Tahiti — tähtitieteellisten havaintojen tietokanta

Tomi Hänninen
Juho Muhonen
Ismo Puustinen
Kai Pääsky
Pekka Simola
Nuutti Varis

Helsinki 12.5.2003

Testaussuunnitelma

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta/Osasto — Fakultet/Sektion — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author

Tomi Hänninen Juho Muhonen Ismo Puustinen Kai Pääsky Pekka Simola Nuutti Varis

Työn nimi — Arbetets titel — Title

Tahiti — tähtitieteellisten havaintojen tietokanta

Oppiaine — Läroämne — Subject

Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Testaussuunnitelma	12.5.2003	45 sivua

Tiivistelmä — Referat — Abstract

Testaussuunnitelma on olennainen osa ohjelmistotuotantoprojektin dokumentaatiota. Testauksen suunnitelma sisältää sekä moduuli–, integraatio– että järjestelmä– eli validointitestauksen hahmottelut. Moduulitestauksessa jokainen moduuli testataan ilman muiden moduuleitten läsnäoloa. Integraatiotestauksessa yhdistetään saman kokonaisuuden moduulit ja testataan niiden yhteistoiminnallisuus. Validointitestauksessa eri kokonaisuudet yhdistetään ja yhtenäinen järjestelmä testataan liittymiensä kautta.

Uusin versio tästä dokumentista on saatavilla Tahiti-ryhmän kotisivuilla osoitteessa `http://www.cs.helsinki.fi/group/tahiti/` .

Versiohistoria:

1. Versio 1.0 (katselmoitu versio)
2. Versio 1.1 (Korjattu versio)

Avainsanat — Nyckelord — Keywords

Tahiti, fotometria, Standard Asteroid Photometric Catalogue

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Versio 1.1

Sisältö

1 Johdanto

Tämä dokumentti on suunnitelma Tahiti–projektin tuottaman ohjelmiston testaamista varten. Dokumentissa kuvataan kaikki ohjelmiston testaamiseen käytettävät testaustapaukset, vaikkakaan tarkkoja testiaineistoja dokumenttiin ei kirjoiteta. Testitapauksista kirjotetaan ohjelmiston oletettava käyttäytyminen kussakin tapauksessa.

Toisessa luvussa kuvataan yleinen testausstrategia, sekä perustellaan joitain tehtyjä päätöksiä. Luvussa kolme käsitellään moduulitestausta, luvussa neljä integraatiotestausta ja viimeisessä, viidennessä, luvussa validointitestausta.

2 Yleistä testauksesta

Testauksen tavoitteena on saada poistettua tuotettavasta ohjelmasta mahdollisimman monta virheitä, jotta toimitettava ohjelma olisi mahdollisimman täydellinen. Testaukselle on varattu projektisuunnitelman mukaan aikaa 3 viikkoa, jona aikana projektiryhmällä on myös muita tehtäviä, kuten esimerkiksi käyttöohjeen tekoa.

Testauksesta pyritään tekemään mahdollisimman kattava käytössä olevien resurssien valossa. Moduulitestauksessa jokainen ohjelmiston metodi testataan lausekattavasti tietyin varauksin. Näihin poikkeamiin lausekattavuudesta palataan modulitestauksen yhteydessä. Kaikki tehdyt testit dokumentoidaan vähintään siten, että jokainen testauksessa käytetty testiluokka on saatavissa.

Toisessa testivaiheessa, integraatiotestauksessa, testataan järjestelmän eri komponenttien yhteistoimintaa. Integraatiotestausta suoritetaan vain Tahiti–Apille, ja Apin toiminnot toteuttaville käsittelijäluokille. Integraatiotestauksen päätavoitteena onkin varmistaa, että Tahiti–Apin eri luokat ja metodit toimivat oikein yhteen. Samalla testausaineistoa pyritään jakamaan ekvivalenssiluokkiin, jotta mahdollisimman moni virhe huomattaisiin. Testiaineiston jakoon palataan integraatiotestauksen yhteydessä.

Testauksen viimeisessä, kolmannessa, vaiheessa tarkistetaan että järjestelmä toimii määrittelyn mukaisesti. Tätä viimeistä testausvaihetta kutsutaan validointitestaukseksi. Validointitestauksen yhteydessä testataan siis järjestelmää kokonaisuutena, ja tarkistetaan että järjestelmä toimii oletetulla tavalla. Myös järjestelmätestauksessa testausaineisto jaetaan ekvivalenssiluokkiin mahdollisten virhetilanteiden löytämiseksi.

Testauksesta siis ensimmäinen vaihe, moduulitestaus suoritetaan ns. White–box–testauksena, jossa testiaineisto muodostetaan ohjelmakoodin mukaisesti. Kaksi seuraava vaihetta, integraatiotestaus sekä järjestelmätestaus suoritetaan nk. Black–box testauksena, jossa järjestelmälle annetaan erilaisia syötteitä, ja tarkastellaan saatuja tuloksia. Black–box –testauksessa testitapausten suunnittelu perustuu järjestelmän oletettuun toimintaan. Käyttöliittymästä on testattu pelkästään niinsanotut Bean–luokat, joiden avulla käyttöliittymä siirtää tietoa apin ja itsensä välillä.

Moduulitestaus toteutetaan itsenäisesti luokkien toteuttajien toimesta, eikä moduulitestausta varten pystytetä erillistä testausympäristöä. Myös integraatiotestaus suoritetaan itsenäisesti, joten siihenkään ei erillistä testausympäristöä tarvita. Järjestelmätestaus sen sijaan suoritetaan yhteisessä testausympäristössä, joka ryhmälle perustetaan koneelle db.cs.helsinki.fi. Testauksessa käytetään

Java versiota 1.4.1_01.

3 Moduulitestaus

Moduulitestaus tullaan suorittamaan järjestelmässä ns. ”White-Box”testauksena, lausekattavasti. Testauksessa ei kuitenkaan pyritä täydelliseen lausekattavuuteen, vaan lausekattavuutta pidetään vain eräänlaisena ohjenuorana. Käytännössä tämä tarkoittaa sitä, että metodit testataan lausekattavasti, mutta muiden metodien tuottamiin virheisiin ei tarvitse varautua.

Rajallisten resurssien vuoksi moduulitestaus on jokaisen luokan kehittäjän vastuulla, sillä metodien rakenne on oletettavasti toteuttajille kaikista selvin.

3.1 AsteroidDataBean

AsteroidDataBean–luokan metodit testataan erillisellä JUnit–testiluokalla. Luokan sisältö talletetaan muuttujaan, jonka arvo voi olla null. Tällöin luokalla ei ole sisältöä.

- *returnData()* Perustapauksena voidaan pitää AsteroidDataBean ilmentymää, jolla on joitain asteroideja. Tällöin metodi palauttaa kaikki kyseisen bean–luokan asteroidit taulukossa. Jos beanilla ei ole mitään sisältöä, metodi palauttaa null–viitteen.
- *returnData(long asteroidId)* Perustapauksena voidaan olettaa, että bean–luokan sisällöstä löytyy kysieisellä asteroidId–parametrilla ilmentymä, jolloin kyseinen ilmentymä palautetaan.

  Tämän lisäksi on testattava tapaukset:

  1. *AsteroidId:tä ei löydy beanista* Tällöin metodi palauttaa null viitteen. Testaustilanteen voi järjestää helposti luomalla asteroiId:n, jota järjestelmästä ei löydy, esimerkiksi negatiivisen arvon.
  2. *Bean–instanssilla ei ole sisältöä* Tällöin metodi palauttaa null viitteen. Tapaus voidaan järjestää luomalla uusi ilmentymä AsteroidDataBean–luokasta ja kutsumalla tätä metodia jollakin parametrilla.
- *getAsteroidName(String aid)* Metodin on tarkoitus palauttaa parametrin aid ilmoittaman asteroidin määritelty nimi. Nimi voi olla neljää eri muotoa: <numero> <nimi> (<tunnus>), <numero> <nimi>, <numero> (<tunnus>), (<tunnus>). Parametrista aid joudutaan hakemaan asteroidin numero käyttämällä Long.parseLong()–metodia.

  Perustapauksena on epätyhjä ilmentymä AsteroiDataBean–luokasta sekä parametrin aid avulla sisällöstä voidaan löytää asteroidi, jolla on sekä nimi että tunnus. Tällöin metodi palauttaa yllä olevista vaihtoehdoista ensimmäisen.

  Lisäksi on testattava seuraavat tapaukset:

  1. *Parametri aid on null–viite.* Parametrin aid ollessa null–viite, metodi palauttaa käyttäjälle tyhjän merkkijonon ””. Tilanne voidaan helposti luoda kutsumalla metodia null–viitteellä.

2. *Bean–instanssilla ei ole sisältöä.* Kuten yllä, jos bean–instanssilla ei ole sisältöä, metodi palauttaa käyttäjälle tyhjän merkkijonon "". Tilanne voidaan luoda tekemällä uusi instanssi AsteroidDataBeanista alustamatta sitä millään tavoin.
3. *Parametrin aid avulla löytyy asteroidi, jolla on vain nimi.* Metodin tulisi palauttaa merkkijono, joka on muotoa <numero> <nimi>. Tilanne voidaan luoda tekemällä yksi testiasteroidi, jolle ei aseteta tunnusta.
4. *Parametrin aid avulla löytyy asteroidi, jolla on vain tunnus.* Metodin tulisi palauttaa merkkijono, joka on muotoa (<tunnus>). Tilanne voidaan järjestää kuten yllä, mutta nimen sijaan asetetaan pelkkä tunnus.
5. *Annettu parametri aid ei ole muunnettavissa kokonaisluvuksi.* Tällöin metodi palauttaa tyhjän merkkijonon "". Tilanne muodostetaan kutsumalla metodia vaikkapa parametrillä "kissa".
6. *Annetulla parametrillä aid ei löydetä asteroidia.* Metodi palauttaa tällöin tyhjän merkkijonon. Tilanne voidaan luoda antamalla parametriksi aid negatiivinen kokonaisluku.

3.2 AsteroidQueryBean

AsteroidQueryBean–luokassa säilytetään käyttäjän antamat asteroidihakuparametrit. Luokka testataan erillisellä testiluokalla. Alla beaneille tyypillisten get- sekä set–metodien testaus on jätetty, koska ne ovat lausekattavasti testattuja jokaisella kutsukerralla, riippumatta syötteistä tai palautteesta. Muutamalla get metodilla on kuitenkin poikkeus tähän, getDetector():lla, getAbsolutePhotometry():llä sekä getLighttimeCorrected():lla, joista on testitapaukset alla. Tilannetta, jolloin mikään luokan kentistä olisi null–viite ei tule, koska konstruktori kutsuu luokan reset()–metodia, joka alustaa kaikki kentät ei-null–arvoiksi.

- *getDetector()* Perustilanteessa syötteenä setDetector()–metodille on taulukko, jossa ei ole sanaa "Other". Tällöin metodi palauttaa kyseisen taulukon takaisin muuttumattomana.

  Ainoa poikkeava tapaus on:

  1. *Detector–taulukossa on elementtinä "Other"* Tällöin metodi korvaa taulukon "Other" elementin luokan detectorOther kentän arvolla. Tilanne voidaan järjestää antamalla setDetector()–metodille parametriksi merkkijonotaulukko, joista yksi jäsen on "Other" ja antamalla setDetectorOther()–metodilla "Other" elementin korvaava teksti.

- *getAbsolutePhotometry()* Perustilanteessa getAbsolutePhotometry()–metodi palauttaa merkkijonon "false" kyseisen kentän ollessa beanissa tyhjä merkkijono "". Jos kyseinen kenttä ei ole tyhjä merkkijono, metodi palauttaa kentän arvon.

- *getLighttimeCorrected()* Perustilanteessa getLighttimeCorrected()–metodi palauttaa merkkijonon "false" kyseisen kentän ollessa beanissa tyhjä merkkijono "". Jos kyseinen kenttä ei ole tyhjä merkkijono, metodi palauttaa kentän arvon.

- *getValueArray(String field)* Metodin tarkoitus on palauttaa parametrin määrittelemän tekstimuotoisen kentän arvo. Perustilanteessa parametri field on merkkijono "detector", jolloin

metodi palauttaa kentän ”detector” arvot. Huomioitavaa on, että metodi ei käytä tähän yllä olevaa getDetector()–metodia.

Muita metodin testitapauksia ovat:

1. *Parametri field on eri kuin ”detector”.* Parametrin ”field” ollessa mitä tahansa muuta kuin null tai ”detector”, metodi palauttaa filters merkkijonotaulun. Testitapaus voidaan luoda helposti luomalla AsteroidQueryBean olio ja antamalla getValueArray()–metodille parametriksi ”kissa”.
2. *Parametri field on null–viite.* Parametrin ollessa null–viite metodi palauttaa tyhjän merkkijonotaulukon.

- *validate()* Validate–metodi tarkastaa muuttujissa olevan tiedon oikeellisuuden. Validointimetodi validoi tiettyjen arvojen oikeellisuuden. Tekstin yksinkertaistamiseksi alla on esitetty testiaineisto, jolla validointimetodi palauttaa epätosiarvon, kysely on siis väärä. Lausekattavuus saavutetaan luomalla syöte, jossa kaikki allaolevat luvut eivät ole lukuja, ts. kokonaisluku on merkkijono ”abc”. Erillistä testitapausta validaation oikeellisuudelle ei tarvita, koska lausekattavuuteen vaaditaan vain erilaisten väärien syötteiden tarkastelu.

  1. Asteroidin numero ei ole positiivinen kokonaisluku.
  2. Valokäyrien minimimäärä ei ole positiivinen kokonaisluku.
  3. Valokäyrän havaintopisteiden määrä ei ole nolla tai positiivinen kokonaisluku.
  4. Minimivaihekulma ei ole desimaaliluku väliltä 0-180.
  5. Maksimivaihekulma ei ole desimaaliluku väliltä 0-180.
  6. Minimimatka auringosta ei ole nolla tai positiivinen desimaaliluku.
  7. Maksimimatka auringosta ei ole nolla tai positiivinen desimaaliluku.
  8. Minimimatka maasta ei ole nolla tai positiivinen desimaaliluku.
  9. Maksimimatka maasta ei ole nolla tai positiivinen desimaaliluku.
  10. Minimilatitudi ei ole desimaaliluku väliltä 0-360.
  11. Maksimilatitudi ei ole desimaaliluku väliltä 0-360.
  12. Minimilongitudi ei ole desimaaliluku väliltä 0-360.
  13. Maksimilongitudi ei ole desimaaliluku.
  14. Havaintovälineeksi ei ole valittu ”Other” ja Otherille on annettu arvo.

- *returnErrorMsg(String field)* Perusoletuksna parametriksi field valitaan jokin beania vastaavan lomakkeen kentän nimi jonka arvo ei ole läpäissyt validointitarkistusta. Metodi palauttaa tällöin tekstimuotoisen virheilmoituksen kyseisestä kentästä.

  Lausekattavuus saavutetaan seuraavilla lisätesteillä:

  1. *Parametri field on null–viite.* Parametrin ollessa null–viite metodi palauttaa null–viitteen. Testi on helppo suorittaa kutsumalla metodia parametrina null–viite.
  2. *Parametriä field ei löydetä lomakkeen kentistä.* Parametrin ollessa kenttä, jota lomakkeessa ei ole, metodi palauttaa null–viitteen.

- *setApiError(int errornum)* Perusoletuksena parametri errornum on positiivinen kokonaisluku tai nolla, jolloin apin virhe liitetään muiden validoinnin yhteydessä tapahtuneiden virheiden joukkoon ja metodi palauttaa tosiarvon. Parametrin ollessa negatiivinen virhetietorakenteeseen lisätään ilmoitus virheellisestä apin virheestä ja metodi palauttaa epätosiarvon..

3.3 AsteroidSubmitDataBean

Alla beaneille tyypillisten get– sekä set–metodien testaus on jätetty, koska ne ovat lausekattavasti testattuja jokaisella kutsukerralla, riippumatta syötteistä tai palautteesta. Myös reset–metodi tulee lausekattavasti testatuksi jokaisella kutsumiskerralla.

generateFormColumns() GenerateFormColumns–metodin tarkoitus on alustaa formColumns-muuttuja. Se ei palauta mitään arvoa vaan kopioi pelkästään muuttujaviittaukset toisikseen.

generateColumns() GenerateColumns–metodi muuttaa formColumnsin apille kelpaavaan muotoon. Se palauttaa arvonaan String-taulukon, jossa ovat kaikki formColumnsin sisältämät arvot mapattuina TahitiLibraryn COL-arvoihin. Se testataan ajamalla se millä tahansa syötteellä ja vertaamalla mappauksia toivottuihin.

validate() Validate–metodin tarkoitus on palauttaa virhearvo, jos joku beanin muuttujista puuttuu tai on tyypiltään virheellinen. Tosiarvon palauttamiseen vaaditaan seuraavat ehdot:

formColumns ei saa sisältää kahta samaa numeroa, vaan kaikkien numeroiden pitää olla erillisiä. Samoin formColumns ei saa sisältää ei-numero-merkkijonoja.

data täytyy olla epätyhjä merkkijono.

