



Samuel Laisaar, Leevi Laaksonen, Aleksandr Liski, Leo Lehtiö

Juna-asemien informaatio taulujärjestelmä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintäteknologian tutkinto-ohjelma

Innovaatioprojekti

2.5.2023

Tiivistelmä

Tekijät:	Samuel Laisaar, Leevi Laaksonen, Aleksandr Liski, Leo Lehtiö
Otsikko:	Juna-asemien informaatiotaulujärjestelmä
Sivumäärä:	34 sivua + 1 liitettä
Aika:	7.5.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintäteknologian tutkinto-ohjelma
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Amir Dirin

Projektissa on toteutettu IoT-ohjelmisto, joka mahdollistaa juna-asemien erilaisten tietojen esittämisen ja erilaisten näyttyyppien nopean rakentamisen asentajan toimesta. Ohjelmiston rinnalle toteutettiin toimiva käyttöliittymä, jonka avulla käyttäjä voi hallita näyttöjen ilmoituksia ja erikoisviestejä. Projekti käyttää hyväkseen avointa rajapintaa, josta ohjelmisto kokoaa välitettäviä junatietoja. Tietojen aitous todennetaan digitaalisella allekirjoituksella. Todennusta varten toteutimme SSL-protokollalla toimivan ratkaisun. Ohjelmiston toimintaa testattiin käyttötapauksella, jossa rakensimme Espoon rautatieaseman informaatiotaulujärjestelmän.

Avainsanat: IoT, Tietoturva, TPM, MQTT, Python, PIS, Digitraffic, API

Abstract

Author:	Samuel Laisaar, Leevi Laaksonen, Aleksandr Liski, Leo Lehtiö
Title:	Platform Information System
Number of Pages:	34 pages + 1 appendices
Date:	7 May 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Amir Dirin, Senior Lecturer

In this project, we developed an IoT software that allows the passengers to read railway information and the maintainers to rapidly build various display types for different data cases, using the IoT Framework template. Displays are managed using the management user interface which can be used to send unique messages and announcements. The software aggregates data from an open-source API. Before transit, the messages are signed and later verified for authenticity. We implemented a solution for data validation using SSL Digital Signature. To test the functioning of the system, we implemented Espoo's railway station as our use case.

Keywords: IoT, TPM, MQTT, Python, Security, PIS, Digitraffic, API

Sisällys

Lyhenteet

1	Johdanto	1
2	Asiakkaan vaatimukset	1
3	Järjestelmän suunnittelu ja arkkitehtuuri	3
3.1	Kokonaiskuva	3
3.2	MQTT-kommunikointi	5
3.3	Näyttöjen kokoonpanot	9
3.4	Management kokoonpano	11
3.5	TPM ja viestin todentaminen	13
3.6	Junatietojen aggregointi ja käsittely	16
3.7	Arkkitehtuuri	18
4	Järjestelmän käyttöönotto	21
4.1	Asennus	21
4.1.1	PyPI asennus	22
4.1.2	Git asennus	22
4.2	Ohjatun toiminnan käynnistäminen (Wizard)	23
4.3	Aggregaattorin käynnistäminen	24
4.4	Näytön käynnistäminen	26
4.5	Ilmoitusten käyttöliittymän käynnistäminen (Manager)	27
4.6	Asetusten hallinta ja todennusavaimen generointi	29
5	Käyttötapaus: Espoon rautatieasema	31
6	Yhteenveto	32
	Lähteet	34

Liitteet

Liite 1: Nimikonventio

Lyhenteet

PIS:	<i>Platform Information System</i> . Juna-asemien informaatiotaulujärjestelmä, joka mahdollistaa junatietojen esittämisen ja hallitsemisen näytöillä.
TPM:	<i>Trusted Platform Module</i> . Fyysinen turvapiiri, jota käytetään laitteiden turvallisuuden parantamiseen.
IoT:	<i>Internet of Things</i> . Fyysiseen verkkoon yhdistettyjen älylaitteiden luoma verkosto.
MQTT:	<i>Message Queue Telemetry Transport</i> . Verkkoprotokolla laitteesta laitteeseen tapahtuvan julkaisun- ja tilausviestinnän jonotuspalvelu.
Rata:	Avoin rajapinta Suomen rataverkolla kulkevien junien aikatauluista, sijainneista ja kokoonpanoista. Viittaa <i>rata.digitraffic.fi</i> lähteeseen.
SSL:	<i>Secure Sockets Layer</i> . Tietoverkkosalausprotokolla. Käytetään viestien allekirjoitukseen ja todentamiseen.
UTC:	<i>Coordinated Universal Time</i> . ISO 8601 aikastandardi, joka mahdollistaa ajan yhtenäisen käytön kaikkialla maailmalla.
JSON:	<i>Javascript Object Notation</i> . RFC 7159 tiedonvaihtoformaatti, jonka tietue koostuu avainarvo pareista.
PyPI	<i>Python Package Index</i> . Python-pakettien varasto tai asennusmenetelmä, joka käyttää pip-ohjelman.
PKS	<i>Private Key Store</i> . PKCS15 standardin salattu avainsäiliöformaatti, jossa todennusavaimet pidetään.

1 Johdanto

Tämän projektin tarkoituksena oli toteuttaa järjestelmä, joka mahdollistaa juna-tietojen esittämisen ja hallinnan erilaisilla näytön tyypeillä sekä erilaisten näyttö-tyyppien nopean rakentamisen. Toteutimme lisäksi käyttötapausesimerkin oike-asta rautatieasemasta.

Projekti toteutettiin yhteistyössä Nokia Bell Labsin kanssa. Työskentely tapahtui osittain etätyöskentelynä ja osittain Nokian *garage*-työskentelytilassa, jonne ke-räännyimme kerran viikossa. Viikkotapaamisissa kävimme asiakkaan kanssa toteutukset läpi ja saimme lisäohjeita seuraaville projektin vaiheille.

Alkuperäinen projektin suunnitelma sisälsi TPM-turvapiirin käytön, jota emme osanneet yhdistää kehitysympäristöömme. TPM:n python-kirjastoissa ilmensi asennusongelmia. Toteutimme oman ratkaisun, joka jäljittelee TPM:n toimintaa mahdollisimman paljon ja projektin loppupäässä saimme turvapiirin toimintakun-toon. Projekti käyttää nyt molempia ratkaisuja viestien todentamiseen.

Projektin arkkitehtuurin toteutuksessa on käytetty Raspberry Pi- pientietokoneita ja aikaisemmin suunniteltua IoT-ohjelmistokehystä, joka pohjautuu MQTT-vies-tintävälitys teknologiaan. Ohjelmisto on toteutettu täysiin Python-ohjelmointikie-lellä ja näyttöjen käyttöliittymä käyttää pythonin tkinter-kirjastoa. Rautatiease-mien aikataulutiedot ja junien kokoonpanot haetaan digitrafficin avoimesta raja-pinnasta.

2 Asiakkaan vaatimukset

Platform Information System on informaatiotaulujärjestelmä, joka mahdollistaa pienrautatieasemien matkustajatietoverkoston rakentamisen. Sillä asiakas voi asentaa näyttöjä erilaisiin kohteisiin ja esittää matkustusasiatietoja. Ohjelmisto voi myös toimia virtuaalisena rautatieasemana, jolloin sitä voi käyttää erilaisten

verkkoteknologioiden testaamiseen ennen kuin teknologia otetaan käyttöön todellisessa järjestelmässä.

Ryhmä oli sopinut asiakkaan kanssa, että ryhmä itse määrittelee ohjelmiston käyttötapauksen ja kriteerit. Asiakas määritteli ohjelmiston teeman, sekä toteutuksen tavoitteet. Alussa ryhmä suunnitteli alkukantaisen konseptin ja arkkitehtuurin, jota sitten lähdettiin toteuttamaan.

Ohjelmistolle on asetettu seuraavat vaatimukset:

- Projekti rakennetaan aikaisemmassa projektissa luodun ohjelmistokehyksen inspiroimana.
- Näyttöjä voi olla useampia ja jotkut näytöt voivat esittää saman sisällön.
- Ohjelmiston pitää toimia yhden tai useamman juna-aseman kanssa.
- Ohjelmiston välisen viestinnän eheys on testattavissa ja tietue validoitavissa, mieluiten TPM 2.0 avulla.
- Käyttää avoimen rajapinnan junatiedot ja kokoonpanot.
- Ohjelmiston on toimittava Nokian Bell Labsin Raspberry Pi:n pientietokoneissa.
- Asiakas pystyy:
 - Asentamaan ohjelmiston helposti näyttöihin.
 - Rakentaa erilaisia näkymiä, jotka voivat esittää monipuolisia tietueita.
- Matkustaja pystyy:
 - Tehdä matkaselvityksen näytöistä.
 - Saada päivitettyä tietoa juna-aseman tapahtumista ja aikataulun muutoksista.
 - Nähdä junien aikatauluihin liittyviä tietoja reaaliajassa.

3 Järjestelmän suunnittelu ja arkkitehtuuri

Asiakkaamme toimesta saimme itse suunnitella järjestelmän, kunhan hyödynnämme jo valmista ohjelmistokehystä IoT-laitteiden kommunikaatioon. Tähän kuuluivat MQTT-protokollan käyttö ja TPM turvapiiri viestien todentamiseen.

