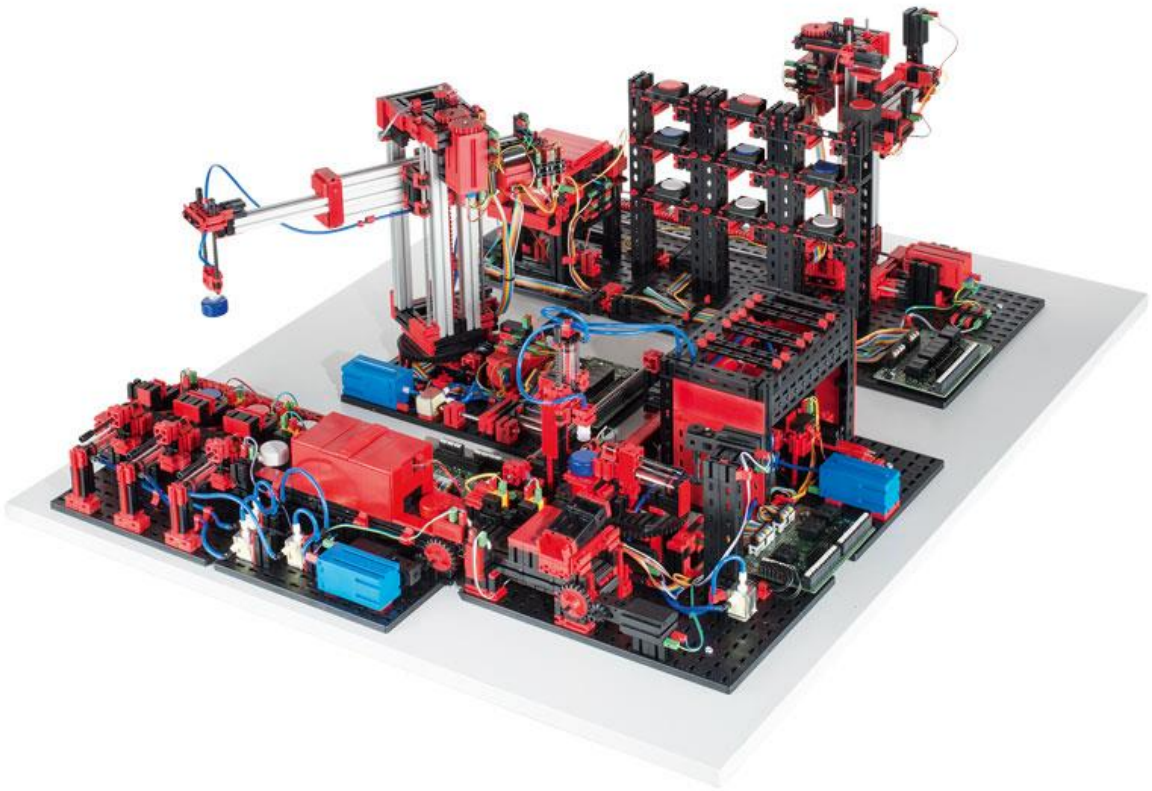
Informationsmanagement

Dokumentation IoT Fehlercodes



Fehlercodes Gruppe 1

Nopper Alexander | Latinović Jovan | Stingl Sarah | Rauch Julia

Aul Tamara | Urrehman Henna

Sommersemester 2020

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN
MÜNCHEN

Inhaltsverzeichnis

1. Einleitung

Die Aufgabe unserer Gruppe Fehlercodes 1 im Projekt Infomanagement war es, über einen Barcode Fehlerbilder zu simulieren. Die Fehlerbilder sollten dann über einen Barcode-Scanner an einen Raspberry Pi übertragen und von dort an das Content Delivery Portal *PRISMA* weitergeleitet werden.

