Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap
Institutt for datateknologi, elektroteknologi og realfag

# SMART OCEAN DATA VISUALISATION DASHBOARD

## Systemdokumentasjon

## Versjon 1.0

# REVISJONSHISTORIE

| Dato | Versjon | Beskrivelse | Forfatter |
|---|---|---|---|
| <15/05/2023> | <1.0> | <Oppsett, statusoppdatering> | <Eirik, Pål> |

# INNHOLDSFORTEGNELSE

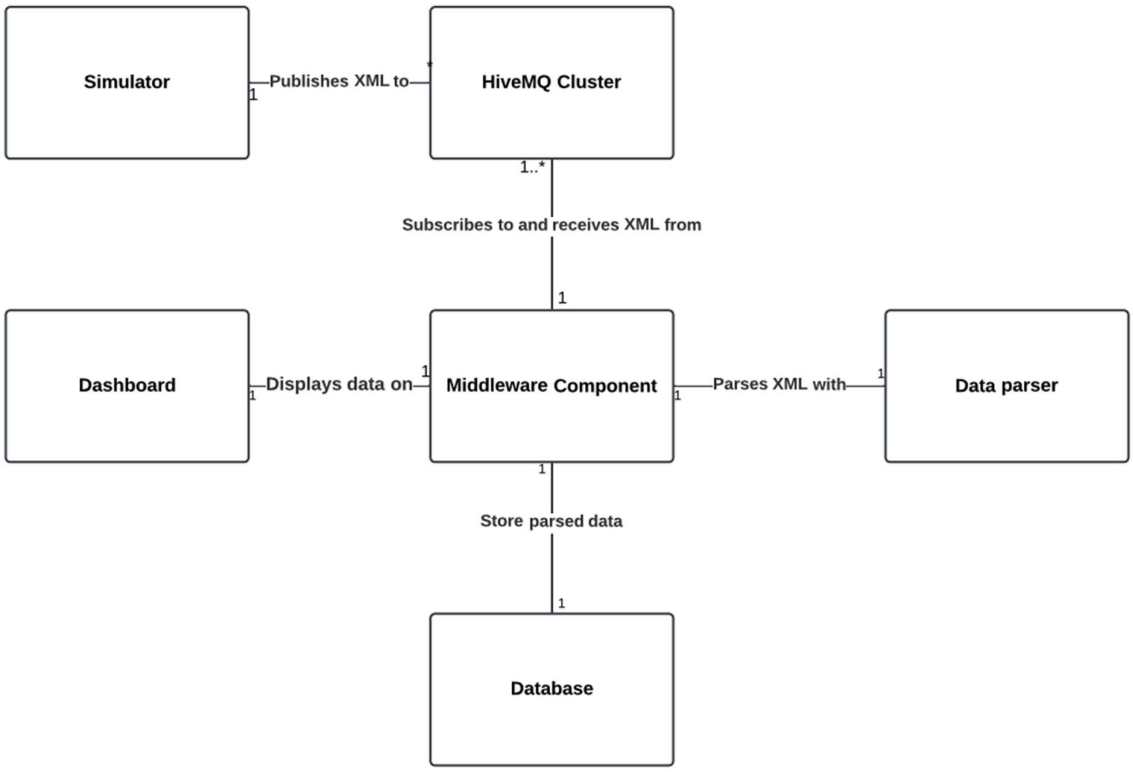
# 1 INNLEDNING

Følgende er en systemdokumentasjon. Det inneholder oversikter over Smart Ocean Dashbordet sin arkitektur, prosjektstruktur, klasser, database modeller og tester.