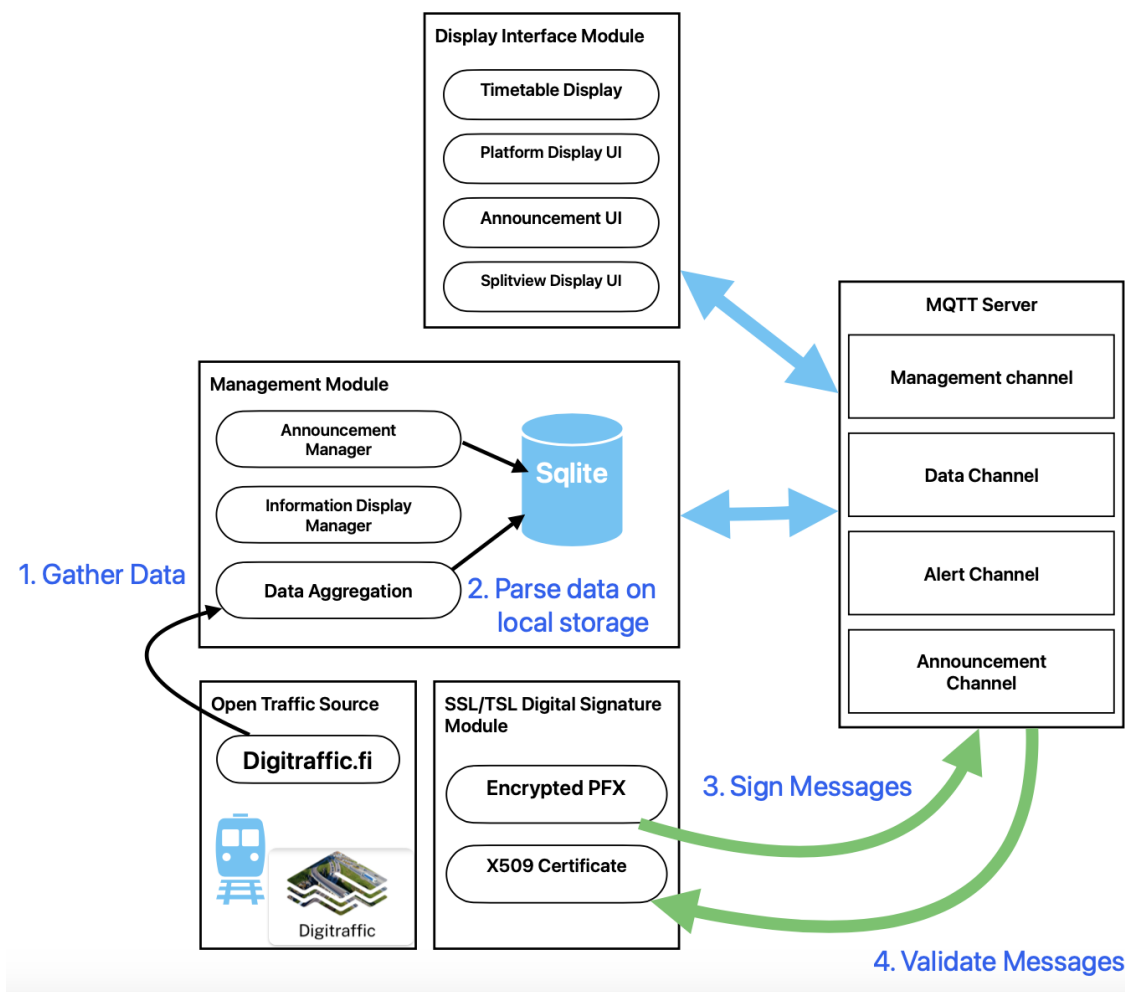
3.1 Kokonaiskuva

Tämän projektin tavoitteena oli toteuttaa järjestelmä, joka mahdollistaa junatietojen esittämisen ja hallinnan erilaisilla näytön tyypeillä ja erilaisten näyttötyyppien nopean rakentamisen abstrakteilla komponenteilla niin, että asentaja voisi luoda oman rautatieinformaatiojärjestelmän ilman koodaustaitoja. Viestiväylien kehys on toteutettu niin, että kehittäjän olisi yksinkertaista rakentaa omia näyttötyyppisiä helposti.

Järjestelmä koostuu eri laitteilla olevista ohjelmista, jotka viestittelevät keskenään MQTT-viestintäprotokollaa käyttäen. Järjestelmä koostuu neljästä päämoduulista: 1) Managerimoduulista ja tiedon aggregoinnista, 2) käyttöliittymästä ja näyttökomponenteista, 3) todennusrajapinnasta, sekä 4) avoimesta rajapintakehyksestä ja aiheväylistä.

Järjestelmä on suunniteltu pyörimään parhaiten, kun managerimoduuli ja käyttöliittymämoduuli asennetaan eri laitteille. On suositeltavaa, että käyttöliittymämoduulista yksi näyttökomponentti toimii yhdellä laitteella kerrallaan, vaikka mikään ei estä järjestelmää toimimasta yhdellä laitteella useammalla tai yhdellä näytöllä. Sama näyttökomponentti voi elää useammalla laitteella. Avoin rajapintakehys käyttää hyväkseen junaliikennetietoja tarjoavan Digitraffic.fi:n [1] lähettä.

Kuvassa 1 on esitetty järjestelmän toiminta ylemmällä tasolla. Junatietojen näyttäminen rautatieaseman näytöillä tapahtuu käyttöliittymämoduulissa (*Display Interface*). Näyttökomponenteista voi rakentaa erilaisia tyyppisiä eri tarkoituksiin, kuten esimerkiksi laituriin saapuvan junan tiedot esittävän näytön.



Kuva 1. Järjestelmän arkkitehtuurin yleiskuva.

Näytöillä esitettävä tieto saadaan Managerimoduulin (*Management Module*) aggregator – nimisestä ohjelmasta. Aggregaattorin (*Data aggregation*) kerää ja käsittelee rata.digitraffic.fi:stä saapuvan junaliikennedatan, sekä jakaa tiedot näytöihin MQTT-protokollan aiheviestiväyliä pitkin. Managerimoduuli sisältää aggregaattorin lisäksi myös Manager-nimisen ohjelman, joka on tarkoitettu näyttöjen hallinnointiin. Manager on erillinen ohjelma, jonka käyttöliittymä sallii käyttäjänsä hallita ja tarkkailla näyttöjen tiloja ja lähettää näytöillä ilmestyviä ilmoitus- ja varoitusviestejä.

Kuvan 1 MQTT Server on palvelinohjelma, joka vastaa MQTT-protokollan viestien välittämisestä Management Moduulin ja Display Interface Moduulien välillä. Projektissa käytettävä ohjelma on Apache Mosquitto.

Todennusrajapinta (*SSL/TSL Digital Signature Module*) vastaa saapuvien viestien validoinnista. Se korvaa tässä projektissa Trusted Platform Module turvapiirimoduulin.

3.2 MQTT-kommunikointi

MQTT-viestintä perustuu julkaisija/tilaaja malliin, jossa yksi laite voi toimia sekä viestien julkaisijana että tilaajana. MQTT on kevyt protokolla, joka soveltuu hyvin laitteille, joiden verkkoyhteys tai prosessointikyky on rajoitettu. [2.]

Järjestelmän eri laitteet käyttävät keskenään MQTT-viestiväyliä. Kaikki laitteet yhdistyvät samaan verkossa olevaan MQTT-palvelinohjelmaan ja lähettävät viestejä konventioiden mukaan nimettyihin kanaviin. Projektin kanavien nimikonventio rakentuu junan ja rautatieaseman kokoonpanoista. Eri näyttötyypit voivat tilata tietoja yhdestä tai useammasta aihekanavasta.

Kanavien nimissä on hyväksikäytetty MQTT-protokollassa sallimia jokerimerkejä. Jokerimerkeillä pystytään suodattamaan tiettyihin kanaviin, jotka tarjoavat tiettyjä junatietoja. Esimerkiksi kuuntelemalla kanavaa *PSL/+/Departure/Commuter*, näyttöön saapuu tietoja lähtevistä lähijunista kaikista Pasilan laitureista. Jokerimerkki "+" viittaa siihen, että kyseisessä nimikonvention tasossa kuunnellaan kaikkia muutoksia, siinä missä *PSL/4/Departure/Commuter* vastaa vain laiturille 4 saapuvien lähijunien tietoja Pasilan rautatieasemalla. Tällä tavalla voidaan suorittaa dynaamisia hakuja, jotka mahdollistavat suuria määriä variaatioita. Pelkästään jo yhden näyttökomponentin avulla voidaan ilmaista yli 12 variaatiota ja jokainen näyttökomponentti vastaa yhtä tiettyä tarvetta. Esimerkiksi laiturinäkymää voi muokata näyttämään saapuvia lähijunia tai lähteviä kaukojuunia tai jopa kaikkien rautatieasemien lähteviä kaikkia junia, laiturilla 11.

näyttö ja aggregaattori tietävät. Hallintokanavan kautta kulkeutuva viesti sisältää event-parametriosion, joka määrittää viestin koostumuksen toiminnan järjestelmässä, esimerkiksi "rollcall"-tapahtuma käskää näytöt päivittämään pariliitoksen aggregaattorin kanssa. [Liite 1.]

Kun näyttö käynnistyy, sammuu tai kaatuu, se julkaisee itsestään tietoa hallintokanavaan. Tämä tieto sisältää näytön esitiedot, tilatun aseman ja näytön näkymän tyypin. Tätä tietoa voi sitten käyttää näytönhallinnan sovelluksessa.

Ilmoituskanava, eli "Announcement" käytetään ohjelmiston toimintana kuljetta-
maan asiakasilmoituksia ja muita tärkeitä tietoja rautatieaseman tapahtumista. Ilmoitukset voivat olla infonäytöllä tai yhdellä laiturilla, esimerkiksi hallintaohjelma voi lähettää varoituksen ratatien huoltotapahtumasta. Ilmoitukset voivat olla alalaidassa pyöriviä "*announcement/info*" ilmoituksia tai "*announcement/alert*" koko näytön kaappaavia varoituksia. "*announcement/passing*" vastaa läpikulkevista junista. Minuutti ennen kuin aseman läpi kulkeva juna saapuu paikalle, aseman laiturinäkymissä ilmestyy erikoisvaroitusta.

Näyttöjen kokoonpanot keräävät junatietoja kanavista, joiden päänimikonventio on "*station*", eli aikataulukkanava. Tämä on koko ohjelmiston osalta merkityksellisin konventio, sillä se vastaa suurimman osan toiminnasta ja mahdollistaa dynaamisten komponenttien käytön. Kuvassa 2 on esitetty vaaleansinisellä pääkonvention erilaisia tasoja. Aikataulukkanavalla on neljä alikonventiota, jotka toimivat sen parametreina.

- `stationCode` Juna-aseman lyhennekoodi. Esim "HKI" (Helsinki)
- `platformID` Tietyn junaraiteen numero
- `transit` Onko lähtö vai saapuminen (arrival, departure)
- `transport` Junan tyyppi, esim. lähijuna tai kaukojuna.

Viestit kulkevat viestiväylissä JSON-muodossa, joka sisältää tapahtumatyypp-
pejä ja tietoja riippuen päänimikonvention tarkoituksesta. Aikataulukkanavan
viesteissä kulkee paljon junatietoa, aikatauluja ja kokoonpanoja. Sen tietueen
rakenteen voi nähdä kuvasta 3.

