# Exercise Project

Tekoälyagentit

Ohjaaja Tapio Pahikkala

Opiskelija Feridun Akpinar (507943)

Tietojenkäsittelytieteen tutkinto-ohjelma

feridun.akpinar@utu.fi

# Sisällysluettelo

1. Johdanto....2
1.1 Tehtävän kuvaus ja analysointi....3
1.2 Tehtävän ympäristö ja käytettävät teknologiat....3
1.3 Rajoitukset....3
2. Ratkaisuperiaate....4
2.1 Agenttien standardit, ominaisuudet ja rajoitukset....4
2.2 Agentin käyttöönottoprosessi....7
2.3 Etusivu....8
2.4 Kojelauta-sivu....8
2.5 Ohjeet-sivu....10
3. Järjestelmän mallinnus....11
3.1 Järjestelmän mallinnus, käyttöliittymä ja ohjaus....11
4. Testaus....21
4.1 Yksikkötestaus....21
4.2 End-to-end testaus....22
5. Kirjallisuusviitteet....23

# 1. Johdanto

Viime vuosina tekoälyn merkittävä kehitys on lisännyt tekoälyagenttien syvällisempää tutkimista. Autonomiset tehtävien suorittamiseen kykenevät tekoälyagentit ovat yleistyneet sekä teollisuudessa että tieteellisissä tutkimuksissa. Tekoälyagentteja käytetään lähes kaikissa teollisuuden sovellusalueissa, mm. materiaalien löytämisessä [1], virtuaaliympäristöissä [2], e-kaupankäynnissä [3] ja esineiden internetin haasteiden ratkaisemisessa [4]. Tekoälyagenttien soveltuvuusalue on hyvin laaja, niitä voidaan soveltaa lähes minkä tahansa ongelman ratkaisemiseen.

Tekoälyagentien määritelmälle ei ole rajaa, se voi olla ohjelmisto-olio, järjestelmäkomponentti tai jopa jääkaapin lämpötilan säätäjä. Tekoälyagentin piirteille ja luokille on sen sijaan olemassa suhteellisen hyvät määritelmät. Mikä tahansa entiteetti, jolla on sensoreita tai havaintokanavia ja kyky päättää itsenäisesti voidaan kutsua tekoälyagentiksi [5]. Tyypillinen tekoälyagentti koostuu havaintokanavista, toimintaosasta ja agenttiohjelmasta. Tekoälyagentteja voidaan luokitella monella eri tavalla, yleisin tapa on agenttiohjelman mukainen luokitus. Agenttiohjelmaa käyttämällä tekoälyagentteja voidaan luokitella neljään eri luokkaan: refleksitagentit, mallipohjaiset refleksiagentit, tavoitepohjaiset ja hyödyllisyyspohjaiset agentit [6].

Refleksiagentti eroaa muista yksinkertaisella agenttiohjelmallaan. Sillä on erittäin triviaali agenttiohjelma, joka perustuu pelkästään ehto-toiminta sääntöihin. Reflektiagentin ehto-toiminta säännöt toimivat niin että havainnoista päätellään toteutuuko joku ehto, jonka jälkeen suoritetaan toteutuneen ehdon toiminta. Tällä yksinkertaisen tehokkaalla menetelmällä on kuitenkin rajoitteensa, se pystyy käsittelemään ainoastaan sen hetken havaintoja. Se ei siis säilytä havaintoja, eikä se kykene pitämään tilaa, jota käyttämällä se voisi tehdä vanhoista havainnoista riippuvia päätöksiä.

Mallipohjainen refleksiagentti eroaa refleksiagenteista sisäisellä tilallaan, jota refleksiagentilla ei ole. Tällaisella agentilla on malli ympäristöstään ja se ylläpitää tilansa siirtymämallilla ja anturimallilla. Siirtymämalli sisältää perustietämystä agentin ympäristöstä ja toimintojen vaikutuksista siihen. Anturimalli taas tarjoaa tietoa ja tietämystä havaintojen tulkkaamisesta ja niiden käsittelemisestä. Sisäisen tilan havaintohistorian ansiosta mallipohjainen refleksiagentti kykenee tekemään paljon monimutkaisempia päätöksiä kuin refleksiagentti.

Tavoitepohjainen agentti on yksi yleisimmistä agenttiluokista agenttipohjaisissa järjestelmissä. Sen ydin perustuu tavoitteeseen eli päämäärään, johon se yrittää päästää. Tällaisen agentin ohjelma on suurin piirtein samanlainen kuin mallipohjaisella refleksiagentilla, ainoa ero on gentin toiminnan valitsemisstrategiassa, joka vaattii valittua toimintoa viemään agentin kohti päämärää. Toimintojen valitseminen saattaa olla sekä triviaalia että paljon resursseja vievää, ja agentti saattaa suunnittella tai etsiä oikeita toimintoja hyvinkin pitkään. Tavoitepohjaisen agentin luonteenpiirteessää korostuu myös dynaaminen tavoite, eli toisin kuin mallipohjainen agentti tavoitepohjainen agentti kykenee vaihtamaan tavoitetta ajon aikana.

Tavoitepohjaisesa agentissa on yksi heikko puoli, se ei nimittäin ota kantaa miten tavoitteeseen tulee päästä, eikä se välitä tavoitteeseen kulutetuista resursseista. Toisin sanoen agentti saattaa päästä tavoitteeseen, mutta ei niin kuin olisi toivottu. Hyödyllisyyspohjainen agentti, joka on kaikista tekoälyagenttiluokista yleisin ja kehittynein luokka ratkaisee tämän ongelman hyödyllisyyfunktiolla. Hyödyllisyysfunktio maksimoi agentin etuja jokaisena hetkenä optimaalisten toimintojen valinnoilla. Agentin joutuessa tilanteeseen, jolloin sillä on useampi toiminto valittavana hyödyllisyysfunktio valitsee ainoastaan sen toiminnon, joka maksimoi sen edut.

## 1.1 Tehtävän kuvaus ja analysointi

Tehtävänä on kehittää web-sovellus, jossa käyttäjä voi luoda ja kokeilla yllä mainittujen luokkien tekoälyagentteja: refleksiagenttia, mallipohjaista refleksiagenttia, tavoitepohjaista agenttia ja hyödyllisyyspohjaista agenttia. Sovellus tarjoaisi erilaisia toimintoja, joilla käyttäjä kykenee luomaan, ajamaan ja poistamaan agentteja sekä tunnistamaan näiden agenttien eroja, hyötyjä ja rajoituksia. Rakennettavien agenttien sovelluskohde on IoT ja sensorit, eli agenttiohjelmat suunnitellaan sovellettavaksi IoT-laitteille ja niiden sensoreille. Tehtävän ydin on kehittää web-sovellus, jota käyttämällä opitaan tekoälyagenttiluokkien erot sovelluksen toiminnallisuuksia käyttämällä.

Tehtävään kuuluu viisi eri vaatimusta. Ensimmäinen vaatimus liittyy rakennettavaan ohjelmistotyyppiin. Sen on oltava web-sovellus ja se on toimittava offline-tilassa. Lisäksi web-sovelluksen tulee olla selkeä ja yksinkertainen, eikä se saisi vaatia rekisteröitymistä ennen käyttöä. Toiseen vaatimukseen kuuluu ohjeet-sivu, jossa on oltava kaikki ohjeet sovelluksesta ja sen toiminnoista. Kolmas vaatimus koskee kojelauta-sivua, johon on keskitettävä web-sovelluksen toiminnallisuudet. Neljäs vaatimus liittyy agenttien tavoitteisiin ja ehto-toiminta sääntöihin, joita on pystyttävä lataamaan web-sovellukseen tekstitiedostolla. Viidennessä vaatimuksessa vaaditaan komentoliittymää, jolla voidaan käyttää sovelluksen päätoimintoja.

## 1.2 Tehtävän ympäristö ja käytettävät teknologiat

Web-sovelluksen sovelluskehys on Aurelia. Aurelia on työpöytä-, mobiili ja web-sovellusten kehittämisalusta, joka on koostettu moderneista JavaScript-moduuleista [7]. Sovelluksen kehittäminen Aurelialla on nopea ja vakaa johtuen sen tarjoamista apukirjastoista ja työkaluista. Aurelialla tehtyjen komponenttien perustuminen Web standardeihin, kuten Web Components-teknologiaan helpottaa merkittävästi web komponenttien nopean kehittämisen ja testaamisen.

## 1.3 Rajoitukset

Johtuen tämän projektin rajoitetusta laajudeesta, kehittämisen kulutettu aika on minimoitava pitämällä kehitettävien toimintojen määrää mahdollisimman vähäisenä. Lisäksi web-sovelluksesta on jätettävä pois turvallisuusmoduulit ja algoritmit, joita löytyy yleensä vastaavissa web-sovelluksissa. Niitä käytetään agenttien käyttämien muisti- ja laskentaresurssien rajoittamiseen. Web-sovelluksesta tulee puuttumaan myös jonotuspriorisointimoduuli, jota käytetään yleensä jonossa olevien agenttien purkaamiseen.