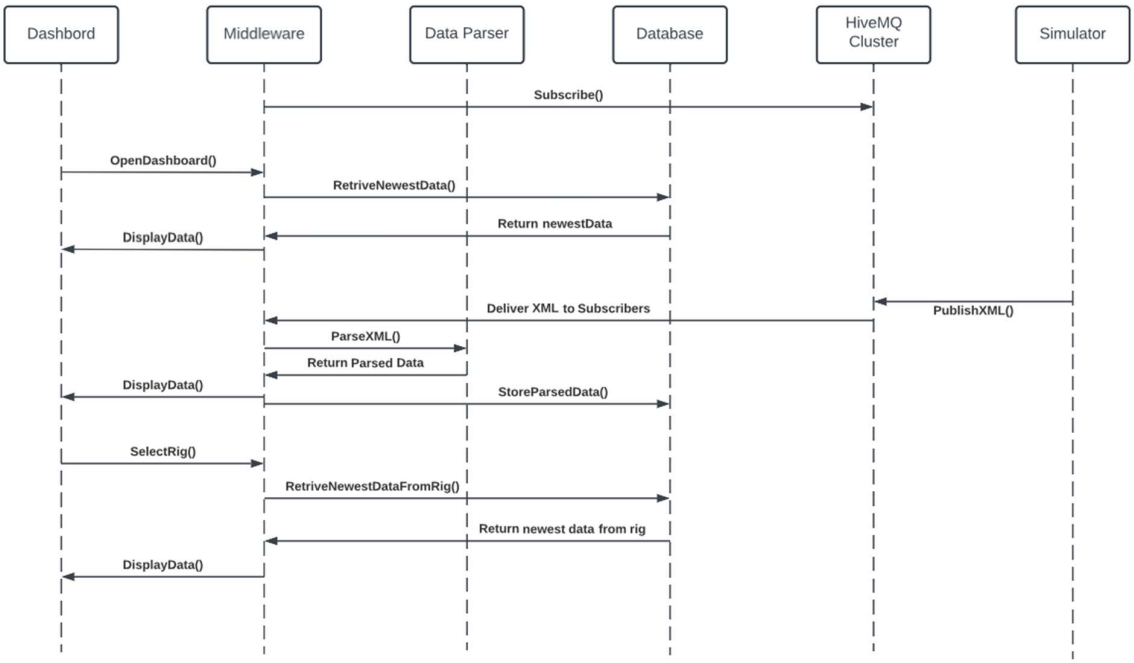
Lucidcharts (Lucid, 2023) ble brukt til moddelering.

# 2 ARKITEKTUR

Software arkitektur diagrammet under viser en oversikt over hovedkomponentene og forholdet mellom dem.