- MessageTimestamp : datetime ✓ Format: %Y-%m-%dT%H:%M:%S.%fZ
- stationFullname: string ✓ Full name of the station where scheduled train activities occur
- + schedule: array
 - id: string ✓ Id is unique to timetable entry
 - train_id: number ✓ Train to which this entry belongs to
 - type: string ✓ Type of activity of the train. "Arrival" or "Departure"
 - cancelled: boolean ✓ True, if activity at station has been cancelled
 - scheduledTime: datetime ✓ Time of activity at station based on schedule
 - differenceInMinutes: number ✓ The delay between scheduled time and actual time
 - actualTime: datetime ✓ Actual time of activity occurring
 - commercialTrack: number|string ✓ Platform track on which the train performs activity
 - trainStopping: boolean ✓ False, if the train is only passing through the station
 - cause: string ✓ Describes the reason why the train was late or early. Only generic causes
 - destination: string
 - + stopOnStations: list
 - stationFullname: string ✓ Full name of the next scheduled station stop
 - commuterLineID : string ✓ Z, P, R, IC77, etc...
 - trainCategory: string ✓ Transport type. "Commuter" or "Long-Distance"
 - runningCurrently: boolean ✓ True, if the train is currently active
- Digital Signature: hex string ✓ All payload messages carry a digital signature at the tail during transit

Kuva 3. Aikataulukanavan JSON-viestin rakenne.

Kuvan 3 rakenne vastaa yhtä saapuvaa viestiä aikataulukanavalle. Yhden näkymän session aikana voi saapua useita samaan tyyppisiä tietueita eri kanavista ja *schedule*-osiossa yleensä on yksi tai useampi merkintä. Nämä merkinnät koostavat yhdessä näytettävän tiedon näkymän käyttöliittymässä.

Lisää tietoa nimikonventiosta ja alikonvention parametreista löytyy tämän dokumentin liitteestä 1.

3.3 Näyttöjen kokoonpanot

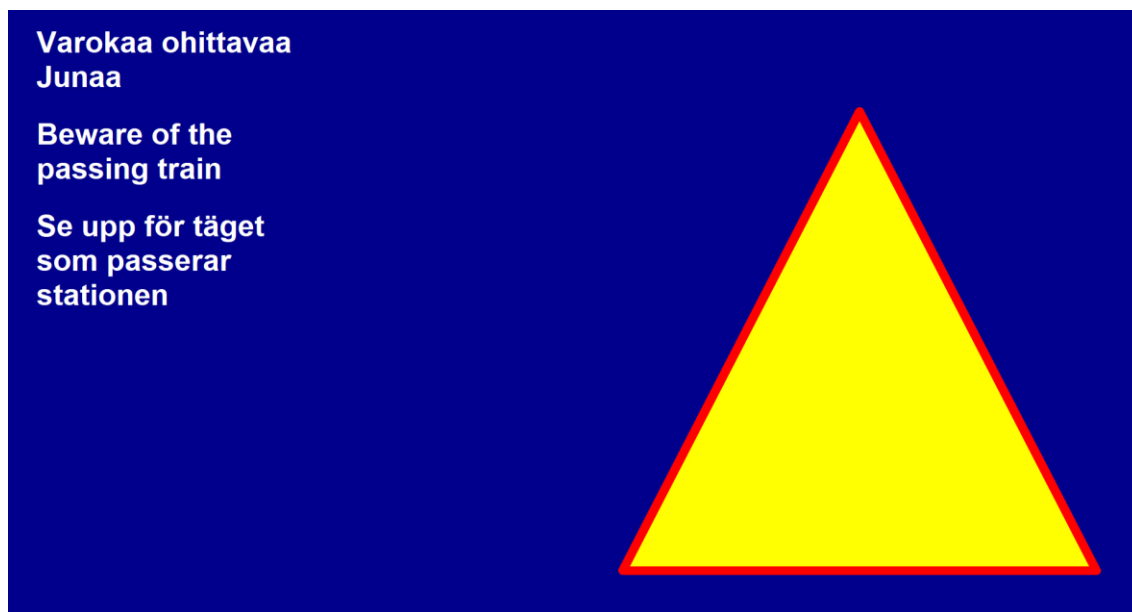
Järjestelmän näytöt koostuvat älykkäistä näkymistä, jotka muodostuvat näyttökomponenteista. Näihin näkymiin julkaistava junatietue voidaan myös määritellä erilaisilla argumenteilla, joilla voidaan erotella yksittäisen laiturin junat, saapuvat tai lähtevät junat, sekä lähi- että kaukojunat. Argumenttien ansiosta voi rautatieasemalle asentaa variaationäyttöjä asiakkaan toiveiden mukaan. Kokoonpanoja on helppo rakentaa ja skaalata juna-aseman suunnitelman kasvaessa.

Näyttöjen kokoonpano käyttää Pythonin Tkinter-käyttöliittymäkehystä. Projektin alussa ryhmä päätti sen käytöstä, perustellen, että sen käyttö on yksinkertaista ja se löytyy valmiiksi asennettuna monissa Python-versiojulkaisuissa.

Projekti sisältää neljä esirakennettua näkymää:

- | | |
|--------------------------|--|
| • Taulukkonäkymä | Junan aikataulut. |
| • Laiturinäkymä | Saapuvan junan esitiedot. |
| • Kahden laiturin näkymä | Saapuvien junien esitiedot / aikataulut. |
| • Informaationäkymä | Juna-aseman tapahtumat ja ilmoitukset. |

Aikataulujen lisäksi, näytöt vastaanottavat ilmoituksia, sekä tiedon ohittavasta junasta. Ilmoituksilla on kaksi tärkeystyyppiä: *info*, joka tulostaa tavallisen ilmoituksen näytön alaosaan ja *alert*, joka tulostaa varoitusviestin keskelle näyttöä. Näytöt, jotka on määritelty näyttämään vain yhtä laituria vastaanottavat ajan seuraavasta ohikulkevasta junasta. Näyttö reagoi heti, kun sellainen tietue ilmestyy järjestelmään. Minuuttia ennen kuin juna ajaa laiturin ohi, näytölle ilmestyy automaattinen varoitusviesti, jolloin muu tieto jää näkyvistä pois. Kun minuutti on saavutettu, niin ilmoitus katoaa ja näytön näkymä palaa ennalleen. Esitys varoitusviestistä löytyy kuvasta 4.



Kuva 4. Ohikulkevan junan varoitus.

Kuvan 4 varoitusviesti on näyttökomponenttien käyttöliittymäkomponentti, joka on rakennettu näkymän päälle. Sillä voidaan näyttää rinnakkaistietoja ilman, että se vaikuttaisi näkymän toimintaan.

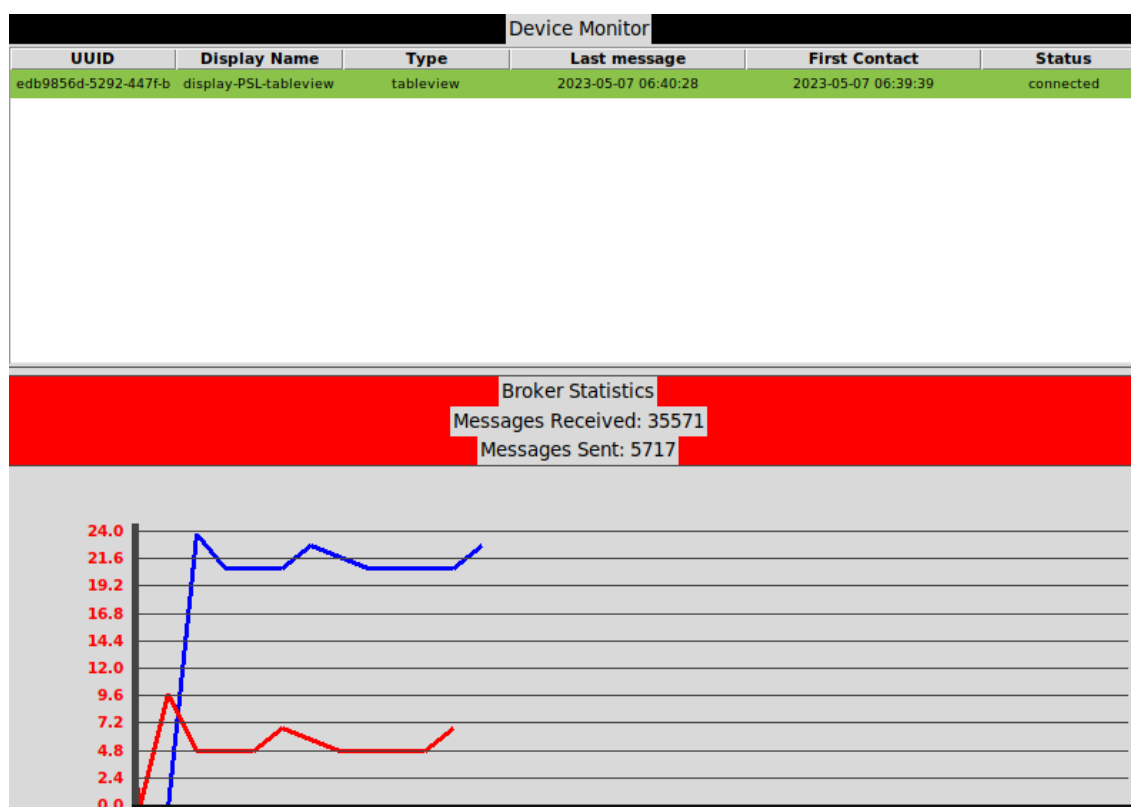
Pasila asema departing trains					22:35:40
Time	Notice	Train	Platform	Destination	
22:36		P	1	Helsinki asema	
22:37		L	11	Kirkkonummi	
22:38		A	10	Helsinki asema	
22:40		I	2	Lentoasema	
22:42		P	11	Lentoasema	
22:42		K	1	Helsinki asema	
22:43		U	9	Helsinki asema	
22:44		R	5	Helsinki asema	
22:45		R	3	Tampere asema	
22:51		IC972	9	Helsinki asema	
Juna- ja lentoliikenteessä yhä häiriöitä huonon sään vuoksi					

Kuva 5. Taulukkonäkymä.

Kuvassa 5 esitetyn taulukkonäkymän tieto koostuu useammasta aihekanavasta saapuneista tietueista. Näkymään siis saapuu kaikki Pasilan asemalta lähtevät junat, mukaan lukien lähi- ja kaukojunat, kaikilta laitureilta.

