Systemdokumentasjon

Martin Johannes Nilsen
Ole Jonas Liahagen
Simon Årdal

Vår 2021

1 Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
04.05.21	1.0	Ferdigstilling, første utkast	Simon Årdal
15.05.21	1.1	Små justeringer på format og illustrasjoner	Martin J. Nilsen

2 Introduksjon

Dette dokumentet er skrevet i forbindelse med vårt bachelorprosjekt, Got Your Back, og har som formål å gi interessenter en innføring i systemets grunnlegende virkemåte, samt gi vedkommende en oversikt over systemets mest essensielle og kritiske komponenter. Målet med dokumentet er at lesere skal sitte igjen med en overordnet god oversikt over hvordan systemet fungerer, hvilke komponenter det er bygget opp av, og hvordan komponentene samhandler med hverandre. Dokumentet vil inneholde seksjoner om arkitektur, prosjektstruktur, klassediagram, server-tjenester, kontinuerlig integrasjon, testing og brukerveiledning angående installasjon og kjøring.

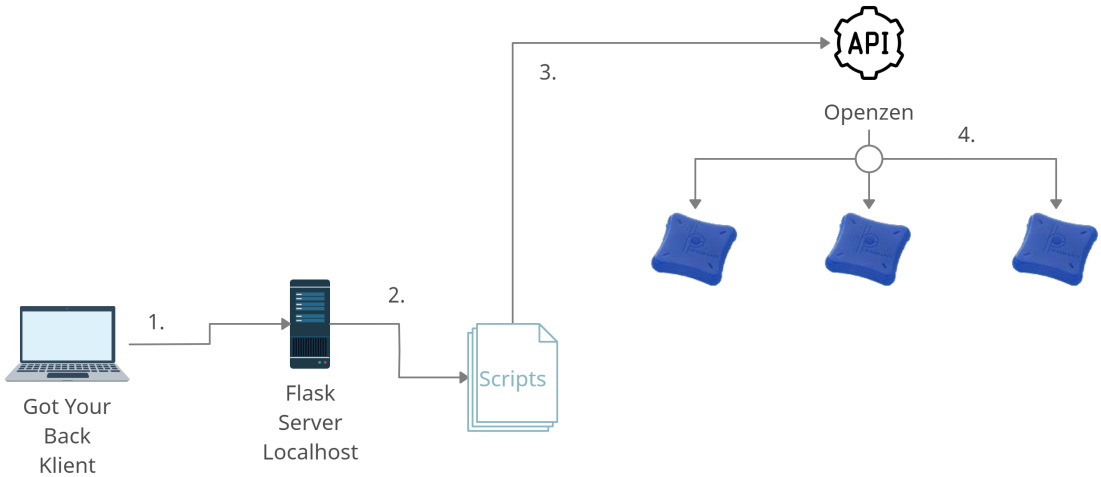
3 Arkitektur

For å kunne gi en tilstrekkelig innføring i systemets arktitektur har vi delt seksjonen opp i fire sub-seksjoner som hver tar for seg situasjoner som i all hovedsak er situasjoner som beskriver systemets kjernefunksjonalitet. Disse fire situasjonene er:

- Kommunikasjon med sensorer
- Uthenting av klassifiseringer
- Klassifisering av sittestillinger og behandling av sensordata på samme tid
- Klassifisering av sittestillinger, sensordatabehandling og uthenting av klassifiseringer på samme tid

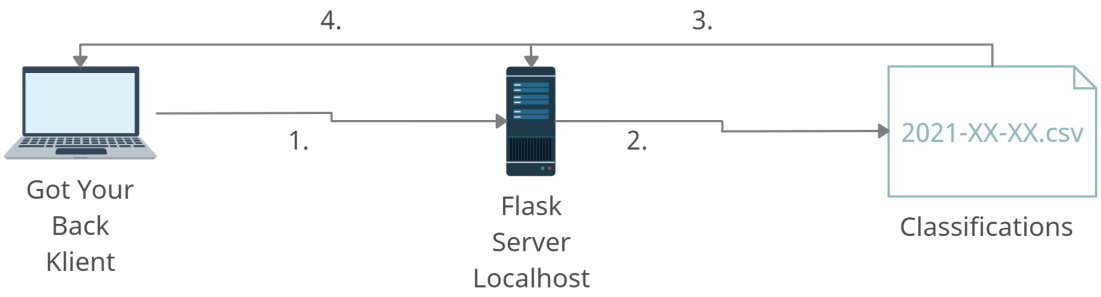
Dersom leseren oppnår en god forståelse av hva som skjer ved hver av disse tre situasjonene vil brukeren grovt sett forstå hva som skjer i systemet ved alle brukstilfeller.