Testaus suoritetaan ajamalla metodi oikealla syöttelllä ja syötteellä, jossa ainakin kahdessa taulukon indeksissä on sama arvo ja ainakin yksi indeksi sisältää ei-numeron, sekä testidata on tyhjä. Tuloksien pitäisi olla true ja false.

3.4 CustomElementSubmitBean

CustomElementsubmitBean–luokassa on sisällä käyttäjän antamat omavalintaiset ratatietoelementit. Luokka testataan erillisellä testiluokalla. Kuten yleensä, beanissa olevien get- sekä set–metodien testaus on jätetty, koska ne ovat lausekattavia jokaisella kutsukerralla. Sama voidaan sanoa konstruktorista sekä reset–metodista. Returnerrormsg–metodi on testattu AsteroidQueryBean–luokan testauksen yhteydessä, joten sen testiä ei tässä ole myöskään tehty.

- *validate()* Kuten AsteroidQueryBean–metodin yhteydessä, validate–metodi tarkastaa käyttäjän antamien kenttien arvojen oikeellisuuden. Lausekattavuus saavutetaan testaamalla luokka niin, että kaikkiin kenttiin annetaan virheellistä numerotietoa. Sen lisäksi lausekattavuuteen vaaditaan, että kentät testataan myöskin arvoilla, jotka eivät ole numeroita. Seuraavassa on listattu oikeellisuusvaatimukset:

  1. Kentän epoch arvon täytyy olla positiivinen desimaaliluku.

2. Kentän longitude tulee olla desimaaliluku väliltä 0-360.
3. Kentän axis tulee olla desimaaliluku.
4. Kentän anomaly tulee olla desimaaliluku väliltä 0-360.
5. Kentän argument tulee olla desimaaliluku väliltä 0-360.
6. Kentän eccentricity arvo tulee olla desimaaliluku väliltä 0-1.
7. Kentän inclination arvo tulee olla desimaaliluku väliltä 0-360.

3.5 EventHistoryDataBean

EventHistoryDataBean sisältää pelkästään get- ja set-metodin sekä reset-metodin, joten sen testaaminen on sivuutettu triviaalina.

3.6 Initializer

Initializer servlet–luokka alustaa järjestelmän alkutilaan ja palauttaa käyttäjän takaisin hänen valitsemalleen sivulle. Servlet myös varmistaa että käyttäjällä on hänen internet–selaimessaan piparit päällä, joita tarvitaan käyttäjän tunnistamiseen yksikäsitteisesti järjestelmälle.

- *doGet(HttpServletRequest request, HttpServletResponse response)* Perustestissä oletetaan seuraavien väitteiden pitävän paikkaansa:

  1. Käyttällä on piparit päällä selaimessa, jolloin käyttäjälle on luotu sessio–ilmentymä.

  Lisäksi testataan seuraavat tapaukset, jolloin luokka on testattu lausekattavasti:

- *Käyttäjällä ei ole pipareita* Metodi palauttaa käyttäjälle sivun, joka mainitsee asiasta. Tilanne voidaan testata hylkäämällä järjestelmän selaimelle lähetettävä pipari.

- *Käyttäjällä on piparit päällä, mutta session–ilmentymässä ei ole ilmentymää TahitiApista.* Metodi luo käyttäjälle ilmentymän TahitiApista ja palauttaa käyttäjän takaisin normaalille jsp–sivulle. Tilanne tulee testatuksi joka kerta kun käyttäjä ensimmäistä kertaa lataa järjestelmän jsp–sivun.

- *Käyttäjällä on piparit päällä, mutta session–ilmentymässä ei ole attribuuttia baseQuery.* Metodi alustaa attribuutin baseQuery arvoksi false ja palauttaa käyttäjän takaisin normaalille jsp–sivulle. Tilanne tulee testatuksi kuten yllä.

3.7 LightcurveDataBean

LightcurveDataBean–luokassa säilytetään tietyn asteroidin kaikkia järjestelmän asteroidihaun yhteydessä löytämiä valokäyrätietoja. Tämän lisäksi luokassa on kaikki asteroidiin sekä maahan liittyvät ratatiedot. Luokkaan liitetään myös kaikki käyttäjän valokäyrille itse luomat ratatiedot. Metodi testataan erillisellä testiluokalla. Alla on testattu lausekattavasti vain metodit, joissa on haarautuvia polkuja. Yksisuuntaiset polut voidaan todeta lausekattaviksi kutsumalla niitä kerran.

- *findLightcurve(long lcid)* Perustestissä metodille annetaan arvoksi jokin valokäyrän numerotunniste, joka löytyy tästä luokan ilmentymästä. Tällöin metodi palauttaa kyseisen valokäyrän sijainnin taulukossa. Testi voidaan suorittaa luomalla instanssi tästä luokasta niin, että haluttu valokäyrän numerotunniste löytyy.

  Lisäksi metodi on testattava sellaisella valokäyrän tunnisteella, jota ei löydy järjestelmästä. Tällöin metodi palauttaa -1 ja on lausekattavasti testattu. Kyseinen testi voidaan luoda helposti testaamalla metodia negatiivisella arvolla.

- *insertCustomElement(ApiTrajectory ce, long lcId, boolean earthTrajectory)* Perustestinä metodi saa maan rataelementtitiedon olemassaolevalle valokäyrälle. Tällöin metodi palauttaa tosiarvon. Perustilanne voidaan simuloida luomalla uusi ratatietoluokan instanssi, antamalla tämän luokan sisällä olevien valokäyrien joukosta jonkin tunnus ja antamalla viimeiseksi parametriksi true.

  Lausekattavuuden saavuttamiseksi metodi on lisäksi testattava seuraavilla tapauksilla:

  1. Halutaan lisätä validille valokäyrälle asteroidin rataelementti. Metodi palauttaa tosiarvon. Testi voidaan luoda kuten yllä antamalla viimeiseksi boolean–parametriksi false.
  2. Haluttua valokäyrää ei löydy järjestelmästä. Metodi palauttaa epätosiarvon. Testi voidaan luoda antamalla valokäyrän tunnisteeksi negatiivinen luku.

- *changeTrajectoryInfo(long lcId, boolean earthTrajectory, long newTrajectoryType)* Perustestissä metodille annetaan arvoksi jokin ilmentymässä olevan valokäyrän tunnus jolle halutaan vaihtaa maan ratatiedoksi jokin viimeisen parametrin määräämä ratatietotunnus. Tällöin metodi palauttaa tosiarvon. Testi voidaan tehdä luomalla instanssi tästä oliosta ja antamalla metodille parametriksi jotkin oikeelliset valokäyrä- sekä ratatietotunnisteet.

  Lausekattavuus syntyy seuraavin lisätestein:

  1. *Valokäyrää ei löydy luokan ilmentymästä.* Metodi palauttaa epätosi. Testi voidaan luoda antamalla valokäyrän tunnukseksi negatiivinen kokonaisluku.
  2. *Valokäyrälle halutaan muuttaa asteroidin rataelementtityyppiä.* Testi palauttaa tosiarvon. Testin suoritus käy kuten perustapauksessa, parametrin earthTrajectory arvoksi asetetaan vain false.

- *getTrajectory(long lcid, boolean earthTraj)* Perustapauksessa metodille annetaan jokin luokan ilmentymässä olevan valokäyrän tunnus jonka maan ratatiedon kutsuja haluaa. Valokäyrälle on määritelty ilmentymässä maan ratatietotyypiksi 0 (”Uppsala”). Metodi palauttaa kutsujalle null–viitteen. Testi voidaan tehdä luomalla uusi instanssi luokasta jolle on annettu oikeelliset valokäyrä- sekä ratatiedot.

  Lausekattavuus varmistetaan testaamalla seuraavat lisäehdot:

  1. *Valokäyrää ei löydy ilmentymästä.* Metodi palauttaa null–viitteen. Tapaus voidaan testata antamalla metodilla negatiivinen kokonaisluku parametriksi lcid.
  2. *Kutsuja haluaa maan ratatiedon valokäyrälle, jonka ratatietotyyppi on ”Custom” (-1).* Metodi palauttaa null–viitteen. Testi voidaan toteuttaa antamalla kyseiselle valokäyrälle maan ratatiedoksi jokin omavalintainen ratatieto insertCustomElement–metodin avulla.

3. *Kutsuja haluaa maan ratatiedon valokäyrälle, jonka ratatietotyyppi on positiivinen kokonaisluku.* Käyttäjä haluaa tällöin jonkin järjestelmässä maalle olevan ratatiedon. Metodi palauttaa kyseisen ratatiedon. Testi voidaan suorittaa antamalla järjestelmälle ratatietoinformaatiota niin, että kyseisen valokäyrän getEarthTrajectoryID() palauttaa kyseisen ratatietoinformaation tunnuksen.
4. *Kutsuja haluaa asteroidin ratatiedon valokäyrälle, jonka ratatietotyyppi on "Uppsala" (0).* Metodi palauttaa null–viitteen. Testi voidaan toteuttaa kuten perustapaus.
5. *Kutsuja haluaa asteroidin ratatiedon valokäyrälle, jonka ratatietotyyppi on "Custom" (-1).* Metodi palauttaa null–viitteen. Testi voidaan toteuttaa antamalla kyseiselle valokäyrälle asteroidin ratatiedoksi jokin omavalintainen ratatieto insertCustomElement–metodin avulla.
6. *Kutsuja haluaa asteroidin ratatiedon valokäyrälle, jonka ratatietotyyppi on positiivinen kokonaisluku.* Käyttäjä haluaa tällöin jonkin järjestelmässä asteroidille olevan ratatiedon. Metodi palauttaa kyseisen ratatiedon. Testi voidaan suorittaa antamalla järjestelmälle ratatietoinformaatiota niin, että kyseisen valokäyrän getDefaultTrajectoryID() palauttaa kyseisen ratatietoinformaation tunnuksen.

- *getTrajectoryOptions(boolean earthTrajectories, int pos)* Peruskutsulla halutaan maan ratatietotekstit luokan ilmentymästä löytyvälle ensimmäiselle valokäyrälle. Testi palauttaa merkkijonon, joka on muotoa "Uppsala, <ratatietopäiväys>, <ratatietopäiväys>, Custom". Testi voidaan järjestää luomalla instanssi luokasta jolle annetaan oikeaa valokäyräinformaatiota, ja jonka datapisteille on määritelty "Uppsala"–arvot.

  Lausekattavuus saavutetaan seuraavin lisätestein:

  1. *Parametri pos on virheellinen.* Kutsujalle palautetaan tyhjä merkkijono. Testi voidaan helposti toteuttaa kutsumalla metodia negatiivisella pos–parametrilla.
  2. *Kutsuja haluaa maan ratatietotekstit valokäyrälle jolla on "Uppsala"–arvot.* Metodi palauttaa merkkijonon, joka on samaa muotoa kuin perustestitapaukesssa. Testi järjestää kuten perustestitapaus.

- *getTrajectoryOptionValues(boolean earthTrajectories, int lcid)* Metodin testitapaukset vastaavat täsmällisesti getTrajectoryOptions–metodin testitapauksia. Tekstin "Uppsala" tilalla on nyt luku nolla, "Custom"–tekstin tilalla on luku -1 ja normaalien ratatietojen luvut ovat kyseisen ratatiedon tunnus (1..n).

- *getTrajectoryType(long lcid, boolean earthTrajectory)* Perustestissä halutaan tietää järjestelmästä löytyvän valokäyrän maan ratatietotyyppi. Ratatietotyyppi voi olla kolmea eri tyyppiä (-1, 0, 1..n). Metodi palauttaa 0. Testi voidaan järjestää luomalla instanssi luokasta ja antamalla sille oikeellista valokäyrätietoa niin, että kyseisellä valokäyrällä on "Uppsala"–arvot. Tämän jälkeen kyseistä metodia kutsutaan.

  Tämän lisäksi lausekattavuuden takia metodi testataan seuraavin lisätestein:

  1. *Valokäyrää ei löydy järjestelmästä.* Metodi palauttaa nollan. Testi voidaan järjestää kutsumalla metodia negatiivisella lcid arvolla.

2. *Halutaan asteroidin ratatietotyyppi.* Metodi palauttaa halutun valokäyrän asteroidin tämän hetkisen ratatietotyypin. Testi voidaan järjestää kuten perustesti.

- *getLightcurve(long lcid)* Perustilanteessa metodia kutsutaan valokäyrän tunnuksella, joka löytyy luokan instanssista. Metodi palautta kyseisen valokäyrän instanssin.

  Lausekattavuus todetaan testaamalla metodi valokäyrätunnuksella, jota ei ole olemassa, esimerkiksi negatiivisella arvolla. Tällöin metodi palauttaa null–viitteen.

3.8 LightcurveQueryBean

Luokka testataan erillisellä testiluokalla. Alla beaneille tyypillisten get- sekä set–metodien testaus on jätetty, koska ne ovat lausekattavasti testattuja jokaisella kutsukerralla, riippumatta syötteistä tai palautteesta.

- *isSelected(long lc)* Perustapauksessa valokäyrä on valittuna. Metodi palauttaa tällöin tosiarvon. Testi voidaan järjestää antamalla metodille setLcid() metodin avulla taulukko, jossa on haluttu lc–parametrin arvo.

  Lisäksi metodi testataan lc arvolla, jota ei ole valittu. Testi voidaan järjestää valitsemalla lc:n arvoksi jokin arvo, jota setLcid() metodin avulla ei ole määritelty.

3.9 LightcurveSubmitBean

Alla beaneille tyypillisten get– sekä set–metodien testaus on jätetty, koska ne ovat lausekattavasti testattuja jokaisella kutsukerralla, riippumatta syötteistä tai palautteesta. Myös reset–metodi tulee lausekattavasti testatuksi jokaisella kutsumiskerralla.

generateFormColumns() GenerateFormColumns–metodin tarkoitus on alustaa formColumns-muuttuja. Se ei palauta mitään arvoa vaan kopioi pelkästään muuttujaviittaukset toisikseen.

generateColumns() GenerateColumns–metodi muuttaa formColumnsin apille kelpaavaan muotoon. Se palauttaa arvonaan String-taulukon, jossa ovat kaikki formColumnsin sisältämät arvot mapattuina TahitiLibraryn COL-arvoihin. Se testataan ajamalla se millä tahansa syötteellä ja vertaamalla mappauksia toivottuihin.

Validate() Käytännössä testattavaksi jääkin vain validate-metodi, joka testataan erityisellä testiluokalla. Validate–metodin tarkoitus on palauttaa virhearvo, jos joku beanin muuttujista puuttuu tai on tyypiltään virheellinen. Tosiarvon palauttamiseen vaaditaan seuraavat ehdot:

number Vaatimukset:

(a) Kentässä on oltava arvo.

(b) Arvon on oltava kokonaisluku.

(c) Kokonaisluvun on oltava suurempi kuin yksi.

Testaus tehdään vaatimukset täyttävällä arvolla sekä kaikilla arvoilla, jotka eivät täytä yhtä vaatimuksista.

name Vaatimukset:

(a) Kentässä on oltava arvo.

(b) Arvon on vähintään yhden merkin pituinen.

Testaus tehdään vaatimukset täyttävällä arvolla sekä kaikilla arvoilla, jotka eivät täytä yhtä vaatimuksista.

designation Vaatimukset:

(a) Kentässä on oltava arvo.

(b) Arvon on vähintään yhden merkin pituinen.

Testaus tehdään vaatimukset täyttävällä arvolla sekä kaikilla arvoilla, jotka eivät täytä yhtä vaatimuksista.

observingSite Vaatimukset:

(a) Kentässä on oltava arvo.

(b) Arvon on vähintään yhden merkin pituinen.

Testaus tehdään vaatimukset täyttävällä arvolla sekä kaikilla arvoilla, jotka eivät täytä yhtä vaatimuksista.

zeroTime Vaatimukset:

(a) Kentässä on oltava arvo.

(b) Arvon on oltava kokonaisluku.

(c) Kokonaisluvun on oltava suurempi kuin yksi.

Testaus tehdään vaatimukset täyttävällä arvolla sekä kaikilla arvoilla, jotka eivät täytä yhtä vaatimuksista.

columns Vaatimukset:

(a) Ainakin yhden radionapin on oltava valittuna.

(b) Yhdessä sarakkeessa ei saa olla kahta radionappia valittuna.

Testaus tehdään vaatimukset täyttävällä arvolla sekä kaikilla arvoilla, jotka eivät täytä yhtä vaatimuksista.

data Vaatimukset:

(a) Kentässä on oltava arvo.

(b) Arvon on vähintään yhden merkin pituinen.

Testaus tehdään vaatimukset täyttävällä arvolla sekä kaikilla arvoilla, jotka eivät täytä yhtä vaatimuksista.