Toiminnot, joita tekoaälyagentit tarjoavat pidetään laskennallisesti kevyinä ja yksinkertaisina. Tavoite on minimaalinen, stabiili ja toimiva versio vaatimuksia täyttävästä sovelluksesta. Painopiste pidetään tekoälyagenttiluokissa ja niiden erojen ja rajoitusten esille tuomisessa. Komentoliittymän komentojen rakennetta pidetään yksinkertaisena, jotta ne voitaisiin jäsentää ilman vaativaa jäsennintä. Sovellus ajetaan virtuaalipalvelimella, joilla on seuraavanlaiset tekniset tiedot.

| **Muisti** | **Tallennustila** | **Prosessori** | **Käyttöjärjestelmä** | **Internet-yhteys** | **Sijainti** |
|---|---|---|---|---|---|
| 2 Gt | 10 Gt | 3.5 Ghz | Ubuntu 20.04.2.0 LTS | 100 Mbps | Yhdysvallat |

*Taulu 1: Palvelimen tekniset tiedot*

## 2. Ratkaisuperiaate

Annetun tehtävän ensimmäisen vaatimuksen täyttämiseksi kehitetään web-sovellus, jossa on kolme sivua: etusivu, kojelauta-sivu ja ohjeet-sivu. Sovellus kehitetään itsenäiseksi web-sovellukseksi ilman tietokantaa ja palvelinta, niin, että koko sovellus pyörii itsenäisenä verkkoselaimessa. Tämä lähestymistapa valitaan pääosin turvallisuusyistä, sillä ensimmäisessa vaatimuksessa on pääsy sovellukseen ilman käyttäjätiliä ja salasanaa. Käyttäjätilitön pääsy sellaiseen sovellukseen, jolla voi suorittaa toimintoja palvelimen puolella altistaisi sen helposti verkkohyökkäyksille.

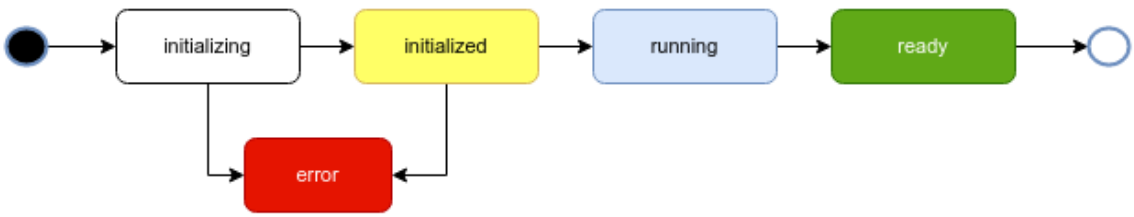
Toisen vaatimuksen täyttämiseksi sovellukseen tehdään ohjeet-sivu, jolle varataan osoite */instructions*-osoite. Tämä sivu pitää sisällään yleistä tietoa sovelluksesta, sen tarkoituksesta ja käytettävistä tekoälyagenttiluokista. Se tarjoaa yksityiskohtia sovelluksen agenteista. Selkeyttä korostetaan välttämällä teknisiä termejä, joiden lukeminen vaikeuttaa ymmärtämistä.

Kolmas vaatimus koskee kojelauta-sivua, jota kehitetään sisältämään sovelluksen kaikki toiminnallisuudet. Sivulle varataan /dashboard-osoite ja se rakennetaan hiearakisella rakenteella koostamalla se komponenteista ja moduuleista. Kojelauta-sivu sisältää logs-view-näkymää logien lukemiseen, neljännessä vaatimuksessa vaaditutta files-view-näkymää tiedostojen lataamiseen ja selailuun ja lopuksi agents-view-näkymää agenttien tilojen tarkkailuun ja niiden poistamiseen. Sivu sisältää myös viimeisessä vaatimuksessa pyydettyä komentoliittymää, jota käyttämällä voidaan luoda ja ajaa agentteja, ja generoida havaintoja.

### 2.1 Agenttien standardit, ominaisuudet ja rajoitukset

Agenttiin liitetään seuraavat ominaisuudet: tyyppi, nimi, tunnus, aloitusaika, lopetusaika ja tila. Agentin nimi koostuu *Agent* sanasta, alaviiva merkistä ja satunnaisista numeroista, jota sovellus arpoo. Agentin tunnus koostuu yhteensä 13 numerosta ja se käytetään agentin tunnistamiseen. Aloitusaika kertoo sen ajan kun agentin ohjelma käynnistettiin ja lopetusaika nimensä mukaisesti lopetusajan, eli silloin kun agentti pääsi tavoitteeseen tai pysäytettiin. Agentin tila kertoo agentin sen hetken tilasta.

Jokaiselle agentille rajataan viisi eri tilaa: initializing, initialized, running, ready ja error. Initializing-tila kertoo agentin olevan alustus-tilassa, jolloin agentti on lukemassa ehto-toiminta sääntöjä tai tavoitteita ladatusta tiedostosta. Initialized-tila tarkoittaa, että agentti on lukenut tiedoston ja valmiina ajettavaksi. Running-tilassa oleva agentti on aktiivisessa tilassa, jolloin se kykenee vastaanottamaan havaintoja. Ready-tilalla agentti ilmoittaa olevansa valmis ja error-tilalla joutuneensa virheelliseen tilaan.




*Kuva 1: Agentin tilat*

| **Agenttiluokka** | **Havaintomalli** | **Ehto-toiminta sääntömalli** | **Kuvaus** |
|---|---|---|---|
| Refleksiagentti | ID *NUMBER*<br>TARGET *STRING*<br>VALUE *NUMBER* | ID *NUMBER*<br>TARGET *STRING*<br>RULE [< \| > \| =] *NUMBER*<br>ACTION *STRING* | Tämä agentti vastaanottaa sellaisia havaintoja ja ohje-toiminta sääntöjä, joiden rakenne sisältää havaintomallin ja ehto-toiminta sääntömallin sarakkeissa listattuja attribuutteja.<br><br>**Havaintoesimerkki**<br>ID 1 TARGET TEMPERATURE VALUE 22<br><br>Tämä havainto koskee laitetta 1 ja kohdetta lämpötilaa. Havainto ilmoittaa lämpötilan arvoksi 22 astetta.<br><br>**Ehto-toiminta sääntöesimerkki**<br>ID 1 TARGET TEMPERATURE RULE <22 ACTION WARM_UP<br><br>Tämä ehto-toiminta sääntö koskee laitetta 1 ja kohdetta lämpötila. Sääntö aktivoi warm_up toiminnan eli lämmityksen mikäli lämpötila laskee alle 22 astetta. Agentti ei koskaan lopeta ohjelmansa vaan pysyy käynnissä ja vastaa havaintoihin yksitellen. |
| Mallipohjainen agentti | ID *NUMBER*<br>TARGET *STRING*<br>VALUE *NUMBER* | ID *NUMBER*<br>TARGET *STRING*<br>RULE [< \| > \| =] : *NUMBER*<br>ACTION *STRING* | **Havaintoesimerkki**<br>ID 1 TARGET TEMPERATURE VALUE 22<br><br>**Ehto-toiminta sääntöesimerkki**<br>ID 1 TARGET TEMPERATURE RULE <22:500 ACTION WARM_UP<br><br>Tämä ehto-toiminta sääntö koskee laitetta 1 ja kohdetta lämpötila. Sääntö aktivoi warm_up toiminnan mikäli lämpötila pysyy alle 22 asteessa yli 500 millisekunttia. Agentti ei koskaan lopeta ohjelmansa vaan pysyy käynnissä ja vastaa havaintoihin yksitellen. |
| Tavoitepohjainen agentti | ID *NUMBER*<br>TARGET *STRING*<br>*VALUE NUMBER* | ID *NUMBER*<br>TARGET *STRING*<br>RULE *NUMBER*<br>ACTION *STRING:STRING* | **Havaintoesimerkki**<br>ID 1 TARGET TEMPERATURE VALUE 22<br><br>**Ehto-toiminta sääntöesimerkki**<br>ID 1 TARGET TEMPERATURE RULE 22 ACTION WARM_UP:COOL_UP<br><br>Tämä ehto-toiminta sääntö koskee laitetta 1 ja kohdetta lämpötila. Sääntö aktivoi warm_up toiminnan mikäli lämpötila laskee alle 22 asteeseen ja cool_up toiminnan lämpötilan ylittäessä |

