# Introduktion til projektet

1. Sensor data sendes til et web api, i dette tilfælde fra en embedded enhed.
2. Web api’et står for datahåndtering og forbindelsen til databasen
3. En cross-platform mobil app benytter dataen og kan administrere dataen, data hentes og sendes mellem web api og mobil app via http requests.

Hele projektet er lavet med fokus på at benytte løs kobling, som betyder at de forskellige enheder og komponenter i projektet er opbygget uafhængigt af hinanden.  
Det giver mulighed for at kunne ændre dele i systemet, uden at påvirke resten af systemet.   
Det gør projektet mere fleksibelt, modulerbart og giver høj skalerbarhed.

# Embedded modul med sensor enhed

Den embeddede del som her i projektet sender sensor data til api’et er opbygget af med en ESP8266 NodeMCU og en DHT11, som sender temperaturmålinger til web api’et ved at lave et url kald med måling som parametre, Det kræver en åben netværksforbindelse

Api’et er egentlig ligeglad med om sensor dataen kommer fra en esp8266 eller en dht11. det kunne have været fra en tredjeparts sensor, eller whatever, så længe at dataen kan modtages via netværk.

# 

# Web Api

Lavet i Symfony, med en MySQL database som ligger i en docker container.

Jeg har benyttet en ORM (Doctrine) - code first approach: opbygget tabellerne i min kode, i entity modeller, og migreret til MySQL fra CLI’en.

For at opdele database logikken med den funktionalitetslogik der foregår i Controllers, har jeg benyttet repository mønstret i mit web api.

Alle http requests har jeg testet løbende og ved ændringer ved hjælp af Postman, hvor jeg har opsat hvert eneste request i en collection, som automatisk kan teste hele collection.

Da Doctrine håndterer database logikken, er symfony api’et egentligt ligeglad med om det er en mysql eller enhver anden database der ligger bag, det giver frihed i at kunne vælge hvilken som helst database der er lettest tilgængelig ved hosting udbyder.

# Mobil App

Jeg har valgt React-native som er et cross-platform framework til app udvikling.

Fra app’en kan der modtages data, men også oprettes, redigere og slette data, det sker via http requests til api’et.

Til det har jeg brugt en indbygget javascript metode kalde FETCH.

React er komponentbaseret, hvilket opdeler kode i genanvendelige små moduler som let kan implementeres rundt omkring i koden, hvor det nu er nødvendigt.   
App’en er lavet uafhængig af resten af systemet, så længe Api logikken er ens og håndteres via http requests, kunne dataen komme fra et hvilket som helst api.

# Hvad har jeg lært?

Hele udviklingsmiljøet i dette projekt, har være en udfordring at arbejde med, det er ekstremt tungt og kræver rigtig mange dependencies for overhovedet at kunne køre.

Fremadrettet vil jeg gerne have fokuseret mere på at opbygge et miljø i docker containers som let kan startes med et opstarts script eller lignende.   
Eller vælge nogle teknologier som kan startes umiddelbart let.

Fremadrettet vil jeg fokusere på at benytte et mere testbart miljø, som eksempelvis .net udvikling i visual studio eller lignende, hvor kode kan debugges direkte fra IDÈ.  
Koden i det nuværende projekt har været vanskeligere at debugge, da der i dette miljø kun er mulighed for at debugge ved at lave print out, som console.log, var\_dump og Print\_f.

Det er helt sikkert ikke sidste gang at jeg benytter komponentbaseret udvikling, og jeg er lidt ked af at jeg ikke fik det udnyttet helt til fulde i dette projekt, da det virkelig gør koden let at genanvende, men også hurtigere at fikse / tilpasse hvis nødvendig, fremfor at skulle rette kode 1.000.000.000 steder.