3.10 LoginBean

LoginBean testataan erillisellä testiluokalla. getErrorMessages-, get- sekä set- ja setApiError–metodit ovat testattu esimerkiksi AsteroidQueryBean–luokan yhteydessä, ja koska tämän luokan vastaavat metodit ovat identtisiä näiden kanssa, niiden moduulitestausta lausekattavasti ei ole alla tehty.

LoginBean–luokan JUnit testit löytyvät luokasta *fi.helsinki.cs.group.tahiti.ui.Test.LightcurveQueryBeanTest*.

- *validate()* Seuraavilla kenttien arvoilla metodi tulee testauksi lausekattavasti:

  1. Kentän loginName pituus on alle neljä merkkiä.
  2. Kentän loginPw pituus on alle neljä merkkiä.

  Testi palauttaa epätosiarvon. Testi voidaan järjestää luomalla uusi LoginBean instanssi ja kutsumalla sen jälkeen kyseistä metodia.

  Metodin testausta varten luodaan uusi LoginBean–luokan ilmentymä, jolle annetaan seuraavat syötteet:

  – Kenttä loginName: ”one”
  – Kenttä loginPw: ”two”

3.11 LoginDataBean

LoginDataBean–luokka testataan erillisellä testiluokalla.

- *getName()* Perustapauksessa LoginDataBean–ilmentymällä on kentässä loginInfo viite User–luokan ilmentymään. Tällöin metodi palauttaa kyseisen User–luokan ilmentymän getName() metodin palauttaman arvon. Testi voidaan luoda luomalla uusi ilmentymä User–luokasta ja annetaan kyseinen ilmentymä konstruktorissa LoginDataBean–luokalle.

  Ainoa lisätestitapaus lausekattavuuden saavuttamiseksi on tapaus, jossa LoginDataBean–luokan ilmentymällä ei ole User–luokan ilmentymää. Tällöin metodi palauttaa tyhjän merkkijonon. Testitapaus voidaan luoda luomalla uusi ilmentymä LoginDataBean–luokasta sen peruskonstruktorilla.

- *getUserID()* Perustapauksessa LoginDataBean–ilmentymällä on kentässä loginInfo viite User–luokan ilmentymään. Tällöin metodi palauttaa kyseisen User–luokan ilmentymän getUserID() metodin palauttaman arvon. Testi voidaan luoda luomalla uusi ilmentymä User–luokasta ja annetaan kyseinen ilmentymä konstruktorissa LoginDataBean–luokalle.

  Ainoa lisätestitapaus lausekattavuuden saavuttamiseksi on tapaus, jossa LoginDataBean–luokan ilmentymällä ei ole User–luokan ilmentymää. Tällöin metodi palauttaa arvon -1. Testitapaus voidaan luoda luomalla uusi ilmentymä LoginDataBean–luokasta sen peruskonstruktorilla.

- *getLevel()* Metodi voidaan testata kuten getUserID–metodi. Perustapauksessa palautetaan User–ilmentymän getLevel–metodin arvo. Samanlaisessa lisätestitapauksessa kuin yllä palautetaan nolla.

- *loggedIn()* Perustestissä LoginDataBean–luokan ilmentymällä on kentässä loginInfo viite User–luokan ilmentymään ja kyseisen User–luokan ilmentymän getLevel–metodi palauttaa positiivisen kokonaisluvun. Tällöin metodi palauttaa tosiarvon.

  Ainoa lisätestitapaus lausekattavuuden saavuttamiseksi syntyy kun User–luokan ilmentymää ei LoginDataBean–luokan ilmentymällä ole. Tällöin metodi palauttaa arvon false.

3.12 LogQueryBean

Alla beaneille tyypillisten get– sekä set–metodien testaus on jätetty, koska ne ovat lausekattavasti testattuja jokaisella kutsukerralla, riippumatta syötteistä tai palautteesta. Myös reset–metodi tulee lausekattavasti testatuksi jokaisella kutsumiskerralla.

validate() Käytännössä testattavaksi jääkin vain validate-metodi, joka testataan erityisellä testiluokalla. Validate–metodin tarkoitus on palauttaa virhearvo, jos joku beanin muuttujista puuttuu tai on tyypiltään virheellinen. Tosiarvon palauttamiseen vaaditaan seuraavat ehdot:

timeStart on päivämäärämerkkijono muotoa DD.MM.YYYY. Päivän on oltava ennen nykyhetkeä ja timeEndiä sekä oikeaa muotoa.

timeEnd on päivämäärämerkkijono muotoa DD.MM.YYYY. Päivän on oltava ennen nykyhetkeä ja timeStartin jälkeen sekä oikeaa muotoa.

Testaus suoritetaan ajamalla metodi oikealla syöttelllä ja syötteellä, jossa muuttujilla on arvoinaan eri tavoin virheelliset arvot. Tuloksien pitäisi olla true ja false.

3.13 MailSubmitBean

Alla beaneille tyypillisten get– sekä set–metodien testaus on jätetty, koska ne ovat lausekattavasti testattuja jokaisella kutsukerralla, riippumatta syötteistä tai palautteesta. Myös reset–metodi tulee lausekattavasti testatuksi jokaisella kutsumiskerralla.

validate() Validate–metodin tarkoitus on palauttaa virhearvo, jos joku beanin muuttujista puuttuu tai on tyypiltään virheellinen. Tosiarvon palauttamiseen vaaditaan seuraavat ehdot:

userName täytyy olla epätyhjä merkkijono.

userID täytyy olla epätyhjä merkkijono.

actionType täytyy olla epätyhjä merkkijono.

Testaus suoritetaan ajamalla metodi oikealla syöttelllä ja syötteellä, jossa muuttujilla on arvonaan null-arvo tai tyhjä merkkijono. Tuloksien pitäisi olla true ja false.

3.14 SiteConfigurationSubmitDataBean

Alla beaneille tyypillisten get– sekä set–metodien testaus on jätetty, koska ne ovat lausekattavasti testattuja jokaisella kutsukerralla, riippumatta syötteistä tai palautteesta. Myös reset–metodi tulee lausekattavasti testatuksi jokaisella kutsumiskerralla.

validate() Validate–metodin tarkoitus on palauttaa virhearvo, jos joku beanin muuttujista puuttuu tai on tyypiltään virheellinen. Tosiarvon palauttamiseen vaaditaan seuraavat ehdot:

adminEMail täytyy olla epätyhjä merkkijono, joka on vähintään kuusi merkkiä pitkä.

mailServer täytyy olla epätyhjä merkkijono, joka on vähintään kolme merkkiä pitkä.

serverPort täytyy olla epätyhjä numeromuotoinen merkkijono, joka on välillä 1 – 65534.

passwordMessage täytyy olla epätyhjä merkkijono, joka on vähintään kolme merkkiä pitkä.

approveMessage täytyy olla epätyhjä merkkijono, joka on vähintään kolme merkkiä pitkä.

referenceMessage täytyy olla epätyhjä merkkijono, joka on vähintään kolme merkkiä pitkä.

passwordMessageHeader täytyy olla epätyhjä merkkijono, joka on vähintään kolme merkkiä pitkä.

approveMessageHeader täytyy olla epätyhjä merkkijono, joka on vähintään kolme merkkiä pitkä.

referenceMessageHeader täytyy olla epätyhjä merkkijono, joka on vähintään kolme merkkiä pitkä.

Testaus suoritetaan ajamalla metodi oikealla syöttelllä ja syötteellä, jossa muuttujilla on arvonaan null-arvo tai tyhjä merkkijono. Tuloksien pitäisi olla true ja false.

3.15 UserInfoDataBean

Alla beaneille tyypillisten get– sekä set–metodien testaus on jätetty, koska ne ovat lausekattavasti testattuja jokaisella kutsukerralla, riippumatta syötteistä tai palautteesta. Myös reset–metodi tulee lausekattavasti testatuksi jokaisella kutsumiskerralla.

validate() Käytännössä testattavaksi jääkin vain validate-metodi, joka testataan erityisellä testiluokalla. Validate–metodin tarkoitus on palauttaa virhearvo, jos joku beanin muuttujista puuttuu tai on tyypiltään virheellinen. Tosiarvon palauttamiseen vaaditaan seuraavat ehdot:

1. userName

   Vaatimukset:

   (a) Kentässä on oltava arvo.

   (b) Arvon on vähintään neljän merkin pituinen.

   Testaus tehdään vaatimukset täyttävällä arvolla sekä kaikilla arvoilla, jotka eivät täytä yhtä vaatimuksista.

Name Vaatimukset:

(a) Kentässä on oltava arvo.

(b) Arvon on vähintään neljän merkin pituinen.

Testaus tehdään vaatimukset täyttävällä arvolla sekä kaikilla arvoilla, jotka eivät täytä yhtä vaatimuksista.

EMail Vaatimukset:

(a) Kentässä on oltava arvo.

(b) Arvon on vähintään kuuden merkin pituinen.

Testaus tehdään vaatimukset täyttävällä arvolla sekä kaikilla arvoilla, jotka eivät täytä yhtä vaatimuksista.

recommender Vaatimukset:

(a) Kentässä on oltava arvo.

(b) Arvon on vähintään neljän merkin pituinen.

Testaus tehdään vaatimukset täyttävällä arvolla sekä kaikilla arvoilla, jotka eivät täytä yhtä vaatimuksista. Tämä kenttä on olemassa vain ylläpitäjän käyttöliittymässä.

2. recommenderEMail

Vaatimukset:

(a) Kentässä on oltava arvo.

(b) Arvon on vähintään kuuden merkin pituinen.

Testaus tehdään vaatimukset täyttävällä arvolla sekä kaikilla arvoilla, jotka eivät täytä yhtä vaatimuksista. Tämä kenttä on olemassa vain ylläpitäjän käyttöliittymässä.

3.16 UserInfoDataBean

Alla beaneille tyypillisten get– sekä set–metodien testaus on jätetty, koska ne ovat lausekattavasti testattuja jokaisella kutsukerralla, riippumatta syötteistä tai palautteesta. Myös reset–metodi tulee lausekattavasti testatuksi jokaisella kutsumiskerralla.

validate() Käytännössä testattavaksi jääkin vain validate-metodi, joka testataan erityisellä testiluokalla. Validate–metodin tarkoitus on palauttaa virhearvo, jos joku beanin muuttujista puuttuu tai on tyypiltään virheellinen. Tosiarvon palauttamiseen vaaditaan seuraavat ehdot:

userName Vaatimukset:

(a) Kentässä on oltava arvo.

(b) Arvon on vähintään yhden merkin pituinen.

Testaus tehdään vaatimukset täyttävällä arvolla sekä kaikilla arvoilla, jotka eivät täytä yhtä vaatimuksista.

3.17 UserSearchDataBean

EventHistoryDataBean sisältää pelkästään get- ja set-metodin, joten sen testaaminen on sivuutettu triviaalina.

3.18 DBControl

3.18.1 Julkiset metodit

- *makeUpdate(String update[])* Metodi saadaan testattua testaus–informaatiota sisältäviä yksittäisiä lauseita lukuun ottamatta lausekattavasti testaamalla tapaus, jossa tietokantayhteys on olemassa ja syötteenä annetut SQL-lauseet ovat toimivia; ja tapaus, jossa tietokantayhteyttä ei ole olemassa tai SQL-lauseista joku on virheellinen.
- *makeUpdate(String sql[], int expectedRowCount[], String log[])* Metodi saadaan testattua lausekattavasti ulkoisten seikkojen aiheuttamia virhetilanteita (tietokantayhteys katkeaa kesken suorituksen) lukuun ottamatta käymällä läpi seuraavat tapaukset:

  - sql– ja expectedRowCount–taulukko eivät ole saman kokoisia Metodi palauttaa false ja virhekentässä arvo TahitiLibrary.ERR_WRONG_ARRAY_SIZES.
  - virheetön suoritus: sql– ja expectedRowCount–taulukot ovat saman kokoisia, tietokantayhteys on olemassa ja SQL-lauseet ovat virheettömiä, expectedRowCountin arvot vastaavat päivitettyjen rivien määrää sekä log syöte ei ole null Metodi palauttaa true.
  - Tietokantayhteyttä ei ole olemassa tai syötteenä annettu SQL on virheeellistä Metodi palauttaa false ja virhekentässä arvo Tahiti.Library ERR_NO_CONNECTION_OR_ERRONEOUS_SQL.
  - expectedRowCountit eivät vastaa päivitettyjen rivien määrää Metodi palauttaa false ja virhekentässä arvo TahitiLibrary.ERR_ROW_COUNT_FAILED.

- *makeQuery(String query)* Metodi muodostuu yhdestä try–catch–osiosta, ja näin ollen testattavia tapauksia on kaksi. Ensimmäisessä tapauksessa tietokantayhteys on olemassa ja syötteenä annetaan toimiva SQL-lause. Jälkimmäisessä tapauksessa tietokantayhteyttä ei ole tai SQL-lause on virheellinen.

- *changeLevel(int level)* Lausekattavuuden saavuttamiseksi metodi testataan tapauksessa, jossa kaikki sujuu niin kuin piti; ja tapauksissa, joissa yhteys katkeaa tai käyttäjätasoksi annetaan epäkelvollinen arvo.

- *resetLevel()* Metodi testataan lausekattavasti testaamalla tapaus, jolloin yhteys saadaan muodostettua; ja tapaus, jolloin sitä ei saada muodostettua.

- *isOk()* Metodi testataan tavallisessa tapauksessa ja tapauksesa, jossa yhteysmuuttujan (conn) arvo on null. Jälkimmäinen tapaus voidaan testata silloin, kun yhteyttä ei ole muodostettuna. Testaamatta jää ulkoisista syistä johtuva catch–osio.

- *createConnection(int level)* Metodi muodostuu kahdesta try–catch–osiosta. Lausekattava testaaminen vaatii näin ollen myös virhetilanteidenn (exception) testaamista. Testikierroksia tulee kolme: tavallinen suoritus; suoritus, jolloin tietokanta–ajuria ei löydy; ja suoritus, jolloin yhteyttä ei saada syntymään muiden syiden kuin tietokanta–ajurin puuttumisen takia. Ensimmäinen virhetapaus saadaan testattua poistamalla tietokanta–ajuri CLASSPATH–ympäristömuuttujasta. Jälkimmäinen virhetapaus voidaan testata siten, että tietokantayhteyttä yritetään käynnistää väärästä osoitteesta, väärällä portilla tai väärillä käyttäjätunnuksilla. Yleisesti ottaen koko metodin testaus tapahtuu luokan ilmentymän luonnin yhteydessä. Konstruktori kutsuu createConnectionia.

- *getErrorMessages()* Metodi suoritetaan aina lausekattavasti.

- *finalize()* Metodi suoritetaan luokan ilmentymän automaattisen tuhoamisen yhteydessä. Metodi kutsuu ainoastaan closeConnection–metodia, joka tulee testattua tässä samassa yhteydessä.

3.18.2 Yksityiset metodit

- *closeConnection()* Metodi suoritetaan aina lausekattavasti catch-osiossa olevaa yhtä testaus–informaatiota antavaa koodiriviä lukuun ottamatta. Tällainen vastaava rivi tulee testattua

eräissä muissa metodeissa ja sen toiminta ei ole oleellista metodin toiminnan kannalta, joten rivi jää testauksen ulkopuolelle.

3.19 LightcurveHandler

3.19.1 Yksityiset metodit

Yksityisten metodien testaamista varten metodeille tehdään julkiset wrapper–metodit, joiden kautta yksityisiä metodeja pystytään käyttämään myös testausluokasta. HandlerDataPoint–apuluokka testataan kopioimalla koko luokka omaan luokkaansa, jossa testaus suoritetaan.

- *getLightcurveByID*

  Lausekattavan testauksen perustapauksena on tilanne, jossa annetulla valokäyrätunnuksella oleva havainto löytyy järjestelmästä. Koska tätä metodia käytetään vain, mikäli valokäyrä on muuttunut kesken hakua, ei tilannetta, jossa valokäyrää ei järjestelmästä löydy, pitäisi esiintyä kertaakaan.

- *parseLightcurveDataPoint*

  Testauksen perustapauksena voidaan pitää aineistoa, josta löytyy sarakkeet aika, error sekä jokin filtteri. Toinen keskeinen testaustapaus on Atlas–syöttimen käyttöön liittyvä, sillä tällöin joitain lauseita ei suoriteta laisinkaan. Vaikkakaan lausekattavassa testauksessa ei normaalisti tarvitsisikaan tällaisia tilanteita testata erikseen, on ero tässä koettu niin suureksi, että yksi perustestitapaus, jossa tiedot ovat oikeita, suoritetaan myös Atlas–syötin simuloituna.

