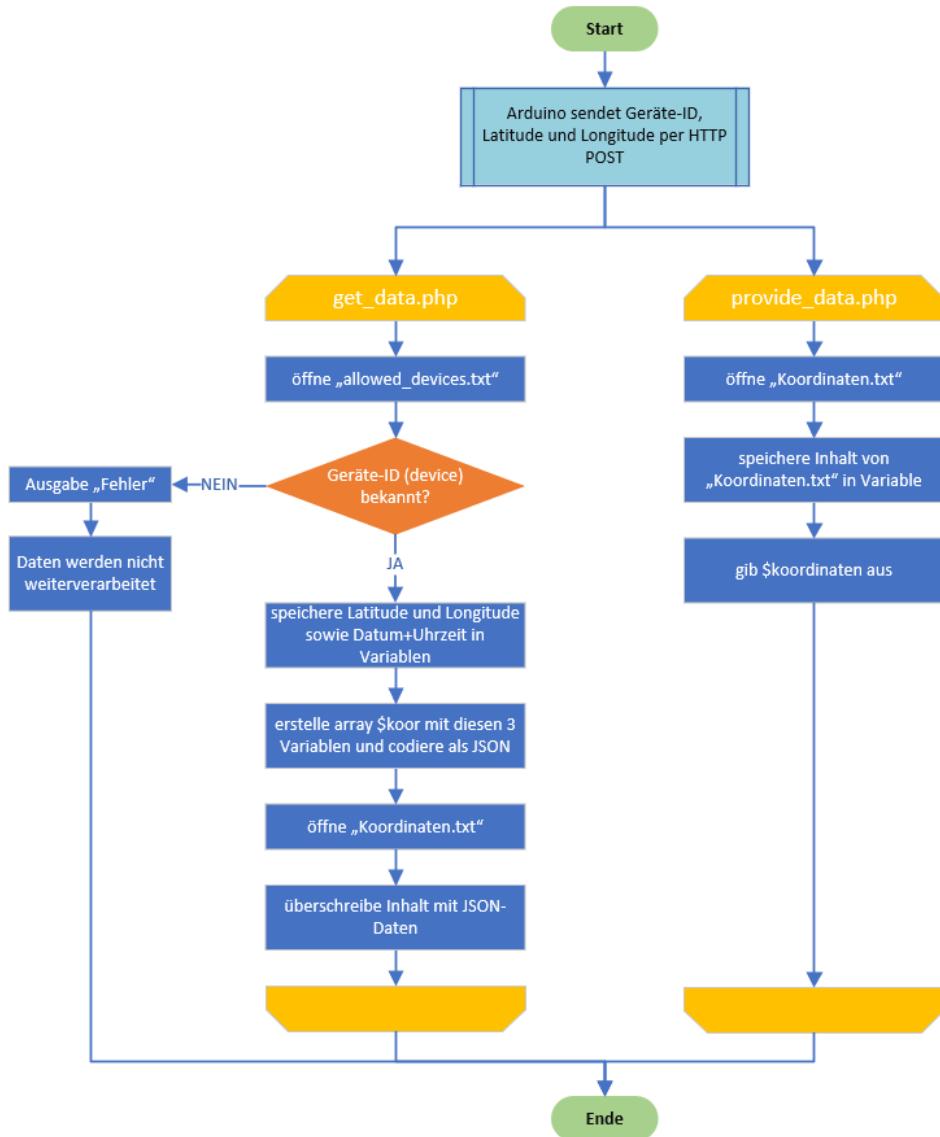


Abbildung 4.6: Programmablaufplan Webserver

Probleme beim Senden vom Arduino

Nachdem das Skript „`get_Data.php`“ in einem funktionsfähigen Zustand war und auch das Testen mit dem Python-Programm „`Senden.py`“ keine Fehler ergab, konnte zum nächsten Entwicklungsschritt übergegangen werden. Dieser besteht darin, die Kommunikation vom Arduino über das GSM-Modul zum Webserver herzustellen, sodass dort die gesendeten Daten auch weiterverarbeitet werden können.

Im Prinzip müssen im Programm auf dem Arduino nur die Koordinaten ins JSON-Format angeordnet werden und per http-POST an die URL

„http://retrorbit.spdns.de/get_Data.php“

gesendet werden. Es erforderte einige Versuche, bis das überhaupt funktionierte, allerdings konnten die Daten auf dem Webserver nicht richtig verarbeitet werden. Dies äußerte sich darin, dass in der Datei „Koordinaten.txt“ für Latitude und Longitude der Wert „null“ eingetragen war. Das Auslesen der Daten mittels POST auf der Seite des Webservers hat also nicht funktioniert. Die Ursache hierfür lag bei der Datenformatierung des Arduinos bzw. dem AT-Befehl zum Senden der Daten via GSM. Dabei werden die Daten immer als String versendet unabhängig von der ursprünglichen Form, welche dann als String angesehen wird. Deshalb funktionierten im PHP-Skript die `$_POST` Befehle für die einzelnen Parameter nicht mehr.

Mit diesem Wissen wurde das „get_Data.php“ Skript nochmals angepasst und als „get_Data2.php“ abgespeichert. Dieses ist im Prinzip genauso aufgebaut wie die ursprüngliche Version, jedoch wurden die einzelnen `$_POST` Befehle gegen einen normalen Arraybezeichner (`$POST_data`) ausgetauscht. Dafür wird nach den Deklarationen der Dateinamen in Codezeile 6 (*siehe Anhang*) mit nur einem `$_POST` der vom Arduino gesendete String ausgelernt und in der Variable `$Arduino_Daten` gespeichert. Es folgt eine JSON-Decodierung, sodass nun die einzelnen Parameter (*Geräte-ID und die beiden Koordinaten*) im Array `$POST_data` gespeichert und referenzierbar sind. Der Zugriff auf die einzelnen Werte geschieht jetzt also nicht mehr über die `$_POST` Befehle, sondern per Referenzierung auf das Array.

4.2.6 Verschlüsselung der Daten

Hintergrund

Im aktuellen Zustand des Webservers und unserer Kommunikation allgemein werden die Koordinaten unverschlüsselt und damit für jedes Gerät bzw. jeden Nutzer klar lesbar auf der Seite

http://retrorbit.spdns.de/provide_Data.php

dargestellt. Sowohl aus datenschutztechnischer Sicht als auch im Interesse des späteren Kunden ist dies natürlich nicht wünschenswert. Deshalb ist es nötig, die bereitgestellten Daten zu verschlüsseln. Der wichtigste Punkt an dieser Stelle ist, dass die verwendeten Chiffrier- und Dechiffrieralgorithmen bzw. Bibliotheken sowohl von PHP als auch von den beiden Programmiersprachen der Apps unterstützt werden.

Symmetrische und asymmetrische Verschlüsselungsverfahren

Die Verschlüsselung unserer Nutzdaten, also Latitude, Longitude sowie die dazugehörigen Datumsinformationen, soll mittels eines symmetrischen Verschlüsselungsverfahren realisiert werden. Symmetrische Verschlüsselungsverfahren sind einerseits sehr performant, selbst bei großen Datenmengen, und andererseits sind sie nur unter unverhältnismäßig großem Aufwand zu entziffern. Ein weit verbreiteter, frei verfügbarer Standard stellt dabei der „Advanced Encryption Standard“, kurz AES, den auch wir bei unserem Projekt nutzen wollen. Bei symmetrischen Verschlüsselungsverfahren wird für das Ver- und Entschlüsseln ein und derselbe Schlüssel benutzt. Beim AES kann dieser Schlüssel wahlweise 128, 192 oder 256 Bit lang sein, wobei gilt: Je länger der Schlüssel ist, desto höher ist auch die Sicherheit der verschlüsselten Daten. Bei der symmetrischen Verschlüsselung besteht aber ein wesentlicher Nachteil, denn der Schlüssel muss beiden Teilnehmern der Kommunikation bekannt sein. Wenn die Chiffrierung also auf Gerät A durchgeführt wird, muss neben den nun verschlüsselten Daten auch der AES-Schlüssel zu Gerät B übertragen werden. Je nach individuellen Gegebenheiten lässt sich dies ggf. auf einem zweiten Kommunikationsweg manuell durch den Menschen realisieren. In der Praxis ist das aber meist nicht möglich. Deshalb wird hier zusätzlich noch ein asymmetrisches Verschlüsselungsverfahren angewandt, mit welchem der AES-Schlüssel chiffriert wird. Diese Kombination beider Verschlüsselungsverfahren wird auch als Hybride Verschlüsselung bezeichnet.

Bei asymmetrischen Verschlüsselungsverfahren hat jeder Kommunikationspartner ein eigenes Schlüsselpaar bestehend aus einem öffentlichen und einem geheimen Schlüssel. Der öffentliche Schlüssel wird immer für die Verschlüsselung von Daten beim jeweilig anderen Teilnehmer genutzt und der geheime Schlüssel für die Entschlüsselung der empfangenen Daten auf dem eigenen Gerät. Der Vorteil ist hier, dass der öffentliche Schlüssel bei der Übermittlung an den Kommunikationspartner nicht extra geschützt werden muss. Eines der bekanntesten asymmetrischen Verschlüsselungsverfahren ist das nach seinen Entwicklern benannte RSA-Verfahren. Da dieses aber schon veraltet ist, werden wir mit der Elliptic Curve Cryptography ein zeitgemäßeres Verfahren nutzen.