3.4 Management kokoonpano

Managerimoduuli rakentuu kahdesta ohjelmasta, aggregator-ohjelmasta ja hallinnointikäyttöliittymästä. Aggregator-ohjelmalla on oma käyttöliittymä, jossa pidetään lokikirjaa järjestelmän tapahtumista ja laitteiden tiloista. Se jäi tässä projektissa hieman keskeneräiseksi ja delegoituu jatkokehitysideaksi (Kuva 6).



Kuva 6. Aggregaattorin käyttöliittymä.

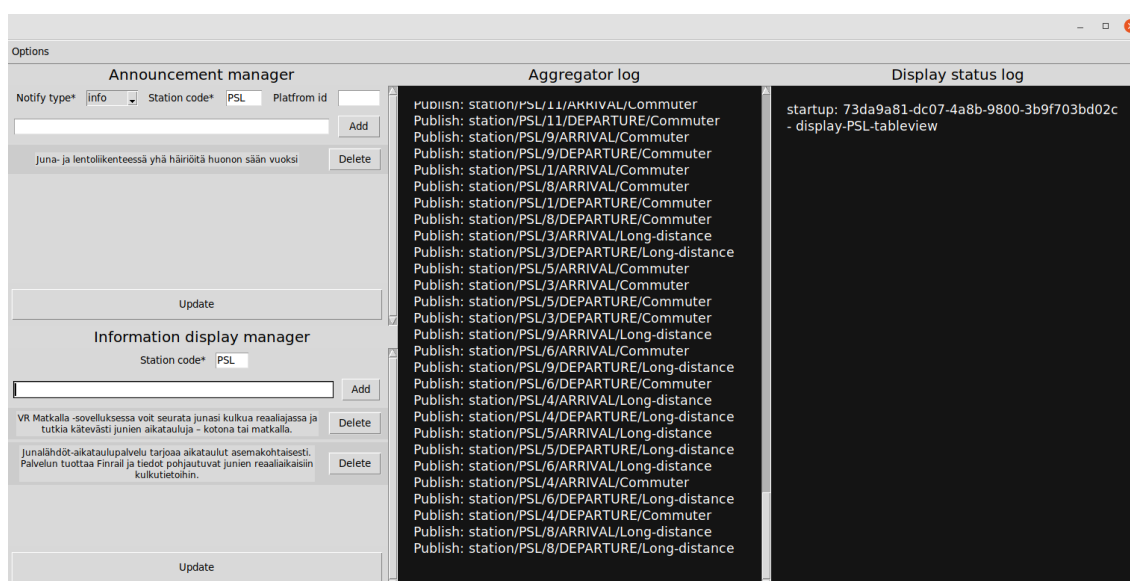
Aggregaattorin käyttöliittymä itsestään on modulaarinen ja oli rakennettu Tkinter-kirjaston päälle. Siihen oli suunniteltu editorin ominaisuuksia, kuten komponenttien lisäys ja näkymien jakaminen oikean hiiren valikosta. Alun perin piti luoda myös erilaisia kaavioita, mutta asiakkaan pientietokoneet eivät pysty

asentamaan pythonin NumPy-kirjastoa, josta monet kaaviokirjastot ja matemaatiikan ohjelmat ovat riippuvaisia. Tämä rajoitus on suunniteltu turvallisuussyistä.

Asiakas ehdotti, että ryhmä löytäisi toisen ratkaisun, jossa ei tarvita numpya. Rakensimme Tkinter-kirjaston päälle siis oman käyttöliittymäkirjaston, joka löytyy tiedostosta `pis/utils/tkui.py` ja oman kaavioluokan, joka löytyy tiedostosta `pis/utils/chart.py`.

Kuvassa 6, käyttöliittymä jakautuu kahteen komponenttiin. Yläosassa on yksinkertainen näyttöjen tilamonitori. Kun merkintä pysyy vihreänä, se tarkoittaa näytön olevan tällä hetkellä yhteydessä aggregaattoriin ja on tehnyt sen kanssa pariliitoksen. Jos merkintä jossain vaiheessa muuttuu oranssiksi, tarkoittaa se sitä, että jossain vaiheessa toimintaa, näyttö oli lähettänyt aggregaattorille valheellisia viestejä joiden eheyttä ei voitu varmistaa. Näin käy myös silloin, kun näyttö itse vastaanottaa sellaisia viestejä. Käyttöliittymän alaosan kaavio kartoittaa tulevien ja lähtevien viestin määrän.

Manager hallinnointikäyttöliittymä, joka toimii näyttöjen hallitsijana. Sen päätoiminnot ovat ilmoitusten lähettäminen näytöille ja aggregator-ohjelman julkaisemien MQTT-aihekanavien ja näyttöjen tilojen seuranta (Kuva 7).



Kuva 7. Managerin käyttöliittymä.

Manager ei itse julkaise ilmoituksia vaan lisää ne tietokantaan, johon aggregator-ohjelmalla on yhteys. Aina kun Managerissa tehdään muutoksia ja käyttäjä painaa Update-painiketta, se julkaisee tiedon `management/<displayID>/update` aihekanavalle, jota aggregator-ohjelma tilaa ja aina, kun aggregaattori vastaanottaa aihekanavalla viestin, se kerää tietokannasta kaikki ilmoitukset ja julkaisee ne näytöille. Aggregator-ohjelman ja managerin tietokantayhteys on paikallinen, joten Manageri tulisi käyttää samalta laitteelta (Management Module).

Ilmoituksilla on kaksi tärkeysastetta, `info` ja `alert`. Ennen ilmoitusten lisäämistä tai muokkaamista tulee käyttäjän valita asema, sekä tarvittaessa myös laiturei. Jos käyttäjä ei määritä laituria, ilmoitusviesti julkaistaan kaikille näytöille, joilla ei ole yhtä määriteltyä laituria. Sellaiset näytöt ovat esimerkiksi aseman päänäyttö tai kahden laiturin näyttö.

Managerilla voidaan myös hallita ilmoituksia, jotka näkyvät näytöissä, joihin on konfiguroitu informaationäkymä. Tämän näkymän tarkoitus on toimia aseman tiedotusten esittäjänä. Tämänhetkisessä toteutuksessa yhdellä asemalla voi olla vain yksi informaationäkymä instanssi. Jos asemalla on useita informaationäkymiä, silloin niihin välittyvät samat ilmoitukset.

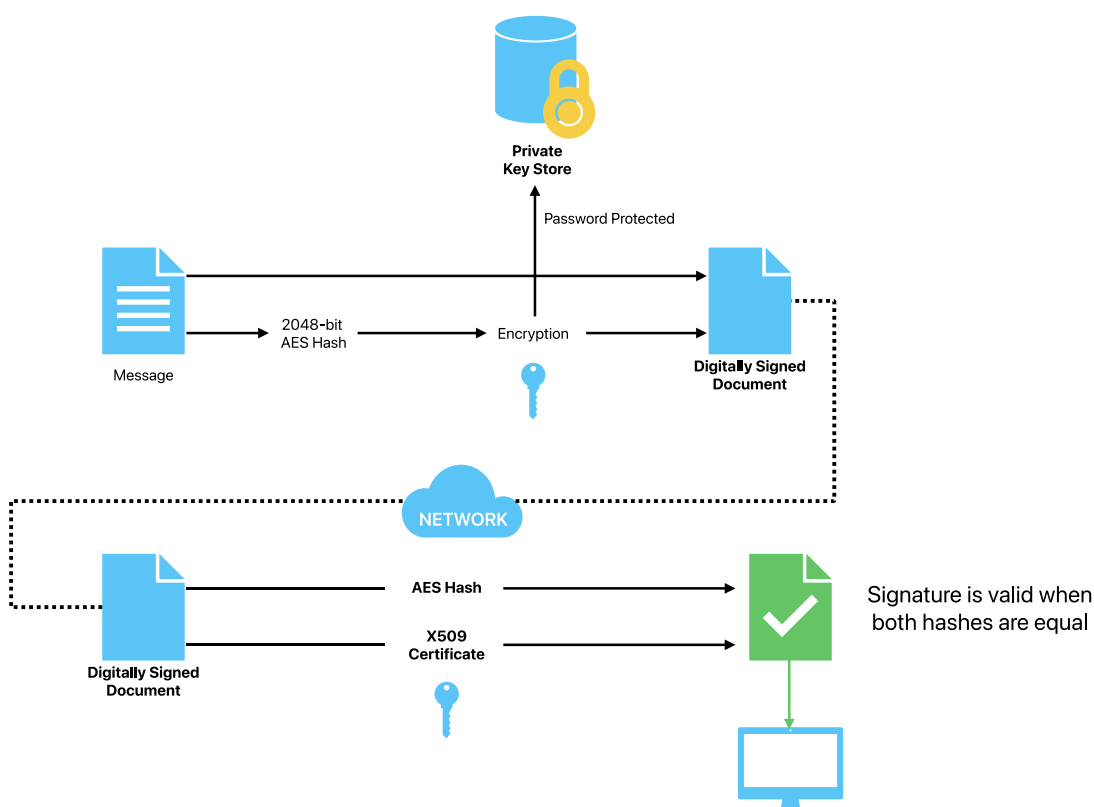
3.5 TPM ja viestin todentaminen

Yksi keskeisempiä projektin vaatimuksia oli välittävän datan validointi ja eheys. Kun MQTT on niin yksinkertainen protokolla, että aggregator-ohjelman ja näytön välinen viestiväylä on turvaton. Oli toivottu, että käyttäisimme apunamme TPM 2.0 turvapiiriä ratkaisuna. TPM:llä voi generoida satunnaislukuja, varmentamaan ja todentamaan todennusavaimia ja projektin näkökulmasta huolehtia siitä, että viestin sisältöä ei muutettu kuljetuksen aikana.

TPM on pieni fyysinen laite, joka löytyy nykyään melkein kaikissa uusissa suorittimissa valmiiksi asennettuna, vaikka poikkeavuuksia löytyy. Esimerkiksi Intel emolevyillä sen ominaisuuksien käyttö voi riippua PTT teknologiasta. PTT (*Plat-*

form Trust Technology) tukee TPM-ohjelmiston joitakin ominaisuuksia prosessorin kautta [3], ilman erillistä turvapiirin moduulia. AMD emolevyt voivat toimia saman periaatteen mukaan eri nimellä – *AdvancedAMD fTPM*. Raspberry Pi -pientietokone vaatii erillisen fyysisen lisäosan asennuksen.