  Perustilanteen lisäksi testataan metodin toimintaa seuraavissa tilanteissa:

  – Null–viitteet

    Lausekattavuuden saamiseksi metodia tulee testata myös tilanteissa, jokin annetuista olio–syötteistä (DBControl, data, columnString) on null–viittaus. Tällöin metodin suorituksen pitäisi keskeytä virheeseen, ja metodi palauttaa null–viiteen.

  – Käytetyt sarakkeet

    Koska metodin toiminta eroaa normaalista hiukan, riippuen siitä sisältääkö annettu data error–saraketta, testaan metodin toimintaa molemmissa tilanteissa. Lisäksi lausekattavuuden saavuttamiseksi tulee testata tilanne, jossa time–sarake, tai varsinainen havaintotieto puuttuu. Mikäli jompikumpi sarakkeista puuttuu, tulee metodin suorituksen keskeytä virheeseen, ja metodi palauttaa null–viiteen.

  – Rataelementtien haun epäonnistuminen

    Metodin testausta varten pitää myös järjestää tilanne, jossa käsiteltävälle asteroidille ei löydy oletusratatietoja. Tämä onnistuu helpoiten käyttämällä sellaista asteroidianumeroa, jota järjestelmästä ei löydy. Mikäli sopivaa rataelementtiä ei löydy, keskeytyyn metodin suoritus virheeseen. Tällöin metodi palauttaa null–viitteen.

  – Vääränlaista dataa

Lausekattavuuden saavuttamiseksi metodia pitää myös testata tilanteissa, joissa kaikkia annettuja sarakkeita ei ole täytetty, tai niihin on laitettu vääränmuotoista tietoa, kuten tekstiä. Riittää testata tilanteet, joissa kaikissa sarakkeissa jokin edellämainituista ehdoista täyttyy, sillä tyhjät sarakkeet ja vääränmuotoinen data tarkistetaan samalla kerralla. Myös näissä tapauksissa metodin suoritus päättyy virheeseen ja metodi palauttaa null–viitteen.

- *getSearchWhere*

  Koska getSearchWhere–apumetodi on hyvin suoraviivainen, ei sen testaukseenkaan tarvita mitään erityisen monimutkaisia syötteitä. Lausekkattava testiaineisto saadaan aikaan yksinkertaisesti tilanteessa, jossa kaikki annettavan query–kontainerin kentät ovat täytettyinä. Tämän lisäksi on syytä kuitenkin testata myös tilanne, jossa ainakin jotkin annettavista kentistä ovat tyhjiä merkkijonoja, ja jotkin null–viittauksia. Näissä tapauksissa metodi vain jättää kyseiset kentät huomiotta SQL–where lauseketta muodostettaessa.

- *HandlerDataPoint–apuluokka*

  Yksityisen HandlerDataPoint–apuluokan metodien testaus on hyvin suoraviivaista. Metodin addRow testiaineistoon tarvitaan lisäys oikeilla tiedoilla, sekä lisäykset, joissa jokin lisättävistä tiedoista puuttuu. Lisäksi tarvitaan testata tilanne, jossa annettu havainto sisältää virhetiedon ja tilanne, jossa virhetietoa ei ole, eli virhearvona annetaan Double.NaN. Myös tilanne, jossa täyteen HandlerDataPoint–olioon yritetään lisätä riviä. Muiden, tiedon noutamiseen käytettävien metodien testaus on helppoa; tarvitsee vain testata metodien toiminta oikealla syötteet, sekä sellaiselle rivinumerolla, jota on joko alle nollan, tai suurempi kuin HandlerDataPoint–olion sisältämä rivimäärä.

3.19.2 Normaalit metodit

SQL–Operaatioita suorittavien metodien testauksessa insert, update ja delete –operaatiot keskeytetään DBControlista tehdystä muunnelmassa ja muodostetut SQL–lausekkeet tulostetaan ruudulle. Normaalit SQL–kyselyt suoritetaan normaalisti ja tieto haetaan kannasta.

- *getAsteroids*

  Koska varsinaisen työn getAsteroids–metodia käytettäessä tekee yksityinen getSearchWhere–metodi, on getAsteroids metodi modulitestaus hyvin yksinkertaista. Testauksessa täytyy kokeilla antaa minLightcurves–muuttujalle ei numeerinen arvo, sekä tyhjää syötettä täytyy testata erikseen. Mikäli minimumLightcurves on tyhjä, pitäisi metodin korvata annettu arvo luvulla 1.

- *getLightcurves*

  Myös getLightcurves metodi käyttää yksityistä getSearchWhere–metodia avukseen, mutta metodi on muuten getAsteroidsia monimutkaisempi. Metodin toiminta on kuitenkin hyvin suoraviivaista, joten testitapausten määrä ei kuitenkaan ole kauhean suuri. Minimum lightcurves –muuttujalle tarvitaan vastaavanlainen testaus kuin getAsteroids–metodissakin.

Normaalien tilanteiden lisäksi täytyy myös testata tilannetta, jossa valokäyrän tiedot muuttuvat otsikko– ja datapistehakujen välissä. Tällöin suoritetaan haku yksityisen getLightcurveByID–metodin avulla. Tämä saavutetaan laittamalla metodiin rivi, joka pyytää käyttäjältä rivivaihtoa jatkaakseen. Tällä välin kannassa olevan havainnon versionumero käydään päivittämässä.

- *insertLightcurves*

  Valokäyrien lisäämisen testauksen perustapaukseksi otetaan sellaisen havainnon lisäys, jossa jokainen tietokenttä sisältää jonkinlaista informaatiota. Tässä ensimmäisessä testitapauksessa vain Atlas–tiedostojen siirtämiseen tarkoitettujen kenttien tulee olla null–vitteitä. Toinen tarvittava perustapaus on Atlas–tyyppisen valokäyrän tallentaminen järjestelmään. Myös Atlas–tapauksessa kaikkien kenttien tulee olla täytettyinä validilla datalla.

  Näiden perustapausten lisäksi myös seuraavanlaiset poikkeamat tulee testata:

  – Pakollisten tietojen puuttuminen

    Koska jokaiseen havaintoon liittyy pakollisia tietoja, joiden on oltava täytettyinä, täytyy metodin toimintaa kokeilla tilanteissa, joissa järjestelmällisesti jokin pakollinen tieto puuttuu. Lisäksi joidenkin tiedoille (zeroTime ja ZeroMagnitude > 0, UnitOfTimen tulee löytyä TahitiLibrarystä) tehtävien tarkistusten toimivuus tulee testata. Näissä tilanteissa suorituksen pitäisi keskeytyä virheeseen, ja metodin palauttaa false.

  – Kohteen puuttuminen järjestelmästä

    Jokaisella järjestelmään syötettävällä valokäyrällä tulee olla kohde, joka on talletettu järjestelmään. Lausekattavuuden aikaansaamiseksi metodia testattaessa tulee myös testata tilanne, jossa kohteena olevaa asterodia ei löydy järjestelmän tietokannasta. Lisäksi tarvitaan myös testitapaus, jossa kohteena on asteroid, jonka sisäinen tunnus on 0, eli maa. Tällaisia havaintoja ei tietokantaan haluta, ja metodin suoritus päättyy tällöin virheeseen.

  – Ongelmat datapisteiden kanssa

    Lausekattavuuden saavuttamiseksi metodia tulee testata myös sellaisilla syötteillä, joista datapistetietoja ei pystytä muodostamaan. Tällaisissa tilanteissa yksityinen parseLightcurveDataPoint–metodi palauttaa null, joten tällainen tilanne pitää myös pystyä simuloimaan. Mikäli näin tapahtuu, keskeytyy metodin suoritus.

  – Vääränlaiset Atlas–tiedot

    Metodin toimintaa pitää myös testata tilanteessa, jossa Atlas–siirron yhteydessä annettavat paikkatiedot ovat vääränpituisia. Tällöin suorituksen tulisi keskeytyä.

- *changeLightcurve*

  Kuten insertLightcurve–metodissa, myös muutoksen perustapauksena pidetään tapausta, jossa kaikki tietokentät on täytettyinä, ja sisältävät jotain muuta tietoa, kuin tietokannassa ennen muutosta on. (Myös datapisteiden tietojen tulee muuttua) Näihin kaikkiin tietoihin sisältyy myös olettamus, että myös valokäyräpisteiden havaintoajat muuttuvat siinä määrin, että myös datapisteisiin liittyvät koordinaatit muuttuvat, jolloin myös havaintoon liittyvät paikkatieto–ominaisuudet (PhaseAngle jne) vaihtuvat. Lisäksi lausekattavuus vaatii, että tekstikenttien AbsolutePhotometry ja LightTimeCorrected arvoina on ”true”.

Perustapauksen lisäksi lausekattavuuden saavuttamiseksi metodin toimintaan pitää testata seuraavissa tapauksissa:

- Pakollisten kenttien puuttuminen

  Kuten lisäyksessäkin, valokäyrän muutoksessakin tulee kaikki pakollisten kenttien olla täytettyinä. Lausekattavuuden saavuttamiseksi pitää metodin toimintaa kokeilla tilanteissa, joissa mikä tahansa pakollinen kenttä on joko tyhjä, tai null–viittaus. Tällöin metodin suoritus keskeytyy virheeseen ja metodi palauttaa false.

- Muutoksen kohteena olevaa valokäyrää ei löydy järjestelmästä

  Metodin toimintaa pitää kokeilla myös tilanteessa, jossa muutoksen kohteen, valokäyrän sisäistätunnusnumeroa ei löydy järjestelmästä. Myös tässä tilanteessa suoritus keskeytyy, ja palautetaan false.

- Atlas–siirretty valokäyrä

  Metodia pitää myös kokeilla tilanteessa, jossa muutoksen kohteena on Atlas–siirtimellä järjestelmään syötetty valokäyrä. Tällöin muuntamisen pitäisi olla mahdotonta. Samaan suoritushaaraan päästään, mikäli muutettavaa valokäyrää ei löydy järjestelmästä.

- Kohteen haku

  Lausekattavuuden saavuttamiseksi myös kaikki asteroidiin liittyvät tietokentät tulee olla parametrinä annettavassa containerissa täytettyinä. Tämän lisäksi tulee testata tilanteet, joissa kohdetta ei löydy järjestelmästä, sekä tilanne, jossa havainnon kohteena on virheellisesti maa. Molemmissa tilanteissa metodin suoritus keskeytyy.

- Valokäyrätietojen käsittely

  Koska valokäyrämuutoksessa muutos voi koskea vain yhtä valokäyrää kerrallaan, on yksityisen parseLightcurveDataPoints–metodin palauttaman HandlerDataPoint–taulukon pituuden oltava tasan yksi. Metodin toimintaa pitää testata erikseen virheellisillä havaintoarvotiedoilla, jolloin parseLightcurveDataPoints–metodi palauttaa arvon null, sekä tilanteessa, jossa purettavat havaintotiedot sisältävät enemmän, kuin yhden havainnon tietoja. Tällaisessa tilanteessa metodin suoritus keskeytetään.

- Vapaaehtoisten arvojen poistaminen kannasta

  Lausekattavuuden saavuttamiseksi metodia pitää kokeilla myös tilanteessa, jossa muutettavasta valokäyrästä on kaikki vapaaehtoiset kentät (Information, Reference, PhotometricSystem, Detector, timeStandard) ovat tyhjiä, mutta tietokannan vastaavat kentät sisältävät tietoa. Tällöin metodin tulisi muodostaa SQL–lause, joka poistaa näiden kenttien sisällön tietokannasta.

- Datapisteiden määrän vaihtuminen

  Koska muunnetussa valokäyrässä saattaa olla eri määrä datapisteitä, kuin alkuperäisessä, täytyy metodin toimintaa testata sekä valokäyrien vähentyessä, että niiden määrän kasvaessa. Lausekattavuuden täyttymiseksi näiden uusien datapisteiden tulee sisältää error–sarakkeessa jotain (oikeata) informaatiota.

- Datapisteen virhetietojen muuttuminen

Metodin toimintaa tulee myös testata tilanteissa, jossa datapisteen virhearvo muuttuu tyhjästä joksinkin luvuksi, sekä tilanteessa, jossa järjestelmässä jo oleva virhearvo poistetaan. Jo järjestelmästä löytyvän virhearvon muuttuminen testataan perustapauksen yhteydessä.

- *deleteLightcurve*

  Koska valokäyrien poistaminen on kaksivaiheista, tarvitaan metodin testausta varten kaksi perustapausta. Ensimmäisessä tapauksessa valokäyrä merkitään poistetuksi ja jälkinmäisessä tapauksessa valokäyrän poistetaan järjestelmästä kokonaan. Testauksessa tulee kokeilla myös molempia tapauksia sekä syöttäjätason oikeuksilla, että ylläpitäjän oikeuksilla. Lisäksi tarvitaan testi, jossa syöttäjä yrittää poistaa jonkin muu, kuin itse järjestelmään syöttämänsä valokäyrän. Täydellisen lausekattavuuden aikaansaamiseksi poiston pitää myös jossain testitapauksessa epäonnistua.

- *restoreLightcurve*

  Myös palautusmetodin toiminta on erittäin suoraviivaista, sillä varsinainen palautus on yksinkertainen SQL–operaatio. Lausekattavan testauksen toeuttamiseksi tarvitaan kaksi päätapausta, onnistunut palautus sekä epäonnistuva palautus. Epäonnistuvaksi palautukseksi käy esimerkiksi yritys palauttaa sellainen valokäyrä, jota ei ole merkitty poistetuksi, tai valokäyrä, jota ei järjestelmästä löydy laisinkaan.

3.20 LogHandler

Luokan LogHandler testaamiseen käytetään erillistä testiluokkaa sekä lopullista tilannetta vastaavaa tietokantaa. Ilman oikeaa tietokantaa luokkaa olisi työlästä testata.

Pakkausnäkymän metodit hakevat lokimerkinnät tietyltä ajanjaksolta ja tekevät syötteistä sql–muotoisen lokimerkinnän tallettamislauseen.

- *getLogEntries*

  Metodi testataan kelvollisella ja epäkelvolla syötteellä. Testitapaukset jäljittelevät seuraavia tilanteita:

  - Null–viite voi olla sekä LogQueryContainerin tai DBControl–olion sijalla. Kummassakin tapauksessa metodin tulee palauttaa null–viite.
  - Virheelliset päivämäärät LogQueryContainerissa aiheuttavat virhetilanteen. Metodin tulisi palauttaa tällöin null–viite.
  - Aikaraja jonka sisälle ei mahdu tapahtumia aiheuttaa tyhjän ApiLogEntry–taulukon palautuksen.
  - Jos aikaraja on kunnollinen, tietokantayhteys pelaa ja aikarajan sisälle mahtuu syötteitä, saadaan ApiLogEntry–taulukko, jossa on lokimerkintöjä.

- *getSQLLog*

  Metodi testataan täydellisillä ja vajailla (null–viitteitä) syötteillä. Seuraavassa kuvaus tilanteista, joita testeissä jäljennetään.

- Parametreissa on null–viitteitä. Tämä tilanne ei aiheuta virheitä, mutta puuttuvan parametrin tilalle SQL–lokitekstiin tulee englanninkielinen teksti, joka ilmoittaa tiedon puuttumisen.
- Syötteet ovat kelvollisia. Tällöin saadaan lokiteksti SQL–muodossa, jossa ei ole vakiotekseillä korvattuja kohtia.

3.21 MailHandler

3.21.1 Julkiset metodit

- *spamRecommender(DBControl conn, User user)* Metodi on testattu lausekattavasti kahdella erilaisella testitapauksella. Ensimmäinen tapaus on niin sanottu virheetön suoritus, mikä tarkoittaa sitä, että metodissa kutsuttava getMessage–metodi palauttaa true. Esitellyssä tapauksessa metodin paluuarvo on true. Jälkimmäisessä tapauksessa kutsuttava getMessage–metodi palauttaa false, jolloin myös spamRecommender palauttaa false. Käytännössä getMessage–metodi saadaan helpoiten palauttamaan false suorittamalla DBControllin (julkiseksi muunnettu) closeConnection–metodi ennen metodin kutsumista.
- *spamUser(DBControl conn, User newUser, String password)* Voidaan testata kuten spamRecommender(DBControl conn, User user).
- *sendNewPassWord(DBControl conn, User user, String newPassword)* Voidaan testata kuten spamRecommender(DBControl conn, User user).
- *getErrorMessages()* Suoritetaan aina lausekattavasti.

3.22 Yksityiset metodit

- *getMessage(DBControl conn,String query)* Metodi testataan virheettömän suorituksen ja virheellisen suorituksen osalta. Jälkimmäiseen tapaukseen päädytään, jos tietokantayhteys ei ole kunnossa tai SQL-lause on virheellinen. Tällöin paluuarvo on false ja virhemuuttuja saa arvon TahitiLibarary.ERR_NO_CONNECTION_OR_ERRONEOUS_SQL. Virheettömän suorituksen paluuarvo on luonnollisesti true.
- *processText(String text,String name,String login,String password,String recommender)* Metodi tulee suoritettua lausekattavasti silloin kun mikään syötteistä ei ole null.