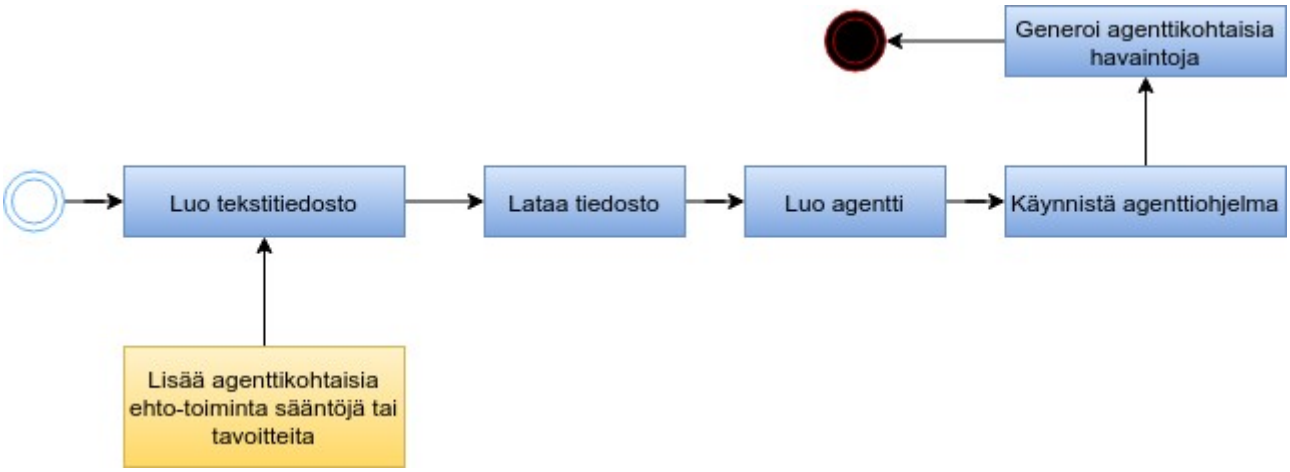
| | | | |
|---|---|---|---|
| | | | 22 astetta. Agenti siirtyy ready-tilaan eli saavuttaa tavoitteensa ja pysäyttää ohjelmansa mikäli se saa havainnon, jossa lämpötilan arvoksi on ilmoitettu 22 astetta. |
| Hyödyllisyys-pohjainen agentti<br><br>(Tämän agentin agenttiohjelma on erikoistunut lämpötilaan) | ID *NUMBER*<br>TARGET *STRING*<br>VALUE *NUMBER* | ID *NUMBER*<br>TARGET *STRING*<br>RULE *NUMBER*<br>ACTION *STRING:STRING* | **Havaintoesimerkki**<br>ID 1 TARGET TEMPERATURE VALUE 22<br><br>**Ehto-toiminta sääntöesimerkki**<br>ID 1 TARGET TEMPERATURE RULE 22 ACTION WARM_UP:COOL_UP<br><br>Tämä ehto-toiminta sääntö koskee laitetta 1 ja kohdetta lämpötila. Sääntö kutsuu hyödyllisyysfunktion ja antaa sille warm_up ja cool_up toiminnat. Agenti siirtyy ready-tilaan eli saavuttaa tavoitteensa ja pysäyttää ohjelmansa mikäli se saa havainnon, jossa lämpötilan arvoksi on ilmoitettu 22 astetta.<br><br>Tämän agentin ohjelma eroaa edellisen tavoitepohjaisen agentin ohjelmasta lämpötilaan erikoistuneella hyödyllisyysfunktiolla. Hyödyllisyysfunktio aktivoi jommankumman toiminnan jos ja vain jos lämpötilanmuutos on pienenevä (smart balance). Jos lämpötila on esim. 18 astetta ja sen jälkeen agentti saa havainnon, että lämpötila on 19 astetta, niin hyödyllisyysfunltio aktivoi lämmityksen. Muussa tapauksessa lämmitystä ei kannata tehdä sillä se voi tarkoittaa, että ulkoinen ovi on auki ja kaikki lämmitys menisi turhaan. Sama koskee jäähdytystä. |

*Taulu 2: Agenttien havainto- ja sääntömallit*

Tehtävänannossa agenttien sovelluskohteeksi on valittu IoT ja sensorit. Tästä syystä agenttiohjelmien sovellusalue rajataan sensoridataan. Refleksiagentin, mallipohjaisen refleksiagentin, tavoitepohjaisen agentin ja hyödyllisyyspohjaisen agentin ohjelmat rakennetaan niin, että ne pystyvät käsittelemään pelkästään sensoridataa ja tarjoamaan ainoastaan seuraavia toimintoja: <, > ja = operaattorit. Käytännössä tämä tarkoittaa, että agentit voivat hyväksyä vain sellaisia ehto-toiminta sääntöjä ja tavoitteita, joissa ehdot koskevat sensoreiden numeerisia arvoja ja joiden vertausoperaattorit rajoittuvat <, >- ja = operaattoreihin. Taulukossa 2 on agenttien havaintomallien ja ehto-toiminta sääntömallien tarkemmat kuvaukset ja esimerkit.

## 2.2 Agentin käyttöönottoprosessi

Ennen agentin käyttämistä, käyttäjän on varmistettava, että luotavan agenttiluokan mukaiset ehto-toiminta säännöt tai tavoitteet ovat kirjoitettu oikein tiedostoon. Sen jälkeen käyttäjä aloittaa käyttöönottoprosessin, joka on viisi vaiheinen ja kaikille agenttiluokille samanlainen.




*Kuva 3: Agentin käyttöönottoprosessi*

**Vaihe 1**

Agentin luomisprosessi alkaa uuden txt-tyyppisen teksititedoston luonnilla omalla työpöydällä. Tiedoston nimellä ja merkistöllä ei ole väliä, kunhan merkistössä ei käytetä ä-, ö- ja å-kirjaimia. Tiedostoon lisätään luotavan agentin ehto-toiminta säännöt tai tavoite agenttiluokkakohtaisella synktaksilla, joita löytyy taulusta 2. Esimerkkejä refleksiagentin ehto-toiminto säännöistä:

*ID 001 TARGET TEMPERATURE RULE <22 ACTION WARM_UP*

*ID 001 TARGET TEMPERATURE RULE >22 ACTION COOL_DOWN*

Ensimmäinen ehto-toiminta sääntö kohdistuu laitteelle 001 ja kohteelle lämpötila. Mikäli lämpötila laskee alle 22 asteeseen agentti suorittaa WARM_UP-toiminnan. Seuraava ehto-toiminta sääntö on samanlainen kuin edellinen, ainoa ero on toiminnassa COOL_DOWN, joka aktivoituu lämpötilan noustessa yli 22 asteeseen.

**Vaihe 2**

Seuraavaksi käyttäjä lataa tiedoston applikaatiolle komennolla *upload*, ja tarkistaa tiedoston latauksen onnistumisen avaamalla tiedostot-näkymää. Tiedosto ilmestyy näkymään mikäli lataus onnistuu. Käyttäjä voi myös ladata tiedoston siirtymällä tiedostot-näkymään ja painamalla lataa tiedosto-tekstiä tai raahamalla tiedoston komponentin kohdalle.

**Vaihe 3**

Tiedoston latauksen onnistuessa, seuraava vaihe on luoda uutta agenttia komennolla *create agent -class reflex -file 0.* Tämä komento luo uuden refleksiagentin 0-indeksin tiedostolla. Indeksi 0 viittaa ensimmäiseen tiedostoon ladattujen tiedostojen listassa. Indeksin arvo ilmestyy tiedoston nimen alle hakasuluissa.

**Vaihe 4**

Seuraavaksi ajetaan agentti komennolla *run agent **agent_id.*** Tässä komennossa agent_id viittaa agentin tunnukseen, jota voi saada selville avaamalla agentit-näkymän. Komento käynnistää agenttiohjelman sille annetulla ehto-toiminta säännöillä. Mikäli tiedoston ehto-toiminta säännöt ovat syntaksiltaan väärin, agentti ilmoittaa tilakseen error-tilan agentit-näkymässä. Muussa tapauksessa agentti ilmoittaa tilakseen initialized, jonka jälkeen agentti on valmis vastaanottamaan havaintoja.

**Vaihe 5**

Viimeinen vaihe on havaintojen generointi. Esimekki havainto luodaan seuraavalla komennolla: generate perception id 1 target temperature value 13. Tässä komennossa luodaan havainto, jonka lähde on id 1(sama kuin 001), kohde lämpötila ja arvo 13. Havainto kertoo siis, että laitteen 1 lämpötila on 13.

Mikäli ongelmia ei esiinny, agenttiohjelma lukee havainnon ja tulostaa log-näkymälle viestin ACTION WARM_UP HAS BEEN ACTIVATED.

## 2.3 Etusivu

Etusivu on web-sovelluksen oletussivu, jolle on varattu /-osoite. Sivussa on perustietoja applikaatiosta ja linkki tälle dokumentaatiotiedostolle.