3.1 Kommunikasjon med sensorer



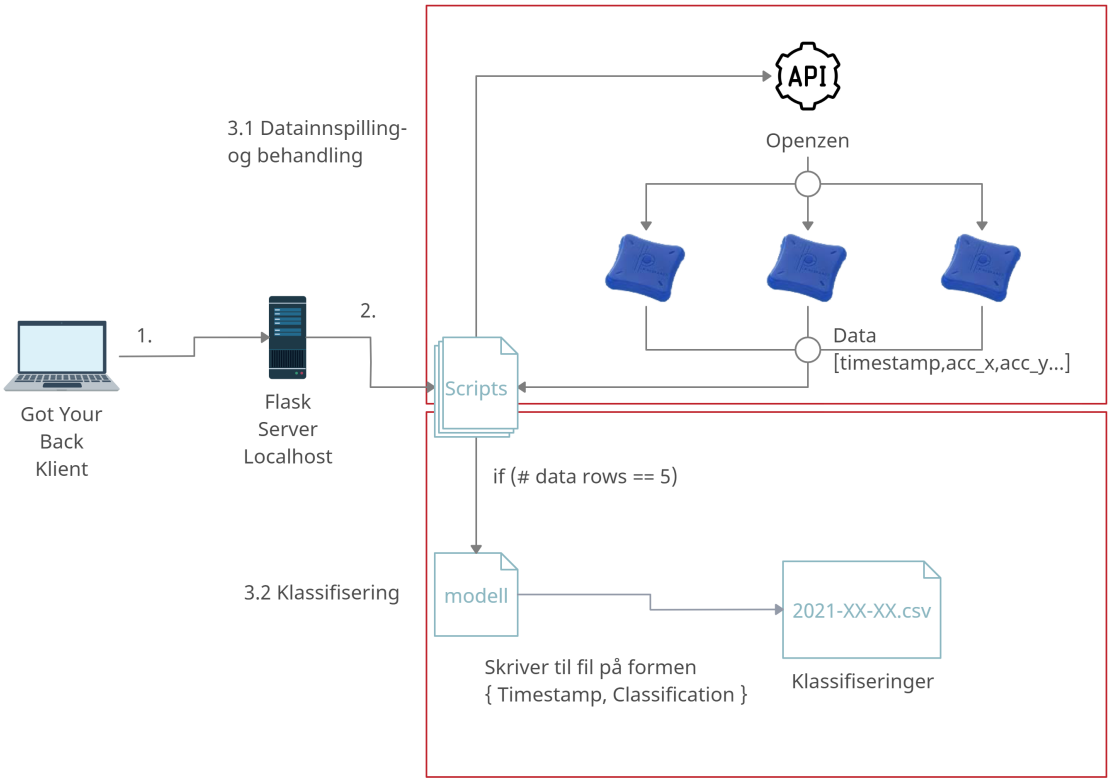
1. Klienten sender forespørsel til Flask server som kjører på localhost
2. Serveren kommuniserer med sensorene via python-scripts. Detaljene for hvordan scriptsene fungerer vil ikke bli gått gjennom i denne seksjonen.
3. Scriptsene benytter seg av API-et Openzen, som er et API som er utviklet av sensorprodusentene som tillater kommunikasjon med sensorene.
4. Via Openzen kan man gi sensorene ulike beskjeder, som f.eks. å koble til, endre innspillingsfrekvens, starte synkronisering og å starte/stoppe innspilling av data. Dersom feil forekommer, vil Openzen fortelle dette til scriptsene, som forteller det til serveren osv. Dersom feil forekommer vil de med andre ord bli sendt nedover hierarkiet.