3.22.1 Aliluokka MailerThread

- *run()* Tulee testattua aina lausekattavasti.
- *send()* Kun muuttuja DEBUG on true, metodi tulee testattua lausekattavasti.

3.23 OutputCreator

Luokan testaamista varten luodaan erillinen testiluokka. Tietokantayhteyttä ei tarvita, mutta TahitiLibraryn metodeita sitten sitäkin enemmän.

Pakkausnäkymän metodit tuottavat järjestelmän tulostustiedostoja. Näitä on kahta tyyppiä, datafile ja rawdatafile.

- *makeDataFile*

  Metodin syötteet ovat ApiLightcurve–oliotaulukko sekä kaksi ApiTrajectory–oliotaulukkoa. Testaus tehdään sekä kelvollisilla että epäkelvoilla syötteillä. Lisäksi valokäyrien lukumäärien tulisi vaihdella. Seuraavasa ovat tapaukset, joiden todenmukaisuutta testeissä kokeillaan.

  - Jos ApiLightcurve–taulukon tilalla on null–viite, palautetaan null–viite.
  - Jos valokäyrän tiedot ovat valmiiksi kannassa, tulisi ne saada sieltä ilman laskentaa. Tämä huomataan ratatietojen puuttumisesta
  - Jos tietoja ei ole laskettu eli ratatiedot ovat parametrina, lasketaan ne itse.
  - Jos havaintoajat ovat valoaikakorjattuja, ne otetaan mukaan sellaisenaan
  - Jos havaintoaikoja ei ole valoaikakorjattu, tulee metodin tehdä operaatio itse
  - Jos haluttiin tiedot magnitudeina, tulisi metodin palauttaa arvot sellaisina
  - Jos haluttiin tiedot intensiteetteinä, palautetaan ne sellaisina

- *makeRawData*

  Metodin syötteet ovat ApiAsteroid–olio sekä ApiLightcurve–oliotaulukko. Testataan sekä kelvoilla että epäkelvoilla syötteillä.

  - Testattava sekä nimen että numeron sisältävällä sekä vain numeron sisältävällä asteroidilla
  - Testattava asteroidilla, jolla on vain tunnus
  - Testattava useampipisteisellä käyrällä
  - Testattava null–syötteillä

3.24 SystemHandler

Luokan testaamista varten luodaan erillinen testiluokka. Myös tietokantayhteys on tarpeen testausta varten.

3.24.1 Kirjastometodit

Kirjastometodit ovat staattisia metodeita, joilla toteutetaan syötteiden tyypin tarkastaminen, päivämäärän muodon muuttamiset sekä poistetaan syötteestä mahdollisesti haitallisia merkkejä. Syötteiden tyyppitarkistus tarkoittaa lukuarvojen etsimistä merkkijonosyötteistä.

- *deSQL*

  Metodin tarkoituksena on poistaa merkkijonosyötteestä SQL-lauseiden suorittamisessa haitallisia merkkejä. Myös html-tulostuksessa haitalliset merkit poistetaan. Poistettavat merkit ovat ', =, < ja >.

  Metodin lausekattava testaaminen suoritetaan antamalla metodille syötteitä, joissa on joitakin sen karsimia ja palauttamia arvoja. Esimerkiksi seuraavat aineistot riittäisivät:

  - *null –viite* pitäisi palautua metodista sellaisenaan
  - *tyhjä merkkijono* pitäisi palautua metodista sellaisenaan
  - *ohjausmerkit <, > ja =* pitäisivät olla poistettuina palautettavasta merkkijonosta
  - *syötetunniste '* tulisi olla kommentoitu syötteestä muotoon "

- *isDouble*

  Metodi palauttaa syötteenään arvon true tai arvon false, riippuen syötteestä. Jos syöte on desimaalilukuarvo, palautetaan true, muutoin false.

  Metodin testaaminen on syötteen puitteissa suoraviivaista. Metodille annetaan syötteeksi sekä puhdas merkkijonomuuttuja että pelkän desimaaliluvun sisältävä merkkijono. Myös tyhjä merkkijono ja null–viite täytyy testata.

- *isInt*

  Metodi palauttaa syötteenään arvon true tai arvon false, riippuen syötteestä. Jos syöte on integer-lukuarvo, palautetaan true, muutoin false.

  Metodin testaaminen on syötteen puitteissa suoraviivaista. Metodille annetaan syötteeksi sekä puhdas merkkijonomuuttuja että pelkän kokonaisluvun sisältävä merkkijono. Myös tyhjä merkkijono ja null–viite täytyy testata.

- *isLong*

  Metodi palauttaa syötteenään arvon true tai arvon false, riippuen syötteestä. Jos syöte on long integer-lukuarvo, palautetaan true, muutoin false.

  Metodin testaaminen on syötteen puitteissa suoraviivaista. Metodille annetaan syötteeksi sekä puhdas merkkijonomuuttuja että pitkän kokonaisluvun sisältävä merkkijono. Myös tyhjä merkkijono ja null–viite täytyy testata.

- *parseDate*

  Metodi muuttaa syötteenä annetun merkkijonon sisältämän päiväyksen toisen syötteen mukaiseen päivämäärämuotoon. Metodin testaaminen voidaan suorittaa lähinnä antamalla kelpoja ja epäkelpoja syötteitä metodille. Kelpojen syötteiden tapauksessa saadaan halutunmuotoinen syöte, muutoin null–viite.

- *toTimestamp*

  Metodi muuttaa syötteen päiväysmuuttujan SQL:n timestamp –muotoon. Testaaminen suoritetaan seuraamalla tulosteiden oikeellisuutta muiden luokkien moduulitestaamisen yhteydessä. Metodi oletetaan toimivaksi.

3.24.2 Julkiset metodit

Pakkausnäkymän metodit vastaavat varsinaisesta järjestelmätoiminnasta, ts. TahitiApista on kutsometodit jokaista pakkausnäkymän metodia kohden.

- *changeSettings*

  Metodin testaaminen on kiinni syötteistä. Testaaminen on suoritettava kelvollisilla, epäkelvoilla ja puoliksi epäkelvoilla syötteillä. Syötteet ovat DBControl–olio sekä SettingsFromContainer–olio.

  - AdminEMail, MailServer ja ServerPort ovat pakollisia kenttiä. Testauksessa tullaan erityisesti painottamaan niiden oikeellisuutta. Metodi palauttaa falsen jos jokin pakollisista arvoista puuttuu.

- *getSettings*

  Metodin testaaminen on suoraviivaista; syötteeksi annettavan DBControl–olion on oltava kunnossa ja tietokannan käynnissä. Testaus suoritetaan toimivan yhteyden kanssa sekä toimimattomaksi muokatun yhteysluokan kanssa.

  - Suoritetaan kannan ollessa päällä ja aputestiluokalla, joka käyttäytyy kuin kantaa ei olisi.

3.25 UserHandler

3.25.1 Yksityiset metodit

Yksityisten metodien testaamista varten luokkaan lisätään väliaikaisesti kolme julkista apumetodia, jonka kautta testausluokka pääsee yksityisiin metodeihein käsiksi. Nämä metodit ovat seuraavanlaiset:

public String encryptTest(String test) return this.encrypt(test);

public String generatePasswordTest() return this.generatePassword();

public ApiUser parseFormTest(UserFormContainer data) return this.parseForm(data);

- *encrypt(String plaintext)*

  Encrypt–metodin toiminta on hyvin suoraviivaista, ja jokaisella käyttökerralla lähes jokainen metodin rivi tulee suoritetuksi. Vain alun poikkeuskäsittely muodostaa poikkeuksen tähän ja poikkeuskäsittelyn testaamiseksi joudutaan luokan sisäiselle HASH_ALGORITHM –muutujalle antamaan järjetön arvo. Jotta kaikki rivit tulevat suoritetuksi, tarkistetaan palautettavasta hexadecimaali jonosta, että se sisältää vähintään yhden nollalla alkavan tavun, sekä yhden tavun, jonka numeerinen arvo ylittää luvun 128.

  Oletus SALT –muuttujan arvolla tälläinen testiaineisto on esimerkiksi ”salasana”, joka palauttaa SHA–algoritmilla ja ”Just type something here--suolalla merkkijonon $Ox2a4a117c1ac82c679472031f498e24174f1ab0c6$. Tämän merkkijonon tavu 03 sisältää tarvittavan lisänollan, ja tavu C8 on selvästi suurempi kuin raja-arvo 128.

Virheellisellä HASH_ALGORITHM –arvolla metodi palautti oletetusti null. Lisäksi virheen nouto getErrorMessages–metodilla onnistui.

- *generatePassword()*

  Koska genereta Passwords metodin koko toimintaidea liittyy satunnaisuuteen, ei sen testaaminen lausekattavasti ole täysin mahdollista ilman muutoksien tekemistä metodiin. Liisämällä jokaiseen metodin ehtoon tulostuslausekkeen voitiin varmistua siitä, että kaikki metodin lauseet tulivat suoritetuiksi. Metodin lopussa olevaa poikkeuskäsittelyä pystyttiin testamaan muuntamalla käytettävä merkkijärjestelmä ASCIIsta johonkin sellaiseen koodistoon, jota ei ole olemassa.

- *parseForm(UserFormContainer data)*

  Koska parseForm ottaa syötteenä UserFormContainerin, tarvittiin sen testaamista varten ilmentymä kyseisestä containerista. Koska pakkauksesta fi.helsinki.cs.group.tahiti.ui löytyy rajapinnan toteuttava UserInfoDataBean–luokka, käytetään testauksessa sitä.

  Seuraavassa on lueteltu luokan eri haarautumisvaihdoehdot, ja näistä saatavat edellytykset testitapauksille.

  - UserID

    Alun try-catch rakenteessa muunnetaan annettu userID long–tyyppiseksi. Testauksessa tarvitaan aineisto, jossa userID on oikeaa muotoa, sekä tapaus, jossa se ei ole oikeaa muotoa. Mikäli kentän sisältö on väärää muotoa, keskeytyy metodin suoritus virheeseen.

  - Level ja version

    Käyttäjätasolle ja versiolle tehdään vastaavat tarkistukset, joten myös niitä kohtaan joudutaan tekemään testi, jossa se on oikean tyypinen, numeerinen sekä testi, jossa kentän arvona on jotain muuta. Tällöin metodin suoritus keskeytyy virheeseen.

  - Käyttäjästatus

    Mahdolliset käyttäjästatuksesta ovat ”Amateur”sekä ”Professional”. Testit pitää suorittaa molemmilla arvoilla, sekä jollain ei–validilla arvolla. Mikäli arvo on väärä, keskeytyy metodin suoritus virheeseen.

  - Muut pakolliset kentät

    Koska kaikista pakollisista kentistä tarkisteen että ne ovat täytettyinä, joudutaan testauksessa jokainen kenttä jättämään järjestyksessä tyhjäksi. Lausekattavuuden täyttämiseksi kentät pitää erikseen täyttää tyhjällä Stringillä (), ja null–viitteellä. Pakollisten kenttien puuttuessa metodin suoritus päättyy virheeseen.

  - Vapaaehtoiset kentät

    Koska metodi ei tee vapaaehtoisille kentille muita tarkistuksia, kuin tyhjät stringit muutetaan null–viittauksiksi, saadaan lausekattavuus täytettyä laittamalla kaikkiin vapaaehtoisiin kenttiin tyhjät String–arvot testauksessa.

3.25.2 Normaalit metodit

Myös normaalien metodien testauksessa käytetään testausluokkaa UserHandlerTester. Koska normaalit metodit saattavat olla riippuvaisia yksityisistä metodeista, SystemHandlerin apumetodeista,

tai DBControlin tarjoamasta tietokantayhteydestä, ei täysin puhdas modulitestaus ole mahdollista. Tietokantaan tehtäviä kirjoituksia ei suoriteta loppuun asti, vaan testauksessa käytetään modifioitua DBControl–luokkaa, joka vain tulostaa mahdollisest Insert, Update ja Delete –lauseet ruudulle. Haut kuitenkin suoritetaan oikeasti.

- *registrationRequest*

  Tämänkin metodin testaamiseen käytetään pakkauksesta fi.helsinki.cs.group.tahiti.ui löytyvää UserInfoDataBean–luokkaa. Perustapauksessa kaikki kentät ovat täytettyinä, eikä tietokannasta tule löytyä annettua loginID:tä vastaavaa käyttäjätunnusta. Seuraavassa on listattu perustapauksen lisäksi tarvittavat testitapaukset.

  - Virheelliset tiedot annetussa UserFormContainerissa

    Koska annetun UserFormContainerin tulkitseminen tapahtuu yksityisessä parseForm–metodissa, ei tämän luokan yhteydessä erikseen testata kaikki eri syöteyhdistelmiä. Testauksessa vain tarkistetaan, ettei satunnainen, epäkelpo syöte mene läpi testistä.

  - Tunnuksen olemassaolon tarkistus

    Olemassaolo tarkistuksessa tarvitaan tietokantaa, joten testausta varten tarvitaan sellainen aineisto, että ensimmäisellä ajolla luotavaa käyttäjätunnusta ei löydy kannasta, ja toisella ajolla se löytyy.

  - Vapaaehtoisten kenttien täyttäminen

    Vapaaehtoisien kenttien täyttämistä tutkivat lauseet testaan antamalla metodille syötteitä, joissa jokaisesta puuttuu vuorollaan jokin vapaaehtoinen kenttä. Tämän jälkeen modifiodun DBControl–luokan ruudulle tulostamaa SQL–lausetta verrataan odotettuun tulokseen.

- *login*

  Kirjautumisen suoraviivaisen luoteen vuoksi testauskin on hyvin yksinkertaista. Testausta varten tarvitaan sellainen aineisto, että ensimmäisellä ajolla käyttäjä löytyy järjestelmästä, ja toisella ei.

- *changePassword*

  Myös salasanan vaihto on erittäin suoraviivainen operaatio. Testauksessa täytyy kokeilla vaihtoa sellaisilla syötteillä, joilla vaihdon pitäisi onnistua, ja lisäksi tilanteita, joissa annettu uusi salasana on liian lyhyt tai vanha salasana on väärä.

- *getUsers*

  Koska getUsers–metodi ei toimi kuin eräänlaisena välittäjänä getUserFullille, tarvitaan sen testaamiseen vain kaksi tapausta. Toisessa getUsersFull palauttaa kannasta löytyviä tuloksia, ja toisessa palautetaan null.

- *getUsersFull*

  GetUsersFull–metodi suorittaa vain yhden SQL kyselyn, ja palauttaa kyselyn tulokset kysyjälle, joten testaustan varten tarvitaan vain suorittaa kysely, jolle löytyy tuloksia (esimerkiksi voidaan hakea tyhjällä merkkijonolla), kysely, jolle ei löydy yhtään vastinetta, sekä kysely

null-viitteelle. Näiden lisäksi myös versionumeron täyttämistä tulee kokeilla. Lausekattavuuden saavuttamiseksi riittää, että yhdessä haussa versionumeroa ei ole määritelty, ja toisessa on.

- *setUserData*

  Käyttäjätietojen muuttaminen voidaan jakaa kahteen pääluokkaan, käyttäjän itsensä tekemiin muutoksiin, sekä ylläpitäjän tekemään muutokseen. Näiden lisäksi testataan vielä tilanne, jossa joku muu tavallinen käyttäjä pyrkiin muuttamaan toisen käyttäjän tietoja.

  Seuraavanlaisia testitapauksia tarvitaan, jotta lausekattavuus täyttyisi:

  - Kirjautumistunnuksen muutos

    Kirjautumistunnuksen muutoksessa tulee testata tilanne, jossa haluttu uusi käyttäjätunnus on jo järjestelmässä. Tällöin muutos keskeytetään.

  - Olemattoman käyttäjän vaihto

    Koska ylläpitäjät pystyvät vaihtamaan myös muiden, kuin itsensä tietoja, tulee testauksessa ottaa huomioon tilanne, jossa muutettavaa käyttäjää ei löydy järjestelmästä laisinkaan. Tällöin muutosta ei luonnollisesti voida suorittaa.

  - Käyttäjätasoon kohdistuva muutos

    Ylläpitäjän tekemien käyttäjätietojen muutoksen testaamiseen riittää yksi onnistuva testitapaus, jossa myös käyttäjätaso vaihtuu. Mikäli käyttäjätasoa ei vaihdeta, suoritetaan muutama rivi vähemmän, joten lausekattavan testauksen kannalta asialla ei ole merkitystä.

- *resetPassword*

  resetPasswordin lausekattavaa testausta varten täytyy yrittää nollata sellaisen käyttäjän salasanaa, jota kannasta ei löydy, normaalin nollauksen lisäksi.