## 2.4 Kojelauta-sivu

Kojelauta-sivu on koko sovelluksen ydin ja se koostuu komentoliittymästä, tiedostot-näkymästä, agentit-näkymästä ja logit-näkymästä. Nämä kaikki osat kehitetään itsenäisinä komponentteina, jotta sovelluksen laajennus olisi helpompi ja kirjoitettavien testien määrä myös vähäisempi. Komentoliittymän vastuuseen kuuluu toimintojen käyttäminen komennoilla, kun taas tiedostot-näkymä on tarkoitettu ladattujen tiedostojen selailuun ja poistamiseen. Agentit-näkymä näyttää sekä luodut agentit että niiden tilatiedot poistamisominaisuudella. Logit-näkymässä voi lukea logeja ja agentin kirjoittamia viestejä.

### Komentoliittymä

Viidennessä vaatimuksessa vaaditaan komentoliittymän kehittämistä, jolla voidaan käyttää sovelluksen päätoimintoja. Komentoja kirjoittamalla voidaan luoda agentteja, käynnistää agenttien ohjelmia, ladata agentin ehto-toiminta sääntötiedostoja, generoida havaintoja ja avata näkymiä. Komentoliittymän toiminnallisuus on yksinkertainen ja helppo, se lukee ensin käyttäjän syöttämän komennon, sitten se tulkkaa komennon ja lopulta se suorittaa komennon mikäli komennon syntaksi on oikein. Jos komennon suoritus onnistuu, sovellus värittää komennon tila-kuvaketta vihreällä värillä. Jos komennon ajaminen epäonnistuu, sovellus värittää tila-kuvakkeen punaisella värillä ja tulostaa virheviestin komennon alle. Tällä pyritään varmistamaan, että käyttäjä saa tiedon siitä oliko komento kirjoitettu oikein.

Komentoliittymä siirtää päätoiminnot komentojen vastuulle, eikä käyttöliittymälle, jotta sovelluksen kehittämistyömäärä vähenisi merkittävästi.

| **Ohjelma** | **Argumentit** | **Asetukset** | **Kuvaus** |
|---|---|---|---|
| Upload | [file/text-file/folder/perceptions] | | Aktivoi tiedostojen liittämiskomponentin<br><br>---------- |

| | | | |
|---|---|---|---|
| | | | upload file |
| Create | agent | class<br>file | Tämä komento luo uuden agentin annetulla luokan ja tiedoston arvoilla<br><br>*class* arvo viittaa agentin ohjelman tyyppiin, joita ovat reflex, mode-reflex, goal ja utility<br><br>*file* arvo viittaa tiedoston indeksiin, joka alkaa 0 (nollalla)<br><br>---<br>create agent -class reflex -file 0 |
| Show | files | | Avaa tiedostonäkymän<br><br>---<br>show files |
| Generate | perception | id<br>target<br>value | Luo uuden havainnon annetuilla tiedoilla<br><br>*id* arvo on on laitteen tunnus<br>*target* arvo on kohteen nimi<br>*value* arvo on kohteen arvo<br><br>---<br>generate perception id 1 target temperature value 22 |
| Run | agent | | Käynnistää sen agentin ohjelman, jonka tunnus on *agent* arvo<br><br>---<br>run agent 5555 |

*Table 1: Terminal commands*

Web-sovelluksen vaatimuksien täyttämiseksi komentoliittymään lisätään yllä olevan taulun 1 komennot. Jokaisen komennon rakenne koostuu ohjelmasta, argumenteista ja asetuksista. Taulukon ensimmäinen rivi listaa ohjelman nimen, toinen rivi argumenttien nimet ja kolmas rivi asetusten nimet. Hakasulkeissa olevat argumentit ja asetukset ovat valinnaisia, muut ovat pakollisia. Neljännessa rivissä on kuvaus sekä komennosta että sen argumenteista ja asetuksista. Kuvaus sarakkeen alin teksi on esimerkkikomento. Komento show and upload ovat apukomentoja, kun taas loput komennoista ovat päätoimintoja. Komentoliittymän kaikki komennot ajetaan ilman jonoon laittamista ja näiden vastaukset tulostetaan suoraan logit-näkymään.

### Logit-näkymä (Logs-view)

Logit-näkymä on sovelluksen logien selailuun tarkoitettu näkymä. Jokainen logiin kuuluu viesti, tulostusaika ja agentin tunnus, jos se on peräisin agentista. Logeja voi sekä poistaa että suodattaa, jos niitä on paljon näkymässä. Logien suodatuksessa voi käyttää agentin tunnusta tai viestin sisältöä. Tiettyjen agenttien logit

suodatetaan kirjoittamalla suodatinkenttään teksti *Agent:* ja agentin tunnus, esim. *Agent: 111*. Suodatus tekstillä tehdään kirjoittamalla *Text:* ja teksti, jonka täytyy sisältää etittävissä logeissa, esim. *Text: ACTION*.

### Tiedostot-näkymä (Files-view)

Tiedostot-näkymässä voidaan selailla ladattuja tiedostoja ja nähdä niiden nimet, tyypit ja indeksit. Lisäksi näkymässä voidaan poistaa tiedostoja. Tiedostot-näkymän tarkoitus on näyttää käyttäjälle ladattuja tiedostoja, niiden nimiä ja indeksejä.

### Agentit-näkymä (Agents-view)

Agentit-näkymän tehtävä on näyttää luodut agentit ja niiden tiedot, joita on mainittu tämä dokumentin kohdassa 2.1. Lisäksi näkymä tarjoaa painikkeita agentin poistamiseen ja agentin logien lukemiseen.

## 2.5 Ohjeet-sivu

Ohjeet-sivulle varataan */instructions*-osoite ja sen vastuuseen kuuluu sovellukseen liittyvien yleistietojen tarjoaminen sekä kaikkien toimintojen yksityiskohtaiset käyttöohjeet. Sivun ohjeita lukemalla käyttäjä ymmärtää sovelluksen tarkoituksen ja oppii käyttämään sitä.

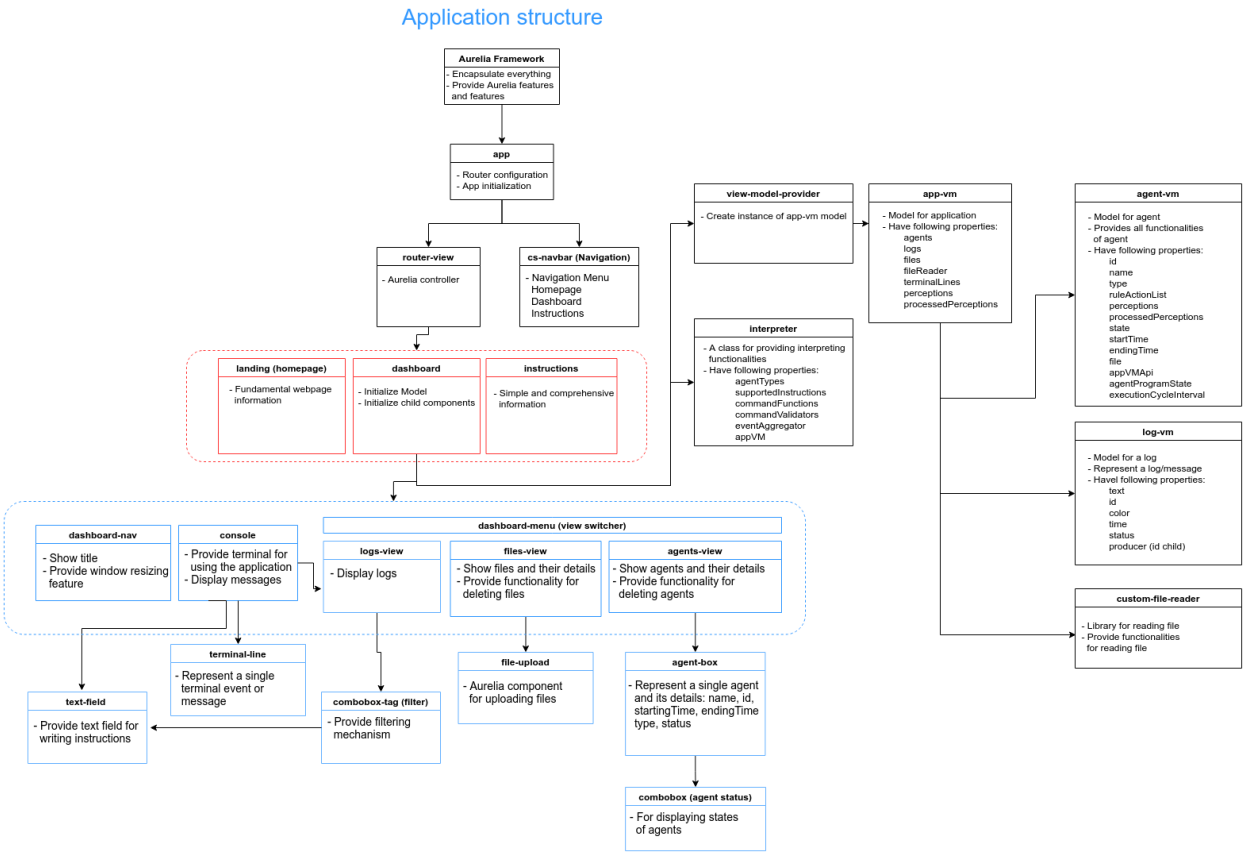
Sivun vasemmalle puolelle tulee tietoa tekoälyagenttiluokista ja niiden eroista. Samalle säiliölle tulee tietoa sovelluksen tarkoituksesta. Sivun oikealle säiliölle taas lisätään ohjeita agentin käyttöönottoprosessissta ja komentoliittymän käyttöohjeista. Tämän dokumentin taulu 1 ja 2 lisätään sivulle englannin kielellä.

