Vedlegg 7 Systemdokumentasjon for ROBPLAN INSIGHT

Systemdokumentasjon for ROBPLAN INSIGHT

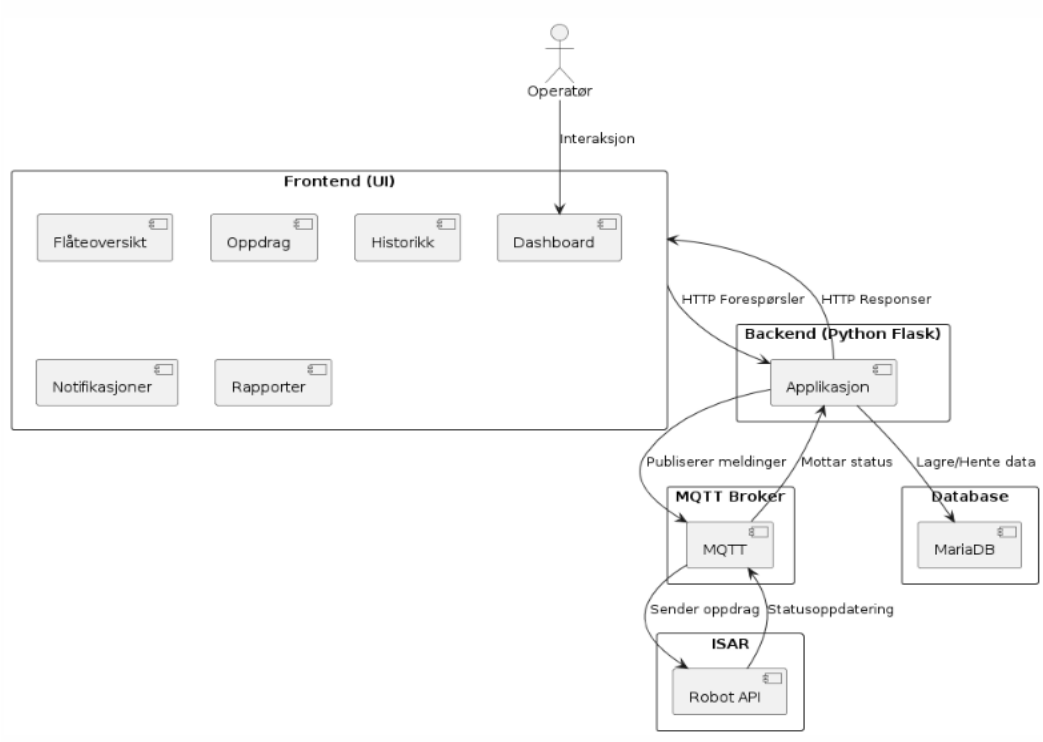
1. Introduksjon

ROBPLAN INSIGHT er et dashboard utviklet for SINTEF og Equinor for å integreres med Equinors ISAR (Integration and Supervisory control of Autonomous Robots). Systemet gir operatører et verktøy for effektiv overvåkning og kontroll av autonome roboter.

2. Systemoversikt

ROBPLAN INSIGHT er designet for å håndtere en flåte av autonome roboter som utfører inspeksjoner på industrielle anlegg. Systemet gir operatøren mulighet til å starte, overvåke og evaluere oppdrag. Det gir sanntidsdata, oppdragsadministrasjon og historikk over utførte oppdrag.

Nedenfor er et diagram som viser systemarkitekturen for ROBPLAN INSIGHT. Diagrammet illustrerer de ulike komponentene i systemet og hvordan de samhandler med hverandre.



Figur 1: UI diagram

- **Frontend (UI)**: Viser de forskjellige delene av brukergrensesnittet, som flåteoversikt, oppdrag, historikk, dashboard, notifikasjoner, og rapporter.

- **Backend (Python Flask)**: Viser hvordan backend håndterer HTTP-forespørsler og -responser, publiserer meldinger til MQTT Broker, og lagrer/henter data fra databasen.
- **MQTT Broker**: Viser kommunikasjonen mellom backend og MQTT Broker, samt MQTT Brokerens interaksjon med ISAR.
- **ISAR**: Viser hvordan oppdrag og statusoppdateringer håndteres av ISAR.
- **Database (MariaDB)**: Viser hvordan data lagres og hentes fra databasen.