3.2 Uthenting av klassifiseringer



1. Klienten sender forespørsel til Flask server som kjører på localhost
2. Serveren henter klassifiseringer som ligger lagret i en lokal csv-fil for den spesifike dagen.
3. Klassifiseringene blir sendt tilbake til server, så videre til klient.

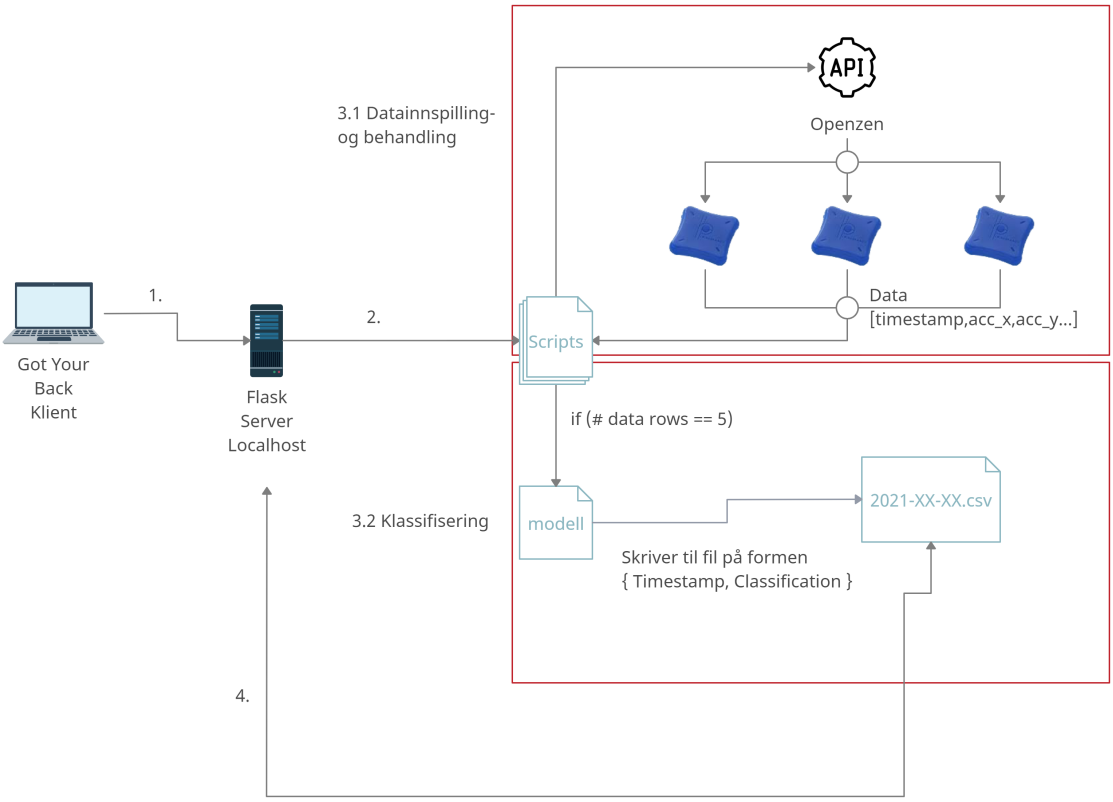
3.3 Klassifisering av sittestillinger og behandling av sensordata på samme tid



1. Klienten sender fetch-kall til en Flask-server som kjører på localhost. Dette fetch-kallet vil i dette tilfellet være en forespørsel om å begynne innsamling og klassifisering av data.

2. Serveren kjører korresponderende endepunkt, og vil benytte seg av python-scripts. I dette tilfellet vil to parallelle tråder bli iverksatt; en for datainnspilling- og behandling (3.1), og en annen tråd som gir den ferdig behandlede dataen til modellen for klassifisering (3.2). De to parallelle prosessene er innrammet i rødt i figuren.