Implementierung der Verschlüsselung auf dem Webserver

Der komplette Verschlüsselungsprozess ist in „encryption.php“ realisiert, also sowohl die Verschlüsselung der Nutzdaten als auch die Chiffrierung des AES-Schlüssels. Für die Übermittlung des öffentlichen Schlüssels wird zusätzlich das Skript „key_from_app.php“ erstellt. Allerdings ist die Implementierung noch nicht im Zusammenspiel mit einer der Apps getestet worden, da es hier Verzögerungen im Projekt gab, sodass das ganze Verschlüsselungsthema erst

in naher Zukunft tatsächlich funktionsfähig sein wird. Für die symmetrische Verschlüsselung nutzen wir in PHP die Funktionen der „OpenSSL“ Bibliothek und für die asymmetrische Verschlüsselung kommen die Funktionen der „Sodium“ Bibliothek zum Einsatz, welche auch für die von uns verwendeten Programmiersprachen der Apps zur Verfügung stehen.

Als erstes werden die zu verschlüsselnden Nutzdaten (Koordinaten und Datum mit Uhrzeit) aus der Datei „Koordinaten.txt“ geladen. Anschließend wird das Verschlüsselungsverfahren auf AES-192-CTR festgelegt und ein entsprechend langer AES-Schlüssel generiert. CTR steht dabei für Counter Mode, welcher zusätzlich einen Initialisierungsvektor erfordert, welcher die gleiche Länge wie der AES-Schlüssel haben muss. Nachdem dieser Initialisierungsvektor generiert wurde, folgt die eigentliche Verschlüsselung unserer Nutzdaten mit der Funktion „openssl_encrypt“. Weiter geht es mit der Generierung des Schlüsselpaares des Webservers für die asymmetrische Verschlüsselung mit der Funktion „sodium_crypto_box_keypair()“.

Nun fehlt noch der öffentliche Schlüssel aus der App. Dieser wird aus der Datei „app_key.txt“ geladen. Diese Datei wurde im Skript „key_from_app.php“ generiert und lokal auf dem Server gespeichert. Dieses Skript ist im Prinzip genau so aufgebaut wie das „get_Data.php“ Skript, nur dass es entsprechend nur auf die beiden Parameter „Geräte-ID“ und „keypair2_public“ abgestimmt ist. So findet auch hier zuerst eine Überprüfung statt, ob die gesendeten Daten von einem bekannten, erlaubten Gerät stammen.

Nachdem dann also alle notwendigen Schlüssel auf dem Webserver vorhanden sind, kann der AES-Schlüssel mit der Funktion „sodium_crypto_box()“, welche noch eine zusätzliche Zufallszahl (nonce) benötigt, verschlüsselt werden. Diese Zufallszahl wurde zuvor erzeugt und muss auch an den Empfänger für die Entschlüsselung übermittelt werden. Im nächsten Schritt werden wieder alle zu sendenden Variablen in ein Array (\$datenpaket) kopiert, siehe untenstehende Abbildung. Dabei ist es wichtig, dass alle Variablen bis auf die verschlüsselten Nutzdaten noch einmal mit base64 codiert werden, da sonst bei der JSON-Codierung des Arrays im nächsten Schritt nicht alle Zeichen der einzelnen Strings korrekt verarbeitet werden können. Bevor diese Strings dann allerdings in den Apps genutzt werden können, muss dort wieder eine base64 Decodierung erfolgen, um wieder das BYTES Format zu erhalten.

Im letzten Schritt werden nun die JSON codierte Variable \$json_data in der Datei „Datenpaket.txt“ lokal abgespeichert. Diese Datei kann dann genau wie zuvor die Datei „Koordinaten.txt“ von dem Skript „provide_Data.php“ geöffnet und deren Inhalt ausgegeben werden. So können die Entschlüsselungsparameter zusammen mit den verschlüsselten Daten von den Apps abgefragt werden.

```
46 $datenpaket = array(
47     "key1pub" => base64_encode($keypair1_public),
48     "nonce" => base64_encode($nonce),
49     "AES_key" => base64_encode($encrypted_AES_key),
50     "AES_iv" => base64_encode($iv),
51     "Nutzdaten" => $encrypted_Nutzdaten
```

Abbildung 4.7: Datenpaket Webserver

Der gesamte Ablauf der Skripte auf dem Webserver ist in der nachfolgenden Übersicht 4.8 zur besseren Nachvollziehbarkeit noch einmal schematisch dargestellt.

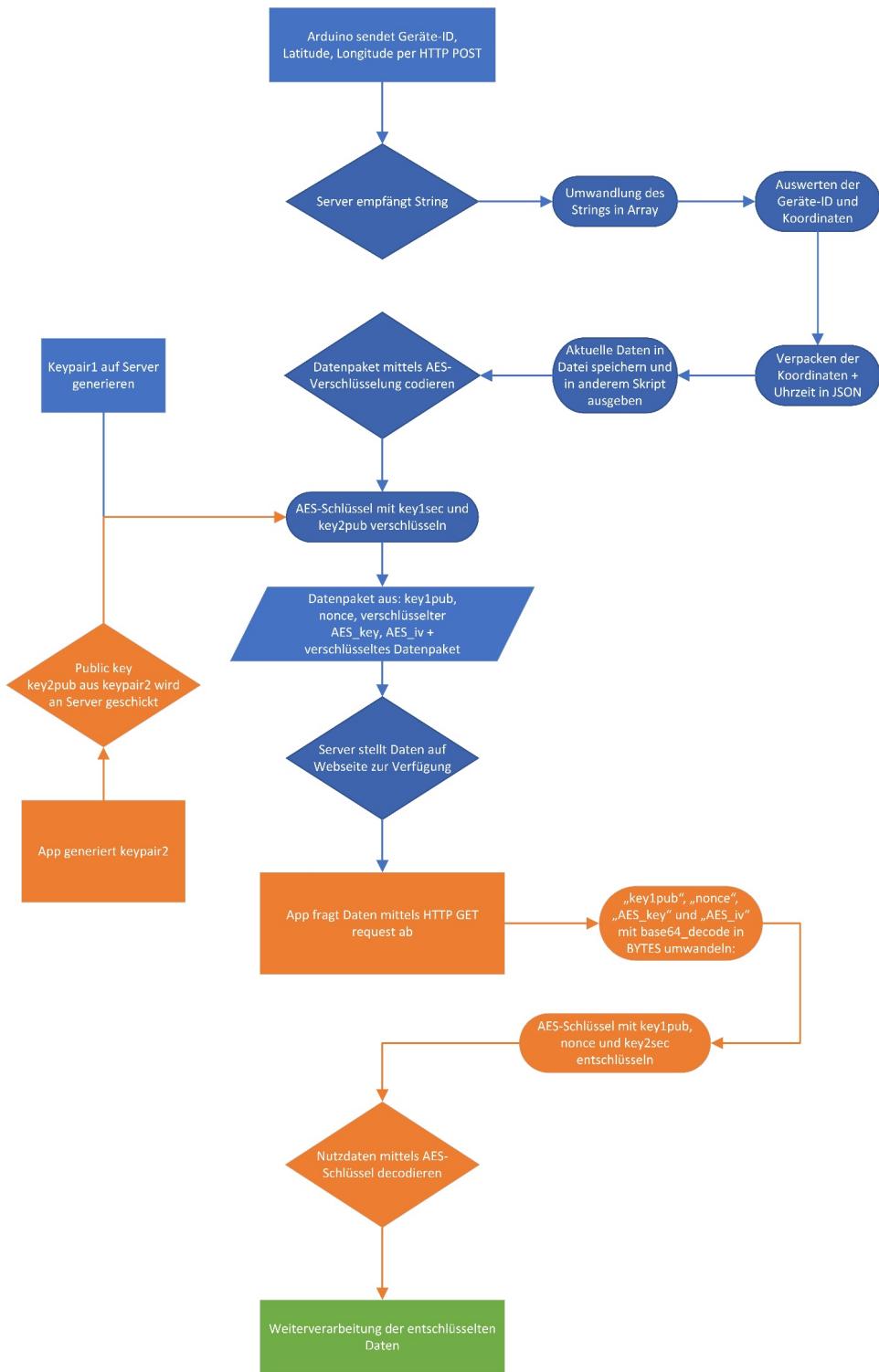


Abbildung 4.8: Skriptablauf Webserver

4.3 Android

Dieses Kapitel enthält wesentliche Informationen und Schritte, die für die Entwicklung einer Android-App erforderlich sind. Als Erstens wird das Konzept vorgestellt, was in einer App-Entwicklung beachtet werden muss. Zweitens werden die Grundlagen vorgestellt. Anschließend wird die Implementierung basierend auf dem Konzept des Projekts beschrieben.

4.3.1 Mobile Apps

Mobile Anwendungen werden unter Berücksichtigung von Faktoren wie Benutzerfreundlichkeit, Zugänglichkeit, Handbuch, Design, User Interface und Interaktion mit der User Experience entwickelt. Es werden dabei unter anderem Features und Services genannt, die einen echten Mehrwert bieten und die Anwendung attraktiv und innovativ erscheinen lassen. Um das volle Potenzial der App aufzudecken und konzeptionell zu benennen, ist es wichtig, mögliche zukünftige App-Eigenschaften und -Funktionen über den Prototyp hinaus vorzuschlagen.

Bei der Umsetzung der mobilen Applikation sollten folgende Aspekte betrachtet werden:

- Nützlich- und Nutzbarkeit
- Effizienz und Effektivität
- Zuverlässigkeit
- Saubere Interaktion
- Dynamik und Flexibilität

4.3.2 Android Studio

Android ist das Betriebssystem und die Softwareplattform für mobile Geräte wie Smartphones, Uhren, Fernseher oder Schnittstellen zur Kommunikation mit Fahrzeugen. Android gehört zu Google und ist das am weitesten verbreitete Betriebssystem von Smartphones. Der weltweite Marktanteil von Smartphones mit Android als Betriebssystem liegt bei etwa 85 %.

