# Informatik

## App

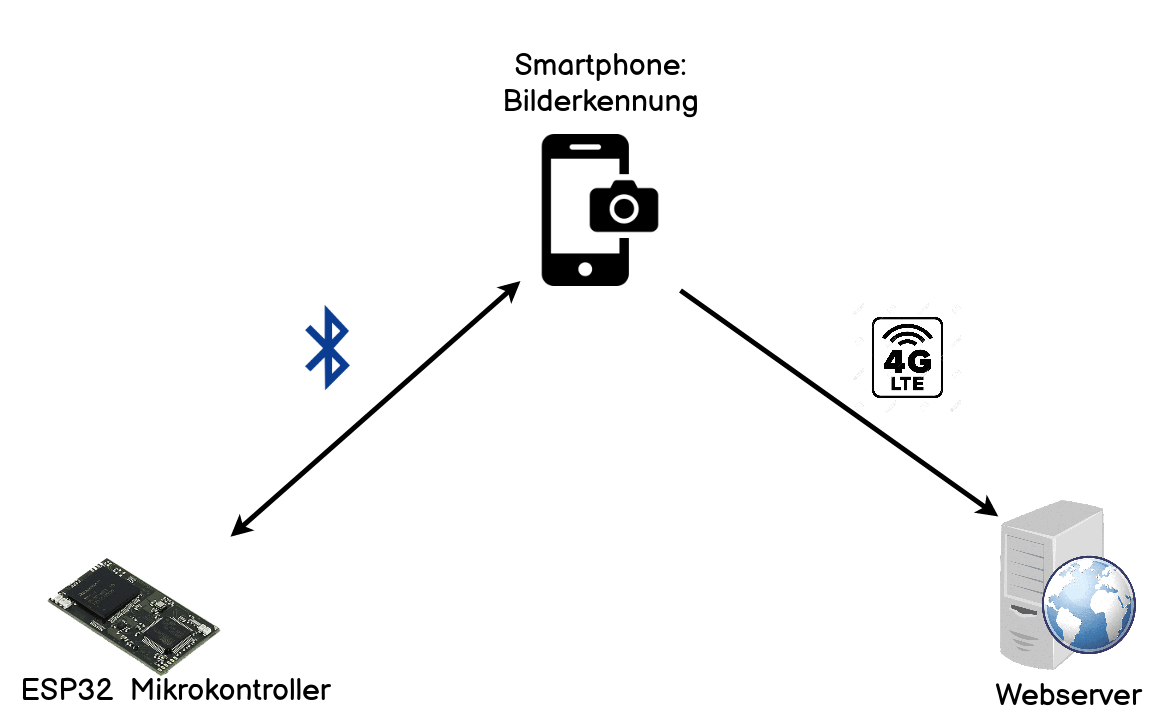
Die Komponenten der Informatik bestehen den Geräten:

* Samsung Galaxy SIII
* Ubuntu 20.04 Webserver

### Architektur

Das Smartphone-App ist die wichtigste Komponente. Es ist wie ein Bindeglied, welches die verschiedenen Komponenten verbindet. Es regelt nicht nur die Kommunikation nach aussen über das Internet (LTE), sondern auch der Informationsaustausch mit dem Mikrocontroller, welcher den Motor steuert.

Diese Verbindungen und ihre funktionale Abhängigkeit sieht man in Abbildung 1. Das ist eine High-Level Übersicht des Systems.



Abbildung

## Bilderkennung

Ganz generell betrachtet, gibt es zwei unterschiedliche Lösungsansätze für die Objekterkennung: Anhalten oder Durchfahren. Anhalten würde bedeuten, an der richtigen Stelle zu bremsen, um dann in einem zweiten Schritt im Stillstand mehrere scharfe Bilder zu schiessen.

Durchfahren würde bedeuten, fortlaufen Bilder zu machen und diese während der Weiterfahrt zu analysieren. Anhalten hat den Nachteil, dass Bremsen und Weiterfahren viel Zeit in Anspruch nehmen kann. Die Qualität der Bilder ist im Stillstand jedoch höher. Durchfahren hat den entscheidenden Nachteil: Die Wahrscheinlichkeit, ein unscharfes oder gar unbrauchbares Bild zu erhalten, ist erheblich höher.

Das muss getestet werden.