- *getUser*

  getUser jakautuu oikeasti kahteen metodiin, kirjautumistunnuksen ja sisäisen käyttäjänumeron perusteella tehtäviin hakuihin. Ensimmäinen, kirjatumistunnukseen perustuva haku hakee vain käyttäjän sisäisen käyttäjänumeron, ja kutsuu tämän avulla käyttäjänumeron perusteella tapahtuvaa hakua. Tätä metodia varten testiaineistoksi riittää järjestelmästä löytyvä kirjautumistunnus, sekä tunnus jota järjestelmästä ei löydy.

  Varsinaisen käyttäjähaun tekevä getUser on myöskin hyvin yksinkertainen. Testiaineistoksi tarvitaan vain käyttäjänumero, joka löytyy järjestelmästä ja numero, jota järjestelmästä ei löydy.

- *removeUser*

  Poistoa testattaessa tarvitaan tietokantaan sellainen aineisto, jossa ensimmäisessä ajossa käyttäjän toiminta ei ole aiheuttanut yhtään merkintään järjestelmän lokiin ja toisessa tapauksessa tälläinen merkintä täytyy löytyä. Lisäksi lausekattavuuden saamiseksi tarvitaan tapaus, jossa poistaminen ei onnistu.

3.26 TahitiLibrary

TahitiLibrary.java modulin oikeellisuutta testataan asiakkaan antamien tietojen perusteella. Metodit ajetaan parametreillä, joista saatava tulos on ennalta tiedossa. Ennakkotiedot on kerätty Mikko Kaasalaisen lähettämistä Atlas-tiedostoista, joiden oikeellisuuteen siis luotetaan. Metodin palauttamaa arvoa verrataan ennakkotietoon oikeellisuuden varmistamiseksi. Metoditestausta varten TahitiLibraryyn tulee väliaikainen main-metodi, jossa testikutsut sijaitsevat.

- *toDate* Metodi muuttaa juliaanisessa muodossa olevan päiväyksen Date-muotoon. Testataan kutsumalla parametrilla 2448188.5, josta tuloksena tulisi olla keskipäivä GMT 24.10.1990.
- *toJulian* Metodi muuttaa Date-muodossa olevan päiväyksen juliaaniseen muotoon. Testataan palauttamalla toDaten muodostama Date-olio juliaaniseen muotoonsa.
- *aspectData* Metodeita distance, solarPhase, eclipticLongitude ja eclipticLatitude varten muodostetaan testiaineisto, johon kuuluu kaksi kolmiuloitteista paikkavektoria,
  $v = (2.819902, -0.349044, 0.026762)$ ja
  $u = (0.855732, 0.507082, 0.000003)$, joita v on asteroidin paikkavektori ja u maan paikkavektori. Näillä paikoilla aspektitietojen tulisi saada seuraavat arvot:
  - distance(v): 2.8416
  - phaseAngle(v,u): 16.50
  - eclipticLongitude(u,v): 336.40
  - eclipticLatitude(u,v): 0.70

  Lisäksi metodit testataan vektoreilla, jotka eivät ole kolmiulotteisia sekä null arvoilla. Näissä tapauksissa palautetta arvo pitäisi olla NaN.

  Kutsutaan parametreilla null, jolloin palauttaa NaN, taulukolla, jonka pituus on eri kuin kolme, joilloin palauttaa NaN.
- *countCoordinates* Metodi testataan rataelementeillä:

```
ApiTrajectory asteroid = new ApiTrajectory();
asteroid.setEpoch(toDate(2452200.5));
asteroid.setAxis(1.3226910);
asteroid.setEccentricity(0.27961274);
asteroid.setInclination(1.72799);
asteroid.setArgument(161.02783);
asteroid.setLongitude(70.92064);
asteroid.setAnomaly(107.8134613);
```

  juliaanisena ajanhetkenä 2451942.732663. Tuloksena pitäisi olla asteroidin koordinaattivektori $v = (-0.951738642, 0.770087699, 0.034729127)$. Pienet heitot lukuarvoissa ovat mahdollisia, koska se miten lähelle todellisia arvoja päästään riippuu siitä, miten vakiot MEAN_ANOMALY_ITERATION_MAX ja ennenkaikkea TRAJECTORY_ERROR_TOLERANCE asetetaan. v:tä laskettaessa on suoritettu 20 iteraatiota.

- *lighttimeCorrect* Metodi testataan valoaikakorjaamalla ajanhetki 2451942.732663. Maan koordinaateilla eC ja asteroidin koordinaateilla aC:

```
aC[0] = -0.951738642;
aC[1] = 0.770087699;
aC[2] = 0.034729127;
eC[0] = (0.274172392 - 0.951738642);
eC[1] = (-0.0544258374 + 0.770087699);
eC[2] = (-0.0347286299 + 0.034729127);
```

  Tuloksena pitäisi olla ajanhetki 2451942.73104.

- *reduceMagnitude, toIntensity* Metodit testataan yhdessä. Ensin reduceMagnitude-metodia kutsutaan magnitudilla 18.344, samoin koordinaatein kuin valoaikakorjauksessa, minkä jälkeen saatu magnitudi muutetaan intensiteetiksi. Tuloksena pitäisi olla lukuarvo 0.5470039E-03.

3.27 TrajectoryHandler

3.27.1 Julkiset metodit

Julkiset metodit testataan luomalla luokasta ilmentymä, ja kutsumalla niitä. Metodit testataan lausekattavasti.

- *getErrorMessages* Metodi hakee sisäisen virhekentän arvon. Suoritetaan aina lausekattavasti.
- *getTrajectory* Metodi hakee kannasta rataelementtietietoja. Lausekattavuus saavutetaan testaamalla seuraavat tapaukset:
  - Parametri dbConn on null Tuloksena null ja sisäisessä virhemuuttujassa arvo TahitiLibrary.ERR_NULL_PARAMETER.
  - Tietokantayhteys ei ole kunnollinen Tuloksena null ja sisäisessä virhemuuttujassa arvo TahitiLibrary.ERR_NO_CONNECTION.
  - Tietokantayhteydessä virhe suoritettaessa kyselyä Tuloksena null ja sisäisessä virhemuuttujassa arvo TahitiLibrary.ERR_NO_CONNECTION_OR_ERRONEOUS_SQL.
  - Kannasta löytyvät ainakin yhdet rataelementit halutulle asteroidille Tuloksena ApiAsteroid-taulukko täytettynä kannasta löytyneiden rataelementtien tiedoilla.
- *addTrajectory* Metodi lisää kantaan rataelementtejä. Lausekattavuus saadaan testaamalla tapaukset:
  - Parametrien tarkistus epäonnistuu (checkParameters palauttaa false) Tuloksena false.
  - Rataelementtisarakkeiden järjestäminen epäonnistuu (arrangeColumns palauttaa false) Tuloksena false.
  - Jonkin rivin jäsentäminen epäonnistuu (parseRow palauttaa false jollekin riville) Tuloksena false.

- Asteroidia, jolle ratatietoja lisätään ei ole vielä kannassa Tuloksena lisäyskutsussa conn.makeUpdate(updateReal, expectedNumber, log) updateReal-taulukko sisältää tämän asteroidin lisäysrivin.
- Tietokantayhteys pettää, kun tarkistetaan, onko rataelementti duplikaatti Tuloksena false ja sisäisessä virhemuuttujassa arvo TahitiLibrary.ERR_NO_CONNECTION_OR_ERRONEOUS_SQL.
- Rataelementti ei ole duplikaatti Tuloksena lisäyskutsussa conn.makeUpdate(updateReal, expectedNumber, log) updateReal-taulukko sisältää tämän rataelementin lisäysrivin.
- Kantayhteys pettää lisäysvaiheessa Tuloksena false ja virhemuuttujassa arvo TahitiLibrary.ERR_NO_CONNECTION_OR_ERRONEOUS_SQL.
- Lisäys, jossa oikeellisessa datassa on sekä sellaisia asteroideja, joita kannasta ei vielä löydy, että toisenlaisia Tämä testitapaus kattaa osan yllåluetelluista tapauksista. Tuloksena conn.makeUpdate(updateReal, expectedNumber, log) kutsu, jossa updateReal-taulukko sisältää kaikki rataelementit, jotka eivät olleet duplikaatteja, expectedNumber-taulukko sisältää yhtä monta ykköstä kuin updateRealissä on alkioita ja log sisältää kaksi lokilisäystä, joista toisessa on rataelementtilisäysten oikea määrä ja toisessa asteroidilisäysten oikea määrä updateReal-taulussa.

3.27.2 Yksityiset metodit

Yksityisten metodien testaamista varten luokkaan lisätään väliaikaisesti julkinen apumetodi jokaista ykstyistä metodia vastaamaa, jonka kautta testausluokka pääsee yksityisiin metodeihein käsiksi. Koska jotkin yksityiset metodit muokkaavat yksityisiä muuttujia myös jokaista yksityistä muuttujaa varten tehdään julkinen get-metodi, jolla siihen päästään käsiksi.

- *getDataRows* Metodi jakaa saamansa tekstin osiin rivinvaihtojen kohdalta tekemättä parametrille minkäänlaisia tarkistuksia. Metodin testataan lausekattavasti seuraavalla kutsulla:
  - Merkkijono, jossa on rivinvaihtoja

    ```
    getDataRows("Rai\nrai\nraa\n\nmina\nhaistan\nihmis\lihaa");
    ```

    Tuloksena tulisi olla merkkijonotaulukko (Rai, rai, raa, mina, haistan, ihmis, lihaa).
- *checkParameters* Metodin ainoa tarkoitus on tarkistaa, löytyykö addTrajectory-metodille annetuista parametreistä null-viitteitä, tyhjiä merkkijonoja tai virheellisiä arvoja. Mahdollisia virhetilanteita ovat:
  - Oikeelliset parametrien arvot Tuloksena true.
  - Jokin parametreistä dbConn, tfc tai user on null Tuloksena kaikissa tapauksissa false ja virhekentässä arvo TahitiLibrary.ERR_NULL_PARAMETER.
  - Tietokantayhteys dbConn on viallinen ja palauttaa metodin isOk arvona false Tuloksena false ja virhekentässä arvo TahitiLibrary.ERR_NO_CONNECTION.
  - jokin trajectoryFormContainerin attribuuteista on null Tuloksena false ja virhekentässä arvo TahitiLibrary.ERR_NULL_PARAMETER.

  - TrajectoryFormContainerin columns-taulukon koko ei ole sama kuin TahitiLibrary.NUMBER_OF_TRAJECTORY_COLUMNS Tuloksena false ja virhekentässä arvo TahitiLibrary.ERR_WRONG_ARRAY_SIZES.
  - TrajectoryFormContainerin columns-taulukko sisältää null viitteitä tai tyhjiä merkkijonoja tai jotakin merkkijonoista ei voida muuttaa numeroksi. Tuloksena tulisi kaikissa tapauksissa olla false ja getErrorMessages-metodin tulisi palauttaa virhe TahitiLibrary.ERR_NULL_PARAMETER.

- *arrangeColumns* Metodi etsii parametrinaan saamasta merkkijonotaulukosta rataelementtien sijainnit. Metodi olettaa, että merkkijonotaulun oikeellisuus (ei null-arvoja jne.) on tarkistettu kutsumalla checkParameters-metodia.

  - Normaali tapaus, eli kaikki rataelementit sijaitsevat jossakin Tuloksena true ja sisäiset muuttujat axis,epoch,designation,eccentricity,inclination argument, longitude,anomaly ja reference on asetettu vastaamaan kunkin rataelementin sijaintia.
  - Jokin rataelementeistä ei ole TahitiLibraryssa kerrottu hyväksyttävä sarake. Tuloksena false ja sisäisessä virhemuuttujassa arvo TahitiLibrary.ERR_INVALID_TRAJECTORY_COLUMN.

- *parseRow* Metodi jäsentää yhden rivin sisäisiin muuttujiin parseAsteroid ja parseTrajectory Metodin testataan lausekattavasti seuraavilla kutsuilla:

  - Hyväksyttävä merkkijono eli rivi, jolla on kaikki rataelementit asetettuina oikein Tuloksena true ja parseTrajectoryn kentät asetettuina riviä vastaavaksi.
  - Merkkijono (rivi), jossa alle TahitiLibrary.NUMBER_OF_TRAJECTORY_COLUMNS välilyöntiä eli liian vähän sarakkeita

    ```
    parseRow("C1 C2 C3");
    ```

    Tuloksena false ja sisäisessä virhekentässä arvo TahitiLibrary.ERR_TOO_FEW_COLUMNS.
  - Nimitietoja ei onnistuta jäsentämään Tuloksena false ja sisäisessä virhekentässä arvo, jonka parseDesignation-funktio on asettanut.
  - Jonkin pakollisen ratatietokentän paikkaa ei tiedetä, eli että jokin ratatietojen sijaintikentistä on -1 Tuloksena false ja sisäisessä virhekentässä arvo TahitiLibrary.ERR_TRAJECTORY_ELEMENTS_NOT_SET
  - Jotakin rivillä olevista numeroarvoista ei voida muuttaa tekstistä numeroksi Tuloksena false ja sisäisessä virhekentässä arvo TahitiLibrary.ERR_DATAFORMAT

- *parseDesignation* Metodi jäsentää asteroidin nimitiedot rivistä, joka on jaettu osiin välilyöntien mukaan, jos nimitiedot on paikallistettu arrangeColumns metodilla.

  - Metodi saa parametrinaan rivin, jolla nimitieto koostuu numerosta, moniosaisesta nimestä ja useampi kuin kaksiosaisesta tunnuksesta. Lisäksi asteroidi ei saa löytyä kannasta eli se joudutaan lisäämään. Tuloksena true ja sisäisessä parseAsteroid kentässä kannan palauttama uusi numero, sekä jäsennetyt nimitiedot.

  - Metodi saa parametrinaan rivin, jolla nimitieto koostuu numerosta, moniosaisesta nimestä ja useampi kuin kaksiosaisesta tunnuksesta. Lisäksi asteroidi löytyy kannasta eli sitä ei lisätä vaan sen järjestelmän sisäinen numero saadaan kannasta. Tuloksena true ja sisäisessä parseAsteroid kentässä kannasta löytynyt numero, sekä jäsennetyt nimitiedot.
  - Metodi saa parametrinaan rivin, jolla nimitieto koostuu numerosta, moniosaisesta nimestä ja useampi kuin kaksiosaisesta tunnuksesta. Lisäksi nimitiedoilla löytyy useita asteroideja kannasta. Nimitiedot tarkistetaan OR vertailulla eli tämä tapaus pätee esimerkiksi jos numero täsmää eri asteroidin kanssa kuin nimi tai tunnus. Tuloksena false ja sisäisessä virhekentässä arvo TahitiLibrary.ERR_INVALID_ASTEROID_DESIGNATION.
  - Kantayhteydessä tulee virhe nimitietoja tarkistettaessa Tuloksena false ja sisäisessä virhekentässä arvo TahitiLibrary.ERR_NO_CONNECTION_OR_ERRONEOUS_SQL.

- *parseAsteroidInsert* Metodi muuntaa saamansa Asteroid-containerin postgre-lisäyslauseeksi.

  - Kentät asteroidID, designation, name ja number ovat kaikki täytettyinä containerissa Tuloksena insert-lause muotoa:

    ```
    Insert into asteroid (asteroidnumber, designation, name number)
    values (1,'Designi','neimi',2);
    ```

  - Kenttä designation on tyhjä merkkijono Tuloksena insert-lause muotoa:

    ```
    Insert into asteroid (asteroidnumber, designation, name number)
    values (1,null,'neimi',2);
    ```

  - Kenttä name on tyhjä merkkijono Tuloksena insert-lause muotoa:

    ```
    Insert into asteroid (asteroidnumber, designation, name number)
    values (1,'Designi',null,2);
    ```

  - Kenttä number on -1 eli parseAsteroid ei ole löytänyt asteroidille numeroa Tuloksena insert-lause muotoa:

    ```
    Insert into asteroid (asteroidnumber, designation, name number)
    values (1,'Designi','neimi', null);
    ```

- *parseTrajectoryInsert* Metodi muuntaa saamansa Trajectory-containerin postgre-lisäyslauseeksi.

  - Normaali tapaus, jossa Trajectory-containerin kentät asteroidID, trajectoryID, axis eccentricity, inclination, argument, longitude, anomaly, epoch ja reference ovat täytettyinä Tuloksena insert-lause muotoa:

    ```
    Insert into trajectory (asteroid, trajectorynumber, axis, eccentricity,
                            inclination, argument, longitude, anomaly, epoch,
                            orbitreference)
    values
    (1,1,0.12,0.11,0.13,0.123,0.412,0.142,0.1234,to_timestamp(\\'DATE','FORMAT'),
    'Referenssi');
    ```

- *aspectUpdate* Metodi tarkistaa, onko kannassa ratatietojen lisäyksen jälkeen valokäyriä, joiden sijaintitietoja tai vaihekulmaa, etäisyyttä auringosta, etäisyyttä maasta pituusastetta tai leveysastetta pitäisi muuttaa.
  - Kannassa on näkymässä lightcurveupdate valokäyriä, joilla on datapisteitä Tuloksena jokaista datapistettä ja valokäyrää kohden on SQL-lause, joka päivittää kantaa.
- *isDuplicate* Metodi tarkistaa, onko kannassa jo ratatietoja, joita sinne ollaan tallettamassa. Metodi suoritetaan joka tapauksessa lausekattavasti.