Für die Entwicklung von Android Apps kann die IDE Android Studio verwendet werden, dabei handelt es sich um die offizielle, integrierte Entwicklungsumgebung für Android. Die IDE hilft beim Entwerfen und Entwickeln von Android-Anwendungen mit vielen Hilfsfunktionen. Das Abhängigkeitsmanagement wird durch eine gute Integration mit Gradle, dem für Android-Anwendungen verwendeten Build-System, unterstützt. Neue Abhängigkeiten

sind normalerweise direkt unter dem Quellcode-Editor verfügbar, es besteht daher keine Notwendigkeit, Build-Skripte manuell zu bearbeiten. Android Studio hat auch viele andere sehr nützliche Funktionen, wie z. B. eine integrierte Software-Versionskontrolle, effektive Debugging-Tools und einen sehr intelligenten Code-Inspektor[pro].

4.3.3 Flutter

Was ist Flutter? Flutter ist ein von Google entwickeltes Framework, um nativ kompilierbare Anwendungen anhand einer einzigen Codebasis zu schreiben. Flutter ist ein Open-Source-Framework von Google zur Entwicklung grafischer Anwendungen, die auf Mobilgeräten, Browsern und Computern ausgeführt werden. Der wichtigste Aspekt sind die verschiedenen gemeinsamen Codebase Plattformen welche mit nativer Geschwindigkeit laufen. Das bedeutet, dass die gleiche Anwendung also auch ohne viele Änderungen auf eine andere Plattform kompiliert werden kann. Flutter bietet zudem umfangreiche Bibliotheken mit vorgefertigten UI-Elementen. Datenströme sind sehr einfach zu implementieren und stellen sicher, dass Benutzer immer auf dem neuesten Stand sind.

Die Verwendung des Flutter-Frameworks bietet mehrere Vorteile. Wie bereits erwähnt, liegt ein besonderer Fokus darauf, eine einzige Codebasis für viele Endgeräte zu verwenden. Die Codebasis wird durch die sogenannte Tree-Form aufgebaut. Das spart im Idealfall Zeit und somit natürlich auch Geld. Mit der “Hot Reload“ Funktion wird es Entwicklern ermöglicht, schnell und einfach Schnittstellen zu erstellen, neue Funktionen hinzuzufügen und Fehler schneller zu finden.[Lan].

4.3.4 Dart

Flutter läuft auf Dart, der hauseigenen Programmiersprache von Google. Das ursprüngliches Ziel war es, JavaScript zu ersetzen und **die** neue Sprache zu werden. Dart ist der moderne Nachfolger von JavaScript und wurde wie die beliebte Web-Skriptsprache entwickelt, um direkt im Browser als Webanwendung ausgeführt werden zu können.

Die nennenswerten Vorteile von Dart sind:

- leicht zu lernende Sprache
- gute Dokumentation
- hoher Leistungsfaktor
- verständliche Syntax
- leistungsstarke Tools zur Unterstützung der App-Entwicklung

4.3.5 Konzept: SchrödingersAlarm

Beim Projekt ist die Mobile Android-App eine wichtige Komponente des Systems. Die App soll für jedes Android Smartphone kompatibel sein. Nachfolgend wird die Funktionalität der App durch die beiden Abbildungen 4.10 und 4.9 dargestellt:

Standard Fall

Die App ist mit dem Webserver verbunden. Alle Informationen, die vom Webserver kommen, werden in der App angezeigt. Die Koordinaten sollen den Standort des Fahrzeugs auf einer Karte in der App anzeigen.

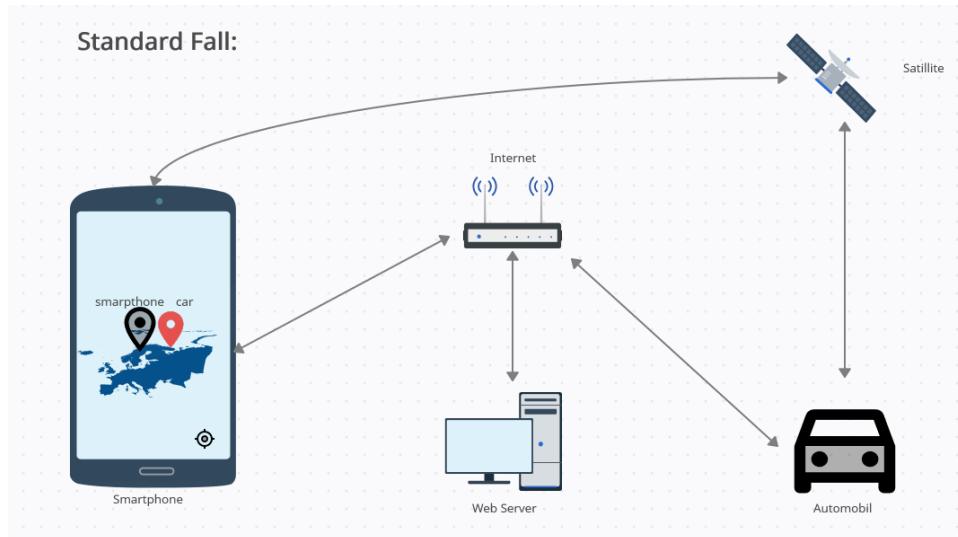


Abbildung 4.9: Konzept der Android App im Standard Fall

Alarm Fall

Im Falle das sich der Standort des Automobils ändert, wird ein Alarm auf dem Handy ausgelöst und der Nutzer bekommt eine Push-Benachrichtigung.

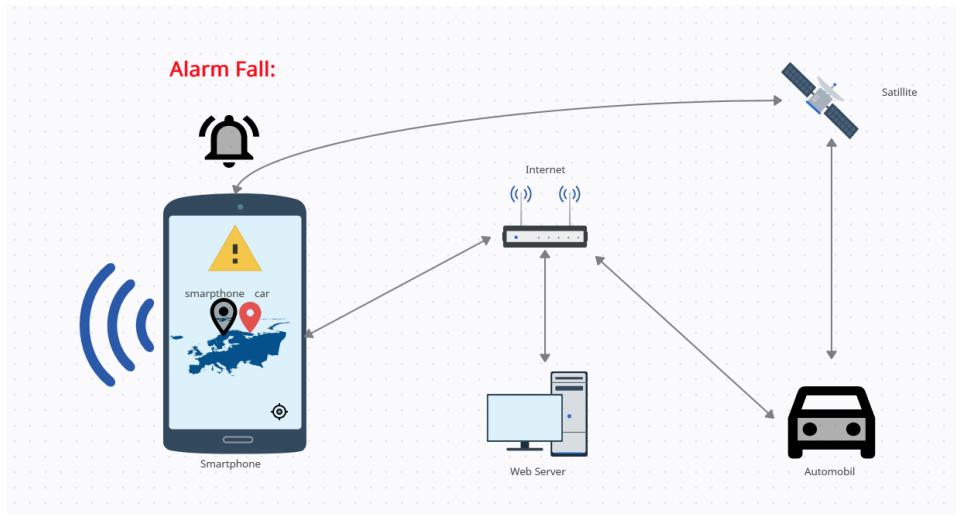


Abbildung 4.10: Konzept der Android App im Alarm Fall

4.3.6 Implementierung

Flutter Funktionalität

Flutter ist sehr einfach zu verstehen und verkürzt die Lernkurve erheblich. Die Grundidee lautet: "Alles ist ein Widget". Es ist dem Entwickler völlig selbst überlassen, wie er sein Projekt strukturieren und hinsichtlich Architektur- und Entwurfsmuster logisch aufbauen möchte. Der Einstiegspunkt zu Flutter ist die main.dart-Datei oder die zentrale main()-Methode. Von dort aus ist es möglich, sich gegenseitig zu verschachteln und zu importieren. Die Entwicklung von Flutter beinhaltet hauptsächlich nur Widgets, weil alles hineingeschrieben wird. Sogar ein einfacher Text "Teil" wird als Widget festgelegt.

In Flutter gibt es ebenfalls eine Baumstruktur "*Widget Tree*". Jede betriebssystemspezifische Komponente, die man in Flutter verwenden möchte, muss zunächst von einem Flutter-Entwickler nativ entwickelt werden. Dies ist jedoch nur dann erforderlich, wenn die Anwendung auch der nativen Anwendung ähnlich sein soll. Aus den Basis-Komponenten, der verschiedenen Pakete, konnte man eigene Widgets schreiben. So kann sich ein eigener, abgerundeter Button beispielsweise durch eine Kombination aus RawMaterialButton, Text und Icon-Widgets darstellen lassen. In Flutter werden Widgets als einfache Klassenvariablen über den Konstruktor der Klasse übergeben und können dort beliebig verwendet werden. Durch die Klassenvariablen lassen sich Komponenten in ihrem Aussehen sowie ihrer Funktionalität individualisieren und anpassen [Vog].

Es gibt zwei Arten von Komponenten in Flutter, die im Code explizit als solche benannt werden. Erstens „*Statful Widgets*“, die einen Zustand beibehalten oder verwalten können. Andererseits gibt es „*Stateless Widgets*“, die nur als statische, einfache Widgets dienen. Ein zustandsbehaftetes Widget erfordert immer eine zugeordnete `createState()`-Funktion. Eine Methode, die ein Zustandsobjekt (*private Klasse*) erstellt, welche wiederum den Widget-Status verwaltet. Abgesetzt durch Aufruf von `setState()` ruft die Methode `build()` auf, um die Komponente neu zu zeichnen. Die `build()`-Methode wird bei dem Umwandeln eines Stateless Widgets in ein Stateful Widget, in die private State-Klasse ausgelagert.