3.4 Klassifisering av sittestillinger, behandling av sensordata og uthenting av klassifiseringer på samme tid



I dette scenarioet er punkt 1 - 3 likt som før, men ved vanlig kjøring vil også enda en prosess være aktuell - nemlig å hente ut klassifiseringer samtidig som prosessene fra 2.3 blir gjennomført. Klienten kan be serveren om å hente ut klassifiseringsdata (enten ferske eller eldre) uavhengig av om datainnspilling eller klassifisering foregår. Dette er for at klienten skal kunne presentere interessant data selv om sensorene spiller inn data på samme tid.

4 Prosjektstruktur

Backend

```
Got Your Back
├── backend
│   ├── openzen.pyd
│   ├── openzen.so
│   ├── server.py
│   └── server_test.py
├── classifications
│   ├── [classification files]
│   └── generate_dummy_data.py
├── data
│   ├── annotation
│   │   ├── testing
│   │   └── training
│   ├── overview_of_data_collection.ipynb
│   ├── sensor_calibrations
│   │   ├── 00043E3036EB_CAL
│   │   ├── 00043E4B31EE_CAL
│   │   └── 00043E4B3326_CAL
│   ├── test_data
│   │   └── [testing datasets]
│   └── train_data
│       └── [training datasets]
├── model
│   ├── models
│   ├── src
│   │   ├── ANN.ipynb
│   │   ├── CNN.ipynb
│   │   ├── LSTM.ipynb
│   │   └── RFC.ipynb
│   └── utils
│       ├── Data_presentation.ipynb
│       ├── declarations.py
│       ├── df_utils.py
│       ├── Preprocess_data.ipynb
│       └── Precision.ipynb
├── reports
├── requirements.txt
├── scripts
│   ├── classification_handler.py
│   ├── data_queue.py
│   ├── openzen.pyd
│   ├── openzen.so
│   ├── realtime_classify.py
│   ├── rnn_utils.py
│   ├── sensor_bank.py
│   └── tests
│       ├── data_queue_test.py
│       ├── openzen.pyd
│       └── sensor_bank_test.py
├── start.bat
├── start.command
├── start_server.bat
└── start_server.command
```

