Isto turunen
Joona Peltola
Mikko Lager

Suunnitteludokumentti ohjelmistojen suunnittelu - ryhmä 37

Tässä dokumentissa käydään läpi toteutettu ohjelma ja sen toimintojen käyttö. Lisäksi pohditaan ohjelman sisäistä rakennetta ja sen muutoksia suhteessa aiempaan dokumenttiin, sekä mitä muutoksia ohjelmaan voisi mahdollisesti vielä tehdä.

Ohjelman toiminnot

Ohjelman tarkoituksena on voida seurata sekä analysoida Kampusareenalla sijaitsevien mittareiden mittaamaa dataa. Vaihtoehtoina on katsoa viimeisin data tai historiadataa viimeiseltä kahdelta viikolta. Analyysipuolella voidaan asettaa kahden suureen ristiinnäyttö, laskea korrelaation arvo sekä laskea valituiden suureiden keskiarvot.

Aloitussivulla käyttäjä pystyy valitsemaan haluamansa mittarit, joiden mittaamat (viimeisimmät) arvot päivitetään ruudulle. Käyttäjä voi myös valita mitkä mittarit näytetään tai piilotetaan klikkaamalla ruudun vasemmassa laidassa sijaitsevia ruutuja, sekä valitsemaan haluamansa kuvan valmiiden kuvien tilalle klikkaamalla kuvasta. Käyttäjä voi myös valita digitaalisen ja analogisen mittarin välillä, sekä valita haluamansa mittaustaajuuden. Näytölle päivittyy automaattisesti uudet arvot taajuuden näyttämän ajan välein. Näytölle voi päivittää uusimmat arvot myös klikkaamalla “Päivitä”-painiketta.

Kuten aloitussivulla, historiasivulla käyttäjä voi näyttää tai piilottaa haluamansa kuvaajat. Jotta historiasivulle päivittyy käyttäjän haluamat arvot hänen valitsemistaan mittareista, on käyttäjän valittava sopiva aikaväli klikkaamalla “Valitse aikaväli”-nappia. Tällöin aukeaa popup-ikkuna, jossa on kalenteri sekä kellonaikoja. Kalenterista voi valita aloitus- ja lopetuspäivän, mutta vain viimeisen kahden viikon ajalta, sekä aloitus- ja lopetus kellonajan. Mikäli aloitus- ja lopetuspäivä ovat samat ja käyttäjä valitsee aiemman lopetusajan, kuin mitä aloitusaika on, ohjelma vaihtaa automaattisesti lopetusajaksi aloitusajan. Historiadataa haettaessa resoluutio lasketaan valitun aikavälin mukaan.

Analyysisivulla on samanlainen ajanvalitsin kuin historiasivulla, mutta suureet on valittava erikseen. Kun aika sekä x- ja y-akselin suureet on valittu, päivittyy korrelaatio, keskiarvo sekä pisteet kuvaajalle. Tämän jälkeen data päivittyy aina, kun käyttäjä valitsee vain joko aikavälin tai toisen akselin muuttujan.

Ohjelma myös tallentaa edellisen session tietoja, jotka säilyvät seuraavalle kerralle. MainPagen, AnalysisPagen ja HistoryPagen istunnossa määritetyt asetukset tallennetaan settings.json tiedostoon, joka sisältää muunmuassa tiedon siitä, mitkä mittarit on valittu näytettäväksi. Data ei kuitenkaan MainPagella ja HistoryPagella tallennu, vaan se haetaan uudestaan ohjelman käynnistyessä seuraavan kerran. AnalysisPagella tallentuu myös senhetkinen haettu data, joka tallennetaan settings.txt tiedostoon ja palautetaan automaattisesti ohjelman käynnistyessä.