Unsere Projektgruppe hat sich daher vor allem mit folgenden Fragestellungen bzw. Anforderungen beschäftigt:

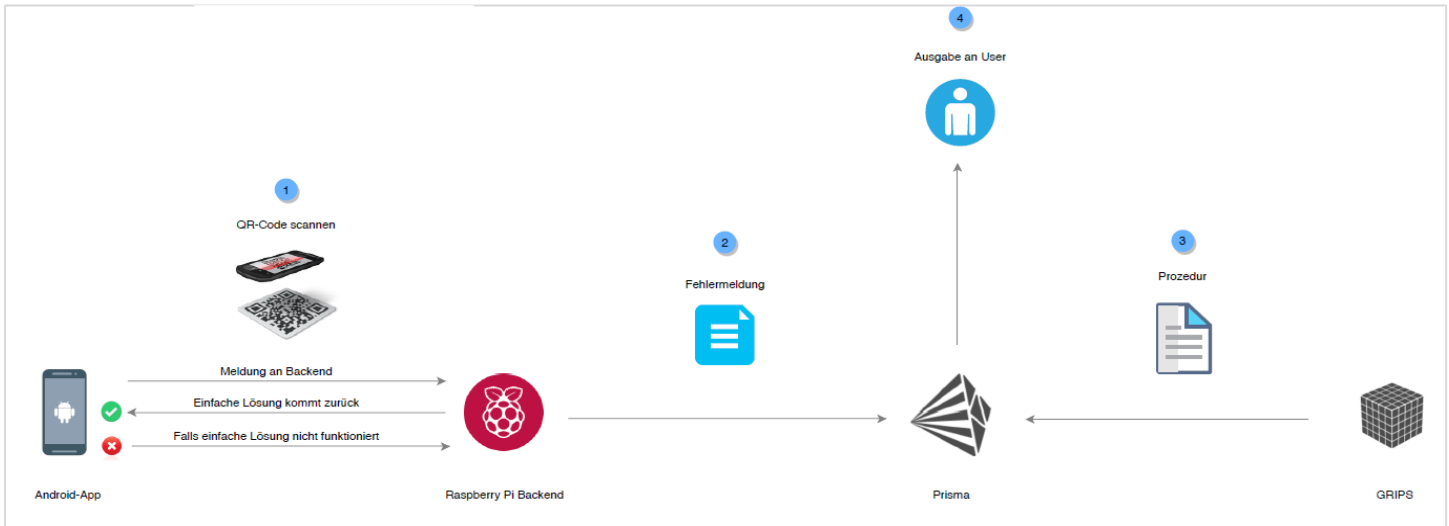
- Wie können die Fehler mithilfe eines Barcode-Scanners erfasst und an einen Raspberry Pi übertragen werden?
- Wie können diese Daten an *PRISMA* übergeben werden? Welche Schnittstellen gibt es bzw. welche Schnittstellen benötigen wir, um die Daten zu übergeben zu können?

Die Anforderungen wurden im Laufe des Projekts etwas abgewandelt und angepasst: Über QR-Codes haben wir Fehlerbilder simuliert, die ein Nutzer dann mithilfe einer Android-App einscannen kann. Dazu haben wir auch folgenden Use Case festgelegt, damit die Scanner-App nicht nur Mittel zum Zweck ist: „Als Servicetechniker würde ich mit für die weiteren Schritte wünschen, dass mir in der App Lösungsvorschläge angezeigt werden, um den Fehler vor Ort zu beheben“.

In der folgenden Dokumentation wird das Ergebnis unserer Gruppenarbeit dargestellt sowie unsere Vorgehensweise in diesem Projekt. Außerdem wird gezeigt, wie nachfolgende Gruppen Zugang zu den Daten unseres Projekts bekommen, um dieses weiterführen zu können.

2. Prozessplan

Der Prozessplan stellt dar, wie unsere Projektgruppe die oben beschriebenen Anforderungen umgesetzt hat, um einem User nach Einscannen eines Fehlercodes eine Prozedur zum Fehlerhandling auszugeben.



Der Prozessplan zeigt den Ablauf – vom Scannen des Fehlercodes bis zur Ausgabe an den User

Scannen des QR-Codes und einfache Lösungsprozedur
Der Bediener der App dreht einen Baustein um und scannt den QR-Code auf der Rückseite des Bausteins. Dieser QR-Code enthält einen bestimmten Fehlercode, dessen Inhalt dann per HTTP-Request an das Raspberry Pi Backend geschickt wird.