Sekvensdiagrammet under viser en mer detaljert programflyt mellom komponentene



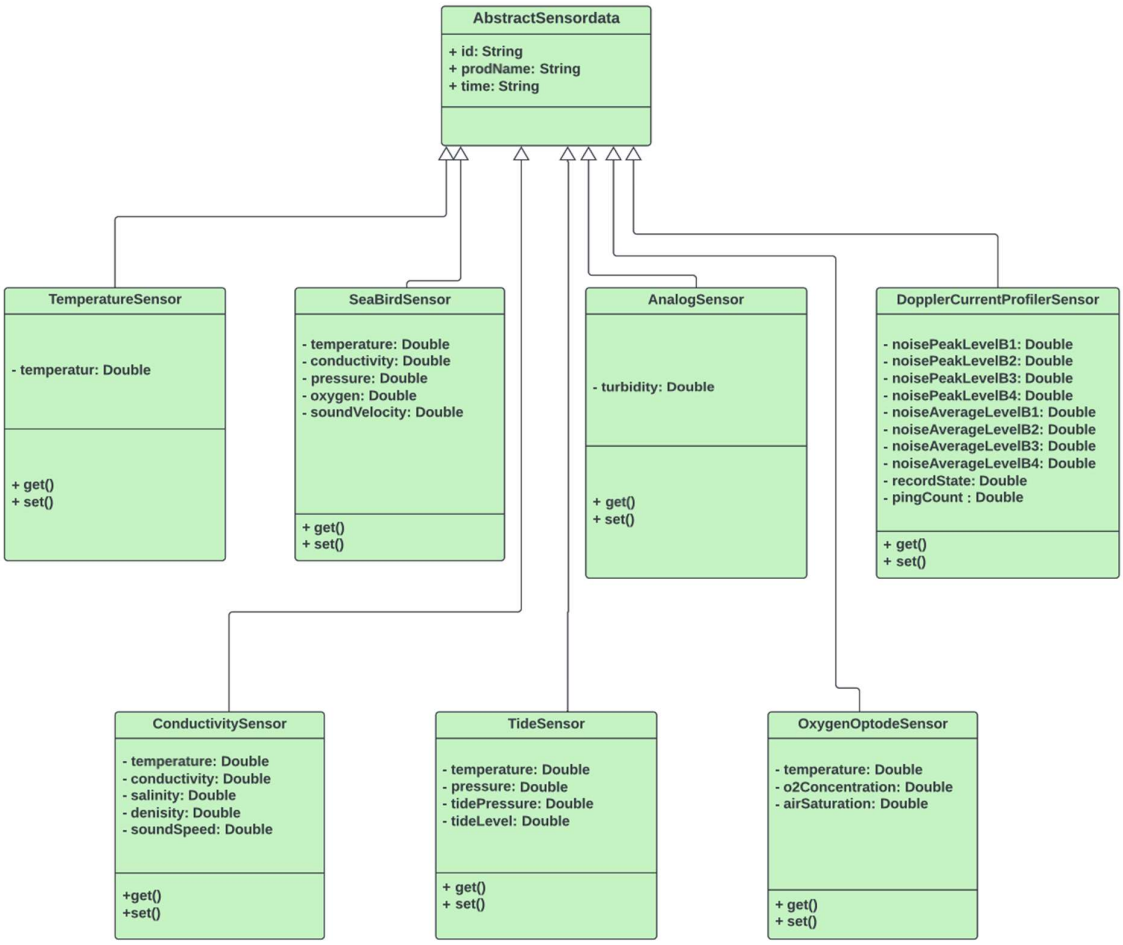
# 3 PROSJEKTSTRUKTUR

- SmartOceanDashboard
  - src
    - main
      - java
        - AustevollNordData
        - AustevollSorData
        - Database
          - DAO
        - Middleware
          - MiddlewareMain.java
          - XMLParser.java
          - Test
            - xmlErrorTestFile.xml
            - XMLParserTest.java
        - SensorClass
          - AbstractSensorData.java
          - AnalogSensor.java
          - ConductivitySensor.java
          - DopplerCurrentProfilerSensor.java
          - ProdNameConstants.java
          - SeaBirdSensor.java
          - TemperatureSensor.java
          - TideSensor.java
        - Servlets
          - UpdateDataServlet.java
        - Simulator
          - HiveMQPublishSimulator.java
        - WebSocket
          - SocketHandler.java
          - SocketServer.java
      - Resources
        - META-INF
          - beans.xml
          - persistence.xml
      - Webapp
        - css
          - SFI_SmartOcean_logo.png
          - style.css
        - js
          - map.js
          - UpdateDataSocket.js
          - UpdateDataServlet.js
        - WEB-INF
        - About.html
        - Admin.html
        - help.html
        - Index.jsp
        - Settings.html
- build.gradle
- gradlew
- gradlew.bat
- settings.gradle