# 3. Järjestelmän mallinnus

Web-sovellus on kehitetty Aurelia-sovelluskehyksellä käyttäen MVC-arkkitehtuuria. Sovellus on rakenteellisesti jaettu kolmeen eri osaan: malliin, ohjaukseen ja näkymään. Malli on useasta luokasta koostuva olio, joka ylläpitää sovelluksen tilaa, tietoja ja metodeja. Näkymäosa on Aurelialla tehdyistä komponenteista koostettu kokonaisuus. Ohjausosan hoitaa pääosin sovelluskehyksen sisäiset ohjainominasuudet, ja komponenttien luokkiin on kirjoitettu kaikki muut olennaiset ohjaukseen liittyvät toiminnallisuudet.

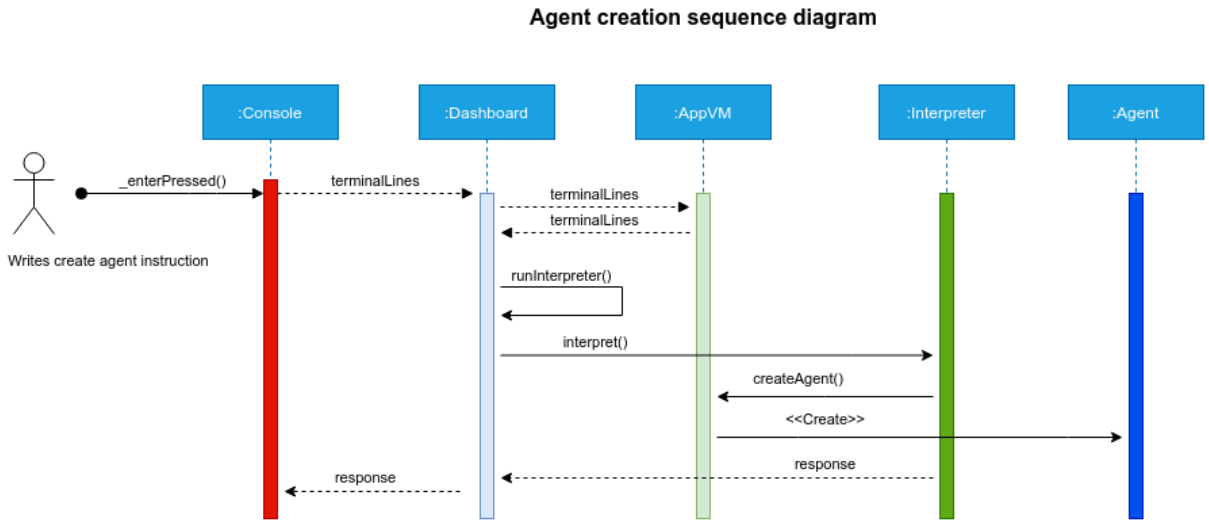
## 3.1 Järjestelmän mallinnus, käyttöliittymä ja ohjaus

Sovellus käyttää Aurelian hierarkista sovellusrakennetta. Ylin komponentti on app-komponentti ja se pitää sisällään kaikki sovelluksen komponentit. App-komponentin sisällä on router-view niminen reititin-komponentti, joka on Aurelian oma komponentti ja se on tarkoitettu sivujen aktivoimiseksi. Sovelluksen reitit rekisteröidään app-komponentin app.js-tiedostossa. Kuvassa 4 on web-sovelluksen rakenne.



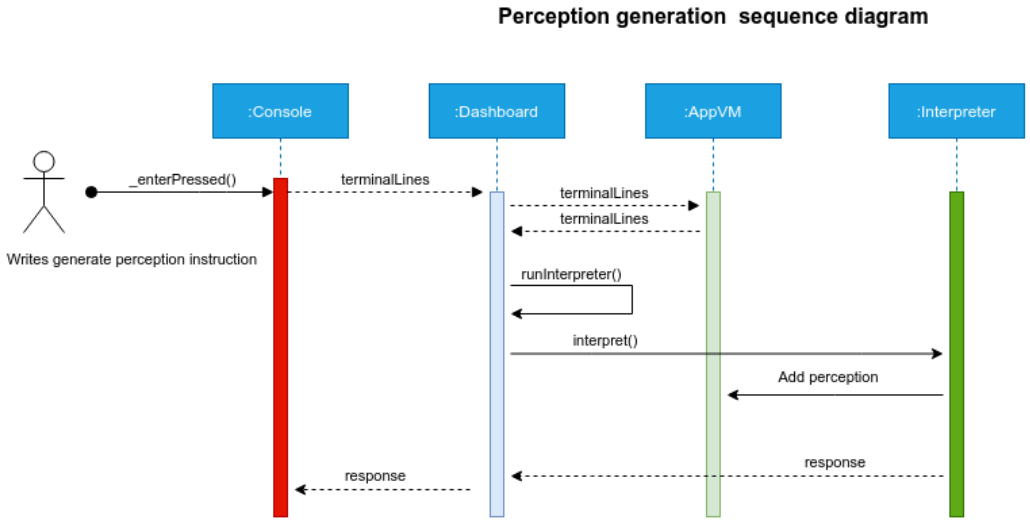

*Kuva 4: Sovelluksen rakenne*

Kuvassa 5 on agentin luonnin sekvenssikaavio. Agentin luomiskomento kirjoitetaan komentoliittymälle jonka jälkeen painetaan enter-näppäintä. Vastaanotettu komento siirtyy sovelluksen kojelauta-komponentille, joka edelleen lähettää se tulkkaajalle tulkattavaksi. Mikäli agentin luomiskomento on oikein ja virheitä ei esiinny komennon oikea ?-kuvake muuttuu vihreän väriseksi.




*Kuva 5: Agentin luonti - sekvenssikaavio*

Kuvassa 6 on havainnon luonnin sekvenssikaavio. Luotaessa havaintoa sovellus tekee syntaksitarkistuksen ja lisää sen jälkeen havainnon AppVM-mallin perceptions-tauluun.




*Kuva 6: Havainnon luonti - sekvenssikaavi*

Aurelia käyttää Web Components-standardin komponentteja. Jokaisella Aurelia komponentilla on html-tiedosto, css-tiedosto ja js-tiedosto. Html-tiedosto pitää sisällään komponentin HTML5-sisällön template-muodossa ja vastaavasti css-tiedosto pitää komponentin tyyliasetuksia. Komponentilla tulee olla myös saman niminen JavaScript-tiedosto ja sen tulee tarjota saman nimisen luokan CamelCase-muodossa. Komponenttien tekeminen vaatii Aurelian dokumentaatioon perehtymistä ja harjoittelua.

Seuraava taulukko listaa näkymän komponentit ja niiden vastuut.