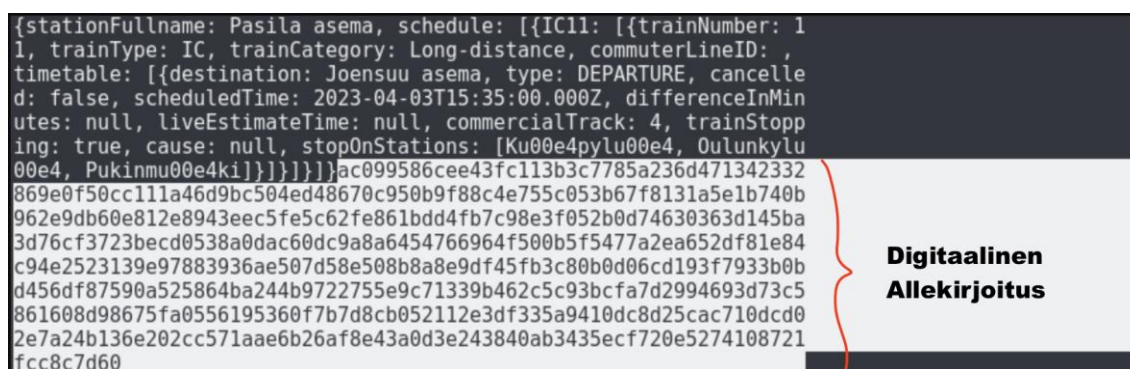
Kuitenkin, koska turvapiiri on fyysinen laite ja käytimme projektin aikana virtuaalista kehitysympäristöä, totesimme turvapiirin integraation ympäristöömme hyvin hankalaksi. Päätimme, että on parempi hakea toista ratkaisua, joka jäljittelee TPM:n toimintaperiaatetta mahdollisimman paljon. Ratkaisuksi valitsimme OpenSSL Digital Signature, eli digitaalisen allekirjoitusmenetelmän. Aggregator-ohjelman lähettämiin junaliikennetietoihin lisätään digitaalinen allekirjoitus, jonka näytöt todentavat.



Kuva 8. OpenSSL Digital Signature.

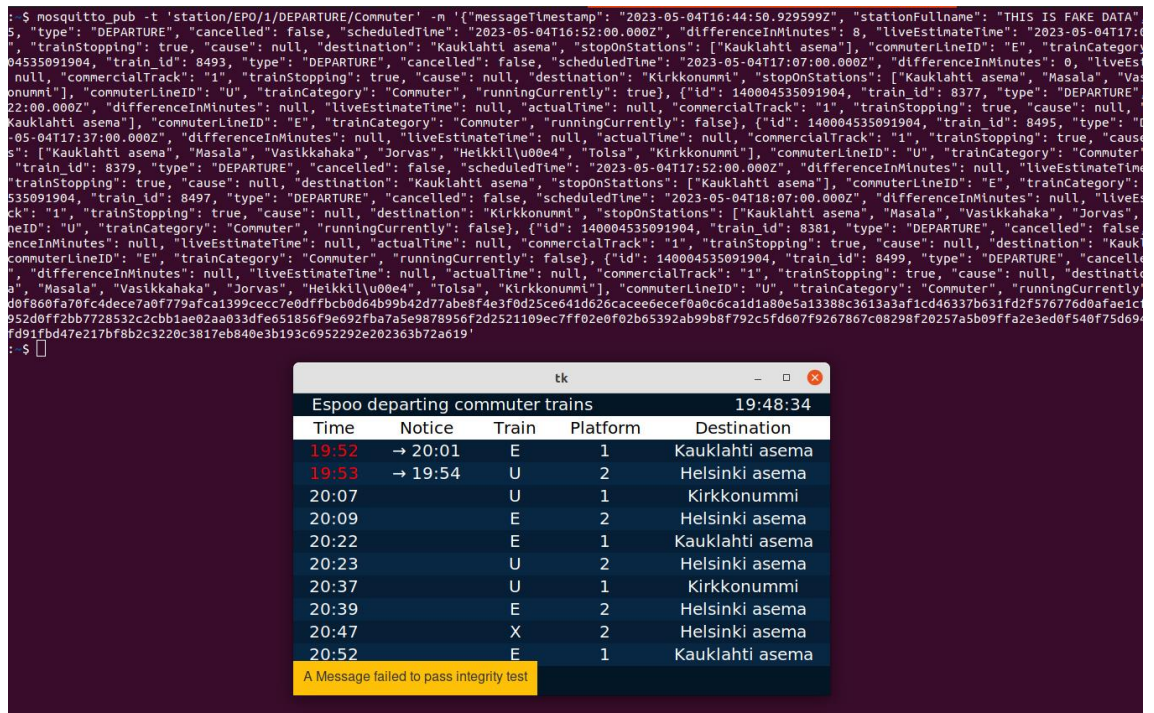
Ohjelmiston asennuksen aikana laitteen muistiin luodaan PKS-tiedosto [4], johon pakataan yksityinen 2048-bittinen avain. Tiedosto salataan käyttäjän antamalla salasanalla. Samalla salasanalla ohjelmisto pääsee myöhemmin käsiksi

yksityiseen avaimeen. Näytön ja aggregator-ohjelman pariliitoksen aikana, aggregaattori jakaa yksityisestä avaimesta julkisen osan näytön kanssa. Kuva 8 esittää miten, ennen kuin aggregator-ohjelma lähettää viestin eteenpäin, sille luodaan digitaalinen allekirjoitus. Viesti siten lähetetään alkuperäisessä muodossaan, niin että sen hännässä on mukana sisällön mukainen digitaalinen allekirjoitus. Allekirjoituksen muodon emme voi ihan suoraan nähdä viestistä, vaan kuvan 9 mukaisesti, allekirjoitus on salatussa muodossa.



Kuva 9. JSON-viestin digitaalinen allekirjoitus.

Todentaminen tapahtuu siten, että kun laite saa MQTT-aihekanavasta tietueen, niin sen allekirjoituksen salattu sisältö avataan X509-sertifikaatilla [5], joka on julkinen avain aikaisemmin saadusta pariliitoksesta. Allekirjoitus verrataan viestin sisältöön ja kun ne täsmäävät, niin voimme olla hyvin varmoja tietojen eheydestä.



Kuva 10. Virheilmoitus ongelmallisesta eheydestä.

Pystymme tarkistamaan datan validoinnin toiminnan kaappaamalla tietuen jostakin aiheväylästä, väärentämällä sen sisällön ja lähettämällä tiedon suoraan näytön tilaamaan MQTT-aihekanavaan. Tuloksena saadaan tapaus, jossa näyttö havaitsee viestin eheyden ongelmalliseksi ja automaattisesti jättää sen tiedon pois järjestelmästä ja tapauksesta tehdään ilmoitus (Kuva 10).

3.6 Junatietojen aggregointi ja käsittely

Aggregaattori-ohjelma hakee junatietoja avoimesta digitraffic.fi lähteestä. Digitraffic on Fintrafficin omistama rajapinta, joka tarjoaa Suomessa kulkevien junien junatietoja useista lähteistä. Siihen tehdään syklinen API-kutsu, joka palauttaa pyydetyn rautatieaseman tämänhetkiset junat ja junien esitiedot ja aikataulut, sekä kaikki rekisteröidyt pysäkit. Haku toimii syklisesti, eli kutsu tehdään jonkin ajan välein. Projektin aikana on kokeiltu erilaisia päivitysaikoja minuuteista tuntiin ja noin minuutin välinen sykli on toiminut parhaiten.

Tietojen kerääminen tapahtuu manager.py-nimisessä ohjelmatiedostossa, jota aggregator päätoimisesti käyttää. Junatiedoista poistetaan ensin ylimääräiset turhat tiedot, jonka jälkeen ne lisätään B+ binäärihakupuihin. B+ puu on yleinen itsetasapainottava puurakenne, jota käytetään paljon tietokannoissa ja käyttöjärjestelmissä. Sen hakuoperaation aikavaatimus on $O(n \log n)$ ja peräkäishaku voi olla $O(n)$, missä n on lisättyjen avainten määrä. B+ puu sopii hyvin tilanteisiin, jossa tapahtuu paljon lisäyksiä ja hakuja. Avainten poisto on monimutkainen ja vaikea prosessi, joten on helpompi poistaa hakupuu muistista. Emme siis tallenna junatietoja ollenkaan, vaan prosessoimme ne ja lähetämme suoraan eteenpäin. B+ hakupuumme salli tiedon lajittelun useammalla avaimella. Välitettävät junien aikataulutiedot pitää ensin järjestää tulo- tai lähtöajan ja sitten junatyypin (lähi- tai kaukojuna) perusteella, mutta myöskin lajittelu junan nimen ja laiturin perusteella pitäisi olla mahdollista.

Lopuksi junatiedot haetaan puusta, sillä hakukriteerillä, että junan saapumis- tai lähtöaika on suurempi kuin nykyinen aika ja niistä tiedoista rakennetaan MQTT-nimikonvention mukainen aihekanava, jonka jälkeen jäsennetyt tiedot lähetetään JSON-muodossa MQTT-viesteinä näytöille.

Kun näyttö vastaanottaa yhden tai useamman aikataulutietueen, aikataulut muutetaan muotoon, joka on helposti käytettävissä näkymän puolella. Ennen kuin junatiedot käsitellään, tulee näytön varmistua, että se on vastaanottanut kaiken mitä aggregaattori-ohjelma on julkaissut. Voidaan esimerkiksi ottaa tilanne, jossa näyttö tilaa junatietoja aseman kaikista laitureista. Tällöin näytölle saapuu jokaista laituria kohtaan yksi MQTT-viesti. Näyttöihin on luotu säie, joka jatkuvasti kuuntelee saapuvia viestejä. Aikataulut ovat alussa järjestetty viestien erillisinä sisältöinä ja viestien tuloaikojen mukaan. Saapuvat viestit yhdistetään ja järjestetään uudelleen B+ puussa pysähtymis- tai lähtöajan mukaan. Kaikki aikatauludatan kanssa tulevat ajat ovat UTC aikamuodossa, jolloin ajan muutos Helsingin ajaksi tapahtuu pythonin "pytz"-kirjastolla. Käyttöliittymän data päivitetään observer-suunnittelumallia hyödyntäen, jossa saapuvan datan päivittyessä, myös näkymän käyttöliittymän elementit päivittyvät samanaikaisesti.