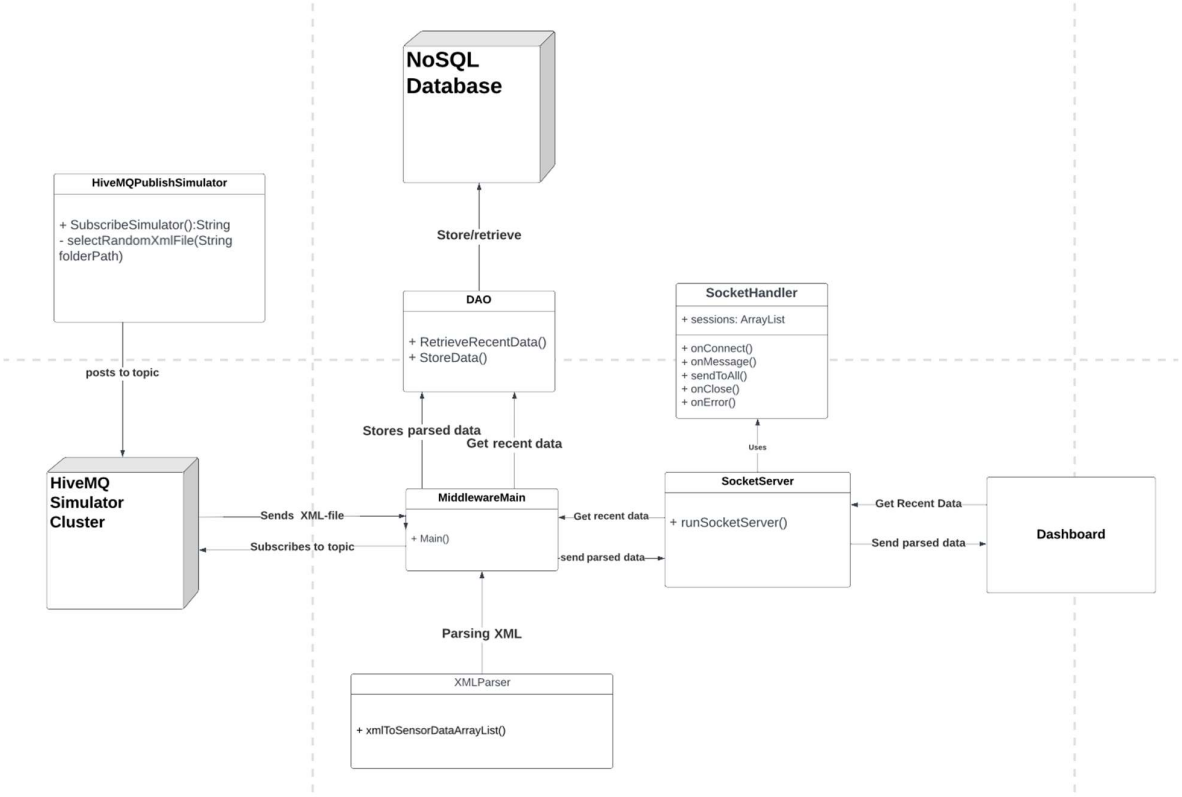
“AustevollNordData” og “AustevollSorData” er mapper som inneholder ca. 50 xml filer. Disse er utelatt fra denne prosjektstrukturen.

# 4 KLASSEDIAGRAM

Figuren under viser klassediagrammet for klassene som inneholder sensordataen etter at de er blitt filtrert ut XML-filene.