  Tuloksena true, jos kannassa duplikaatti false muutoin.

3.28 AtlasImporter

AtlasImporteria ei testata sen erikoisluontoisuuden vuoksi metodi kerrallaan, vaan testaus suoritetaan kyseistä ohjelmaa suorittamalla. Käytössä on oikeastaan AtlasImporterin testiluokka, joka ei kutsu atlasimporter–packagen ulkopuolisia luokkia, vaan esimerkiksi tulostaa tietokantaan oikeasti syötettävät arvot näytölle. Testaus voidaan jakaa kahteen osaan. Ensiksi testataan ohjelmalle annettavat argumentit, ja sen jälkeen testataan testausta varten rakennettu atlas–tiedosto. Tällaisella testauksella päästään ohjelman olennaiset osat koskevaan lausekattavuuteen. Ensimmäisessä vaiheessa testataan tapaukset: väärä määrä argumentteja ja atlashakemisto tai tiedosto, jota ei ole olemassa. Jälkimmäisessä vaiheessa AtlasImporterin pitäisi tulostaa ruudulle testi–atlas–tiedostossa annetut tiedot tai antaa virheilmoitus, jos pakollisia kenttiä ei ole täytetty. Testi–atlas–tiedosto on tämän dokumentin liitteenä.

4 Integraatiotestaus

Integraatiotestaus suoritetaan ilman käyttöliittymää ns. Black–box testauksena bottom–up –mallin mukaisesti. Integraatiotestauksessa pyritään siihen, ettei sama henkilö suorita sekä moduulitestausta että integraatiotestausta tekemälleen luokalle. Tästä pyrkimyksestä saatetaan kuitenkin joutua joustamaan käytettävissä olevien resurssien mukaisesti.

Black–Box –testaustyppistä joustetaan kuitenkin siinä määrin, ettei jokaisen merkkijonoja parametrinään ottavan metodin yhteydessä testata erikseen erilaisia SQL–injektio tapauksia. Integraatiotestauksen yhteydessä luetetaan siihen, että handler–luokat suorittavat jokaiselle String–tyyppiselle muuttujalle SystemHandlerin deSQL–metodin, jolloin mahdolliset SQL–lauseet ja erikoismerkit tehdään vaarattomiksi. Tämän metodin toiminta on jo testattu moduulitestauksen yhteydessä, ja se oletetaan toimivaksi. Validointitestauksen yhteydessä tätä toimintaa testataan pistokokein.

Tahiti–Apin integraatiotestaus suoritetaan bottom–up mallin mukaisesti. Tämä tarkoittaa sitä, että aluksi testataan Apin alapuolella olevat metodit ja luokat, ja vasta tämän jälkeen varsinaisen Apin toiminta yhdessä eri toiminnot toteuttavien metodien ja luokkien kanssa. Alapuolella olevien metodien toiminta on kuitenkin suoritettu jo moduulitestauksen yhteydessä, joten testaaminen voidaan aloittaa suoraan kokonaisen TahitiApin kanssa.

Seuraavissa aliluvuissa käsitellään jokaisen eri metodin testaukseen tarvittavat aineistot. Aineistot on pyritty muodostamaan siten, että mahdollisimman moni metodin oikea käyttötapaus tulisi testatuksi, sekä mahdolliset virhetilanteet saataisiin selville. Virhetilojen löytämiseksi metodien toimintaa kokeillaan raja–arvoilla, jotka ovat sallittujen rajojen molemmin puolin. Lisäksi luokkien toimintaa kokeillaan selvästi virheellisillä arvoilla.

4.1 getAsteroids

Metodi testataan aluksi *tyhjällä haulla*, jonka ainoana ehtona on, että valokäyrien lukumäärä on yksi tai useampia. Tämän jälkeen tutkitaan tapaus, jossa valokäyrien lukumääräksi asetetaan arvo, joka on pienempi kuin yksi.

Tyhjän haun testauksen jälkeen hausta testataan vuorotellen alla luetellut tapaukset. Jokaisen tapauksen yhteydessä, jossa oletetaan haun toimivan testataan sekä tapaus, että tietokannasta löytyy ehdon täyttävä tapaus, että tapaus, jossa hakuehto ei täyty millään kannan alkiolla.

- Hakuehdoksi asetetaan null-viite
- Johonkin Suunnitteludokumentin luvusta 3.2 löytyvään kenttään asetetaan null-viite
- Suunnitteludokumentin luvusta 3.2 löytyvien lukuarvoehtojen mukaiset ylä- ja alarajat sekä ylärajat ylittävät ja alarajat alittavat arvot.
- Validi haku jokaisen suunnitteludokumentin luvusta 3.2 löytyvän kentän kohdalla erikseen. Testauksessa suoritetaan sekä tapaus, jossa haun tulisi tuottaa tulosta, että tapaus, jossa tuloksena on tyhjä joukko.
- Haku, jossa kaikki kentät on täytetty. Sekä tyhjä, että tuottoisa tapaus.
- Erikseen testataan vielä tapaus, jossa filttereitä on hakuehdoissa useampia kuin yksi

4.2 getLightcurves

Metodi testataan samoin kuin getAsteroids. Aluksi siis testataan *tyhjä haku*, jonka ainoana ehtona on, että valokäyrien lukumäärä on yksi tai useampia. Tämän jälkeen tutkitaan tapaus, jossa valokäyrien lukumääräksi asetetaan arvo, joka on pienempi kuin yksi.

Tämän jälkeen hausta testataan vuorotellen alla luetellut tapaukset. Jokaisen tapauksen yhteydessä, jossa oletetaan haun toimivan testataan sekä tapaus, että tietokannasta löytyy ehdon täyttävä tapaus, että tapaus, jossa hakuehto ei täyty millään kannan alkiolla.

- Hakuehdoksi asetetaan null-viite
- Johonkin Suunnitteludokumentin luvusta 3.2 löytyvään kenttään asetetaan null-viite
- Suunnitteludokumentin luvusta 3.2 löytyvien lukuarvoehtojen mukaiset ylä- ja alarajat sekä ylärajat ylittävät ja alarajat alittavat arvot.

- Validi haku jokaisen suunnitteludokumentin luvusta 3.2 löytyvän kentän kohdalla erikseen. Testauksessa suoritetaan sekä tapaus, jossa haun tulisi tuottaa tulosta, että tapaus, jossa tuloksena on tyhjä joukko.
- Haku, jossa kaikki kentät on täytetty. Sekä tyhjä, että tuottoisa tapaus.
- Erikseen testataan vielä tapaus, jossa filttereitä on hakuehdoissa useampia kuin yksi

4.3 getTrajectories

Metodin vastaanottama long–tyypinen muuttuja ei anna testaukselle paljoakaan riippumavaraa. Metodin toimintaan testataan virheellisellä syötteellä (negatiivinen asteroidinumero), järjestelmästä löytyvällä asteroidinumerolla sekä oikealla, positiivisella asteroidnumerolla, jota järjestelmästä ei löydy.

4.4 makeDataFile

Metodi testataan ensin antamalla syötteenä null-arvot vuorotellen kuhunkin taulukkopaikkaan. Tämän jälkeen testataan tapaukset:

- Taulukoissa on eri määrät alkioita
- Kussakin taulukossa on yksi oikeellinen alkio ja system on TahitiLibrary.SYSTEM_MAGNITUDE
- Kussakin taulukossa on yksi oikeellinen alkio ja system on TahitiLibrary.SYSTEM_INTENSITY
- Kussakin taulukossa on yksi oikeellinen alkio ja system on -1
- Kussakin taulukossa on yksi alkio, ja lightcurve-taulun alkio on null-viite
- Kussakin taulukossa on yksi alkio, ja earthTrajectory-taulun alkio on null-viite
- Kussakin taulukossa on yksi alkio, ja asteroidTrajectory-taulun alkio on null-viite
- Kussakin taulukossa on yksi alkio, ja molempien Trajectory-taulujen alkio on null-viite
- Tauluissa on nolla alkiota
- Taulussa lightcurve on useampia kuin yksi alkio ja molempien Trajectory taulujen jokin alkio on null-viite

4.5 makeRawData

Metodi testataan siten, että joku syötteistä on null; ja siten, että syöttenä annetussa taulukossa tai AsteroidContainerissa on null–arvoja.

4.6 registrationRequest

Koska rekisteröintipyynnön suorittamiseen ei tarvita aikaisempia oikeuksia, keskitytään integraatiotestauksessa vain UserFormContainer–containerin sisältämien kenttien arvoihin, ja näiden arvojen eri variaatioihin. Normaalin käyttäjäpyynnön, jossa kaikki pakolliset kentät ovat täytteyinä testataan metodin toimintaa tilanteissa, joissa jokin pakollisista kentistä on null ja/tai tyhjä merkkijono. Lisäksi status–kenttää kokeillaan syöttää järjettömiä statuksia, joita järjestelmän ei tulisi hyväksyä.

Metodin toimintaa pitää myös testata myös vapaaehtoisten kenttien ollessa täytettyinä. Koska vapaaehtoisia kenttiä on viisi kappaletta, ei kaikkia erilaisia täyttöyhdistelmiä tulla erikseen testaamaan (tällöin testitapauksia olisi pelkästään näitä varten 5! (=120). Sen sijaan vapaaehtoisten kenttien toimintaa tulla testaamaan kuudella eri testillä, jossa viidessä ensimmäisessä yksi jokaisesta vapaaehtoisesta kentästä on täytettynä, ja kuudennessa tapauksessa kaikki kentät sisältävät tietoa.

Myös tilannetta, jossa haluttu kirjautumistunnus on jo käytössä tulee testata. Tällöin metodin pitäisi ilmoittaa käytössä olevasta tunnuksesta, sekä suorituksen keskeytyä. Muita tällaisia tilanteita, jotka vaativat erityistä testausta ovat sellaiset, jossa jokin kentistä, joista metodin ei tulisi välittää, ovat täytettyinä. Tälläisiä kenttiä ovat

- UserNumber (Järjestelmän sisäinen tunnus)
- Level (Käyttäjätaso, uusilla käyttäjillä aina nolla)
- Comment (Ylläpitäjän käyttäjästä antama kommentti)
- Version (Versionumero, uusilla käyttäjäpyynnöillä aina nolla)

Tämä toiminnallisuus voidaan testata tekemällä käyttäjärekisteröintipyyntö, jossa kaikki edellämainitut kentät ovat täytettyinä. Toivottu tulos on, että metodi kirjaa käyttäjäpyynnön tietokantaan, mutta ei välitä edellämainittujen kenttien sisällöstä.

4.7 login

Metodin toiminnallisuutta testataan kolmenlaisilla syötteillä; kelvollisilla, epäkelvoilla ja väärän tyyppisillä. Kelvollisessa syötteessa sekä tunnus että salasana ovat oikein, epäkelvoissa jompi kumpi tai molemmat ovat väärin ja väärän tyyppisessä syötteessä arvona on esim. null–viite. Oikeellisuus tarkoittaa tiedon löytymistä kannasta ja sen vastaavuutta toisen tiedon kanssa.

4.8 insertLightcurve

Metodin käyttö edellyttää syöttäjätason oikeuksia, joten sen käyttöä testataan ensin kaikilla kolmella eri käyttäjätasolla, kun syötteenä on oikeellinen yhden datapisteen ja aika ja visual sarakkeet sisältävä LightCurveForm-container. Tämän jälkeen testitapauksia varioidaan varioimalla LightcurveFormContainerin sisältöä seuraavasti. Jatkossa kenttiä AsteroidName, AsteroidNumber ja AsteroidDesignation kutsutaan *nimitiedoiksi*.

- Kaikki nimitiedot tyhjiä merkkijonoja.
- Nimitiedoista on annettu vain numero
- Nimitiedoista on annettu vain nimi
- Nimitiedoista on annettu vain tunnus
- Nimitiedot on asetettu siten, että numero vastaa yhtä asteroidia ja nimi jotakin toista
- ZeroTime kenttä on nolla tai pienempi kuin nolla (koska suunnitteludokumentissa oli näköjään virhe)
- Jokin pakollisista kentistä on null-viite tai tyhjä merkkijono
- Suunnitteludokumentin luvusta 3.5 lyötyvät ylä- ja alarajat sekä ne ylittävät että alittavat arvot

Kun ollaan varmistuttu, että metodi toimii muiden kuin data- ja columns- kenttien osalta, pidetään seuraavissa testeissä muut kentät vakioina ja varioidaan data ja columns-kenttiä. Testataan seuraavat tapaukset.

- Columns-taulukossa eri määrä sarakkeita kuin datakentässä
- Columns-taulukossa kaksi samaa saraketta
- Columns-taulukossa ei aikasaraketta
- Columns-taulukossa vakio, joka ei ole TahtitiLibraryssä
- Tyhjä datakenttä
- Tyhjiä rivejä oikeiden rivien välissä
- Yksi epäkelpo rivi
- Kaikki datapisteet ovat täsmälleen samanlaisia
- Oikeanlainen data usealla datapisteellä

Lopuksi testataan vielä täsmälleen samojen tietojen syöttämistä kaksi kertaa peräkkäin.

4.9 insertLightcurveAtlas

Metodi testataan muutoin kuin insertLightcurve, paitsi että ensimmäisessä testausvaiheessa testataan myös earth-, asteroid-, aspectTime- ja atlasAspect-kenttiä. Ensimmäisen testausvaiheen testien lisäksi edellä mainittuja kenttiä testataan doublen jollakin erityisarvolla (MAX_VALUE, MIN_VALUE, NaN, NEGATIVE_INFINITY, POSITIVE_INFINITY).

4.10 changePassword

Metodi vaatii syöttäjätason oikeudet, joten aluksi metodia testataan kaikilla käyttäjä- tasoilla oikeellisilla parametreilla eli oikein olevalla vanhalla salasanalla sekä 5-merkkisellä uudella salasanalla. Tämän jälkeen testataan tapaukset:

- Null-arvot parametreina
- Vanha salasana on väärin

4.11 getUsers

Metodin syöte sisältää vain etsittävän merkkijonon, joten testaus on helppo rajata. Testattavat tapaukset:

- Null–arvo parametrinä
- Merkkijono, jonka avulla löydetään ainakin yksi käyttäjä
- Merkkijono, jonka avulla ei löydetä käyttäjiä
- Eri tasoisilla käyttäjillä hakeminen

Metodin tulisi palauttaa null ensimmäisessä tapauksessa ja tyhjä Users–taulukko viimeisessä. Toisessa tapauksessa tuloksena olisi vähintään yhden alkion kokoinen Users–taulukko.

4.12 setUserData

Metodille annetaan parametrina UserFormContainer–olio, jonka kenttien kelvollisuudesta riippuu metodin paluuarvo. Testitapauksia ovat siis epäkelvoilla arvoilla täytetty UserFormContainer, null–viite ja kelvollinen UserFormContainer.

Lisää testattavaa aiheuttaa käyttäjäoikeusvaatimus. Metodia ei luonnollisestikaan voi käyttää selaajan oikeuksin, joten paluuarvon tulee olla tällöin false. Lisäksi pakolliset kentät, joita on suurin osa syötteestä, tulee olla täytettynä, tai muutos ei onnistu. Syötteen kentistä lisäksi UserID:n on oltava positiivinen.

Paluuarvo epäonnistuneissa muutoksissa on false. Onnistuneissa muutoksissa palautetaan true.

4.13 resetPassword

Testataan tapauksilla: loginID on null, loginID:ssä annettu käyttäjä on olemassa ja loginID:ssä annettua käyttäjää ei ole olemassa.

4.14 changeLightcurve

Valokäyrien muuntaminen on mahdollista vain ylläpitäjä–tason käyttäjille. Tämän takia metodin ensimmäisesti testitapaukset yksinkertaisesti koostuvat käyttöyrityksistä ilman tarvittavia oikeuksia. Metodin toimintaa pitää kokeilla ennen kirjautumista, sekä syöttäjän oikeuksilla. Kummassakaan tapauksessa suorittamisen ei pitäisi onnistua.

Annettava syöte, LightcurveFormContainer sisältää lukuisia kenttiä, jotka aiheuttavat runsaasti erilaisia testaustapauksia. Osa annettavista kentistä sisältää pakollista tietoa, joten metodin toimintaa tulee ainakin testata tilanteessa, jossa jonkin näistä pakollisista kentistä on tyhjä, tai sisältää pelkän null–viittauksen.