Reactive-luokka tekee datasta tai elementistä reaktiivisen. Kun sen sisältö muuttuu, se voi laukaista jonkun toisen tapahtuman samaan aikaan. Sitä käytetään näkymään saapuvien tietojen synkronoimiseen käyttöliittymän kanssa.

Integrity-luokka sisältää OpenSSL:n avaintodentamisen toimintoja, mukaan lukien PKS-säiliön lukemiseen ja rakentamiseen tarkoitettuja metodeja.

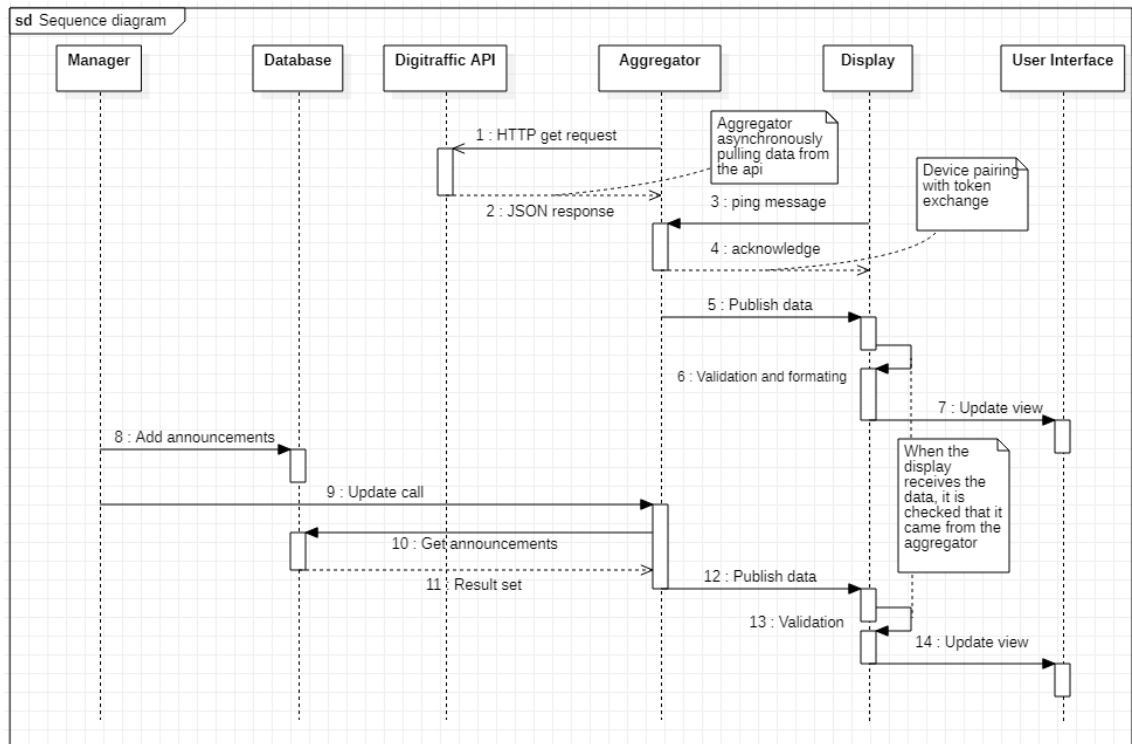
ManagementNode.Manager- luokkaa käytetään datan aggregointiin ja näyttöjen hallintaan. Luokasta käynnistetään Aggregator-ohjelman toiminnan. Se kerää tietoja yhdestä lähteestä ja jakaa ne viestiväyliin, jotka ovat tässä projektissa MQTT-aihekanavia.

Telemetry.Connection-luokka on yhteydessä MQTT-palvelimeen. Se tarjoaa paho.mqtt-paketin päälle rajapinnan, jolla voidaan nopeasti rakentaa tilauksia yhdellä kutsulla.

BPTree-luokka on pythonissa toteutettu B+ binäärihakupuu. Sen hyötykäyttöä varten löytyy query-niminen singletoniluokka, jolla puun hakutoimintoja laajennetaan. Puuta käytetään paljon aggregator-ohjelmassa junatietojen käsittelyyn.

Rata-luokka on yhteydessä rata.digitraffic.com sivustoon, jonne se voi tehdä API-kutsuja ja saada sieltä junien ja asemien aikataulut ja kokoonpanot.

Arkkitehtuurista löytyy kaksi announcement_manager-luokkaa. Yksi löytyy Management-moduulista ja toinen löytyy dashboard-moduulista. Managerimoduulin luokka vastaa ilmoituksien päivittämisestä ja lähettämisestä, kun Näyttömoduulin luokka vastaa ilmoituksien hallinnoinnista ja käyttöliittymästä.



Kuva 12. Järjestelmän sekvenssikaavio.

Kuvassa 12 on esitetty järjestelmän toiminta, joka alkaa aggregator-ohjelmasta. Digttrafficista haetaan junatietoja asynkronisesti, kunnes sieltä saadaan saapuvan tietuen JSON-muodossa. Samaan aikaan, käynnistyvä näyttö lähettää tekee pariliitoksen aggregaattorin kanssa kohdassa 3, jolloin aggregaattori lähettää sille "acknowledge"-tunnistuskutsun, joka sisältää julkisen todennusavaimen, jolla näyttö todentaa kaikki seuraavat tulevat viestit. Saapuvat viestit todennetaan kohdassa 6, ja mikäli kyseessä on aito junatieto, niin näyttö päivittää käyttöliittymänsä reaktiivisesti.

Manager on järjestelmässä ulkopuolinen ohjelma, jonka toiminta on näytöstä ja datan aggregoinnista riippumaton. Sen ainoa tehtävä on lisätä ilmoituksia ja varoituksia järjestelmän sisäiseen tietokantaan. Tarvittaessa aggregaattori lukee ilmoitukset tietokannasta ja lähettää ne näytöille julkaistavaksi. Kohdassa 13 tapahtuu samanlainen validointi, kuten kohdassa 6.

4 Järjestelmän käyttöönotto

Järjestelmä vaatii toimiakseen Python tulkin. Ennen järjestelmän asentamista tarkista, että Python-versiosi on vähintään 3.9.

```
python3 -version
```

Suosittellemme Pythonin virtuaalisen ympäristön käytön. Kehitysympäristösämme oli käytössä *virtualenv*-työkalu.

Jos asennus tehtiin pip:llä, niin järjestelmän ohjelmat voi käynnistää terminaalikomennoina. Jos asennus tehtiin Git-asennuksen ohjeiden mukaisesti, niin tiedostojen käynnistäminen sujuu parhaiten src-kansiosta: `cd platform-info-system/src` ja siellä, ohjelmiston paketti lisätään ympäristömuuttujaksi: `export PYTHONPATH=pis/:$PYTHONPATH`.

4.1 Asennus

Järjestelmän voi asentaa manuaalisesti tai Pythonin PIP pakettiasennusohjelmalla. Asennus on testattu toimivan ainakin DietPI, Ubuntu ja Rasbian Linux-distroilla. Asennuksen jälkeen ohjelmisto on käytettävissä terminaalikomennoina ja python kirjastona nimellä *pis*. Toinen tapa asentaa järjestelmä on kloonamalla projekti suoraan GitHub-sivulta [6], jolloin sen saa käyttöön myös Windows-ympäristössä. Suosittelemme pip-asennuksen käytön aina kun mahdollista.

Varmista, että git ja pip ovat asennettuna:

```
sudo apt-get update
sudo apt install git
sudo apt install python3-pip
```

Jos pythonissa ei ole sisäänrakennettua tkinter-pakettia, sen voi asentaa suoraan: `sudo apt-get install python3-tk`

Asenna Eclipse Mosquitto koneeseen, joka toimii hallinnointikoneena:

```
sudo apt-add repository ppa:mosquitto-dev/mosquitto-ppa
sudo apt-get update
sudo apt-get install mosquitto
sudo apt-get install mosquitto-clients
```

Käytä Mosquitto käynnistyspalveluna:

```
sudo systemctl enable mosquitto.service
```

4.1.1 PyPI asennus

Suosittellemme virtuaaliympäristön käytön, kun käyttäjällä on rajoitetut käyttöä-oikeudet, joten sellainen luodaan ja aktivoidaan ennen asennusta:

```
Python3 -m venv develop
Source /develop/bin/activate
```

Asenna ohjelma pipillä seuraavasti:

```
Python3 -m pip install git+https://github.com/InnovationProject4/platform-information-system.git
```

Asennuksen jälkeen alustetaan tarvittavat konfigurointitiedostot:

```
pis init
```

4.1.2 Git asennus

Git-asennus sopii tilanteeseen, kun halutaan tehdä kehitystyötä lähteelle. Asennus vaatii vähän enemmän toimintaa käyttäjältä:

1. Menee terminaalissa säilytyspaikkaan, esim. `/opt/`
2. Kloonaa projektin lähdekoodi: `sudo git clone https://github.com/InnovationProject4/platform-info-system`
3. Siirry `src` kansioon: `cd platform-info-system/src`
4. Määritä python-ympäristömuuttuja: `export PYTHONPATH=pis/:$PYTHONPATH`
5. Alusta tarvittavat asetustiedostot: `python3 pis/install/wizard.py init`

4.2 Ohjatun toiminnan käynnistäminen (Wizard)

Wizard on komentorivin ohjaus- ja käynnistys käyttöliittymäohjelma, jonka avulla järjestelmän ohjaus onnistuu sujuvasti ja nopeasti näytöllä, ohjeita seuraamalla.

Pip asennuksella: `pis`

Git asennuksella: `python3 pis/install/wizard.py`

```
Welcome to the Platform Information System wizard!

Command      Description
-----
display      create a display
aggregator   create an aggregator (management node)
config       edit configuration file
uninstall    uninstall the application
quit         exit the wizard

wizard#
```

Kuva 13. Ohjatun toiminnan alkunäkymä.

Kuvasta 13 ilmenee wizardin alkunäkymä. Sen komentorivistä löytyvät seuraavat komennot:

- `Display`: Näytön vaiheittainen käynnistys ohjeiden avulla.
- `Aggregator`: Aggregaattorin vaiheittainen käynnistys ohjeiden avulla.
- `Config`: Muokkaa määritystiedostoa, joka sisältää välittäjän IP-osoitteen, portin ja koko näytön tilan. Käyttäjä voi itse luoda tai poistaa määrittäjiä.
- `Uninstall`: Poistaa ohjelman esiasennetut tiedostot.