4. Beskrivelse av komponenter

Backend

Backend er utviklet med Flask og håndterer API-endepunkter for oppdragsadministrasjon, robotstatus, m.m. Eksempel på views.py for oppdragshåndtering:

```python
@missions_bp.route('/start-mission', methods=['POST'])
@login_required
def start_mission():
    default_mission_data = request.json

    subquery = db.session.query(
        RobotInfo.isar_id,
        db.func.max(RobotInfo.id).label('max_id')
    ).group_by(RobotInfo.isar_id).subquery()

    latest_robot_info = db.session.query(
        RobotInfo.isar_id, RobotInfo.robot_name, RobotInfo.battery_level, RobotInfo.robot_status, RobotInfo.port
    ).join(subquery, RobotInfo.id == subquery.c.max_id).all()

    available_robot = next((robot for robot in latest_robot_info if robot.battery_level > 60 and robot.robot_status != 'busy'), None)

    if not available_robot or not available_robot.port:
        return jsonify({"error": "No available robot with sufficient battery, idle status, or valid port"}), 400

    isar_api_url = f'http://localhost:{available_robot.port}/schedule/start-mission'

    transformed_data = transform_mission_data(default_mission_data)

    try:
        response = requests.post(isar_api_url, json=transformed_data)
        response.raise_for_status()
    except requests.exceptions.RequestException as e:
        return jsonify({"error": "Failed to start mission", "details": str(e)}), 500

    return jsonify({"message": "Mission successfully started", "data": response.json()}), 200
```

Figur 2: start_mission-funksjon

Funksjonen er definert som en rute som håndterer POST-forespørsler til **/start-mission**. Ruten er beskyttet med **@login_required**, noe som krever at brukeren må være autentisert for å få tilgang.

Forespørselens JSON-data mottas og lagres i **default_mission_data**.

For å finne den nyeste informasjonen om hver robot, utføres en subquery for å finne den høyeste **RobotInfo.id** for hver **isar_id**. Deretter hentes den nyeste informasjonen om hver robot basert på resultatene fra subqueryen.

Funksjonen går gjennom den hentede robotinformasjonen for å finne en robot som har et batterinivå over 60 og som ikke er opptatt (**robot_status != 'busy'**). Hvis ingen slik robot finnes, returnerer funksjonen en feilmelding.

```
available_robot = next((robot for robot in latest_robot_info if robot.battery_level > 60 and robot.robot_status != 'busy'), None)

if not available_robot or not available_robot.port:
    return jsonify({"error": "No available robot with sufficient battery, idle status, or valid port"}), 400
```

Basert på den tilgjengelige robotens port, bygges URL-en for å sende forespørselen til robotens API for å starte oppdraget.

```
isar_api_url = f'http://localhost:{available_robot.port}/schedule/start-mission'
```

Oppdragsdataene transformeres til riktig format for robotens API ved hjelp av **transform_mission_data**-funksjonen.

```
transformed_data = transform_mission_data(default_mission_data)
```

En POST-forespørsel sendes til robotens API med de transformerede oppdragsdataene. Hvis forespørselen feiler, returneres en feilmelding med detaljer om feilen.

```
try:
    response = requests.post(isar_api_url, json=transformed_data)
    response.raise_for_status()
except requests.exceptions.RequestException as e:
    return jsonify({"error": "Failed to start mission", "details": str(e)}), 500
```

Hvis oppdraget ble startet suksessfullt, returnerer funksjonen en suksessmelding sammen med dataen fra robotens API.

```
return jsonify({"message": "Mission successfully started", "data": response.json()}), 200
```

HTTP-statuskoder

- **200 OK**: Oppdraget ble lagt til.
- **400 Bad Request**: Ugyldig forespørsel.
- **401 Unauthorized**: Mangler eller ugyldig autentisering.
- **500 Internal Server Error**: Serveren feilet ved behandling av forespørselen.

Frontend

Frontend er bygget med HTML, CSS, JavaScript og Bootstrap. Brukergrensesnittet gir en oversikt over oppdrag, roboter, notifikasjoner og historikk.

Database

Databasen er implementert med MariaDB og administreres via phpMyAdmin. Tabellene inneholder data om brukere, roboter, oppdrag og notifikasjoner.

Integrasjoner

Systemet integreres med ISAR for robotadministrasjon og MQTT for sanntidskommunikasjon.

5. Teknologier og Rammeverk

- **Backend:** Python, Flask
- **Frontend:** HTML, CSS, JavaScript, Bootstrap
- **Database:** MariaDB, Docker
- **Kommunikasjon:** MQTT, WebSockets
- **Utviklingsmiljø:** Visual Studio Code, GitHub

6. Installasjon og oppsett

For detaljerte instruksjoner om installasjon og oppsett, vennligst se README-filen i prosjektets rotmappe. README-filen inneholder instruksjoner for hvordan du setter opp utviklingsmiljøet, kjører Docker-container, og installerer nødvendige avhengigheter.

Repoet ligger her: https://github.com/patricuj/RobPlan-UiT

7. Brukerveiledning

For en fullstendig brukerveiledning, inkludert hvordan man bruker systemets hovedfunksjoner, vennligst se README-filen i prosjektets rotmappe. README-filen gir instruksjoner om hvordan du kan navigerer i systemet, administrerer oppdrag, og se på rapporter.

Repoet ligger her: https://github.com/patricuj/RobPlan-UiT