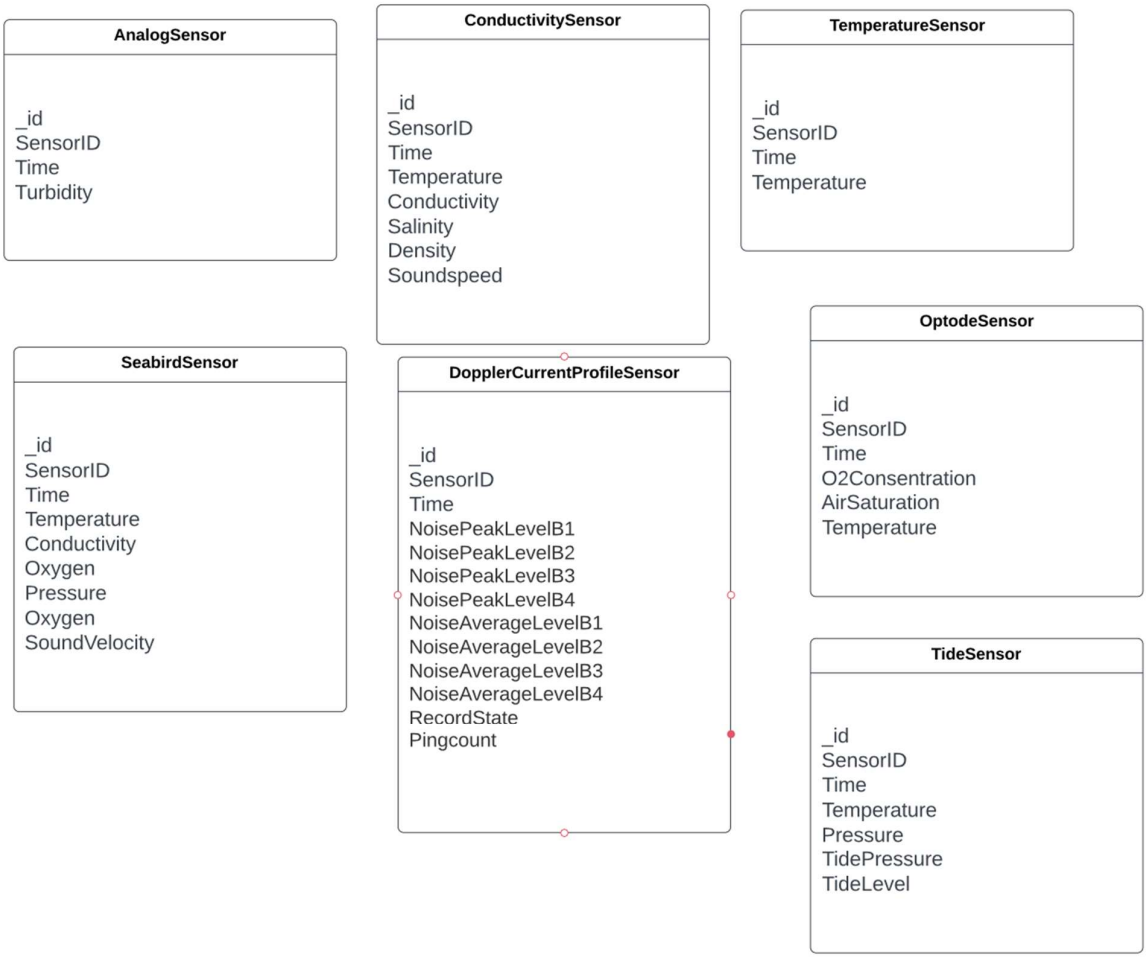


Figuren under viser et klassediagram for de fleste klassene i applikasjonen med unntak av klassediagrammet som er vist over.



# 5 DATABASEMODELL

Applikasjon bruker MongoDB sin NoSQL database. Denne databasemodellen viser dokumentene som blir opprettet. Her vises syv forskjellige sensordokumenter med nøkler som vil bindes til verdier. Videre blir de lagret i en samling for den riggen de hører til.



# 6 SERVER-TJENESTER

Applikasjonen benytter Jetty WebSockets. WebSockets er implementer med en Jetty WebSocket server og en SocketHandler.  Koden under viser SocketHandler klassen med de ulike metoder for håndtering av meldinger samt åpning og lukking av koblinger. Det er også en “sessions“ liste med alle klient som er tilkoblet.

```
@WebSocket
public class SocketHandler {
    public static final List<Session> sessions = new ArrayList<>();

    @OnWebSocketConnect
    public void onConnect(Session session) {
        sessions.add(session);
        System.out.println("WebSocket connected: " +
session.getRemoteAddress().getHostName()
                + "Number of sessions: " + sessions.size());
    }
    @OnWebSocketMessage
    public void onMessage(Session session, String message) {
        System.out.println("Received message from JS: " + message );
        // Echo the message back to the client
        try {
            ArrayList<AbstractSensorData> sensorDataList =
DAO.RetrieveRecentSensorData(message);
            Gson gson = new Gson();
            String data = gson.toJson(sensorDataList);
            session.getRemote().sendString(data);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static void sendToAll(String message) {
        for (Session session : sessions) {
            try {
                System.out.println("Sending message to client: " + message);
                session.getRemote().sendString(message);
            } catch (Exception e) {
                // handle exception here
                System.err.println("Error sending message to client: " +
e.getMessage());
                e.printStackTrace();
            }
        }
    }
    @OnWebSocketClose
```

```
    public void onClose(Session session, int statusCode, String reason) {
        sessions.remove(session);
        System.out.println("WebSocket closed: " + statusCode + " - " +
reason);
    }

    @OnWebSocketError
    public void onError(Session session, Throwable throwable) {
        System.out.println("WebSocket error: " + throwable.getMessage());
    }
}
```