Figur 1: Trestruktur for backend

Frontend

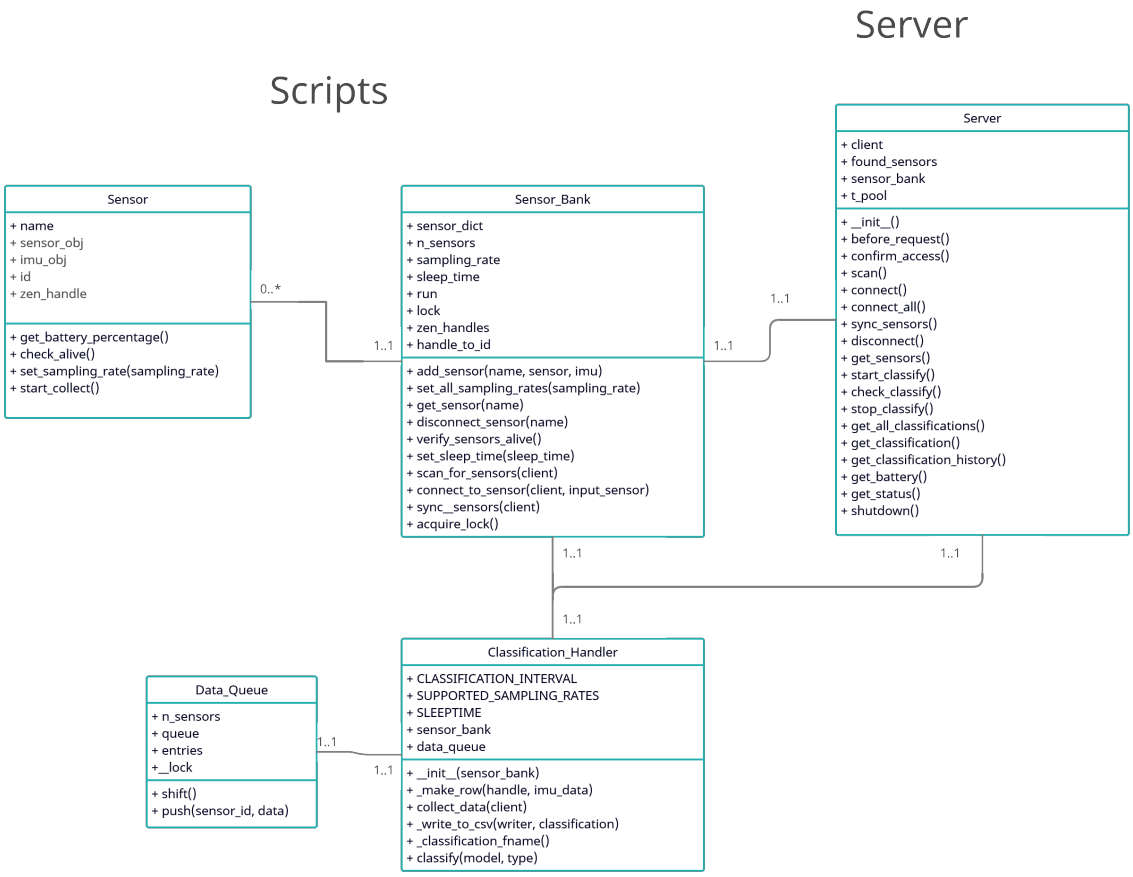
```
Got Your Back/frontend
├── README.md
├── assets
├── build
├── dist
├── package-lock.json
├── package.json
├── public
│   ├── electron.js
│   ├── favicon.ico
│   ├── index.html
│   ├── manifest.json
│   └── robots.txt
├── src
│   ├── @types
│   ├── App.tsx
│   ├── Routing.tsx
│   ├── assets
│   ├── components
│   │   ├── Buttons
│   │   │   ├── Button.component.tsx
│   │   │   └── SensorButton.component.tsx
│   │   ├── ClassificationContent
│   │   │   └── ClassificationContent.component.tsx
│   │   ├── ColumnChart
│   │   │   └── ColumnChart.component.tsx
│   │   ├── ContentBox
│   │   │   └── ContentBox.component.tsx
│   │   ├── HomeShade
│   │   │   └── HomeShade.component.tsx
│   │   ├── InfoTooltip
│   │   │   └── InfoTooltip.component.tsx
│   │   ├── LineChart
│   │   │   └── LineChart.component.jsx
│   │   ├── NavBar
│   │   │   └── NavBar.component.tsx
│   │   ├── PieChart
│   │   │   └── PieChart.component.jsx
│   │   ├── RecordContent
│   │   │   └── RecordContent.component.tsx
│   │   ├── SensorListContent
│   │   │   └── SensorListContent.component.tsx
│   │   ├── SensorModal
│   │   │   └── SensorModal.component.tsx
│   │   ├── SensorRow
│   │   │   ├── SensorRowHome.component.tsx
│   │   │   └── SensorRowModal.component.tsx
│   │   ├── StatusBar
│   │   │   └── StatusBar.component.tsx
│   │   ├── StatusGraphPopup
│   │   │   └── StatusGraphPopup.component.tsx
│   │   └── StatusPopup
│   │       └── StatusPopup.component.tsx
│   ├── index.css
│   ├── index.tsx
│   ├── theme.tsx
│   ├── utils
│   │   ├── dateUtils.ts
│   │   ├── handleErrors.ts
│   │   ├── postureNames.ts
│   │   ├── sensorPlacement.ts
│   │   ├── serverUtils.ts
│   │   └── useInterval.ts
│   └── views
│       ├── HelpView
│       │   └── HelpView.tsx
│       ├── HistoryView
│       │   └── HistoryView.tsx
│       ├── HomeView
│       │   └── HomeView.tsx
│       └── ReportView
│           └── ReportView.tsx
└── tsconfig.json
```