Das Styling in Flutter ist grundlegend speziell. Wenn man das Widget zentrieren möchten, könnte man beispielsweise das Center-Widget verwenden. Um ein Widget von anderen Widgets zu entfernen, muss es dafür mit einem Padding-Widget umschlossen werden. Als einfaches Beispiel kann ein Text-Widget von einem Center-Widget umgeben werden, welche wiederum von einem Container-Widget umgeben sind.

“Center“, “Raw“, “Column“, “Container“-Widgets und viele andere Widgets sind die Bausteine in der Entwicklung von Flutter App.

4.3.7 Implementierung SchrödingersAlarm App

Login Menü

Das Starten der App ist hier Standard, wie bei jeder App. Zuerst kommt die Willkommensseite, in der ein Login Menü erscheint. Jedes Alarmsystem soll eine eigene IMEI und dazugehörigen PIN Code haben und den Kunden mitgeliefert werden. Diese beiden Informationen sind in der Datenbank der App hardcodiert eingefügt, da dieses Feature noch aussteht. Das Logo des Produkts wird oben angezeigt und unten sind zwei Eingabefelder definiert, in denen man die Login-Informationen(IMEI/PIN) eingeben kann. Der Login-Prozess hat einen genau definierten Mechanismus. In den beiden Feldern muss eine bestimmte Anzahl an Zeichen eingegeben werden, ansonsten wird es nicht akzeptiert. Die IMEI muss 15 Ziffern beinhalten. Der PIN Code muss 6 Ziffern beinhalten. Nur bei einer korrekten Dateneingabe wird man zu der nächsten Seite geleitet. Andernfalls erscheint eine rote Fehlermeldung, die um korrekte Informationen bittet.

Wie vorher beschrieben ist, wird die Horizontale aufgeteilt. Der obere Teil ist das importierte Bild vom Logo und im unteren Teil sind die genannten Eingabefelder, siehe Abbildung 4.11] Diese Aufteilung wird mit einem “Column“ Widgets definiert. Der unter Teil wird ebenfalls in viele Widgets aufgeteilt.

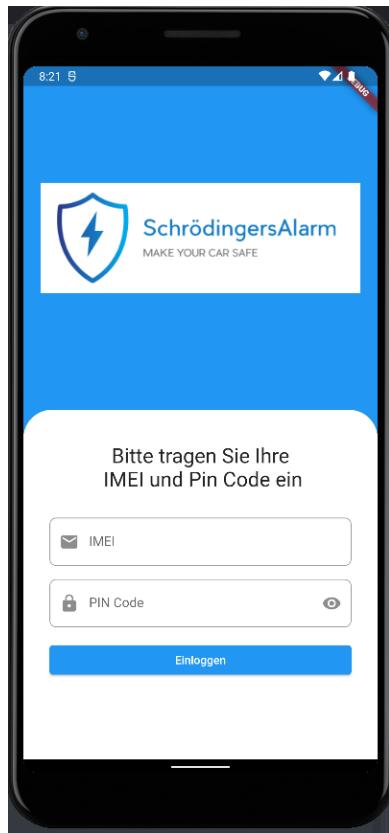


Abbildung 4.11: Android App *Login Seite*

Hauptseite

Nach erfolgreichem Login leitet die App den Nutzer auf die Hauptseite. Auf der Hauptseite gibt es eine horizontale Appbar mit 3 Kategorien (*Home/Status/Maps*), links oben ein Menu Item für Support und rechts oben ein Logout item für das Ausloggen aus der App. Im Hometab wird das Logo und der aktuelle Standort angezeigt. Ist die App aktiviert, wird ein grüner Kreis mit der Meldung *“your car is safe”* angezeigt. Ist sie deaktiviert, wird ein roter Kreis mit der Meldung *“Dangerous situation”* ausgegeben.

Durch das Drücken auf „Status“ in der horizontalen Appbar, werden die Informationen, die vom Webserver kommen, in einem Text Widget angezeigt. Wichtig ist hier, auf eine Webserver Verbindung zu prüfen. Besteht eine Verbindung, soll ein grünes Verbindungs-Icon angezeigt werden, andernfalls ein Icon „Nicht verbunden“.

Das dritte Item in der horizontalen Appbar ist die Karte. In dieser wird die Google Maps Anwendung in der App geöffnet. Die Koordinaten, die von

dem Webserver kommen, werden dann mit einem Marker in der Google Maps Karte markiert, wie in Abbildung 4.12 zu sehen. Dazu kommt der gespeicherten Parking Standort in einem anderen Marker angezeigt.

Falls die Koordinaten vom Webserver zu sehr von den gespeicherten Koordinaten abweichen, bekommt der Nutzer eine Push-Benachrichtigung. Bei der Umrechnung werden Longitude und Latitude von beiden Koordinaten abgezogen und die Differenz berechnet. Falls einer der beiden Werte höher als der festgelegten Wert ist, erstellt die App eine Push-Benachrichtigung.

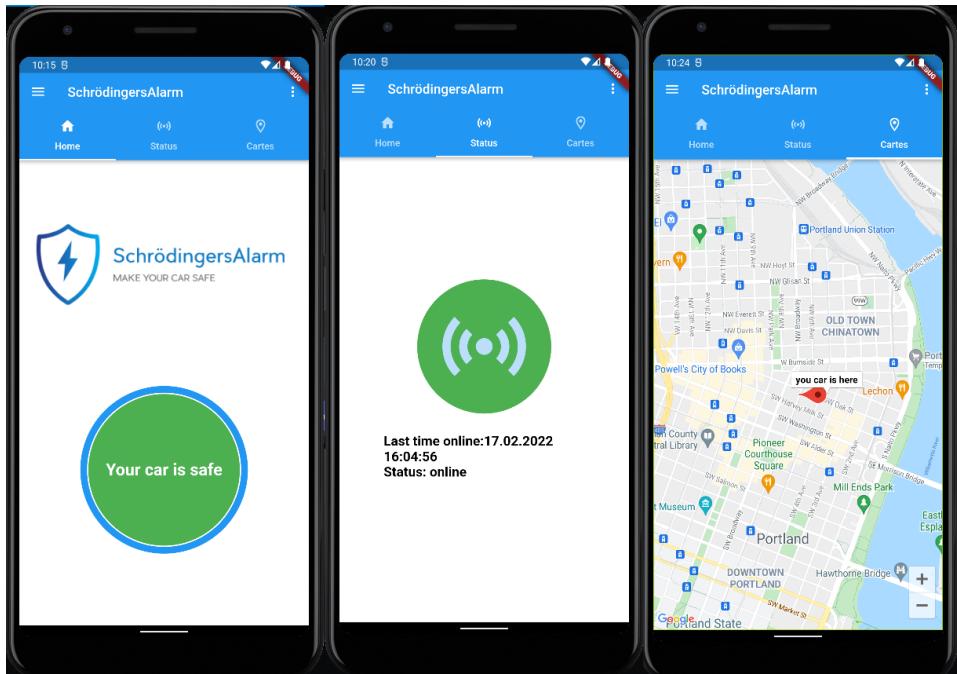


Abbildung 4.12: Android App Seitenanichten

4.4 App - iOS

Um möglichst viele Endnutzer für unser Produkt zu erreichen, war es uns wichtig, für die zwei verbreitetsten Smartphone Betriebssysteme Apps bereitzustellen. Für die Entwicklung der iOS-App wurde die Sprache SwiftUI unter Nutzung der kostenlosen IDE Xcode genutzt. Da es sich hier um neues Terrain handelte, wurde mehr Zeit zur Einarbeitung und späteren Problemfindung benötigt, als anfänglich gedacht, weshalb diese App eher einem Design-Prototypen entspricht.

4.4.1 Xcode

Mit Xcode können Programme für macOS, iPadOS, iOS, watchOS und tvOS entwickelt werden. Diese IDE ist für die Programmiersprachen Swift und Objective-C unter Verwendung der Cocoa-Frameworks vorgesehen, wobei aber auch die Programmiersprachen C und C++ verwendet werden können. Aufgrund seiner Modularität ist es jedoch auch möglich Programme in anderen Sprachen zu schreiben – beispielsweise in Java, Ruby, Perl oder Pascal. [Ahm20]

Wie von Apple gewohnt, bietet Xcode eine übersichtliche wie aufgeräumte Benutzeroberfläche. Beim Starten eines neuen Projektes hilft ein Assistent bei den richtigen Einstellungen. Neben dem Programmieren von Apps unterstützt das Tool auch das visuelle Gestalten der späteren Benutzeroberfläche sowie das Testen und Debuggen von Software. [Ahm20]

Die IDE kommt mit einem grafischen Interface-Design-Tool namens Interface Builder, über das Benutzeroberflächen entwickelt werden können. Menüs, Fenster, Controls und weitere visuelle Elemente lassen sich gestalten oder aus vorhandenen Objekten, aus einer integrierten Bibliothek ziehen. Alternativ können natürlich auch eigene entwickelt werden. Die Anbindung bestimmter Elemente an den dahinterliegenden Code (*in Form von ausführbaren Aktionen beispielsweise*) ist mit dem Interface Builder ebenfalls möglich.[Ahm20]

Für Xcode wurde sich einerseits entschieden, um später keine Komplikationen mit der Kompatibilität auf dem Endgerät zu riskieren. Bei der Recherche, welche IDE wir nutzen möchten, wurde relativ schnell klar, dass dies bei anderen kostenlosen IDE Tools passieren könnte. Andererseits überzeugten die vielen hinterlegten Simulatoren - von iPad, über Macbook bis hin zu sämtlichen iPhone Generationen ist alles dabei - welche ein Arbeiten mit schnellem Feedback auf unterschiedlichen Geräten ermöglichen, ohne diese physisch Zuhause haben zu müssen.