Für Pflanzenerkennung können öffentliche APIs verwendet werden. Diese Lösung hat den Vorteil, dass die technische Implementation relativ simpel ist, es muss lediglich ein HTTP-Request an die Pflanzen Identifikations-API geschickt werden. Danach wird das Bild abgeglichen mit einer Datenbank von Millionen von Pflanzenbildern. Zurück kommt eine Liste von Pflanzen, welches diese Pflanze bezeichnen könnten.

Die Bilderkennung kann lokal auf dem Smartphone ausgeführt werden. Es gibt Architekturen «for portable and constrained systems», wie ein Smartphone. (Martinez-Alpiste et al., 2021) In den letzten Jahren haben Deep-Learning-Techniken bei der Objekterkennung Spitzenergebnisse erzielt. Üblicherweise wird dafür sehr leistungsfähig Hardware benötigt. Es gibt jedoch auch Softwareframeworks, welche spezifisch für Geräte mit niedriger Rechenleistung konzipiert sind. Als prominentes Beispiel sei hier TensorFlow Lite Android genannt.

Theoretisch ist es auch möglich, Bilderkennung auf dem Webserver durchzuführen. Dieser Ansatz hat zwei Vorteile: Erheblich leistungsfähiger Hardware und natürlich gibt es viel mehr Wahlmöglichkeiten für die Infrastruktur. Keine Abhängikeit vom Android-Betriebssystem. Jedoch erhöht es die Latenz, da das Bild auf den Webserver geladen werden muss.

## Webserver

Eine erste Analyse der Situation ergab folgendes Szenario.

Der Webserver sollte hochperformant sein, und das auch bei relativ hoher Last von zweihundert Clients. Die Webseite zeigt Statusinformationen des Fahrzeugs an.

Diese Informationen werden in Echtzeit aktualisiert. Basierend auf dieser Prämisse sind einige Designentscheide gefällt worden.

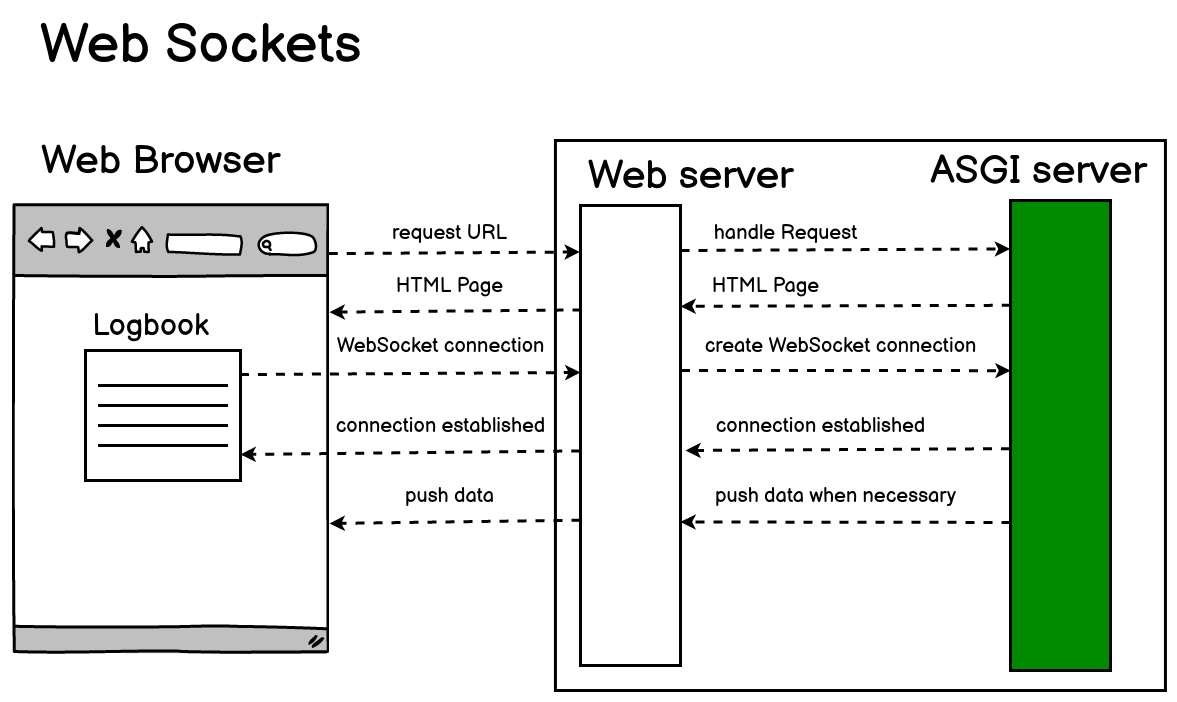
* Frontend minimal halten, lightweight: So wenig javascript wie möglich
* Bilder so klein wie vertretbar.
* Statusupdates erzwingen nicht ein reload der gesamten Website.
* Asynchrones Webframework in Python