4.3 Aggregaattorin käynnistäminen

Pip asennuksella: `pis-aggregator`

Git asennuksella: `python3 pis/aggregator.py`

Aggregator komennon argumentit:

`<aseman_koodi>`

Yhden tai useamman aseman koodi kirjoitetaan välilyönnillä komennon perään. Usean aseman aggregointi onnistuu, kun koodit erotellaan välilyönneillä.

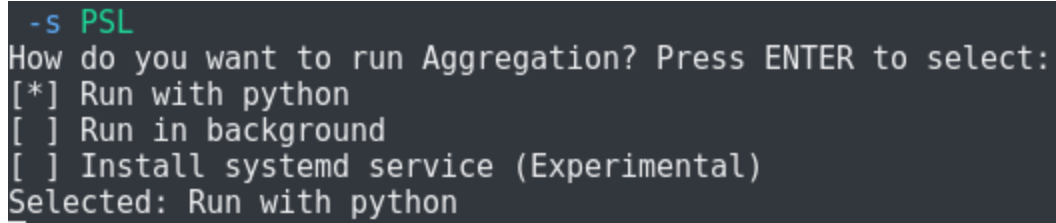
Esimerkkikoodi 1. `pis-aggregator PSL HKI`

Aggregaattorin voi myös käynnistää wizardin ohjatulla käynnistyksellä alikunäköstä komennolla `aggregator`, jolloin ruudulle ilmestyy ohjattu käynnistysvalikko (kuva 14).

```
-s None
Select railway station(s) you want to listen for. Press SPACE to select:
> [ ] AHO Ahonpää
  [ ] AHV Ahvenus
  [ ] AIN Ainola
  [ ] ARL Airaksela
  [ ] ATL Aittaluoto
  [ ] AJO Ajos
  [ ] APT Alapitkä
  [ ] ALV Alavus
  [ ] ALH Alholma
  [ ] ARO Arola
  [ ]
```

Kuva 14. Ohjatun käynnistuksen asemavalikko.

Kuvan 14 asemavalikko sisältää listan suomen rautatieasemista. Asentaja voi hakea halutun aseman nimen liikuttamalla listan nuolinäppäimillä tai hakemalla nimen kirjoittamalla aseman nimestä tuttuja kirjaimia. Aseman valinta tapahtuu välilyönnillä. Käyttäjä voi valita listasta enintään 10 vaihtoehtoa. Siirto seuraavaan näkymään tapahtuu enter-painikkeella.



```
-s PSL
How do you want to run Aggregation? Press ENTER to select:
[*] Run with python
[ ] Run in background
[ ] Install systemd service (Experimental)
Selected: Run with python
```

Kuva 15. Ohjatun käynnistyksen käynnistysvaihtoehdot.

Ohjatun käynnistyksen lopussa käynnistysohjelma kysyy käyttäjältä, miten ohjelman käynnistetään. Käynnistysvaihtoehtoja on kolme:

1. Käynnistetään suoraan pythonilla.
2. Käynnistetään komentorivillä rinnakkaisena ohjelmana.
3. Käynnistetään Linux-käyttöjärjestelmän palveluna.

Vaihtoehto 3 vaatii, että käyttöjärjestelmässä on Systemd-palvelunhallinta ja se toimii vain tietyissä tapauksissa. Kyseessä on kokeellinen ominaisuus ja sitä voi ajatella jatkokehitysideana. Suosittelemme ohjelman käynnistyksen vaihtoehdolla 1 (kuva 15).

4.4 Näytön käynnistäminen

Pip asennuksella: `pis-display`

Git asennuksella: `python3 pis/display_client.py`

Display komennon argumentit:

```
-view <näytön_näkymä> -s <aseman_koodi> -p <laituri> -left <laituri>
-right <laituri> -transit <kauttakulku> -transport <kuljetus>
```

Selitys argumenteille:

<code>-view "tableview"</code>	Taulukkonäkymä. Vaatii argumentin <code>-s</code> , mutta <code>-p</code> , <code>-transit</code> , <code>-transport</code> ovat valinnaisia.
<code>-view "splitview"</code>	Kahden laiturin näkymä. Vaatii argumentteja <code>-s</code> , <code>-left</code> ja <code>-right</code> , mutta <code>-transit</code> ja <code>-transport</code> ovat valinnaisia.
<code>-view "platformview"</code>	Laiturinäkymä. Vaatii argumentteja <code>-s</code> ja <code>-p</code> , mutta <code>-transit</code> ja <code>-transport</code> ovat valinnaisia.
<code>-view "infoview"</code>	Taulukkonäkymä. Vaatii argumentin <code>-s</code> .
<code>-transit</code>	Hyväksyy "departures" tai "arrivals".
<code>-transport</code>	Hyväksyy "commuter" tai "long_distance".
<code>-transit</code>	Hyväksyy "departures" tai "arrivals".

Esimerkkikoodi 2. `pis-display -view "tableview" -s PSL -p 11`

Display voi myös käynnistää wizardin ohjatulla käynnistyksellä alku näkymästä komennolla `display`, jolloin ruudulle ilmestyy ohjattu näkymävalikko.

```
-view None -s None -p None -left None -right None -transit None -transport None
Select display viewtype:
[*] tableview
[ ] platformview
[ ] splitview
[ ] infoview
Selected: tableview
```

Kuva 16. Ohjatun käynnistuksen näkymävalikko.

Kuvan 16 näkymävalikko sisältää pienen listan esirakennetuista näkymistä. Valikossa siirrytään nuolinäppäimillä ja valitaan yksi vaihtoehto enter-painikkeella, jolloin siirrytään seuraavaan valintanäkymään. Ohjatun käynnistyksen lopussa käynnistysohjelma kysyy käyttäjältä, miten ohjelman käynnistetään, kuvan 51 mukaisesti.

4.5 Ilmoitusten käyttöliittymän käynnistäminen (Manager)

Pip asennuksella: `pis-dashboard`

Git asennuksella: `python3 pis/manager_client.py`

Managerin yläpalkista Options-pudotusvalikosta voi tarkkailla kaikkia tämänhetkisiä ilmoituksia, joita tietokantaan on tallennettu, sekä poistaa kaikki tämän hetkiset ilmoitukset kaikista näytöistä. Kolmas valinta antaa mahdollisuuden tyhjentää kaikki ilmoituksiin liittyvät taulut tietokannasta. Tässä pitää ottaa huomioon, ettei valintaa hyödynnä, jos näytöissä on samanaikaisesti ilmoituksia näkyvissä. Ongelmaksi voi johtua se, ettei ilmoituksia saa poistettua näytöistä, ellei julkaise uusia tai käynnistä näyttöä uudelleen.

The screenshot shows two web-based management interfaces. The top interface, 'Announcement manager', has a tab labeled 'Options'. It contains three input fields: 'Notify type*' with a dropdown menu (labeled 'a.'), 'Station code*' with the value 'PSL' (labeled 'b.'), and 'Platform id' (labeled 'c.'). Below these is a text input field and an 'Add' button (labeled 'd.'). A large 'Update' button is below that (labeled 'e.'). The bottom interface, 'Information display manager', has a 'Station code*' field with 'PSL' (labeled 'f.'). Below it is another text input field and an 'Add' button (labeled 'g.'). A message in Finnish is displayed: 'Caikkien junien aikatauluissa on muutoksia olosuhteiden vuoks'. To the right of this message is a 'Delete' button (labeled 'h.'). At the bottom of this interface is another 'Update' button (labeled 'i.').

Kuva 17. Ohjatun käynnistyksen näkymävalikko.

Kuten kuvassa 17 näkyy, managerin käyttöliittymä jakautuu kahteen paneeliin. Ylin hallitsee yksittäisen näytön ilmoituksia ja alin hallitsee informaationäkymän ilmoituksia, jos sellainen on käynnistetty.

Käyttöliittymän selitykset:

- a) Ilmoitustyyppi, pakollinen parametri. Se voi olla `info` tai `alert`.
- b) Kohdistettu juna-asema, pakollinen parametri.
- c) Laiturin numero, jos esimerkiksi kyseessä on laiturinäkymä.
- d) Ilmoituksen lisäys. Add-painikkeesta rekisteröi viestin näytölle.
- e) Päivittää näytön ilmoitukset ja lähettää ne näytöille.
- f) Etsitään infonäkymä juna-asemalta. Pakollinen parametri.
- g) Ilmoituksen lisäys, Add-painikkeesta rekisteröi viestin infonäkymälle.
- h) Poista-painike. Poistaa viestin tietokannasta ja näytöstä.
- i) Päivittää infonäkymän ilmoitukset ja lähettää ne infotauluun.

Yksittäisten näyttöjen ilmoitukset:

1. Etsitään haluttu näyttö kirjoittamalla kohtiin (a.), (b.) ja tarvittaessa (c.) näytön argumentit. Esimerkiksi, jos näyttö sijaitsee Pasilan aseman laiturilla 11, niin kohtaan (b.) lisätään "PSL" ja kohtaan (c.) kirjoitetaan 11.
2. Lisätään tai poistetaan viesti kohdassa (d.)
3. Painetaan kohdan (e.) painiketta, joka päivittää ilmoitukset ja lähettää ne näytölle. Tämän vaiheen voidaan toteuttaa myös ennen vaihetta 2.

Informaationäkymän ilmoitukset:

1. Etsitään haluttu informaationäkymä juna-aseman koodilla kohdassa (f.). Esimerkiksi, jos infonäyttö sijaitsee Pasilan asemalla, niin kohtaan siihen kohtaan kirjoitetaan "PSL". Tässä on muistettava, että tämänhetkinen järjestelmä sallii vain yhden infonäytön asemaa kohti.
2. Lisätään ja poistetaan viesti kohdassa (g.)
3. Painetaan kohdan (i.) painiketta, joka päivittää informaationäkymän ilmoitukset ja lähettää ne infonäytölle. Tämän voi toteuttaa myös ennen vaihetta 2.