4.4.2 SwiftUI

Bei SwiftUI handelt es sich um ein GUI-Framework aus dem Hause Apple, welches auf MVVM⁴ basiert. Es bietet Ansichten, Steuerelemente und Layoutstrukturen zum Deklarieren der Benutzeroberfläche einer App. Das Framework bietet Event-Handler für die Bereitstellung von Taps, Gesten und anderen Arten von Eingaben an der App sowie Tools zur Verwaltung des Datenflusses von den Modellen der App bis hin zu den Ansichten und Steuerelementen, die Benutzer sehen und mit denen sie interagieren.[Incb]

⁴Model View ViewModel = ein Entwurfsmuster, zur Trennung von Darstellung und Logik der UI

Die App-Struktur wird mithilfe des App-Protokolls definiert und füllt sie mit Szenen, welche die Ansichten enthalten, aus denen die Benutzeroberfläche der entstehenden App besteht. Es können eigene benutzerdefinierte Ansichten erstellt werden, die dem Ansichtsprotokoll entsprechen, sowie SwiftUI-Ansichten zum Anzeigen von Text, Bildern und benutzerdefinierten Formen mithilfe von Stapeln, Listen und mehr. [Incb]

4.4.3 Hauptkriterien an die Funktionalität

Damit die App erkennt, ob sich das Auto bewegt oder nicht, muss sie mit dem Webserver kommunizieren können, um sich dort die vom Arduino gesendeten GPS-Daten zu holen. Zusätzlich muss eine Logik implementiert werden, welche bei eingeschaltetem System eine Veränderung der GPS-Daten erkennt und der Nutzer eine Benachrichtigung erhält. Weiterhin benötigen wir eine Funktion, dass die Koordinaten nicht als plain-Text angezeigt werden, sondern direkt auf einer Karte, wie man es beispielsweise von Google Maps gewohnt ist. Eine weitere sinnvolle aber eher optionale Implementierung wäre es, wenn die App erkennt, ob der Server gerade erreichbar ist und ein Icon dementsprechend die Farbe ändert. Um die theoretischen Prozesse zu strukturieren und auch übersichtlich zu visualisieren, haben wir über das Tool Camunda einen ersten groben Ablaufplan erstellt, wie in Abbildung 4.13 zu sehen.

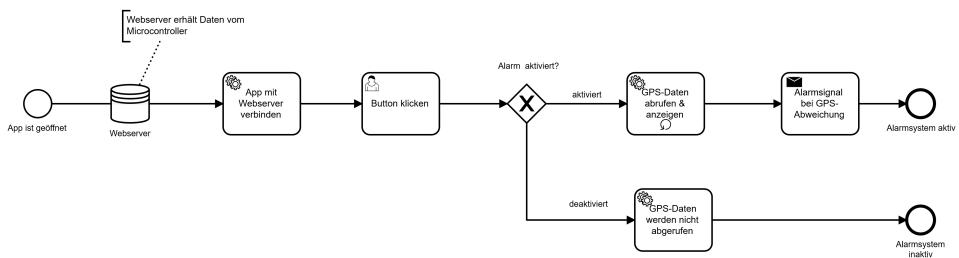


Abbildung 4.13: Entwurf der Hauptfunktionalitäten

4.4.4 Design

Das Design der beiden Apps unterscheidet sich stark und das wurde auch bewusst von uns zugelassen - es war uns wichtig, Hauptkriterien der Funktionalitäten zu definieren, wie beispielsweise in Abbildung 4.13 zu sehen, aber der Kreativität jedes einzelnen von uns wollten wir keine Grenzen vorschreiben.

Kreativ werden wir dann, wenn wir unsere immer gleichen Muster, unsere üblichen Gedankenabläufe unterbrechen und unseren Geist auf ungewohnte Wege schicken⁵

Der Stil dieser App, besticht durch sein sehr minimalistisches und dennoch durch moderne Farben geprägtes Design. Wir haben hierbei auf Intuitivität geachtet und auf *viel Text* verzichtet, da die App so keine möglichen Sprachbarrieren aufwirft und zeitgemäßer wirkt.

Unscheinbar aber wichtig! Der Nutzer sieht als erstes das AppIcon im Appstore sowie später auf seinem Bildschirm. Hierbei wollten wir erreichen, dass dieses Icon nicht nur zu unserem Produkt passt, sondern auch optisch sehr anspricht. Daher haben wir uns für ein Icon entschieden, welches die Farben der iOS-App enthält sowie ein Auto, wie in Abbildung 4.14 demonstriert.

⁵Bod22.



Abbildung 4.14: AppIcon auf dem Bildschirm

Nach dem berühmten Tap auf das Icon hängt es oftmals vom Alter des genutzten Gerätes ab, wie schnell eine App sich aufbaut. Damit auch hier der User ein Feedback erhält, ob etwas passiert, haben wir einen Launchscreen implementiert, welcher ebenfalls dem Design der App entspricht.(Abbildung 4.15)

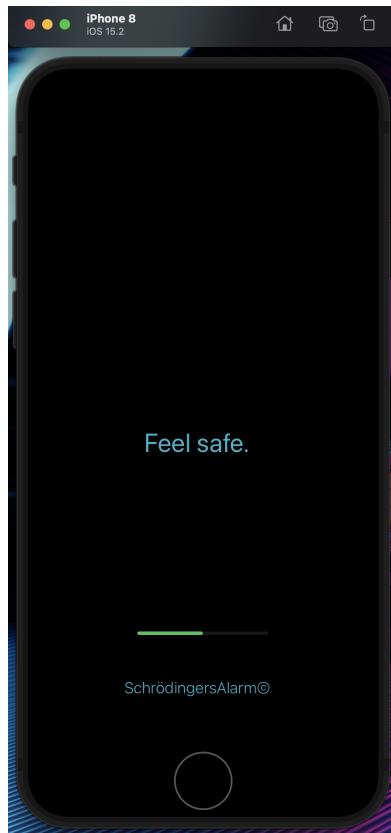


Abbildung 4.15: Launchscreen

Beim Blick auf den Home-Bildschirm, im Navigations-Menü am kleinen Häuschen unten auf Daumenhöhe zu erkennen, sticht sofort das große Schloss ins Auge. Dieses kann auch wortwörtlich verstanden werden - ist es grün, dann ist der Alarm aktiviert (*verschlossen, gesichert*), tapt der Nutzer erneut darauf, ist es rot und der Alarm deaktiviert (*geöffnet, ungesichert*).



Abbildung 4.16: Navigationstab *Home* im aktivierten Modus

Bei der Umsetzung des Kartendesigns war es wichtig, dass der Nutzer die vom Server bereitgestellten Daten angezeigt bekommt. Das Feature, dass diese nicht als plain-Text dargestellt, sondern die Koordinaten direkt als Standort auf einer Karte übersetzt werden, haben wir ebenfalls umgesetzt, wie in Abbildung 4.17 visualisiert.

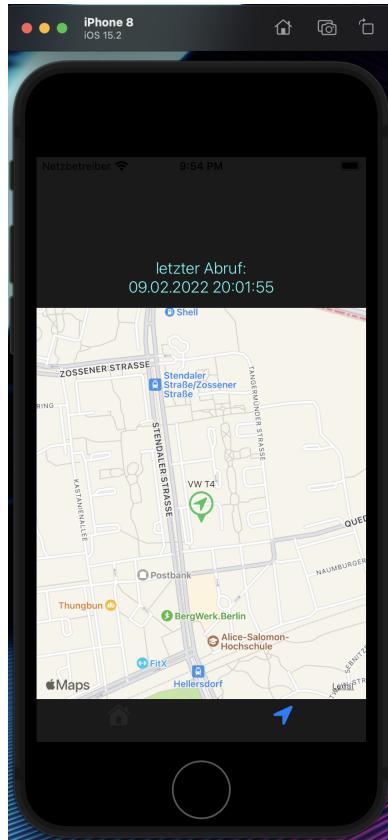


Abbildung 4.17: Navigationstab *Map*

Zur noch besseren Vorstellung, wie das Handling der beiden Apps ist, sind unter **Anhänge** die Links zu den jeweiligen auf Youtube hochgeladenen Demonstrations-Videos hinterlegt.

4.4.5 Probleme

Wie bereits angedeutet, gab es bei der Entwicklung einige Probleme, es waren kleinere wie anfänglich kein ausreichend leistungsstarkes und zudem mit aktuellem Betriebssystem ausgestattetem Macbook und größere, wie nachfolgend genauer beschrieben.

Koordinaten in JSON als String hinterlegt

Zu Beginn wurden die Koordinaten, sowie der Abrufzeitpunkt als Strings vom Server zur Verfügung gestellt, was ein Parsen sehr erschwerte, da für

die Verarbeitung dieser Daten die iOS-Map integer beziehungsweise double-Werte benötigt. Dieses Problem konnte jedoch nach interner Rücksprache schnell behoben werden, da es sich lediglich um eine Einstellung beim Encode des Webservers handelte.

Http? Abrufen verboten!