Figur 2: Trestruktur for frontend

5 Klassediagram

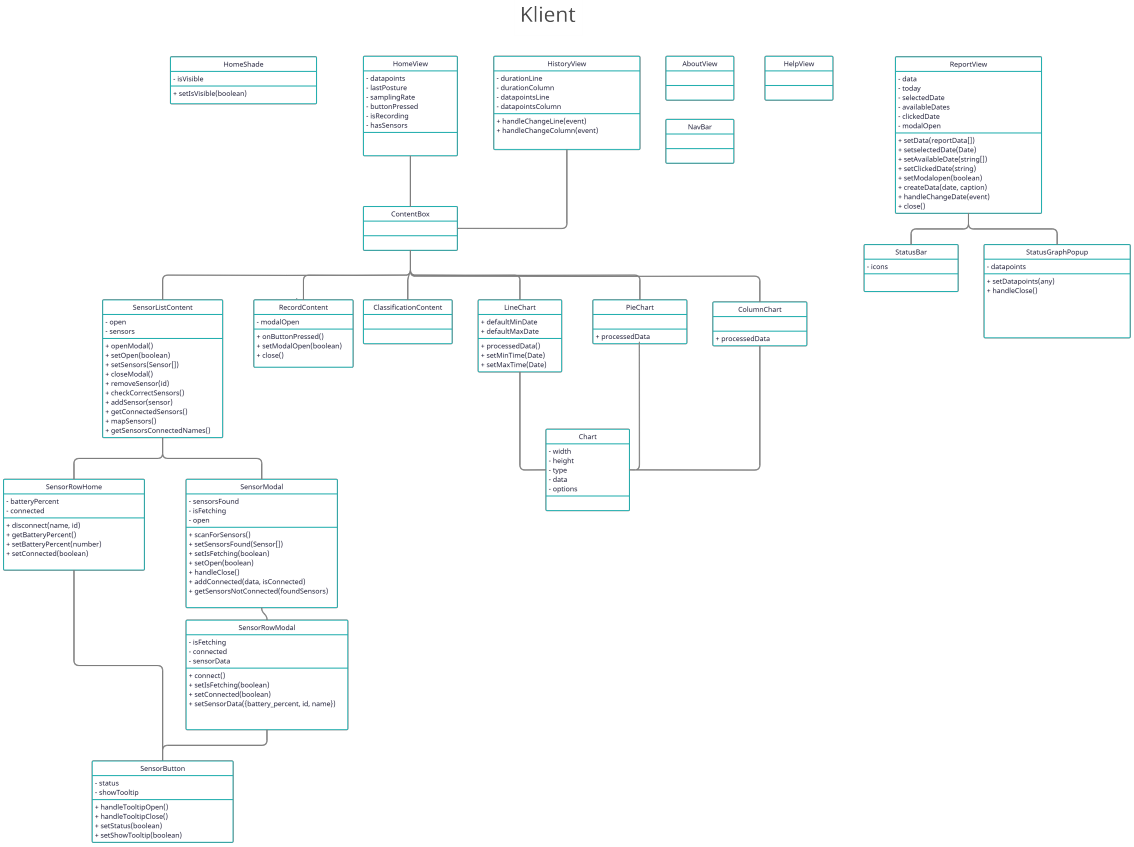
Ettersom klassediagrammene kan være vanskelige å lese er de også vedlagt i prosjektmappen. Disse er av høy oppløsning og er lettere å zoome inn på og lese.

Backend



Figur 3: Klassediagram for backend

Frontend



Figur 4: Klassediagram for frontend

6 Server-tjenester

6.1 Initialisering

```
@app.before_first_request
def init():
```

Initialiserer Sensor_Bank og clientobjekter.

```
@app.before_request
def before_request():
```

Etablerer hvilke headere som skal godkjennes ved OPTIONS-kall.

```
@app.route("/")
def confirm_access():
```

For å bekrefte at serveren er oppe og kjører.