→ Der Bediener kriegt dann vom Backend die Information, ob es eventuell an einem bestimmten Fehler liegt, der mithilfe einer einfachen Lösungsprozedur behoben werden kann. Liegt es an einem solchen Fehler, kann der Bediener diesen mithilfe der entsprechenden Lösungsprozedur einfach selbst beheben. Die einfache Lösungsprozedur wird dem Bediener direkt in der Smartphone-App angezeigt. (Hierfür haben wir drei einfache Fehlertypen beispielhaft selbst entwickelt.)

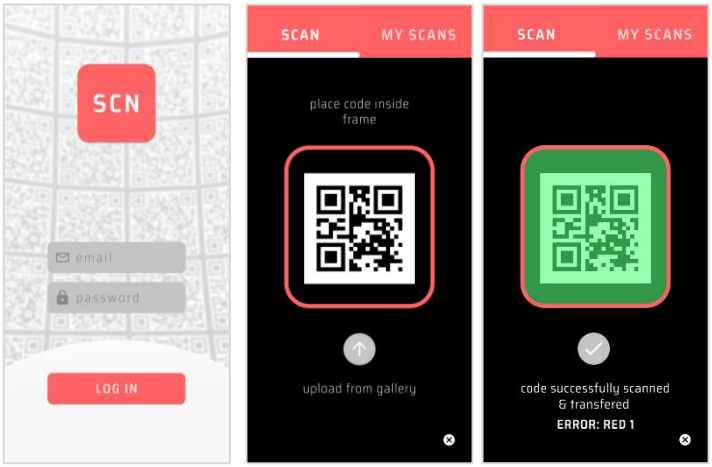
→ Der Nutzer kann dann durch den Button *OK* bestätigen, dass die Prozedur Abhilfe geschaffen hat. Durch das Backend wird dann eine neue Zeile in der Monitoring-Excel-Datei auf dem *PRISMA*-Server mit dem Status *solved* angelegt.

→ Wenn die Prozedur das Problem nicht löst (also wenn es nicht an einem bestimmten Fehler mit einer einfachen Lösungsprozedur liegt oder die einfache Fehlerprozedur nicht geholfen hat), dann betätigt der Nutzer den Button *Hilfe.* Nach Betätigung des Hilfe-Buttons wird der Fehlercode weitergeleitet:

Weiterleitung des Fehlercodes an *PRISMA* unter Zuhilfenahme von *GRIPS*
Das Raspberry Pi Backend legt in der Monitoring-Excel-Datei eine neue Zeile mit dem Status *new* an. Das Autoren-System *PRISMA* wird dann aktiv und zieht den gesamten Entscheidungsbaum (die Prozedur zur Fehlerbehebung) aus dem Component Content Management System GRIPS. Anschließend wird die Prozedur zur Fehlerbehebung an den User ausgegeben.

3. Konzipierung und Nutzung der Scanner-App

Um die Fehlercodes zu scannen, hat unsere Projektgruppe zuerst unter Einsatz eines Mockups eine Scanner-App konzipiert.

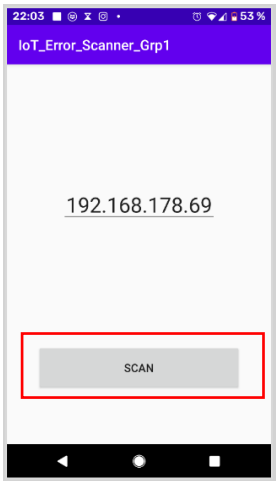


Mockup der Scanner-App

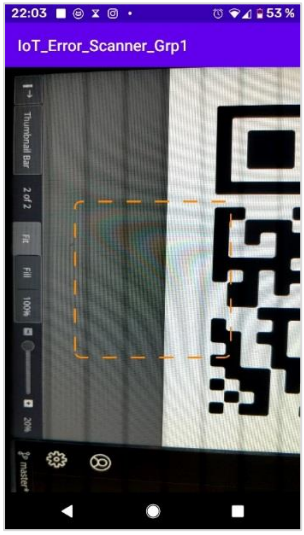
Anschließend haben wir die App mithilfe von Android Studio in Java geschrieben, um diese auf einem Android-Smartphone nutzen zu können.