Die benötigten Daten vom Server werden nicht innerhalb der App angezeigt und es erweckte auch nie den Anschein, dass diese überhaupt abgerufen werden. Es wurde sehr lange das Problem innerhalb der Methoden gesucht, jedoch ohne nennenswerte Erkenntnisse. Erst als extern um Hilfe gebeten wurde, kam schnell heraus, dass SwiftUI den Abruf von JSON-Daten einer Http-Seite verhindert. Nach gezielter Platzierung von print()-Aussagen im Log wurde das eigentliche Problem sichtbar, siehe Abbildung 4.18. Bei Nutzung einer Beispiel JSON-Datei von einer Https-Seite trat keine Fehlermeldung mehr auf.

The screenshot shows the Xcode interface with a Swift file open. A warning message is displayed in a callout box over the code:

```

func loadData() {
    guard let url = URL(string:
        "http://retorbit.spdns.de/provide_Data
        .php") else {fatalError("Invalid URL")}
    let urlRequest = URLRequest(url: url)
    let dataTask =
        URLSession.shared.dataTask(with:
            urlRequest)
    dataTask.resume()
}

guard let response = response as?
    HTTPURLResponse else {
    return
}
if response.statusCode == 200 {
    guard let data = data else {
        return
    }
    DispatchQueue.main.async {
        do {
            let str = String(data: data, encoding:
                data, as: UTF8.self)
        }
    }
}

```

The warning message reads:

App Transport Security has blocked a cleartext HTTP connection to `retorbit.spdns.de` since it is insecure. Use HTTPS instead or add this domain to Exception Domains in your Info.plist.

Abbildung 4.18: Fehlermeldung *NSAppTransportSecurity*

Auf Apple-Plattformen verbessert eine Netzwerkfunktion namens App Transport Security (ATS) den Datenschutz und die Datenintegrität für alle Apps und App-Erweiterungen. ATS erfordert, dass alle HTTP-Verbindungen,

die mit dem URL-Ladesystem hergestellt werden – normalerweise unter Verwendung der URLSession-Klasse – HTTPS verwenden. Es erlegt außerdem erweiterte Sicherheitsprüfungen auf, die die vom TLS-Protokoll⁶ vorgeschriebene Standard-Server-Vertrauensbewertung ergänzen. ATS blockiert Verbindungen, die die Mindestsicherheitsanforderungen nicht erfüllen. [...] Diese Schutzmaßnahmen können umgangen werden, indem der NSAppTransportSecurity-Schlüssel zur Information Property List-Datei der App hinzugefügt und ein ATS-Konfigurationswörterbuch als Wert angegeben wird [Inca]

Dieses Problem kann also behoben werden, wurde aber leider nicht mehr innerhalb der Bearbeitungszeit geschafft und wird später nachgeholt.

⁶Transport Layer Security

Kapitel 5

Produkt

Unser Prototyp setzt sich derzeit aus 3 unterschiedlichen Hauptkomponenten zusammen: dem Arduino samt Shield für die Möglichkeit der Ortung und Simkartenslot, den Webserver zur Datenverteilung und -bereitstellung sowie eine App für die Smartphone Betriebssysteme Android und iOS, wie in Abbildung 5.1 festgehalten.



Abbildung 5.1: *Schrödingers Alarm*

5.1 zukünftiges Produktangebot

Wir haben uns zwei Produktkategorien überlegt, mit denen wir Gewinn generieren können. Zum einen ist das ein monatliches Bezahlkonzept. Hierbei

haben wir uns an dem Konzept einiger Mobilfunkanbieter orientiert. Die Überlegung ist, dass wir zwei hinzu buchbare Optionen anbieten und man nur das bezahlt, was wirklich gebraucht wird. Eine Optionen ist die Google Standort API für eine genauere und zuverlässigere Standorterkennung. Die andere Option ist das Bestellen einer Sim-Karte mit Mobilfunkvertrag. Die Idee ist hier, dass so gegen einen kleinen Aufpreis der Ease-of-Use unseres Produkts weiter gesteigert wird.

Für das Endgerät haben wir uns neben der aktuellen Basis Variante für eine Base+ und eine Pro Variante entschieden. Bei der Base+ Variante ergänzen wir die Basis Variante lediglich um eine Sirene und etwas mehr Akkuleistung. Für die Pro Variante erweitern wir das Basismodell um einen WLAN Empfänger. Des Weiteren überlegen wir unseren modularen Aufbau für eine weitere Umsatzsteigerung zu nutzen und Akkumodule zu verkaufen, mit denen die Akkulaufzeit unserer Produkte vom Nutzer beliebig verlängert werden kann. Ein weiterer Vorteil ist, dass so bei nachlassender Akkuleistung unser Produkt nicht entsorgt werden muss, sondern der Nutzer einfach mit offiziellen Produkten den Akku wechseln kann. Dies soll verhindern, dass der Kunde auf Drittanbieter zurückgreift. Das Ziel ist, dass wir unsere Apps und den Webserver unabhängig von der Produktversion verwenden können.

5.2 Vergleich

Eine kurze Suche im Internet nach ähnlich funktionierenden Systemen lässt schnell die Frage auftreten : Wo liegt bei uns der Unterschied? Für die direkte Konkurrenzanalyse haben wir uns für den Platzhirsch Amazon entschieden, da es wahrscheinlich nirgendwo sonst so viele ähnliche Produkte aus den unterschiedlichsten Ländern mit diversen Qualitätsmerkmalen gibt. Wir haben dafür Produkte im niedrigeren Preissegment für rund 30€, im mittleren für rund 60€ und im höherpreisigen für weit über 100€ mit unserem verglichen.

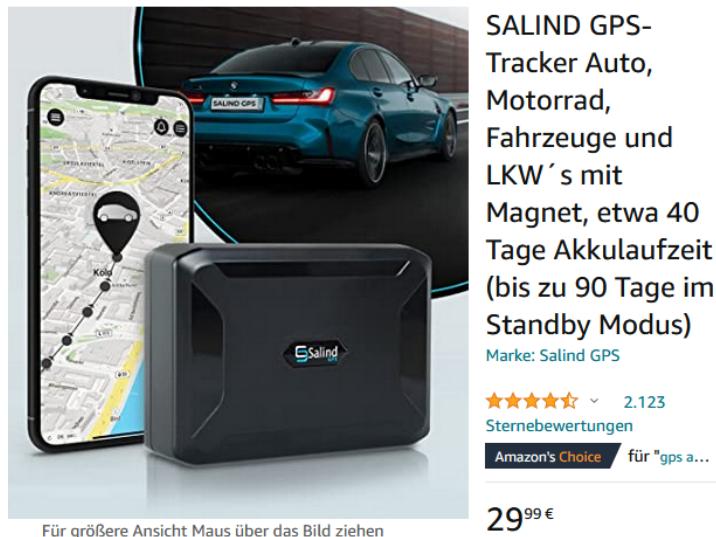


Abbildung 5.2: Beispiel Vergleichsprodukt im unteren Preissegment [GPS]

Es lässt sich festhalten, dass alle Systeme die gleiche Grundfunktion besitzen: eine App, mit der das sich im Auto befindliche Gerät über einen Server Koordinaten-Daten abrufen und auswerten kann. Dennoch sind uns wesentliche Punkte aufgefallen, die sich abheben:

- unser System ist nicht fest in sich verbaut, das bedeutet bei möglichen Defekten kann es leicht geöffnet werden und bei Bedarf einfach selbstständig repariert werden
- Die Akkus sind austauschbar und können mit einem handelsüblichen Ladegerät bequem daheim aufgeladen werden, so muss das Gesamtsystem nicht aus dem Auto entnommen werden. *Anmerkung:* alle Systeme, die wir uns angesehen haben, haben fest verbaute Akku Systeme, die nur über Micro USB aufgeladen werden können
- Aus diesen Punkten ergibt sich auch direkt der nächste positive Effekt: da alle Bestandteile leicht zugänglich und austauschbar sind, ist unser Produkt sehr viel nachhaltiger - zudem haben wir auf unnötige Kunststoff Bestandteile verzichtet
- Des weiteren ist unser System durch sein zugängliches Design leicht erweiterbar und erinnert so unter anderem an die UrIdee des *Fairphone* mit seiner Komponenten Bauweise: man kauft nur, was man braucht, erhält aber ein garantierter funktionierendes Basissystem.

- Ebenfalls geht man bei unserem Produkt keine aufgezwungene Vertragspartner Bindung ein - der Internetanbieter kann sich der Kunde selbst aussuchen

Es lässt sich daher behaupten - Ja, Produkte wie dieses sind in der Basis nicht neu, dennoch haben wir uns Gedanken in ganz unterschiedliche Richtungen gemacht und unseren Ideenhorizont frei gehalten, so dass sich unser Produkt stetig mit und für den Kunden weiterentwickeln kann.

Kapitel 6

Rück- und Ausblick

Ein kurzer Rückblick soll Revue passieren lassen, was wir anfänglich erreichen wollten und wie weit wir gekommen sind.

Zuerst sollte festgehalten werden, dass wir uns zu Beginn trotz unserer vielen Ideen für ein Basisprodukt entschieden haben, da unsere Erfahrung ausreicht, um einschätzen zu können, dass die Bearbeitungszeit knapp ist, Lieferanten lange Lieferzeiten aufrufen und auch ein Lockdown nicht auszuschließen war.

Think big, build small! [Ful22]

Dennoch haben wir den Großteil geschafft, wie im Ausschnitt in Abbildung 6.1 unseres aktuellen Trello-Boards zu erkennen ist.