4.6 Asetusten hallinta ja todennusavaimen generointi

Järjestelmän asennuksen jälkeen laitteelle tallentuu esirakennettu konfiguraatio-tiedosto. Sen hakemistosijainti riippuu käyttöjärjestelmästä ja käyttöäioikeuksista. Tämä määritellään tiedostoissa `pis/utils/conf.py` ja `pis/install/postinstall.py`. Mikäli asennus ei löydä sopivaa hakemistoa, niin se rakentaa tiedoston käyttäjän kotihakemistoon. Tiedoston hakemiston voi selvittää ohjatusta käynnistysohjelmasta (Wizard), kuten se kuvassa 18 näkyy. Asennuksen jälkeisessä tilanteessa, asentaja voi tarvita, esimerkiksi IP-osoitteen muuttamista

konfigurointitiedostossa tai ehkä asentaja haluaa luoda uuden todennusavaimen. Molemmat toiminnot onnistuvat ohjatun käynnistysohjelman kautta. Ohjatun konfiguroinnin näkymä käynnistetään wizardin alkunäkymästä komennolla `config`, jolloin ruudulle ilmestyy asetusten komentorivi.

```
Configuration Wizard!

Command      Description
-----
edit          edit a stored value in sections
add <section> <key> <value> add a new key-value pair to section
rm <section> <key>      remove a key-value pair from section
grep <some key|some value> search for a key-value pair
gen           Generate encrypted validation key for data signing
ls            list all sections
save          commit and save all changes
exit          save modified config and exit the wizard
help          show this banner
config location /usr/local/etc/pids/config.ini

wizard# (config):
```

Kuva 18. Asetusten komentorivi.

Kuvasta 18 ilmenee asetusten komentorivi, jossa on seuraavat komennot:

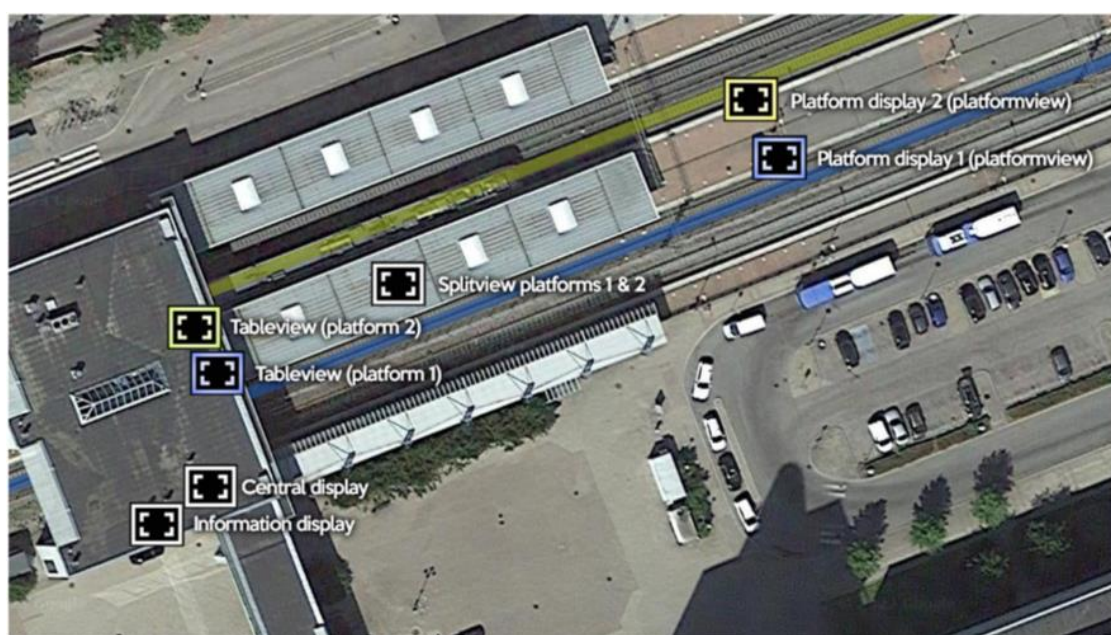
- `Edit`: Muokkaa olemassa olevan asetuksen.
- `Add`: Lisää uusi asetusta tai asetusosio.
- `Rm`: Poistaa asetusta konfiguraatitiedostosta.
- `Grep`: Hae jokin rivi konfiguraatitiedostosta. Esim. `grep token`.
- `Gen`: Generoi todennusavaimen.
- `Ls`: Palauttaa listan konfiguraatitiedoston osioista.
- `Save`: Tallentaa tehdyt muutokset konfiguraatitiedostoon.

Kun asetuksia muokataan ohjatussa käyttöliittymässä, ne eivät heti tallennu konfiguraatitiedostoon. Tallennus tapahtuu komennolla `save`. Samasta komentorivistä asentaja voi generoida uuden todennusavaimen komennolla `gen`, jolloin käynnistyy generoinnin näkymä. Jos laitteella on jo olemassa esiasennettu PKS-tiedosto, silloin ohjelma pyytää käyttäjältä sen tiedoston salasanan. Ilman salasanaa tiedostoa ei voi purkaa, eikä tällöin uutta avainta generoida.

Todennusavaimen generoinnin päätteeksi laitteelle tallentuu pfx-päätteinen sallittu säiliö, joka sisältää projektin järjestelmän yksityisavaimen. Konfiguraatiodostoon tallentuu tämän säiliön salasanan hajautusarvo, token-nimisenä asetuksena.

5 Käyttötapaus: Espoon rautatieasema

Päätimme testata ja esitellä järjestelmämme käyttöönottoa reaali maailman esimerkillä. Valitsimme käyttötapausesimerkiksi Espoon aseman ja toteutimme sille järjestelmämme avulla eri näytöistä koostuvan, junatietoja esittävän kokonaisuuden. Käyttötapausten esittämisen tarkoituksena on visualisoida mihin eri näyttötyypit voidaan asentaa juna-asemalla ja mitä tietoja ne voivat esittää.



Kuva 19. Espoon aseman käyttötapaus esimerkki

Kuvassa 19 on satelliittikuva Espoon asemasta, johon on lisätty yksinkertaistettu kokonaisuus eri näyttötyypeistä. Näyttötyypit ja selitys niiden sijainnista vasemmalta oikealle:

- Information display: Tässä esimerkissä aseman sisällä sijaitseva informaationäkymää käyttävä näyttö. Esittää Espoon asemaa koskevia ilmoituksia.

- Central display: Aseman sisällä sijaitseva taulukkonäkymää käyttävä näyttö, joka esittää koko aseman junatietoja kuten seuraavat 10 Espoon asemalta lähtevää junaa.
- Tableview (platform 1 & 2): Taulukkonäkymää käyttävät näytöt, jotka esittävät lähtevien junien tietoja raiteilla 1 ja 2. Näytöt on sijoitettu uloskäynnin läheisyyteen, josta astutaan laiturille raiteiden 1 ja 2 väliin.
- Splitview platforms 1 & 2: Näyttö, joka esittää kahden laiturin näkymää raiteille 1 ja 2. Näyttö on sijoitettu uloskäynnille, josta astutaan aseman laiturille raiteiden 1 ja 2 väliin.
- Platform display 1 & 2: Aseman laiturille sijoitettavat näytöt, jotka käyttävät "platformview"-laiturinäkymää. Näyttö esittää raiteen seuraavan lähtevän junan tietoja, sekä varoituksia ohiajavista junista.

6 Yhteenveto

Projektista jäi muutamia osio jatkokehitysideoiksi, mutta yleisesti se on onnistunut projekti. Asiakkaan kanssa käytiin viikoittaisia tapaamisia, jossa kävimme läpi kehitysideoita. Asiakas olisi toivonut ryhmältä enemmän suunnittelumallin käyttöä ja parempaa kommunikointia, mutta muuten asiakas on hyvin tyytyväinen lopputulokseen.

TPM:n käytössä ilmeni ongelmia ja ryhmä teki oman ratkaisun, joka toimii yhtä hyvin ja onnistuu validoimaan tietoja. Ryhmämme oppii projektin aikana paljon pienlaiteverkostoista ja viestiväylien turvallisuudesta. TPM:stä tuli tuttu käsite, jota varmasti käydään lisää läpi tulevaisuudessa.

Projektista kehittyi rautatieasemien informaatiojärjestelmä. SSL-validointi huolehtii tietojen aitoudesta, kun junatiedot lähetetään aggregaattorista ja kun ne saapuvat näytöille. Järjestelmä voi toimia yhden tai useamman juna-aseman

kanssa. Manageri-ohjelmalla julkaistaan erillisiä ilmoituksia paikallisista tapahtumista ja muista ilmoituksista.

Lähteet

- 1 Digitraffic. Rautatieliikenne. 2020. < <https://www.digitraffic.fi/rautatieliikenne>> (luettu 2.3.2023).
- 2 OASIS MQTT Version5.0. 2019. <<https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>> (luettu 2.3.2023).
- 3 Darek Fanton. Intel Platform Trust Technology (PTT): TPM For The Masses. 2022. <<https://www.onlogic.com/company/io-hub/intel-platform-trust-technology-ptt-tpm-for-the-masses/>> (luettu 10.3.2023).
- 4 PKCS. 2023. <<https://en.wikipedia.org/wiki/PKCS>> (luettu 9.4.2023).
- 5 SSL.com Support Team. What Is an X.509 Certificate? 2019 <<https://www.ssl.com/faqs/what-is-an-x-509-certificate/>> (luettu 9.4.2023).
- 6 GitHub. 2023. <<https://github.com/InnovationProject4/platform-info-system>>.

Nimikonventio

Listening to train information

`station / {stationCode} / {platformID} / {transit} / {transportType}`

Train status, schedules and advance information

Topic Level Parameters

stationCode	Ex. HKI	Location of transit
platformID	Positive Integer. 1 - 99	Platform number
transit	Departure Arrival	Type of activity for train at station
transportType	Commuter Long-distance Cargo	A type of train that is either regional or local

Publishing Service Announcements

`announcement / {notifyType} / {stationCode} / {platformID}`

Provides public announcements, service updates and other passenger notifications

Topic Level Parameters

stationCode	Ex. HKI	Location of transit
notifyType	Info Alert Infoview Passing	Notification status or action type
platformID	Positive Integer. 1 - 99	Train activity status on track or station

Payload

json : trainData

- type : info, alert, infoview, passing
- message : string (or json-string)

Device Communication

`management / {displayID} / {action}`

Manage communication between displays and their manager. When display awakens, it identifies itself. Managers perform may perform roll calls.

Topic Level Parameters

displayID	Uuid4 format	Unique device identification
-----------	--------------	------------------------------

Payload

json : deviceEvent

- event : startup, rollcall
- message : string
- messageTimestamp : datetime