Die folgende Prozedur zeigt die Funktionsweise der Scanner-App:

1. Der Nutzer drückt in der geöffneten App auf den Button **SCAN.**
   → Die Kamera öffnet sich.



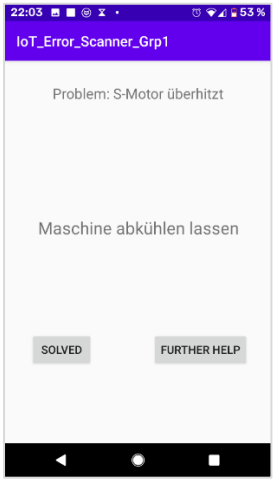
2. Mit der geöffneten Kamera kann der Nutzer den QR-Code erfassen und einscannen.



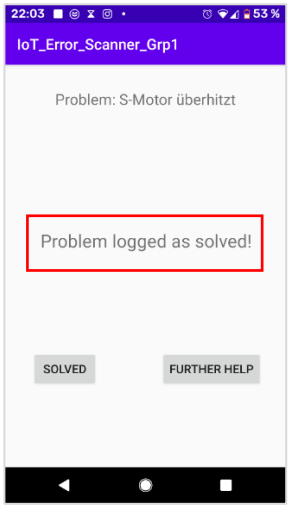
3. Die Anfrage wird von der App an das Backend weitergeleitet.
   Anmerkung: Die Verwendung des Raspberry Pi hätte nur Sinn gemacht, wenn ein Hardware Scanner verwendet worden wäre, statt den Fehlercode über eine App einzuscannen. Da wir uns aber für den Einsatz einer App entschieden haben, ist es nicht relevant, ob für den Austausch ein Raspberry Pi oder ein normaler Server verwendet wird.

4. Die http-Request (Übermittlungsstandard) geht von der App an das Backend.
   In dieser Anfrage stehen folgende Informationen:
   {„number“:“02“, „color“:“rot“,“is_first“:“true“}
   → Das Backend speichert diese Informationen und gibt sie über die Excel-Datei an die richtige Stelle weiter.

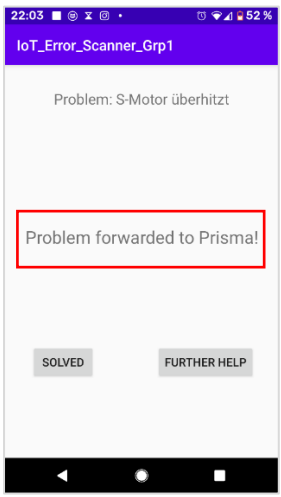
5. Das Backend gibt eine mögliche erste (einfache) Lösung an das Smartphone zurück.
   Beispiel: Maschine abkühlen lassen.



6. Der Nutzer kann dann in der App angeben, ob die Informationen aus dem Backend ausreichend bzw. hilfreich waren oder nicht.
   → Wenn in der App die Taste *solved* betätigt wird, gilt das Problem als gelöst und wird geloggt.



7. Wenn in der App angegeben wird, dass die Informationen aus dem Backend nicht ausreichend waren, wird die Anfrage an *PRISMA* weitergeleitet.



8. *PRISMA* zeigt dem User anschließend an, welche weiteren Lösungsmöglichkeiten zur Verfügung stehen, um den Fehler zu lösen.
   Voraussetzung hierfür ist, dass die erforderlichen Informationen zuvor in GRIPS eingepflegt worden sind.

4. Techstack und Dokumentation des Projekts in GitHub

Die in dem Projekt verwendeten Technologien sind hier nochmal übersichtlich zusammengefasst:

- Die Android-App zum Scannen eines Fehlercodes wurde in **Android Studio** in der Programmiersprache **Java** geschrieben.
- Das Rapsberry Pi Backend wurde in der Entwicklungsumgebung **PyCharm** in der Programmiersprache **Python 3.7.5** geschrieben.
- Um den Webserver in Python 3.7.5 korrekt aufbauen zu können, wurde das **Webframework Flask** eingesetzt.