Luokkien vastuualueet

Seuraavaksi kuvataan luokkien vastuualueet ja niiden tärkeimpien metodien vastuualueet. Esitellään luokat yksi kerrallaan:

Isto turunen
Joona Peltola
Mikko Lager

DataHandler

DataHandler ottaa käyttäjän syötteet UI-puolelta ja välittää syötteiden tiedot luokille, jotka käsittelevät ja antavat tarvittava datan. Kun tarvittava käsitelty data on saatu, Datahandler välittää sen UI:lle. Esitellään seuraavaksi luokan metodit:

- *updateLatest:* palauttaa viimeisimmät datan arvot valituille mittareille käyttöliittymään.
- *updateHistory:* palauttaa datan arvot ja ajat valituille mittareille käyttäjän haluamalta aikaväliltä.
- *setTimer:* päivittää ajastimen arvon halutulla aikavälillä.
- *updateAnalysis:* palauttaa käyttäjän pyytämän analyysidatan halutulta aikaväliltä sekä välittää datan CalculateCorrelation luokalle. Lisäksi se tallentaa datan ja laskettujen suureiden keskiarvot ja korrelaation luokaan.
- *extrctDataList:* purkaa dataListisä pisteiden arvot korrelaation laskemista varten.
- *dataToAnalysisUI*: yhdistää haetuista x ja y datapisteistä QPointF arvoja datan päivitystä varten.
- *setResolution:* asettaa resoluution datahakua varten. Resoluutio valikoituu minuuteiksi, tunneiksi, päiviksi tai kuukausiksi riippuen valitusta aikavälistä.
- *convertQmlDatetimes:* muokkaa qml:n aikasyötteistä json-datan parsintaan sopivan muodon ajalle.

CalculateCorrelation

Laskee annetusta datasta korrelaation ja hakee CalculateDeviation luokalta keskiarvot. Seuraavaksi luokan metodit:

- *saveData:* tallentaa datalistat luokkaan ja määrittää datalistalle arvopisteen NaN, jos datalista on tyhjä.
- *calculateCorrelation:* laskee hajonnan calculateDeviation luokan olioiden avulla ja laskee näiden perusteella korrelaation.

CalculateDeviation

Laskee hajonnan ja keskiarvon CalculateCorrelationin asettamalle datalle ja palauttaa arvot CalculateCorrelationille.

DataPoint

Tämän tyyppisiin muuttujiin tallennetaan yksittäisten mittauspisteiden tiedot, eli niiden arvot, suureet sekä mittauksen ajankohdan. Tämän luokan metodit ovat:

- *value ja setValue:* palauttaa arvon, asettaa arvon.
- *unit ja setUnit:* palauttaa suureen, asettaa suureen.
- *date ja setDate:* palauttaa mittausajan, asettaa mittausajan.

DataList

DataList on lista, joka sisältää DataPoint-osoittimia. Seuraavaksi luokan metodit.

- *append:* lisää mittauspisteen listan loppuun.
- *empty:* poistaa listan alkiot.
- *data:* muuttaa QList-muotoisen muuttujan QVariantList muotoon, ja palauttaa tämän käyttöliittymälle.

- *min/max:* palauttaa DataListin pienimmän/suurimman arvon (value)

DataRequester

DataRequester-luokka kommunikoi datalähteen DataHandler-luokan välillä. DataRequester lähettää pyynnön datalähteelle ja palauttaa joko yksittäisiä DataPoint-osoittimia, tai listan näistä riippuen pyynnön laadusta. Tämän luokan metodeja ovat:

- *makeLatestRequest:* palauttaa viimeisimmät mitatut arvot osoittimina DataPoint-muuttujiin.
- *makeHistoryRequest:* palauttaa listan DataPoint-osoittimia UI:ssa valitulta aikaväliltä.

JsonDataReader

Parsii tietokannasta jsondatan käyttäjän pyytämillä ominaisuuksilla. Luokan metodeja ovat:

- *parseEntities:* palauttaa viimeisimmät mitatut arvot osoittimina DataPoint-muuttujiin parsimalla entityitä parseEntity metodin avulla.
- *parseEntity: parsii entitystä halutun kyselyn mukaisen datan.*
- *parseContextResponse: Ottaa halutulle aikavälille datajoukon ja käsittelee niitä elementti kerralaan parseContexElements metodin avulla.*
- *parseContextElement:* Ottaa syötteenä parseContextResponselta elementin ja parsii siitä datan pyydetyn operaation mukaisesti.