| **Komponentti** | **Tiedostot** | **Kuvaus** |
|---|---|---|
| app | app.js<br>app.css<br>app.html | Ylin näkymäkomponentti (elementti). Pitää sisällään loput sovelluksen komponenteista.<br><br>App-komponentin alustuksen yhteydessä alustetaan myös reititin ja menu. |
| router-view | Ei tiedostoja | Tämä komponentti on Aurelian käyttämä reititin, jolla aktivoidaan sivujen komponentteja. |
| cs-navbar | cs-navbar.js<br><br>(ei ole tehty Aurelialla) | Tämä on ylimmän menun komponentti ja se on oletuskomponenttina jokaisessa sivussa eli se näkyy kaikissa sivussa. |
| landing | landing.js<br>landing.css<br>landing.html | /homepage-osoitteen näkymä (komponentti)<br><br>Kotisivu eli oletussivu sovelluksessa. Komponentissa on ainoastaan teksti ja linkki tälle dokumentille. |
| dashboard | dashboard.js<br>dashboard.css<br>dashboard.html | /dashboard-osoitteen näkymä(komponentti)<br><br>Dashboard on sovelluksen kojelauta ja se sisältää sovelluksen kaikki toiminnallisuudet. Kohdassa 2.4 on lisätietoja komponentista.<br><br>Dashboard-komponentin tehtäviin kuuluu myös AppVM mallin alustus, sen rekisteröinti Interpreter-moduulille ja AppVM-mallin rekisteröinti FileReader-moduulille. |
| instructions | instructions.js<br>instructions.css<br>instructions.html | /instructions-osoitteen näkymä(komponentti)<br><br>Instructions on ohjesivu, jossa on yleistietoa sovelluksesta ja ohjeet käyttölliittymän käyttämiseen. |
| dashboard-nav | dashboard-nav.js<br>dashboard-nav.css<br>dashboard-nav.html | Komponentti käytetään säiliönä Terminal-otsikolle ja komentoliittymänäkymän suurennus toiminnan kuvakkeelle. |
| console | console.js<br>console.css<br>console.html | Interaktiivinen komentoliittymä komentojen kirjoittamiseen ja selaamiseen.<br><br>Alikappaleessa 2.4 on tarkat kuvaukset |

| | | |
|---|---|---|
| | | komentoliittymän komennoista. |
| logs-view | logs-view.js<br>logs-view.css<br>logs-view.html | Komponentti pitää sisällään logit-näkymän kokonaisuuden. Logit-näkymässä voi lukea sovelluksen logeja ja myös agenttien kirjaamia logeja.<br><br>Hakee logit AppVM-mallin instanssista. |
| files-view | files-view.js<br>files-view.css<br>files-view.html | Tiedostot-näkymä on sovelluksen tiedostojen hallintanäkymä. Näkymässä voi selata ladattuja tiedostoja, nähdä niiden tietoja ja poistaa ne.<br><br>Hakee tiedostot AppVM-mallin instanssista. |
| agents-view | agents-view.js<br>agents-view.css<br>agents-view.html | Agentit-näkymä on agenttien hallintapaneeli. Näkymässä voi selailla luotuja agentteja, lukea niiden tiloja, tunnuksia, aikatietoja ja nimet.<br><br>Hakee agentit AppVM-mallin instanssista. |
| text-field | text-field.js<br>text-field.css<br>text-field.html | Itsenäinen tekstikenttä, jota käytetään sekä komentoliitymässä että muissa komponenteissa.<br><br>Sirtää kenttään kirjoitetun tekstin value-attribuutilla. |
| terminal-line | terminal-line.js<br>terminal-line.css<br>terminal-line.html | Terminal-line on komponentti, joka edustaa yhtä logia tai viestiä. Se pitää sisällään viestin merkkausvärin, sisällön, tunnuksen, ajan ja suoritustilan. |
| combobox-tag | combobox-tag.js<br>combobox-tag.css<br>combobox-tag.html | Itsenäinen geneerinen komponentti, jossa on tekstikenttä ja mahdollisuus kirjoittaa tägejä. Käytetään suodatusta varten. |
| file-upload | file-upload.js<br>file-upload.css<br>file-upload.html | Komponentti tiedostojen lataamiseen. Komponenttiin on mahdollista syöttää tiedostotyyppirajoituksia. |
| agent-box | agent-box.js<br>agent-box.css<br>agent-box.html | Agent-box-komponentti edustaa agenttia agentit-näkymässä. |
| combobox | combobox.js<br>combobox.css<br>combobox.html | Combobox on combobox-tag tyylinen komponentti, jossa on tekstikenttä ja valikko.<br><br>Sovellus käyttää tätä komponenttia agentin tilan näyttämiseen. |

*Taulu 3: Sovelluksen näkymäkomponentit*

**view-model-provider.js**

Moduuli tarjoaa (export) luokan nimeltä ViewModelProvider. Luokan tehtävä on tarjota getAppVM-funktion, joka palauttaa AppVM-luokan instanssin. Moduuli luo uuden instanssin ja lisää muuttujalle appVM, mikäli sellaista ei ole tehty. Dashboard-komponentti käyttää tätä moduulia AppVM-luokan instanssin luomiseen.

**app-vm.js**

Tämä moduuli edustaa sovelluksen näkymän mallia tarjoamalla AppVM-luokan. Se pitää sisällään sovelluksen tilan, välttämättömät sovelluskohtaiset toiminnot ja muuttujia väliaikaisille tapahtumille. Kojelauta luo jokaista istuntoa varten yhden AppVM-luokan instannin ja kiinnittää instanssin muutujat lapsikomponenttien attribuutteihin. Seuraavassa taulussa on moduulin tarkka kuvaus.

| **Muuttuja** | **Kuvaus** | **Metodi** | **Kuvaus** |
|---|---|---|---|
| agents : Array | Säilyttää luodut agentit (Agent-instanssit). | constructor : void | Alustaa luokan alustamalla _appVMAPi muuttujan ja luomalla intervallin havaintojen tarkistulle. |
| logs : Array | Säilyttää luodut logit (Log-instanssit) | destroy() : void | Tuhoaa intervallin. Tarvitaan testeissä, muuten testit eivät sulkeudu. |
| files : Array | Säilyttää ladadut tiedostot (Files-instanssit) | registerFileReader (fileReader: CustomFileReader) : void | Rekisteröi tiedostolukijan eli tallentaa viittauksen fileReader-muuttujalle. |
| fileReader : CustomFileReader | Säilyttää viittauksen CustomFileReader-instanssiin. | createAgent (type: number, file: File) : void | Luo agentin annetulla tyypillä ja ehto-toiminta säännön tiedostolla. Tyyppi 0-3 viittaa seuraaviin agentteihin: refleki, mallipohjainen refleksi, tavoitepohjainen ja hyödyllisyypohjainen. |
| terminalLines : Array | Säilyttää komentorivin viestit (TerminalLine-instanssit) | log (message: string, producerId: number, status: Object) : void | Luo login annetulla viestillä, agentin tunnuksella ja status-objektilla. |
| perpceptions : Array | Säilyttää havainnot Array-muodossa seuraavien muuttujien arvoilla [id, target, value]<br><br>Esim. [“1”, “tempterature”, “22”] | createTerminalLine (text: string, isReponse: boolean) : void | Luo komentoliittymän rivin/viestin annetulla viestillä ja isResponse-kontrollilla. IsResponse viittaa siihen, että onko rivi vastaus edelliseen viestiin vai ihan oma rivinsä. |
| processedPerceptions: Array | Siirtää käsiteltyt havainnot tähän tauluun. | | |

| | | | |
|---|---|---|---|
| _appVMApi: Object | Objekti, joka pitää sisällään viittauksen AppVM-luokan funktioihin ja muuttujiin. Annetaan agentille ja logille tarvittaessa.<br><br>Objekti sisältää viittaukset deleteAgent- ja log-funktioihin ja viittaukset logs- ja fileReader-muuttujiin. | | |
| _perceptionCheckInterval : Interval | Muuttuja, joka säilyttää intervallin, jolla säännöllisesti tarkistetaan saapuneet havainnot perceptions-taulukkoon. | | |

*Taulu 4: AppVM-luokan muuttujat ja funktiot*

**agent-vm.js**

Agent-luokka edustaa sovelluksen tekoälyagenttia ja se enkapsuloi kaikki agentin toiminnallisuudet, ominaisuudet ja muuttujat, joita on mainittu ylhäällä olevissa alikohdissa. Se vastaanottaa havaintoja, käsittelee havaintoja ja osaa kertoa valitsemansa toimintoja kirjoittamalla logeja. Seuraavassa taulussa on Agent-luokan muuttujat ja metodit kuvauksineen.

| **Muuttuja** | **Kuvaus** | **Metodi** | **Kuvaus** |
|---|---|---|---|
| id : number | Agentin tunnus. Tällä tunnistetaan agentti. | constructor : void | Alustaa agentin. |
| name : string | Agentin nimi. Alkaa Agent_ ja satunnaisella numerolla. | changeState (state:string):void | Pyytää agentti siirtmään toiseen tilaan. |
| type : string | Tyyppi (arvo 0-4) | addPerceptions(perceptions: array) : void | Lisää havaintoja. |
| ruleActionList: Array | Ehto-toiminta säännöt muodossa [id, target, [rule(2-3)], [toiminnat]] | runAgentProgram() : void<br><br>runReflexAgentProgram<br>runModelReflexAgentProgram<br>runGoalAgentProgram<br>runUtilityAgentProgram | Ajaa ensimmäisen funktion runAgentProgram ja sitten siitä kutsuu agentinohjelman mukaisen funktion.<br><br>Varsinainen agenttiohjelma. |
| perceptions: Array | Havainnot, joita ei ole käsitelty vielä. | executeOperation (perpcetions: Array: rule: Array) : boolean<br><br>executeOperationForReflexAgent<br>executeOperationForModelReflexAgent | Suorittaa operaation <, > tai = annetulle arvolle. Jos esim. Havainnossa on 22 astetta ja säännössä on <22, niin se katsoo onko havainnon 22 aste pienempi kuin annettu <22. |
| processedPerceptions : Array | Käsiteltyjä havaintoja. | parseRules (rules: string) : boolean | Jäsentää säännöt. Palauttaa true, jos jäsennys sujuu |