6.2 Setup

```
@app.route("/setup/scan")
def scan():
```

Benytter Classification_Handler for å skanne etter sensorer. Dersom de blir funnet blir de lagt til i found_sensors.

```
@app.route("/setup/connect", methods=["OPTIONS", "POST"])
def connect():
```

Benytter Classification_Handler for å koble til en sensor med navn sendt fra klienten, dersom denne sensoren finnes i found_sensors. Når de blir tilkoblet blir de lagt til i Sensor_Bank.

```
@app.route("/setup/connect_all")
def connect_all():
```

Kobler til alle sensorene som er lagt til i found_sensors. Sensorene som blir tilkoblet blir lagt til i Sensor_Bank.

```
@app.route("/setup/sync")
def sync_sensors():
```

Benytter Classification_Handler for å synkronisere sensorene som er lagt inn i Sensor_Bank.

```
@app.route("/setup/disconnect", methods=["OPTIONS", "POST"])
def disconnect():
```

kobler fra de sensorene hvis navn blir sendt fra klienten.

```
@app.route("/setup/get_sensors")
def get_sensors():
```

Henter ut informasjon om alle sensorene som ligger i Sensor_Bank.

6.3 Klassifisering

```
@app.route("/classify/start")
def classification_pipe():
```

Starter datainnsamling- og klassifiseringstråder som kjører i parallell.

```
@app.route("/classify/status")
def check_classify():
```

Returnerer om sensorene i Sensor_Bank samler inn data eller ikke.

```
@app.route("/classify/stop")
def stop_classify():
```

Setter sensorene i Sensor_Bank til å stoppe å klassifisere.

6.4 Henting av klassifiseringer

```
@app.route("/classifications")
def get_all_classifications():
```

Henter samtlige klassifiseringer fra dags dato.

```
@app.route("/classifications/latest")
def get_classification():
```

Henter ut den ferskeste klassifiseringen.

```
@app.route("/classifications/history")
def get_classifications_history():
```

Henter ut klassifiseringer fra de siste 7, 14 eller 30 dagene dersom det finnes klassifiseringer fra de dagene.

```
@app.route("/classifications/reports")
def get_report_classifications():
```

Henter klassifiseringer for å vise hvordan man har sittet i tilknyttning til en rapport som har blitt skrevet.

6.5 Sensor informasjon

```
@app.route("/sensor/battery")
def get_battery():
```

Returnerer batteriprosenten til sensoren som oppgis med navn fra klienten.

```
@app.route("/status")
def get_status():
```

Returnerer om sensorene i Sensor_Bank samler inn data og hvor mange som ligger der.

6.6 Brukerrapporter

```
@app.route("/reports", methods=[POST])
def write_report():
```

Skriver rapport til fil.

```
@app.route("/reports", methods=[GET])
def get_report():
```

Henter rapporter fra fil.

```
@app.route("/reports/available")
def get_report_months_available():
```

Finner måneder som det finnes rapporter for.

7 Installasjon og kjøring

7.1 Forhåndskrav

1. Python https://www.python.org/downloads/
2. pip https://pip.pypa.io/en/stable/installing/
3. node.js https://nodejs.org/en/download/

7.2 Installasjon

1. Last ned siste release fra https://github.com/OleJonas/Got-Your-Back og pakk ut innholdet.
2. Åpne en terminal i prosjektets 'root'-mappe
3. Kjør

   ```
   pip install -r requirements.txt
   ```

4. Start applikasjonen ved å kjøre

   ```
   start.bat
   ```

   på Windows, og

   ```
   start.command
   ```

   på macOS.

8 Testing

Tester er skrevet hvor vi har funnet det relevant. For å sikre at metoder fungerer som de skal, er det blitt skrevet tester med dummy-data som etterligner virkelige situasjoner, og som kan kjøre uten at serveren er oppe eller at sensorene er tilkoblet. Ved å inkludere disse i en kontinuerlig integrasjon og kjøre for hver push, sikrer man at metodene oppfører seg som de skal, og får beskjed dersom de ikke gjør det.

Modulene som testes er (med testfilenes plassering i parantes):

- Server (Got-Your-Back/backend/server_test.py)
- Data Queue (Got-Your-Back/scripts/tests/data_queue_test.py)
- Sensor Bank (Got-Your-Back/scripts/tests/sensor_bank_test.py)

9 Kontinuerlig integrasjon

En CI-pipeline er blitt inkludert i prosjektet via GitHub Actions. Denne inkluderer

- Python-setup og tester for python-filer
- Node-setup og bygging av frontend
- Bygging og deploying av API-dokumentasjon

og finnes på https://github.com/OleJonas/Got-Your-Back/actions