Das gesamte Projekt ist außerdem in einem **GIT Repository** – einem digitalen Archiv – in GitHub eingepflegt: **GitHub** ist eine Online-Versionsverwaltung für Softwareprojekte, um Projektdaten unter anderem auch anderen zugänglich zu machen. Für die Fortführung des Projekts in den nächsten Semestern haben wir daher einen Account in GitHub erstellt, um der nachfolgenden Gruppe für die nächste Projektschritte bezüglich des Fehlercode-Handlings alle benötigten Informationen zur Verfügung zu stellen. Die Gruppe kann dadurch nachvollziehen, wie die Inhalte installiert und die verwendet werden können.

Mit den folgenden Daten bekommt die Gruppe Zugang zu dem GitHub-Account:
Login: iottrk2020@gmail.com
Name: Projekttrk
PW: IoTTRK2020!

In der README-Datei steht, wie die Dateien aus dem Repository in einem *Python Virtual Environment* verwendet werden können und wie die Python Applikation gestartet wird. In der requirements-Datei sind außerdem die Namen der Libraries hinterlegt, die installiert werden müssen. Diese Datei hat außerdem den Vorteil, dass über den Python Package Manager (PIP) alle benötigten Libraries automatisch in der richtigen Version installiert werden können.

Einen ähnlichen Aufbau gibt es auch für die Error-Scan-App, die wir entwickelt haben. Hier ist eine Android-Scan-App samt vollständigem Code hinterlegt. Unter dem Reiter Releases -> Assets kann die kompilierte App (IOT_Scanner_grp1.apk) direkt für Android heruntergeladen werden.

5. Schnittstellen-Spezifikation

Eigentlich hätte mithilfe von WebDAV (einem Netzwerkprotokoll zur Bereitstellung von Dateien über das Internet) eine Schnittstelle zu *PRISMA* hergestellt werden sollen, den Zugang haben wir von STAR allerdings nicht rechtzeitig erhalten. Im Backend wird der Fehlercode, der als JSON Datei vorliegt, empfangen und auch ausgegeben (der Fehlercode wird in ein Excel-Sheet, das von *PRISMA* verarbeitet werden kann, ausgegeben).

Der markierte Bereich zeigt Datum, Uhrzeit, Fehlercode-Nummer, Color und einen Status an.

```
44  INFO:root:2020-06-28 18:24:06: number: 02 color: rot status: new
45  INFO:werkzeug:192.168.178.66 - - [28/Jun/2020 18:24:06] "[37mPOST /api HTTP/1.1[0m" 200 -
46  INFO:root:2020-06-28 18:24:11: number: False color: rot status: solved
47  INFO:werkzeug:192.168.178.66 - - [28/Jun/2020 18:24:11] "[37mPOST /api HTTP/1.1[0m" 200 -
48  INFO:root:2020-06-28 18:35:43: number: 02 color: rot status: new
49  INFO:werkzeug:127.0.0.1 - - [28/Jun/2020 18:35:43] "[37mPOST /api HTTP/1.1[0m" 200 -
50  INFO:root:2020-06-28 18:36:16: number: 02 color: rot status: new
51  INFO:werkzeug:127.0.0.1 - - [28/Jun/2020 18:36:16] "[37mPOST /api HTTP/1.1[0m" 200 -
52  INFO:root:2020-06-28 18:36:25: number: 02 color: rot status: moved to prisma
53  INFO:werkzeug:127.0.0.1 - - [28/Jun/2020 18:36:25] "[37mPOST /api HTTP/1.1[0m" 200 -
54
```

Wenn das Problem gelöst wurde, ändert sich der Status den Python Package Manager zu *new*. Andernfalls wird der Status zu *moved to prisma* geändert. Der Status *solved* gibt an, dass ein Fehler mithilfe einer entsprechenden Prozedur gelöst werden konnte.