SettingsReader

SettingsReader-luokka lukee asetuksia settings.json-tiedostosta, sekä tallentaa asetuksia kyseiseen tiedostoon reaaliajassa. Tärkeimmät metodit:

- *readSettingsFile:* lukee settings.json-tiedoston, mikäli tiedostoa ei löydy, lukee default.json-tiedoston. Tämä metodi kutsuu *setCurrentSettings*-metodia.
- *setCurrentSettings:* asettaa nykyiset asetukset SettingsReaderin jäsenmuuttujiin ja kutsuu *save*-metodia.
- *save:* tallentaa jäsenmuuttujiin asetetut arvot settings.json-tiedostoon.
- *getCurrentSettings:* palauttaa jäsenmuuttujiin asetetut arvot.

AnalysisUi

Hoitaa Qml UI:n päivitystä saapuvan datan ja annettujen kyselyiden mukaan. Se hyväksyy kyselyn lähettämisen kun tarvittavat suureet ja ajat on valittu, käsittelee ja siirtää DataHandlerilta saapuvan datan serriesiin sekä laskee ja asettaa ison arvon omaaville datajoukoille kertoimia. Tämän luokan tärkein metodi:

- *UpdateSeries*: generoi datahandlerin datan XYseriesissä esitettävään muotoon ja päivittää seriesin UI:n.

Muutoksia suunnitteluvaiheen dokumenttiin

Aiemmassa suunnitelmassa pohdittiin sitä, näytetäänkö käyttäjälle mittareiden mittaamia kumulatiivisia arvoja, vai näytetäänkö kahden peräkkäisen arvon välinen erotus. Tämä ratkaistiin siten, että aloitussivulla näkyy kumulatiiviset arvot (ei lämpötila), mutta historia- ja analyysisivuilla näytetään ei-kumulatiivisia arvoja.