Her er WebSocket serveren:

```
public class SocketServer {

    public static void runSocketServer() throws Exception {
        try {
            Server server = new Server(8081);
            WebSocketHandler wsHandler = new WebSocketHandler() {
                @Override
                public void configure(WebSocketServletFactory factory) {
                    // Register the WebSocket handler class
                    factory.register(SocketHandler.class);
                }
            };
            server.setHandler(wsHandler);
            server.start();
            System.out.println("Socket server started");

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Serveren oppretter et WebSocketHandler instans som er registrert til SocketHandler klassen gitt over.

# 7 SIKKERHET

Det ble ikke tid til å impementere grensesnittet for admin til å legge til nye rigger. Derfor er det ikke enda utviklet metoder for bla. hashing av brukernavn/passord. Tilgang til databasen gjøres uten brukerinput, så det var ikke nødvendig med beskyttelse mot SQL- injeksjon.

# 8 INSTALLASJON OG KJØRING

Applikasjonene har flere avhengigheter/programvarebiblioteker.
Avhengighetene/programvarebiblioteker ligger i “build.gradle” filen. De er gjengitt her:

```
dependencies {
 implementation 'top.jfunc.json:Json-Gson:1.0'
 implementation 'com.google.code.gson:gson:2.10.1'
 implementation 'junit:junit:4.13.1'
 compileOnly('jakarta.platform:jakarta.jakartaee-web-api:9.1.0')

 testImplementation("org.junit.jupiter:junit-jupiter-api:${junitVersion}")
 testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:${junitVersion}")
 implementation("com.hivemq:hivemq-mqtt-client:1.3.0")
 implementation(platform("com.hivemq:hivemq-mqtt-client-websocket:1.3.0"))
 implementation(platform("com.hivemq:hivemq-mqtt-client-proxy:1.3.0"))
 implementation(platform("com.hivemq:hivemq-mqtt-client-epoll:1.3.0"))
 implementation("com.hivemq:hivemq-mqtt-client-reactor:1.3.0")
 implementation 'com.jayway.jsonpath:json-path:2.6.0'
 implementation 'org.mongodb:mongodb-driver-sync:4.3.4'
 implementation 'javax.json:javax.json-api'
 implementation 'org.glassfish:javax.json'
 implementation 'org.eclipse.jetty.websocket:websocket-server'
 implementation 'com.fasterxml.jackson.core:jackson-databind'
 implementation 'javax.websocket:javax.websocket-api:1.1'
 implementation 'org.eclipse.jetty.websocket:websocket-
server:9.4.44.v20210927'
 implementation 'org.eclipse.jetty.websocket:websocket-
servlet:9.4.44.v20210927'
 implementation 'javax.json:javax.json-api:1.1.4'
 implementation 'org.glassfish:javax.json:1.1.4'
 implementation 'com.fasterxml.jackson.core:jackson-databind:2.12.4'
 implementation 'com.google.code.gson:gson:2.8.9'
 implementation 'javax.ws.rs:javax.ws.rs-api:2.1'
 testImplementation 'junit:junit:4.13.2'
}
```

De viktigste avhengighetene her er MongoDB API-et,Jetty Websockets og HiveMQ API-et i tillegg til bla. JSON til formatering.

For å kjøre løsningen må webapplikaksjonen kjøres på en webserver, som f.eks. Tomcat. I tillegg kjøres “MiddlewareMain” klassen og “HiveMQPublishSimulatoren” klassen for å sende data til HiveMQ clusteret. Det vil også være nødvendig å legge til sin IP-adresse på MongoDB sin nettside, dette er et sikkerhetstiltak hos MongoDB.

# 9 DOKUMENTASJON AV KILDEKODE

Det er lagt til kommentarer i kildekoden for å gi nødvendig beskrivelse.

# 10 KONTINUERLIG INTEGRASJON OG TESTING

Github (Github, 2023) ble brukt til kontinuerlig integrasjon. Det ble satt opp en mastergren som gruppen forket fra og videre pushet til.

Det er laget unit tester for XML-filtreringsmodulen. Det ble laget XML-filer som inneholdt feil og unit testene passerte disse. Videre er det testet at det filtreres ut alle de ønskede sensordataene fra en mengde XML-filer. For å kjøre unit testene, trenger man bare å kjøre unit test klassene.

# 11 REFERANSER

Github, 2023. *Github.* [Online]
Available at: https://github.com/
[Accessed 10 01 2023].
Lucid, 2023. *Lucidcharts.* [Online]
Available at: https://www.lucidchart.com/pages/
[Accessed 15 02 2023].