| | | | |
|---|---|---|---|
| | | | ongelmitta, muuten false. |
| state : string | Tila, jossa agentti on. | reportInvalidAttributeValue (e: string) : void<br><br>agentNotInitialized | Kirjoittavat login joko väärästä attribuutin arvosta tai siitä, että agentti ei ole vielä alustettu. |
| startTime : string | Agentin ohjelman käynnistysaika. | Delete : void | Poistaa agentin. |
| endingTime : string | Aika, jolloin agentti lopetti toimintansa tai pysähtyi. | getFormattedTime(date : Date) : string | Palauttaa ajan formatoidussa muodossa. |
| file : File | Agenti ehto-toiminta sääntöjen tai tavoitteen tiedosto. | IsSmaller (value: string, compValue: string)<br><br>isLarger<br>isEqual | Vertaa arvot nimen mukaisesti ja palauttaa true tai false. |
| appVMApi : object | An object which contains references to AppVM functions and properties. | parseActionValue (value: string) : string or Array<br><br>parseRuleValue (value: string) : number, string or Array<br>parseTargetValue (value: string) : number<br>parseIdValue (value: string) : number | Jäsentää kyseisen (nimen mukainen) kohdan ehto-toiminta säännöstä. |
| agentProgramState : string | Idle tarkoittaa, että se on seissyt.<br><br>Busy tarkoittaa, että se edelleen pyörii. | isRulesValid (rules: Array) : boolean | Purkaa rules-taulun ja jäsentää säännöt. Palauttaa true jos jäsennys sujuu oikein. |
| executionCycleInterval : Interval | Muuttuja, joka pitää agenttiohjelan intervallin. | containsElement (value: string, arr: Array) : boolean | Tarkistaa onko arr-taulussa value-muttujan arvo. |

*Taulu 5: Agent-luokan metodit ja funktiot.*

**interpreter.js**

Tulkkaaja on itsenäinen moduuli, joka tarjoaa itsenäisen Interpreter nimisen luokan. Luokan vastuuseen kuuluu komentoliittymän komentojen tarkistus ja tulkkaaminen. Se on rakenteeltaan yksinkertainen, eikä sen sisäinen jäsennin kuluta paljon resursseja.

| **Muuttuja** | **Kuvaus** | **Metodi** | **Kuvaus** |
|---|---|---|---|
| agentTypes : Array | Sisältää sallitut eli tuetut tekoälyagenttiluokat agenttiohjelman mukaan. | constructor : void | Alustaa agentin. |

| | | | |
|---|---|---|---|
| | Sallitut arvot ovat reflex, model-reflex, goal ja utility. | | Alustaa commandFunctions ja commandValidators tyhjillä Map-instansseilla. Rekisteröi molemmille tuettavien komentojen funktiot ja validaattorifunktiot. |
| supportedInstructions : Object | Objekti, joka sisältää tuetut komenot ja sen osat. Jokainen komento voidaan jakaa ohjelmaan, argumentteihin ja asetuksiin.<br><br>Objektilla on kolme muuttujaa, commands, arguments and options. Jokainen näistä pitää nimensä mukaiset sallitut arvot. | setAppVM(appVM: AppVM) : void | Tallentaa AppVM-instanssin viittauksen appVM-muuttujaan. |
| commandFunctions<br>commandValidators | JavaScript Map-luokan instansseilla alustetaan ja niihin rekisteröidään ohjelman funktiot ja validointifunktiot. | interpret (text: string) : Array | |
| eventAggregator : EventAggregator | Aurelian moduuli moduulien välisten viestin lähettämiseen ja vastaanottamiseen, sekä tapahtumien hallintaan. | isRunInstructionValid (text: string) : Object<br><br>isGenerateInstructionValid<br>isUploadInstructionValid<br>isCreateInstructionValid | Ottaa vastaan tekstin, jossa on komento ja tarkistaa onko komennon syktaksi oikein. Palauttaa objektin, jossa on response ja errorMessage-muuttujat. |
| appVM | AppVM-luokan instannsi, jota kojelauta-komponentti on luonut. | isInstructionValid (text: string, isArgumentsOptional: boolean, isOptionsOptional: boolean) : boolean | Funktio, jota on tehty create- ja upload-komennolle. Tekee synktaksitarkistuksen ja palauttaa true tai false-arvon. |
| | | hasValidCommand (text:string) : boolean<br><br>hasValidArguments<br>hasValidOptions | Tarkistaa komennon, argumenttien tai asetusten oikeellisuuden. |
| | | ReadCommand (text:string) : string<br><br>readArguments<br>readOptions | Palauttaa sen komennon osan, jota funktion nimi koskee. |

| | | | |
|---|---|---|---|
| | | generate (params: Array) : void<br><br>create<br>uploadFile | Suorittaa komentoa koskevan toiminnon.<br><br>Generate luo havainnon.<br>Create luo agentin.<br>Lataa komennon. |
| | | runFunction(fn: Function, params: Array): string | Yleinen funktio, joka kutsuu annetun funktion annetulla parametreilla.<br><br>Palauttaa tyhjän tekstin, jos funktion ajaminen onnistuu, muuten palauttaa virheviestin. |

*Taulu 6: Tulkkaaja ja sen muuttujien ja funktoiden kuvaus*

**custom-file-reader.js**

CustomFileReader-luokka on tiedostojen lukemista varten kehitetty itsenäinen moduuli. Se sisältää pelkästään *readFile* ja *isFileTypeValid* funktiot. *ReadFile*-funktio vastaanottaa File-tyyppisen argumentin, jota se lukee ja sitten palauttaa rivinä. Funktio palauttaa objektin, jossa on response-muuttuja, joka sisältää tiedostosta luetut rivit ja errorMessage-muuttujan, joka kertoo tapahtuiko tiedoston lukemisessa virhe. *isFileTypeValid*-funktio tarkistaa, onko tiedostonpääte .txt.

**log-vm.js**

Log-luokka edustaa käyttöliittymässä ja logit-näkymässä esiintyvää logia.

| **Muuttuja** | **Kuvaus** | **Metodi** | **Kuvaus** |
|---|---|---|---|
| text : string | Login viesti. Oletusarvo N/A. | constructor (text: string, id: number, color: string, producer: object, status: object): void | Alustaa luokan ja asettaa arvot muuttujille. Mikäli argumentit ovat väärin tai tyhjiä, metodi asettaa oletusarvot muuttujille. |
| id: number | Login tunnus. | generateNewId : number | Palauttaa satunnaisen numeron. |
| color: string | Login väri style-attribuutille. | GenerateRandomColor: string | Palauttaa satunnaisen värin style-attribuutille sopivassa muodossa (rgba(…)). |
| time: Date | Aika, jolloin logi luotiin. | | |
| status: Object | Objekti, joka pitää sisällään tietoa siitä oliko logi alunperin | | |
| producer: Object | Mikäli logi on alunperin agentilta, producer-objektin id-muuttuja kertoo agentin | | |

| | | | |
|---|---|---|---|
| | tunnuksen. | | |

*Taulu 7: Log-luokka ja sen muuttujat ja funktiot kuvauksineen*

**main.js**

Tämä moduuli on automaattisesti generoitu tiedosto ja se tarjoaa konfiguroinnin Aurelialle. Se asettaa sovelluksen juureksi (root) app.js-tiedoston ja antaa polun index.js-tiedostolle, jossa sovelluksen komponentit ladataan sovellukselle niin että ne voidaan käyttää globaalisti. Vastaavasti **index.ejs** ja **index.html** tiedostot ovat Aurelian generoimia tiedostoja, jotka tarjoavat peruskonfigurointia.

# 4. Testaus

Web-sovelluksen testaus on suunniteltu ottaen huomioon sovelluksen kriittisimmät toiminnallisuudet. Testattavien kohteiden painopiste on niissä komponenteissa ja palveluissa, joista sovellus ja sen logiikka ovat eniten riippuvaisia. Tämän projektin testausstrategia jakaa testit kahteen eri kategoriaan. Ensimmäisessä kategoriassa on komponenttien, moduulien, palvelujen, luokkien ja muiden osien yksikkötestejä. Toisen kategorian testit ovat E2E-testejä, joilla verifioidaan web-sovellusta: varmistetaan, että sovellus toimii juuri niin kuin se on tässä dokumentissa suunniteltu.

## 4.1 Yksikkötestaus

Web-sovelluksen yksikkötestit on jaettu kahteen eri luokkaan: komponenttien yksikkötestit ja kaikki muiden luokkien ja osien yksikkötestit. Komponenttien yksikkötestit ovat kirjoitettu Aurelian ohjeilla ja kirjastoilla käyttämällä StageComponent menetelmää. Näissä testeissä komponentin tiedosto annetaan Aurelian testi-kirjastolle, joka lataa se vasta luotuun applikaation missä on vain ainoastaan testattava komponentti ladattu DOM:ille. Komponenttien yksikkötestauksissa testataan attribuutit, funktiot tai toiminnallisuudet. Yksikkötestit löytyvät /test/unit-kansiosta.