Isto turunen
Joona Peltola
Mikko Lager

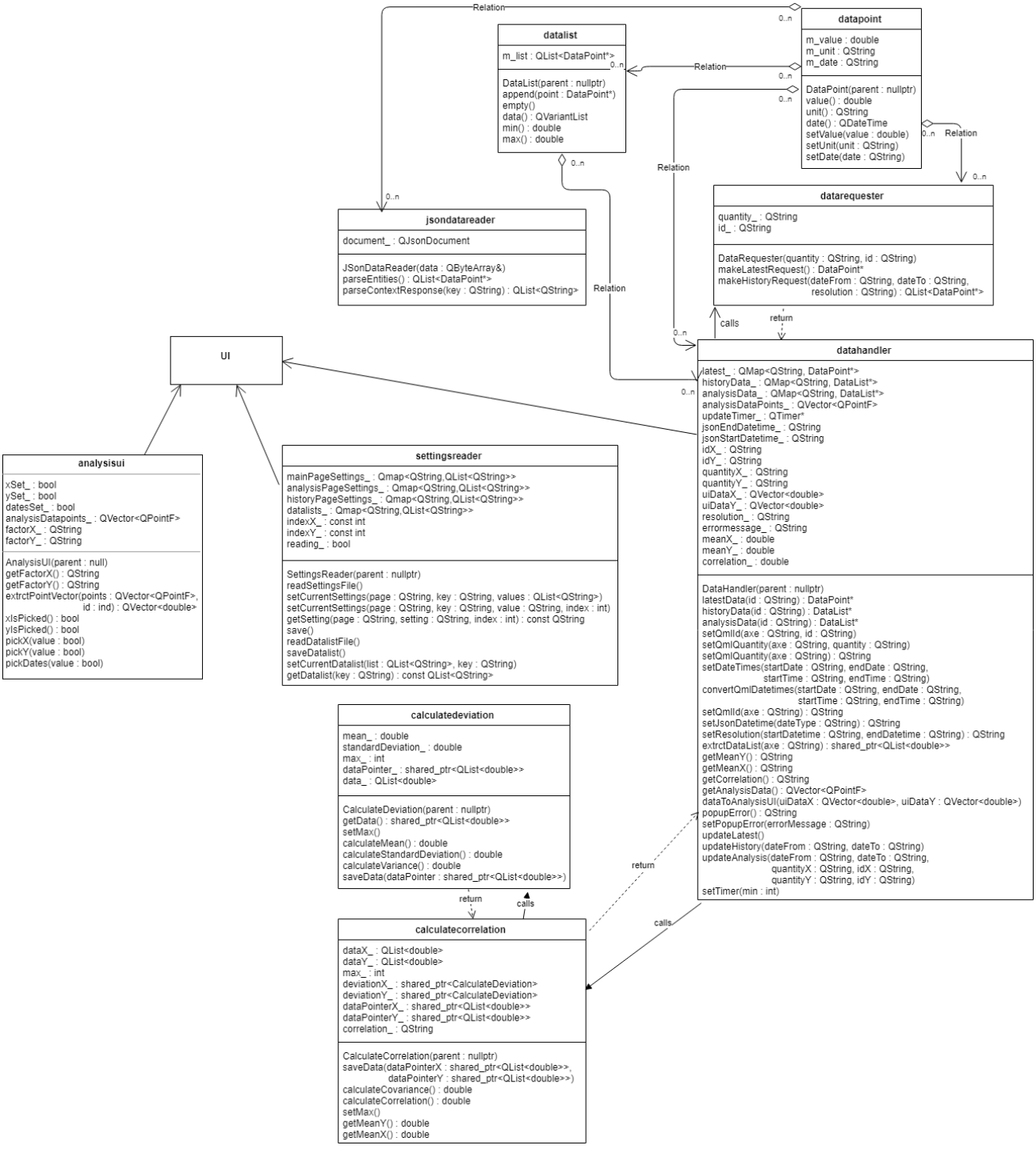
Alkuperäisessä suunnitelmassa oli kaksi erillistä luokkaa historiadatan ja viimeisimmän datan hakemiselle (historyrequester ja latestrequester). Viimeisessä versiossa näistä on tehty yksi luokka(datarequester), sillä niissä oli niin paljon yhtäläisyyksiä.

Kuten aiemmin oli päätetty, ohjelma pystyy hakemaan maksimissaan kaksi viikkoa vanhaa dataa. Tämän vuoksi minimi ja maksimi päivät määritellään qml:n DatePickerillä aina reaaliaikaisen päivän mukaisesti. Näin ollen, vaikka settingsreader tallentaa vanhan istunnon datan haun päiväykset, ne eivät ole enää käytettävissä seuraavalla istunnolla, jos päiväyksistä on kauemmin kuin kaksi viikkoa verraten uuden istunnon aikaiseen päivään. AnalysisPagen datapisteet tallentuvat tiedostoon, joten ne palautuvat kauemmaltakin ajalta.

Ohjelman rakenne

Alla olevassa UML-kaaviossa näkyy ohjelman sisäinen rakenne.