Für das Backend wird fastapi verwendet. Fastapi ist ein hochperformantes web Framework. Es bietet ein Schnittstelle für Python Webserver, die asynchronen Code ausführen können. Dieser Zuwachs von Performance lässt sich darauf zurückzuführen, dass das Framework ASGI basiert ist. Das ist ein Protokoll, welches asynchronen Python-Code erlaubt.

### Websockets ‐ Ein fundamentaler Designentscheid

Websockets werden verwendet, zwecks Aufrechterhaltung der bidirektionalen Kommunikation zwischen Smartphone und Webserver. Theoretisch hätten auch einfach HTTP-Requests verwendet werden können. Doch liegen triftige Gründe vor, Websockets zu verwenden.

* Es werden fortlaufend neue Informationen angezeigt. Das Handy schickt Statusupdates, wenn es eine neue Situation gibt. Zum Beispiel kann eine neue Situation sein, dass der Roboter die Pflanze erkannt hat. Sobald das App das merkt, wird auf der Website protokolliert «Pflanze erfolgreich erkannt: Pfefferminze.» Eine Schlüsselerkenntnis ist: *Pro Update ändert sich immer nur sehr wenig, bezogen auf die Menge an Inhalt auf der ganzen Website*. Folglich wäre es unermesslich ineffizient, die Seite komplett neu zu laden. Nur ein kleiner Teil der Website wird jeweils aktualisiert. Mit Websockets lässt sich dafür eine geeignete Architektur implementieren.
* Ein weitere Grund ist ganz einfach Geschwindigkeit. Der Benutzer soll die Website als schnell empfinden. Eine Studie von Google deutet darauf hin, dass 53% aller Besuche auf Webseiten von Mobiltelefon aus abgebrochen werden, wenn diese länger als 3 Sekunden zum Laden braucht. (“Mobile Site Abandonment After Delayed Load Time.”, 2016) Fakt ist, dass Websockets schneller sind als simple HTTP-Requests. Ein Grund dafür ist, dass der Handshake nur beim ersten mal gemacht werden muss, und fortan nicht mehr (bis die Verbindung abgebrochen wird). In anderen Worten, der HTTP-Header ist kleiner, weil er weniger Redundanz enthält.

****

# Hardwarenahe Kommunikation mit dem ESP

Die dem App innewohnende Funktion ist vergleichbar mit einem Pilot: Es gibt Start- und Stoppbefehle indem es den Motor indirekt steuert. Das App kommuniziert mit dem ESP-Modul, welches wiederum den Motor steuern kann.

Die Übertragungsgeschwindigkeit per Bluetooth ist nicht grosser Latenz unterworfen. Es stimmt zwar, dass der initiale Verbindungsaufbau über Bluetooth ziemlich viel Zeit braucht. Das kann schnell fünf Sekunden dauern. Das ist kein Problem, denn die Verbindung wird vor dem Start schon aufgesetzt. Wenn Bluetooth Client und Server verbunden und gekoppelt sind, sind keine grossen Geschwindigkeitseinbussen zu erwarten.

**Quellen**

Mobile Site Abandonment After Delayed Load Time. (o. J.). Think with Google. Abgerufen 28. Oktober 2021, von https://www.thinkwithgoogle.com/consumer-insights/consumer-trends/mobile-site-load-time-statistics/

Martinez-Alpiste, I., Golcarenarenji, G., Wang, Q., & Alcaraz-Calero, J. M. (2021). Smartphone-based real-time object recognition architecture for portable and constrained systems. Journal of Real-Time Image Processing. https://doi.org/10.1007/s11554-021-01164-1