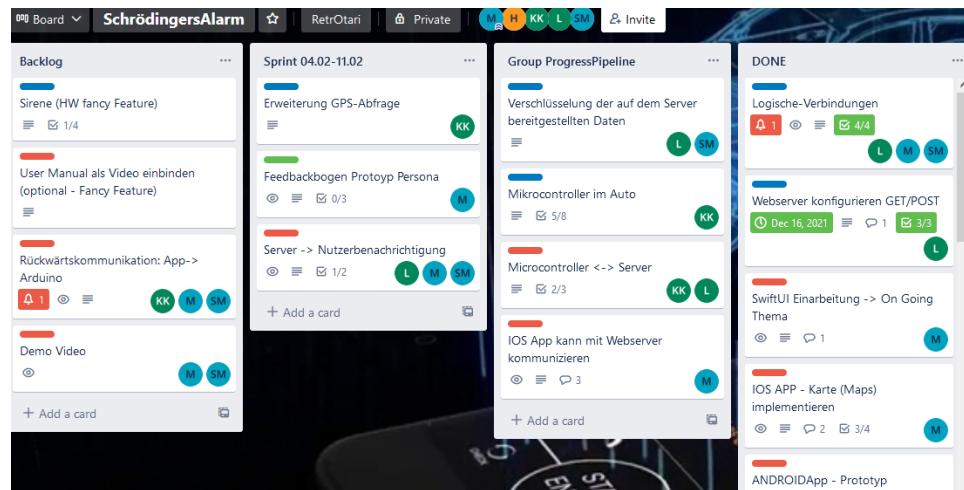


Abbildung 6.1: Abschlussstand Trello-Board

6.1 Lessons Learned

Es ist wichtig dass das Minimum Viable Product einen Nutzen bringt und einfach in seiner Bedienung ist.[Ful22] Durch Feedback von unseren Testern konnten wir herauskristallisieren, dass unsere Apps durch ihr derzeitiges unterschiedliches Design irreführend sind - *Anmerkung:* Ein einheitliches Design an sich war uns bis hier nicht wichtig, da hier tatsächlich der Lernerfolg und Spaß an der Bearbeitung an erster Stelle standen. Dennoch ist dies ein ernstzunehmender Punkt.

Des Weiteren wurde sich gewünscht, dass kleine User Manuals für beispielsweise die Systemerweiterung in Form von Kurzvideos bereitgestellt werden - so muss kein Papier verschwendet werden, und bei notwendigen Informations-Updates oder generellen Neuerungen kann dies schnell und einfach digital passieren. Ebenfalls wäre ein Routentracking vorteilhaft, da derzeit immer nur der letzte Standort abgerufen werden kann - so könnte das gestohlene Auto besser nachverfolgt werden.

Nach so einigen Verbesserungsvorschlägen möchten wir natürlich auch noch festhalten, dass besonders die offene und so leicht erweiterbare Bauweise, die erste ergänzende Idee mit der Sirene, der Geofencing-Mechanismus sowie dass es Apps für beide Betriebssysteme gibt, für sehr positiv empfunden wurden.

6.2 Zukünftige Entwicklung

Arduino

Als Erweiterungen für den Arduino sollen in Zukunft noch eine Alarmsirene vom Typen BSE128(Abbildung 6.2) verbaut und für die Standorterkennung noch die Google API zur besseren Lokalisierung genutzt werden.



Abbildung 6.2: Sirene Digisound B/SE 128

Für die Sirene muss das Gesamtsystem noch dahingehend erweitert werden, dass eine Kommunikation zum Arduino hin möglich wird. Auf dem Arduino muss hierfür noch realisiert werden, dass der Rückgabewert vom Server abgespeichert wird und in regelmäßigen Abständen geprüft wird, dass sich das Gerät nicht aus dem vom Anwender definierten Bereich entfernt hat. Es muss also noch Geofencing stabil implementiert werden, um zu erkennen wann die Sirene auslösen soll. Es ist aber auch eine einfache Lösung denkbar, bei der der Benutzer den Alarm aus der Entfernung selbst steuern muss beziehungsweise kann.

Aus Hardware-Sicht muss dann nur die Sirene mittels Transistorschaltung an den Arduino angeschlossen und mit Strom versorgt werden, damit der Arduino das Alarmsignal im Fall einer Entwendung geben kann. Die Schaltung dafür sieht wie folgt aus (Abbildung 6.3):

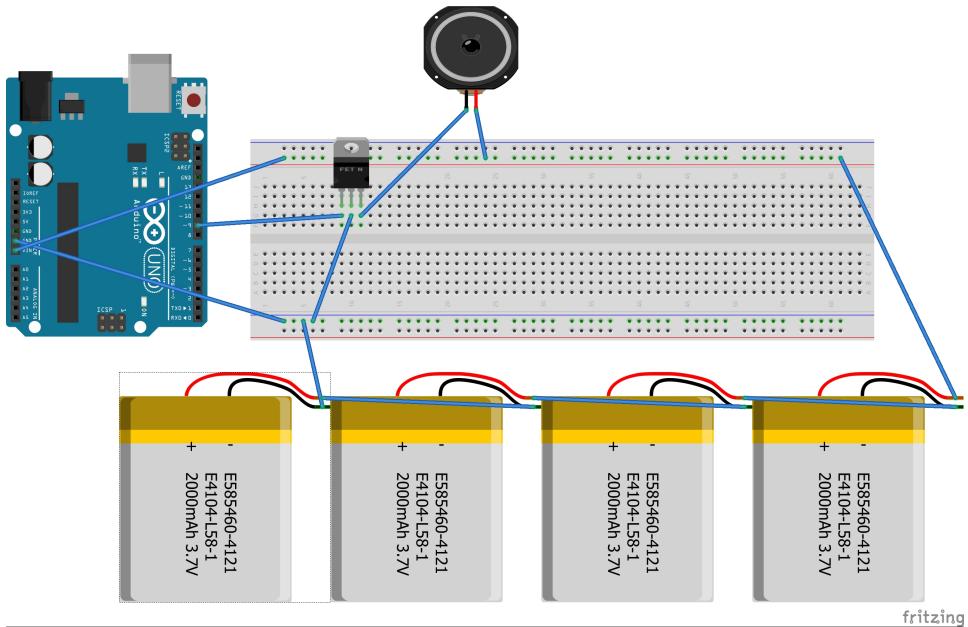


Abbildung 6.3: Systemerweiterung um Sirene

Dabei kann die Sirene, da sie 12-24V Eingangsspannung akzeptiert, an die Stromversorgung des Arduinos mit angeschlossen werden. Es muss dafür jedoch eventuell die Akkuleistung ebenfalls erweitert werden. Dies muss jedoch erst noch getestet werden.

Die Sirene soll zukünftig durch eine einfache N-Channel Mosfet Schaltung gesteuert werden. Um den Google Standort Service effektiv verwenden zu können, ist die Erweiterung um einen WLAN Empfänger sinnvoll. Somit kann neben den Informationen, die durch GSM und GPS zur Verfügung stehen, auch die lokalen WLAN-Netze zur Standortermittlung genutzt werden. Dies erhöht weiter die Genauigkeit kann aber theoretisch auch wegfallen. Die Verwendung der API ist jedoch kostenpflichtig und dementsprechend ist das Geschäftskonzept eventuell weiter zu überdenken. Eine mögliche Lösung hierfür wäre ein monatliches Bezahlkonzept, über das man je nach Bedarf die API Standorterkennung hinzubuchen kann.

Webserver

Finalisierung der Verschlüsselung

Der erste Punkt, der in Zukunft bearbeitet werden soll, ist die vollständige Implementierung und ein anschließendes Testen der Verschlüsselung der

Nutzdaten. Für den Webserver ist wie zuvor beschrieben bereits alles fertig. Es gilt nun also die entsprechend umgekehrten Abläufe in den beiden Apps zu realisieren.

Darüber hinaus sollte geprüft werden, ob es auch möglich ist, die Koordinaten bereits auf dem Arduino zu verschlüsseln. So könnte zusätzliche Sicherheit der Daten gewonnen werden.

Umstellung auf HTTPS

Bei der Entwicklung der iOS-App trat an der Stelle des Auslesens der JSON codierten Daten ein Problem auf. Die Fehlermeldungen waren zunächst nicht eindeutig zu interpretieren, doch es stellte sich heraus, dass mit der von uns gewählten Version nur von HTTPS-Seiten JSONs mittels HTTP-GET abgefragt werden können.

Um derartige Probleme in Zukunft zu umgehen, soll der Webserver von HTTP auf HTTPS umgestellt werden. Wir hatten auch erst überlegt, dies im aktuellen Bearbeitungszeitraum umzusetzen, jedoch fand sich für die iOS-App eine Alternativlösung. Auch erste Recherchen zur Umstellung auf HTTPS zeigten uns, dass dies doch ziemlich aufwendig werden und wieder neue Probleme mit sich bringen könnte.

Bidirektionale Kommunikation ermöglichen

Ein weiteres Feature, das in Zukunft realisiert werden soll, ist eine Kommunikation auch in die andere Richtung. Es soll also dann möglich sein, dass die App Befehle oder sonstige Daten an den Webserver schickt und der Arduino diese dann auch abfragen kann. Eine solche Kommunikation wäre zum Beispiel für das Aktivieren der ebenfalls noch geplanten Sirene im Innenraum des Fahrzeugs erforderlich.