Isto turunen
Joona Peltola
Mikko Lager



Ohjelman tulevaisuus

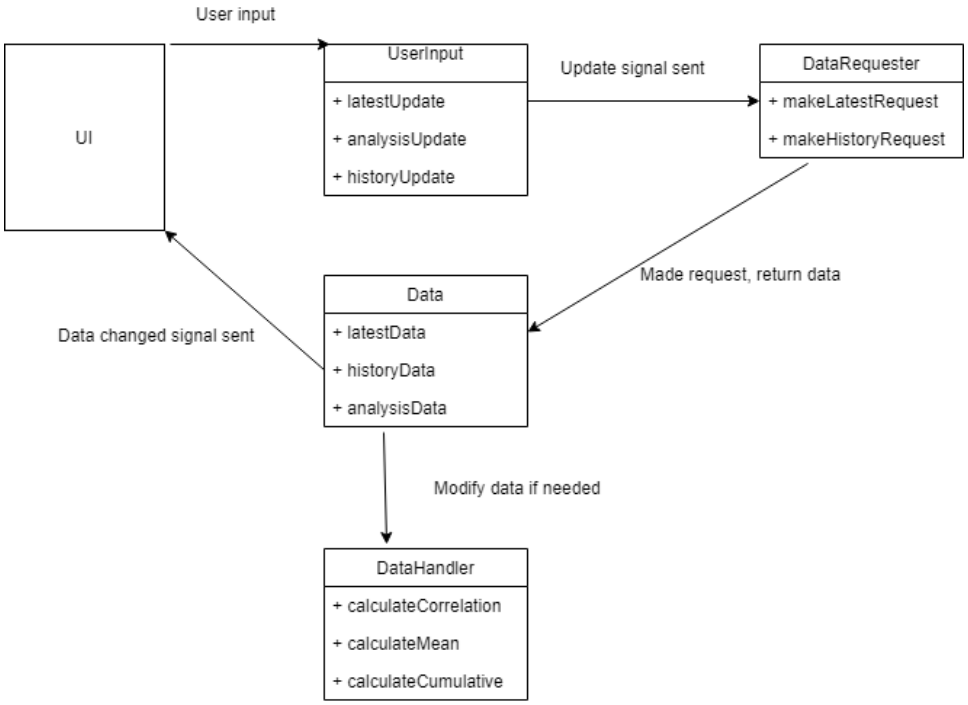
Mikäli meille olisi jäänyt enemmän aikaa ohjelman toteutukseen, ehdottomasti ensimmäisenä mieleen tuleva muutos olisi DataHandlerin vastuiden jakamista useampaan luokkaan. Meillä voisi olla oma luokka, UserInput, joka pitää kirjaa käyttäjän tekemistä syötteistä ja lähettää signaalin, jos jokin muuttuu.

Isto turunen
Joona Peltola
Mikko Lager

Jos esimerkiksi ohjelman pääsivulla käyttäjä painaa ”päivitä” nappia, UserInput ilmoittaa DataRequester luokalle, että viimeisin data pitää päivittää. DataRequester tällöin hakee tietokannasta viimeisimmän datan ja palauttaa sen Data luokalle, joka pitää kirjaa UI:lla näytettävästä datasta. Dataluokka tällöin ilmoittaa takaisin UI:lle, että data on muuttunut.

Itse DataHandlerin vastuulle jäisi sen nimen mukaisesti datan käsittely, eli korrelaation ja keskiarvon laskeminen sekä kumulatiivisen datan muuttaminen hetkelliseen arvoon, jos tälle on tarve.

Esimerkkikaavio:



Tässä mallissa on hyödynnetty MVC- arkkitehtuuria. Modelina, eli tiedon tallentajana ja ylläpitäjänä toimii Dataluokka, viewinä toimii itse UI, kontrollerina toimii UserInput. Kutakin näistä on helppo muuttaa ilman, että muita osia täytyisi muuttaa.

Tämä malli tukee paremmin SOLID- periaatetta. Tässä mallissa kullakin luokalla on paremmin määritellyt vastuualueet, eikä niillä ole liikaa vastuita: UserInput pitää kirjaa käyttäjän syötteistä, DataRequester tekee pyyntöjä tietokantaan, Data pitää kirjaa haetusta datasta, DataHandler muokkaa datan haluttuun muotoon ja laskee datasta tarvittavat tiedot. Tässä mallissa on myös helppo laajentaa toiminnallisuuksia periyttämällä kyseisistä luokista. Esimerkiksi DataRequester voisi olla rajapintaluokka, josta voidaan tehdä erilaisia toteutuksia esim. eri tietokantoja varten.