**App.spec.js**-tiedosto sisältää app-komponentin renderöintitestin. Testissä tarkistetaan, että komponentti latautuu oikein vamistamalla *.app_container* nimisen elementin latautumista vain yhden kerran.

**Logs-view.spec.js**-tiedosto sisältää logs-view-komponentin yksikkötestit. Testejä on yhteensä kaksi kappaletta, yhdessä tarkistetaan komponentin oikeanlainen renderöinti ja toisessa suodattimen toiminnallisuus. Renderöintitestissä komponentti ladataan ja sen jälkeen tarkistetaan, että container-elementti ja console-komponentti ovat myös latautuneet. Näiden molempien olemassaolo vahvistaa komponentin renderöinnin oikeellisuuden. Suodattimen testissä testataan koko suodatinprosessin toiminnallisuus: syötetään testisuodatustekstit ja tarkistetaan, että oikeat logit tulevat näkyville. Se tehdään niin että testissä lisätään kaksi keinotekoista logia ja tarkistetaan suodattimen toiminnallisuus käyttäen *Agent* ja *Text* suodattamia.

**Files-view.spec.js**-tiedosto sisältää files-view-komponentin yksikkötestit logs-view.spec.js-testien rakenteella. Ensin tarkistetaan komponentin renderöintiä ja sen jälkeen valitaan toiminto, jonka oikeanlainen toimivuus vaatii kaikkien funktioiden toimivuutta. Files-view-komponentin kohdalla tämä toiminnallisuus on tiedostojen tulostus näkymään. Testissä luodan tiedosto, jota syötetään komponentille, jonka jälkeen varmistetaan että komponentti tulostaa sekän tiedoston että sen tietoja oikein tiedostot-näkymälle.

**Agents-view.spec.js**-tiedosto sisältää agents-view-komponentin yksikkötestit logs-view.spec.js-testien rakenteella. Testissä luodaan ehto-toiminta sääntöjen tiedosto, ladataaan se sovellukselle ja sen jälkeen luodaan testiagentti, jonka tiedot tarkistetaan agentit-näkymästä, johon se lopulta ilmestyy viimeistään kahden sekunnin viiven jälkeen.

**Combobox.spec.js**-tiedosto sisältää combobox-komponentin yksikkötestit. Tiedoston yksikkötestit on jaettu kolmeen eri luokkaan: attribuutteihin, elementteihin ja funktioihin. Attribuuttitesteissä tarkistetaan komponentin attribuuttien toiminnallisuus. Tiedoston ainoa elementtitiedosto tarkistaa nuoli alas-kuvakkeen toimivuuden eli tarkistetaan, että kuvake avaa combobox-komponentin valikon ja sulkee sen. Funktioiden yksikkötestit tarkistavat funktoiden toimivuutta syöttämällä testi argumentteja ja tarkistamalla niiden toiminnallisuudet.

**Console.spec.js**-tiedosto on console-komponentin yksikkötestitiedosto. Se sisältää neljä yksikkötestiä, josta ensimmäinen muiden yksikkötestien tapaan tarkistaa renderöinnin oikeellisuuden. Attribuutin yksikkötesti testaa terminalLines-attribuutin toimivuutta syöttämällä testi TerminalLine-instansseja ja tarkistamalla niiden esiintyvyyttä näkymään. *_enterPressed* ja *_deleteTerminalLine* funktoiden testeissä tarkistetaan näiden funktoiden toimivuus.

**Custom-file-reader.spec.js**-tiedosto sisältää yksikkötestit CustomFileReader-luokalle, sen muuttujille ja funktoille. Ensimmäinen yksikkötesti tarkistaa readFile-funktion toimivuutta varmistamalla, että se lukee kaikki rivit testitiedostosta. Toisessa testissä verrataan funktion puretut rivit testitiedoston riveihin ja kolmannessa testissä varmistetaan, että luettavan tiedoston tiedostopääte on .txt.

**Dashboard.spec.js**-tiedosto kuuluu nimensä mukaisesti dashboard-komponentille. Renderöintitestin lisäksi se testaa runInterpreter-funktiota luomalla testi komennon ja tarkistamalla runInterpreter-funktion tulosta. Lisäksi testataan expandMode-attribuuttia laajentamalla ja pienentämällä komentoliittymän näkymää.

**Interpreter.spec.js**-testit ovat kirjoitettu testaamaan Interpreter-luokan muuttujia ja funktioita. Yksikkötestit on jaettu kolmeen eri kategoriaan: muuttujien testeihin, funktoiden testeihin ja tulkkauksen standardien testeihin. Muuttujien testeissä tarkistetaan, että muuttujat sisältävät oikeita nimiä ja standardin mukasia vakioita. Funktoiden testeissä testataan luokan funktioiden palautusarvoja ja toimintoja. Tulkkauksen standarditesteissä tarkistetaan standardikomentoja ja niiden tuottamia tuloksia.

**Terminal-line.spec.js**-testit liittyvät terminal-line-komponenttiin. Ensimmäisessä testissä testataaan komponentin latautumista DOM:ille. Toisessa testissä varmistetaan, että TerminalLine-komponentti tulostaa text-muuttujan sisällön oikein. Kolmannessa testissä testataan ajan tulostumista oikein ja neljännessä testissä testataan olion poistamista.

**Text-field-spec.js**-tiedoston testit koskevat text-field-komponentia. Testeissä käytetään combobox.spec.js-rakennetta.

**test-utils.js** on toisesta projektista tuotu suppea apukirjasto, joka nopeattaa testien kirjoittamista.

## 4.2 End-to-end testaus

Yksikkötestit ovat pääosin moduulien testausta, jossa testataan annettuja rajapintamäärittelyjä. Monissa yksikkötestissä testin laajuus rajoittuu funktioon tai luokkaan, joka ei ole riittävä läpäisemään vaatimuksia testaussunnitelmissa. Tästä syystä sovelluksen toimivuus testataaan myös sovellustasolla. E2E-testit on kirjoitettu dashboard.js-tiedostolle ja upload-komponentille ottamalla huomioon tämän dokumentin vaatimukset.

**Upload.e2e.js**-tiedostossa on kirjoitettu e2e-testiä, jolla testataan upload-komponenttia. Testissä käytetään apuna *cypress-file-upload*-kirjastoa tiedoston viemiseen upload-komponentille. Tiedosotn ainoa e2e-testi luo testitiedoston ja lataa se komponentille. Sen jälkeen testi varmistaa, että tiedosto ilmestyy tiedostot-näkymään.

**Dashboard.e2e.js**-tiedosto sisältää sovelluksen monipuolisimmat ja kattavimmat testit. Tässä tiedostossa kirjoitettujen testien päätavoite on verifoida sovelluksen toimivuus ja että sovellus on tehty vaatimusten mukaisesti. Testit on jaettu kolmeen ktegoriaan: agenttien luomistesteihin, agenttien ajamistesteihin ja agenttien oikeellisuustarkistusteseihin. Ensimmäisen kategorian testeissä testataan, että sovelluksella voidaan ylipäätään luoda agentteja. Toisen kategorian testit testaavat agentin ohjelman toimiivuutta ajamalla agenttiohjelmat. Kolmannen kategorian testit vertaavat agenttien reaktiot testihavaintoihin.

# 5. Kirjallisuusviitteet

[1] Montoya, Joseph H., Kirsten T. Winther, Raul A. Flores, Thomas Bligaard, Jens S. Hummelshøj, and Muratahan Aykol. "Autonomous intelligent agents for accelerated materials discovery." Chemical Science 11, no. 32 (2020): 8517-8532.

[2] Suarez, Joseph, Yilun Du, Phillip Isola, and Igor Mordatch. "Neural MMO: A massively multiagent game environment for training and evaluating intelligent agents." arXiv preprint arXiv:1903.00784 (2019).

[3] Liang, Chih-Chin, Wen-Yau Liang, and Tzu-Lan Tseng. "Evaluation of intelligent agents in consumer-to-business e-Commerce." Computer Standards & Interfaces 65 (2019): 122-131.

[4] Sayedahmed, H. A. M. "Intelligent Agent approaches for Internet of Things Challenges: A Review." Int Rob Auto J 4, no. 1 (2018): 00092.

[5] Wooldridge, Michael. "Intelligent agents." Multiagent systems 6 (1999).

[6] Norvig, P. Russel, S. Artificial Intelligence. A modern approach. Pearson; 4th edition (2002).

[7] Aurelia, What is Aurelia?, https://aurelia.io/docs/overview/what-is-aurelia, 2021.