MQTT als redundanten Übertragungsweg einrichten

Um nicht nur den bisherigen Übertragungsweg über den HTTP-Server nutzen zu können und somit anfällig für Systemausfälle zu sein, soll ein zweiter, redundanter Übertragungsweg mit Hilfe des MQTT-Protokolls errichtet werden. Hierfür ist als erstes das Bereitstellen eines eigenen MQTT-Brokers erforderlich, um direkt unabhängig von Dritten zu sein. Auch muss ein MQTT-Topic definiert werden, in welchem die Kommunikation stattfinden soll. Im nächsten Schritt muss auf dem Arduino ein entsprechendes Skript programmiert werden, mit welchem die Koordinaten mittels PUBLISH zum MQTT-Broker übermittelt werden können. Dieses muss dann entweder parallel zum bisherigen Skript ausgeführt werden oder es werden beide in einem neuen Skript vereinigt.

Anschließend muss in den beiden Apps ein MQTT-SUBSCRIBE implementiert werden.

tiert werden, um die vom Arduino veröffentlichten Daten zu erhalten. Auch dies muss parallel zur Datenabfrage vom Webserver geschehen. Zusätzlich soll in den Apps dann ein Abgleich beider Daten (HTTP und MQTT) implementiert werden. Sind beide Daten gleich werden die neuen Koordinaten übernommen. Für den anderen Fall, dass die beiden Datenpakete unterschiedlich sind, muss noch überprüft werden, welche Daten dann gespeichert werden.

Apps

Datenbanken

Für spätere Versionen sollten Möglichkeiten einer Datenbank-Einbindung beispielsweise mit "Firebase", um die Datenbankverwaltung besser zu verwalten, implementiert werden.

Routentracker

Die Idee ist nicht neu und dennoch sehr praktisch - damit nicht nur der aktuelle Standort angezeigt wird, sondern auch eine Nachverfolgung, wäre eine Art Routentracker sinnvoll. Dafür müssten in der JSON-Datei mehrere Daten in einem zuvor definierten Intervall abgespeichert werden, welche dann als verbundene Punkte auf der Karte angezeigt werden. So kann im Falle eines Diebstahl die ganze Strecke nachvollzogen werden, die bereits zurückgelegt wurde.

Android- & Apple-Watch Kommunikation

Smartwatches liegen im Trend und Smartphones werden immer größer und dadurch schnell unhandlich, so dass sie gerne in einer Tasche verstaut werden. Daher ist es eine nützliche Überlegung, dass die Apps auch für Smartwatches bereitgestellt werden, so dass der Nutzer noch schneller bemerkt, wenn ein Alarm ausgelöst wird.

Eine App - Mehrere Fahrzeuge

Es ist sinnvoll, dass wenn man von mehreren Alarmsystemen in Fahrzeugen ausgeht, dass die App gesagt bekommt, welches Gerät der Nutzer Zuhause besitzt - daher sollte es eine Konfigurationsmöglichkeit innerhalb der App für das Alarmsystemgerät geben. Des weiteren wäre es auch sinnvoll, dass wenn beispielsweise eine Familie mit mehreren Systemen und unterschiedlichen Fahrzeugen diese sich in der App gebündelt anzeigen lassen könnte - da wir derzeit jedoch nur ein System besitzen, wurde es hier erst einmal als Placeholder implementiert.

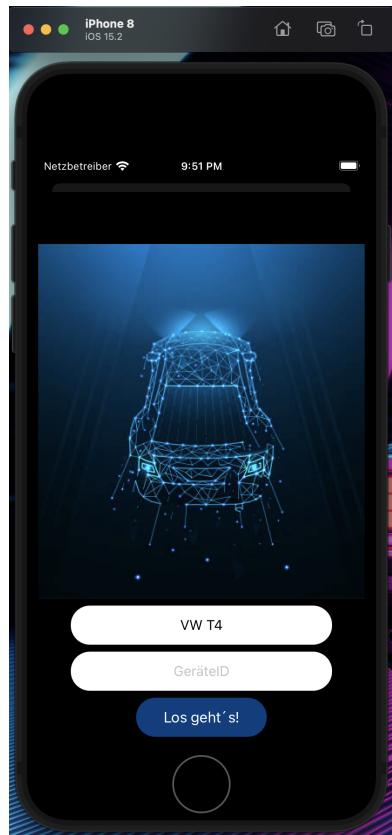


Abbildung 6.4: Navigationstab *Fahrzeug hinzufügen*

6.3 Fazit

Das Projekt ist eine wertvolle Erfahrung in vielen Hinsichten - so haben wir nicht nur aus dem Studium bekannte, sondern auch neue Technologien kennengelernt und erprobt. Des Weiteren haben wir erfahren, was es bedeutet, ein Produkt von einer kleinen Idee bis hin zu einem umfangreichen funktionierenden Prototypen zu gestalten, entwerfen und aufzubauen. Das Projekt wird uns sicherlich mit all seinen Facetten noch lange im Gedächtnis bleiben und der ein oder andere wird es sicherlich noch weiterentwickeln.

Kapitel 7

Anhänge

7.1 Quellcode

Für die Versionierung der Fortschritte sowie ein vereinfachtes Zusammenarbeiten auf Distanz haben wir das Versionsverwaltungstool Git genutzt - hier ist auch der Code für den Webserver, Arduino und die Apps einzusehen. Der Übersicht halber haben wir darauf verzichtet diesen in der Dokumentation extra aufzuführen.

Öffentliches Git-Repository : <https://github.com/MiN0DE/SchroedingersAlarm>

7.2 Videos

Android-App: https://www.youtube.com/watch?v=_CBTdDctr2k

iOS-App: <https://www.youtube.com/watch?v=gWaSZPfxT9s>

Abbildungsverzeichnis

3.1	Persona <i>Linus Schrödinger</i>	9
3.2	Systemkonzept	10
4.1	GSM und Http-Befehle	13
4.2	GPS-Befehle	14
4.3	Programmablaufplan	14
4.4	Systemaufbau für das KFZ	16
4.5	fehlerhafte und korrekte Ausgabe	17
4.6	Programmablaufplan Webserver	25
4.7	Datenpaket Webserver	29
4.8	Skriptablauf Webserver	30
4.9	Konzept der Android App im Standard Fall	33
4.10	Konzept der Android App im Alarm Fall	34
4.11	Android App <i>Login Seite</i>	36
4.12	Android App <i>Seitenanichten</i>	37
4.13	Entwurf der Hauptfunktionalitäten	39
4.14	AppIcon auf dem Bildschirm	41
4.15	Launchscreen	42
4.16	Navigationstab <i>Home</i> im aktivierten Modus	43
4.17	Navigationstab <i>Map</i>	44
4.18	Fehlermeldung <i>NSAppTransportSecurity</i>	45
5.1	<i>Schrödingers Alarm</i>	47
5.2	Beispiel Vergleichsprodukt im unteren Preissegment	49
6.1	Abschlussstand Trello-Board	51
6.2	Sirene Diginet B/SE 128	53
6.3	Systemerweiterung um Sirene	54
6.4	Navigationstab <i>Fahrzeug hinzufügen</i>	57

Quellenverzeichnis

- [Ahm20] Suhali Ahmad. *XCode*. Techn. Ber. World of VR, 2020. URL: <https://worldofvr.de/xcode/>.
- [Bod22] Katja Nele Bode. „Anleitung zum Fliegen lernen“. In: *centaur* (Feb. 2022).
- [Dü21] Katharina Dümmер. „Autodiebstahl: Welche Pkw bei Kfz-Knackern besonders beliebt sind“. In: *ADAC* (Okt. 2021). URL: <https://www.adac.de/rund-ums-fahrzeug/autokatalog/markenmodelle/auto/autodiebstahl-statistik/>.
- [Ful22] Dipl.-Ing Hanna Full. *Prototyping*. moodle. Jan. 2022. URL: https://moodle.htw-berlin.de/pluginfile.php/1394565/mod_resource/content/1/06DesignThinking-Prototyping.pdf.
- [GPS] Salind GPS. *SALIND GPS-Tracker Auto, Motorrad, Fahrzeuge und LKW's mit Magnet, etwa 40 Tage Akkulaufzeit (bis zu 90 Tage im Standby Modus)*. URL: https://www.amazon.de/GPS-Tracker-Motorrad-Fahrzeuge-Akkulaufzeit-Standby/dp/B07TV4W8H4/ref=sr_1_8?keywords=gpsautotracker&qid=1645548524&sprefix=gpsaut,aps,97&sr=8-8.
- [Inca] Apple Inc. *NSAppTransportSecurity*. Techn. Ber. URL: https://developer.apple.com/documentation/bundleresources/information_property_list/nsapptransportsecurity.
- [Incb] Apple Inc. *SwiftUI*. Techn. Ber. Apple Inc. URL: <https://developer.apple.com/documentation/swiftui>.
- [JL] Michael Lewerenz Josefine Lepzien. *Persona-Methode Eine Methode zur Illustrierung von Bildungsbedarfen*. Techn. Ber. unirostock. URL: <https://www.uni-rostock.de/storages/unirostock/UniHome/Weiterbildung/KOSMOS/Persona.pdf>.
- [Lan] Jesko Landwehr. *Was ist Flutter?* Techn. Ber. URL: <https://it-talents.de/it-wissen/was-ist-flutter/>.
- [pro] programmierenlernenhq. *Android Studio Tipps und Tricks*. Techn. Ber. URL: <https://www.programmierenlernenhq.de/android-studio-tipps/>.

- [RD19] Holger Koschek und Carsten Sahling Rolf Drähter. „Scrum kurz gut“. In: *O'Reillys Taschenbibliothek* (2019).
- [Vog] Jonas Hungershausen Lars Vogel. *Mobiles Entwickeln mit dem Flutter SDK*. Techn. Ber. URL: <https://entwickler.de/mobile/flutter-haft>.