Arbejdsportfolio

Dette dokument fungerer som projektets arbejdsportfolio. Formålet er at dokumentere processen "fra idé til færdig løsning". Her beskrives alle ideer og beslutninger, som fører til løsning af projektets elementer – fra den spæde idé, via skitser, til den endelige implementering.

Teknologier

Vi har valgt at skrive projektets software i **Rust**.

Til dette benytter vi **esp-rs/esp-idf-template**, som er community udviklet af specifikt til at understøtte Rust på ESP32-platformen. Dette giver os fordelene ved Rusts hukommelsessikkerhed kombineret med direkte adgang til hardwaren gennem værktøjer og biblioteker, der er målrettet netop denne platform.

Overvejelser

Master-Slave Arkitektur (Fravalgt)

Vi overvejede først en master-slave arkitektur:

- Hver ESP32 får et tilfældigt tal ved opstart
- Enheden med laveste tal bliver master
- Slaves sender RSSI-data til master
- Master beregner positioner og videresender til server

Fordele:

- Mindre netværkstrafik (kun én enhed kommunikerer med server)
- Decentral beregning

Ulemper:

- Kompleks failover-logik ved master-nedbrud
- Begrænset processorkraft på ESP32 til triangulering
- Svært at debugge og overvåge

Centraliseret Web-Beregning (Valgt)

Vi valgte i stedet en centraliseret arkitektur:

- Alle ESP32-enheder sender RSSI-data direkte til MQTT broker
- Web GUI abonnerer på MQTT og aggregerer data fra alle stationer
- Triangulering beregnes centralt i web-serveren (Rust/Axum)
- Positioner broadcastes til browser via WebSocket

Fordele:

- Simplere ESP32-kode (kun sniffer + MQTT publish)
- Fuld processorkraft til triangulering på server
- Nem debugging via web interface
- Alle stationer er ligeværdige (ingen single point of failure)

- Lettere at tilføje nye stationer

Ulemper:
- Mere netværkstrafik (hver station sender individuelt)
- Afhængig af central server

Konklusion: Den centraliserede løsning gav os hurtigere udvikling og bedre mulighed for at eksperimentere med trianguleringsalgoritmer.

Produktmaal
- Opstilling med flere ESP32 stationer placeret i hjornerne af bord/lokale
- Hver station sniff er wifi i promiscuis mode og sender {id, timestamp, RSSI, (x,y)} til MQTT
- Web backend (Rust) samler data, laver simpel triangulering og sender positioner videre via WebSocket
- Browser GUI viser livedata for at kunne validere opstillingen

Proces og forsog
- Logbog og portfolio opdateres løbende i stedet for en traditionel rapport
- Startede med hurtige proof-of-concepts: wifi sniffing i promiscuis mode, MQTT publish fra ESP32, MQTT subscribe i Rust
- Når en strategi gav brugbare data (RSSI stabilitet), blev den udvidet med ekstra noder for at teste triangulering

Triangulering og dataflow
- Hver ESP32 møler RSSI mod mobile enheder og gemmer egen (x,y)
- Data skubbes til lokal MQTT broker med et simpelt topic pr. station
- Serveren oversætter RSSI til grov afstand og beregner krydspunkt ud fra flere stationer
- Nøjagtighed forventes lav, men tilstrækkelig til at kende hvilket område enheden er i

Persondata og GDPR
- MAC-adresser betragtes som personhenførlige; de hashes på ESP32 inden afsendelse
- Ingen kobling til brugere eller andre metadata; kun tid, station-id og koordinat videregives
- Data gemmes kortvarigt (kun det der behøves til live visning og simpel statistik) og slettes derefter
- Adgang begrænses til lokal udviklings broker og webserver; ingen eksterne integrationer

Datasikkerhed

- MQTT kører lokalt; til udvikling kan en test-broker bruges, men produktion holdes på det lukkede net
- Trafik mellem ESP32 og broker kan sikres med simple pre-shared keys hvis støttet af klienten
- Webserver validerer payloads (skema: id, timestamp, rssi, x, y) for at undgå skrammel-data

Videre arbejde

- Finjustere RSSI til afstand-kurver pr. station for bedre estimering
- Bygge et simpelt heatmap i browseren for at visualisere, hvor mange enheder der er i området over tid

Ik gå ned på sikerhed

- TLS implementering på alle connections
- username/password authentication for MQTT