Valinnaiset kentät (Information, Reference, PhotometricSystem, Detector, TimeStandard) antavat myös erilaisia testaustapauksia. Testauksessa pitää ainakin olla tapaukset, jossa näitä tietoja ei alunperin tietokannassa ole, ja niille annetaan uudet arvot. Lisäksi tarvitaan tilanne, jossa tietokannassa olevat kentät sisältävät informaatiota, mutta uudet kentät eivät. Kolmas muutostyyppi saadaan, kun tietokannassa oleva informaation muuttuu. Nämä kaikki ehdot saadaan yhteen testitapaukseen.

Koska muutos voi koskea vain yhtä valokäyrään, täytyy metodin toimintaa myös testata sellaisessa tapauksessa, jossa syötteenä annettava data sisältää enemmän, kuin yhden valokäyrän havaintotietoja. Tällöin metodin ei pitäisi tehdä minkäänlaisia muutoksia tietokantaan. Lisäksi datapisteisiin liittyen pitää tehdä testejä, joissa käytettävä filtteri vaihtuu, mittausarvot vaihtuvat sekä mittausajat vaihtuvat. Näiden lisäksi tarvitaan myös testi tilanteesta, jossa mahdollinen error–kentässä jo oleva arvo katoaa, sekä tilanteesta, jossa errortieto talletetaan tietokantaan.

Koska changeLightcurve–metodilla voidaan myös vaihtaa asteroidin kohdetta, täytyy metodin toiminta testata tilanteissa, joissa uutta kohdetta ei löydy järjestelmästä lainkaan. Toinen vastaanvanoloinen testitapaus on havainnon muuttaminen siten, että sen kohteena on maapallo, eli asteroid 0.

Atlas–siirtimellä järjestelmään lisättyjä valokäyriä ei pitäisi pystyä muuttamaan, joten metodia pitää myös kokeilla tällaisen valokäyrän muuttamiseen. Lisäksi muuttamista pitää yrittää sellaiselle varlokäyrälle, jota järjestelmästä ei löydy laisinkaan, sekä väärälle versionumerolle, jolloin muutoksen ei myöskään pitäisi onnistua.

Koska tiettyjen kenttien (UnitOfTime, zeroTime, zeroMagnitude, columns–taulukon sisältö) annettavassa datassa tulee sisältää vain tietynlaisia arvoja, pitää metodin toimintaa testata oikeiden arvojen ulkopuolella olevilla arvoilla sekä rajoilla olevilla arvoilla. Seuraavassa on lueteltu tällaiset arvorajat:

- UnitOfTime

  Kentän unitOfTime tulee olla joko 0 (UNIT_DAY) tai 1 (UNIT_HOUR). Toimintaa kokeillaan näiden arvojen lisäksi arvolla 2.

- Magnitude

  Magnitude ei voi olla negatiivinen. Metodin toimintaa testataan tilanteessa, jossa jonkin havaintopisteen magnituden arvoksi tulee negatiivinen arvo.

- columns–taulukko

  Columns–taulkon sisältäminen arvojen tulee olla TahitiLibrarystä löytyviä vakioita, eli valideja arvoja ovat luvut väliltä 0...7. Lisäksi jokaisesta syötteenä annettavasta colums–taulokosta tulee löytyä aikasarake (0), sekä vähintään yksi datasarake. Error–saraketta (1) ei lasketa varsinaiseksi datasarakkeeksi. Tämän perusteella metodin toimintaa pitää ainakin kokeilla tilanteissa, jossa saraketiedot sisältävät sarakkeet -1 ja 8, sekä tilanteessa, jossa varsinainen havaintotieto puuttuu. Colums–taulukkoon yritetään myös laittaa ei–numeerista tietoa, jolloin suorituksen pitäisi keskeytyä.

4.15 logoff

Koska logoff-metodi ei tarvitse minkäänlaista syötettä, on sen testaaminen hyvin yksinkertaista. Metodin toimintaa tulee testata tilanteessa, joissa on ensiksi kirjauduttu sisään, sekä tilanteessa, jossa käyttäjä ei vielä ole kirjautunut sisään. Jotta metodin toiminnasta voidaan varmistua, tulee ennen ja jälkeen metodin suorittamista yrittää suorittaa jokin erillisiä käyttöoikeuksia tarvitseva metodi.

4.16 removeUser

Metodi tarvitsee syötteenä saamastaan User–oliosta vain käyttäjän järjestelmän sisäisen tunnistenumeron. Arvon oikeellisuudesta riippuu poiston onnistuminen. Tunnisteen on oltava positiivinen, minkä lisäksi kyseiseen numeroon sidotun käyttäjän on oltava olemassa järjestelmän tietokannassa. Myös tietokantayhteyden on oltava kunnossa, jotta operaatio voisi onnistua.

Testiaineiston on koostuttava ainakin null–viitteestä, epäkelvosta User–oliosta (epäkelpo tunniste) sekä kelvollisesta User–oliosta. Myös tietokantayhteyden tilan vaikutus tulokseen testataan.

Metodi palauttaa onnistuessaan arvon true. Epäonnistuessaan metodi palauttaa arvon false.

4.17 getSettings

Metodin toiminta edellyttää ylläpitäjätason oikeuksia, joten sen kutsumista testataan kaikilla käyttäjätasoilla. Testataan, että metodi palauttaa kantaan talletetut asetukset.

4.18 changeSettings

Metodia testataan ensiksi selaaja–, syöttäjä– ja ylläpitäjätason oikeuksilla siten, että syötteenä annettavan SettingsFormContainerin jokaiseen kenttään laitetaan validi arvo (so. kenttä ei ole null ja ServerPort>=0). Tämän jälkeen ylläpitäjätason oikeuksilla testataan seuraavat tapaukset:

- Syötteenä annettu newSettings on null
- Joku syötteenä annetun SettingsFormContainerin kentistä on null
- ServerPort<0

4.19 getLogEntries

Koska suunnitteludokumentin mukaan getLogEntries–metodin käyttämiseen tarvitaan ylläpitäjän oikeudet, testataan metodin toimintaa tilanteissa, joissa kirjatuneella käyttäjällä on syöttäjän oikeudet, sekä tilanteessa, jossa käyttäjä ei ole kirjautunut järjestelmään sisään laisinkaan.

Metodin syötteenään ottava logQueryContainer sisältää kaksi merkkijonokenttää, timeStart ja timeEnd. Metodin toimintaa tullaan testaamaan tilanteissa, joissa molemmat tai jompikumpi merkkijono on null tai tyhjä. Tällaisissa tilanteissa metodin tulisi toimia rajoittamatta, toisinsanoen, esimerkiksi timeEndin ollessa tyhjä, palautetaan kaikki kenttää timeStart myöhäisemmät havainnot. Luonollisesti metodin toimintaa testataan myös molempien kenttien ollessa täytettyinä.

Lisäksi kenttiin kokeillaan laittaa satunnaista tekstiä, jolloin päivämäärien purkaminen syötekentistä pitäisi olla mahdotonta, ja haun epäonnistua. Vaikkakaan metodin ei sinäänsä tarvitsekaan varautua tilanteeseen, jossa timeEnd on aikaisempi, kuin timeStart, testataan metodin toimintaa myös tällaisessa tilanteessa. Oletettavasti tällaisessa tilanteessa metodi palauttaa logEntry–taulukon, jossa ei ole yhtään alkiota.

4.20 deleteLightcurve

Metodille annetaan syötteenä vain valokäyrän tunnistenumero, joten testiaineisto rajoittuu kelvolliseen ja epäkelpoon lukuun. Onnistuessaan metodi palauttaa arvon true, muutoin false. Ensimmäisellä kerralla käyrä vain poistomerkitään, joten sen löytyminen kannasta voidaan tarkistaa tarkoitukseen sopivalla metodilla. Toisella kerralla käyrä poistetaan kokonaan järjestelmästä.

4.21 restoreDeletedLightcurve

Metodin käyttö vaatii ylläpitäjätason oikeudet, joten sen kutsumista testataan ensin kaikilla käyttäjätasoilla oikeellisella valokäyrän tunnuksella valokäyrälle, joka on merkattu poistetuksi. Sen jälkeen testataan seuraavat tapaukset:

- Valokäyrätunnusta ei löydy kannasta
- Valokäyrätunnuksen valokäyrää ei ole merkattu poistetuksi

4.22 addTrajectory

Metodi testataan seuraavilla eri tapauksilla:

- Syöte on null–viite
- Syötteenä annetun TrajectoryFormContainerin joku kenttä on null
- Filttereitä on liikaa tai liian vähän data–sarakkeisiin verrattuna
- Data ei ole muodoltaan kunnollista tai se arvot eivät ole suunnitteludokumentin kappaleen 3.11 asettamien raja–arvojen ulkopuolella (liian pieni/liian suuri).

4.23 spamRecommender

Myös spamRecommender–metodi saa syötteenä vain long–tyyppisen muuttujan, joka tässä tapauksessa edustaa käyttäjän sisäistä järjestysnumeroa. Metodin toimintaa tulee testata selvästi virheellisellä syötteellä (negativiinen käyttäjänumero), sellaisella positiviisella käyttäjänumerolla, jota järjestelmästä ei löydy, sekä järjestelmästä löytyvillä käyttäjänumeroilla. Lisäksi metodin toiminnan testaamiseksi tietokantaan tarvitaan käyttäjä, jonka ilmoittama suosittelijan sähköpostiosoite on epäkelpo.

Toinen testityyppi metodille saadaan sen tarvitsemista käyttöoikeuksista. Metodin toimintaa tulee testata tilanteessa, jossa sen käyttäjä ei vielä ole kirjautunut järjestelmään ollenkaan, sekä tilanteessa, jossa kirjautunut käyttäjä on käyttäjätasoltaan syöttäjä, eikä ylläpitäjä.

4.24 spamUser

Metodille annetaan syötteena vain käyttäjän järjestelmän sisäinen tunnistenumero, joten testi voidaan suorittaa kahdella lukuarvolla. Toinen on epäkelpo, toinen kelvollinen. Kelvollisen luvun ilmaisema käyttäjä löytyy järjestelmän tietokannasta, epäkelvon ei. Paluuarvo toiminnon onnistuessa on true, muutoin false.

5 Validointitestaus

Validointitestausta varten ryhmälle perustetaan koneelle db.cs.helsinki.fi ryhmähakemisto, jossa järjestelmätestaus suoritetaan. Näin tehdään, jotta testaus olisi mahdollisimman deterministä, ts. testitapausten ulkopuoliset ärsykkeet olisivat minimoituja. Käytännössä tätä kontrolloidaan siten, että ennen muutoksien tekemistä tietokantaan tai testiluokkiin tulee asiasta saada lupa ryhmän IRC–kanavalla olijoilta. Käytännössä tämä tarkoittaa sitä, että aina testausta suoritettaessa testaajien pitää olla tavattavissa ryhmän IRC–kanavalla (EFNet: #ohtu-tahiti).

Validointitestauksen testaustapaukset ovat jossain määrin samanlaisia integraatiotestauksen kanssa. Validointitestauksessa kuitenkin kiinnitetään enemmän huomiota järjestelmän toimiseen kokonaisuutena, ja testitapausten perustapauksina pidetään määrittelydokumentista löytyviä toimintoja. Validointitestauksen yksi päätavoitteista onkin tarkistaa, että kaikki määritellyt järjestelmän toiminnot toimivat oikealla tavalla. Lisäksi validointitestauksessa suoritetaan testausta integraatiotestauksessa esiintyvien tapausten mukaisesti, mutta testit suoritetaan käyttöliittymän kautta.

Validointitestauksessa myös tarkistetaan, että järjestelmä toimii määrittelydokumentissa mainituilla selaimilla. Näitä selaimia ovat Microsoft Internet Explorer 4.0, Netscape Navigator 4.0+ ja Mozilla 1.1.0.

5.1 Valokäyrien haku

Valokäyrän haun testauksen perustapaukset löytyvät integraatiotestauksen kohdasta getLightcurves. Näiden testitapausten ohessa validointitestauksessa kiinnitetään erityistä huomiota saatujen tietojen oikeellisuuteen. Testauksessa tarkistetaan, että tiedot näytetään käyttäjille juuri sellai-

sina, kuin ne tietokannassa ovat. Lisäksi tarkistetaan, että valokäyrää visualisoiva kuva näyttää oikealta.

5.2 Valokäyrien syöttäminen järjestelmään

Testauksen perustapaukset löytyvät integraatiotestauksen insertLightcurve–osiosta. Validointitestauksessa tarkistetaan, että lisätyt valokäyrän näkyvät järjestelmässä oikealla tavalla, ja että käyttöliittymän antamat virhetiedot vastaavat todellisuutta.

5.3 Valokäyrien tulostaminen

Testauksen perustapaukset löytyvät integraatiotestauksen makeDataFile–osiosta sekä makeRawFile–osiosta, riippuen kumman tulostusmuodon käyttäjä valitsi. Validointitestauksessa tarkistetaan, että muodostetut tiedostot tulevat muodostetuiksi käyttäjän valitsemien parametrien mukaisesti. Käyttäjä voi valita jokaiselle erilliselle valokäyrälle oman maarataelementin sekä asteroidirataelementin. Lisäksi laskettavan tulostiedoston yhteydessä käyttäjä saa määritellä haluaako hän tulostiedoston havaintopisteet intensiteetti vai magnitudimuodossa.

5.4 Valokäyrien muuttaminen

Testauksen perustapaukset ovat integraatiotestauksen changeLightcurve–osiossa. Validointitestauksessa tarkistetaan, että muutettu valokäyrä todellakin on muutettu oikein, ja että virheellisistä arvoista kerrotaan tyydyttävällä tavalla.

5.5 Omien tietojen muuttaminen

Käyttäjän on oltava rekisteröitynyt järjestelmään ja kirjautunut sisään muuttaakseen tietojaan. Tietojen muuttamisen toiminta on testattu setUserData metodin yhteydessä. On kiinnitettävä huomiota, ettei käyttäjä saa muutettua tietoja, joiden muuttamiseen ei hänellä ole oikeutta. Lisäksi käyttäjä ei saa edes nähdä ylläpidolle varattuja tietokenttiä.

5.6 Salasanan nollaaminen

Salasanan nollaamisessa käytetyt perustestitapaukset on käsitelty integraatiotestauksen resetPassword–metodin yhteydessä. Järjestelmätestauksessa katsotaan myös, että järjestelmän lähettämä sähköposti on oikeaa muotoa ja varmistetaan että uusi, muutettu salasana toimii järjestelmään kirjautumisessa.

5.7 Käyttäjien tietojen hakeminen

Käyttäjien tietojen hakemisen perustestitapaukset ovat käsitelty integraatiotestauksen getUserssekä getUsersFull–metodien yhteydessä. Validointitestauksen tarkoituksena on varmistaa että käyttäjän antaessa syötteenä osittaisen tekstin käyttäjän nimestä, järjestelmä palauttaa käyttäjälle

kaikki osittaista nimeä vastaavat käyttäjät sekä antaa käyttäjätasosta riippuen mahdollisuuden poistaa, muokata tai tarkastella valittavan käyttäjän tarkempia tietoja.

5.8 Käyttäjien lisääminen ja poistaminen järjestelmästä

Käyttäjien lisääminen järjestelmään koostuu ylläpitäjän tekemästä rekisteröinninpyynnöstä, ja tämän pyynnön hyväksymisestä. Koska nämä ovat erillisiä toimintoja, jotka testataan myös itsenäisinä operaatioina, ei käyttäjän lisääminen vaadi erillistä testitapausta.

Käyttäjän poistamisen perustestitapaukset löytyvät integraatiotestauksen removeUser–metodin yhteydestä. Validointitestauksessa myös tarkistetaan, että käyttäjä todella poistui järjestelmästä, ja että operaation aiheuttama lokimerkintä on oikea.

5.9 Asetusten muuttaminen

Asetusten muuttaminen on käyty aiemmin läpi metodin changeSettings osalta. Testiin kuuluu kelpojen ja epäkelpojen arvojen läpimenon tarkistaminen. Lisäksi pakolliset kentät AdminEMail, MailServer ja ServerPort tulee olla täytettyinä.

5.10 Järjestelmän tapahtumien selaaminen

Järjestelmän tapahtumien selaamisen perustestit ovat kuvattu metodin getLogEntries integraatiotestauksen yhteydessä. Validointitestauksessa tarkastetaan, että käyttäjän antama aikahaarukka vastaa ruudulle ilmestyvien tapahtumien aikahaarukkaa. Myös varmistetaan tyhjän aikahaarukan hakevan kahden edellisen kuukauden tapahtumat.

5.11 Kirjautuminen

Testitapaukset esitellään integraatiotestauksen login–osiossa. Testattavana ovat kelvolliset ja epäkelvolliset syötteet.

5.12 Käyttäjätietojen muuttaminen

Käyttäjätietojen muuttaminen on vastaa integraatiotestauksen setUserData–metodin testausta. Testi muodostuu kelvollisten ja epäkelvollisten arvojen läpimenon tarkastamisesta.

5.13 Rataelementtien lisääminen

Testataan kuten integraatiotestauksen addTrajectory. Validointitestauksessa tarkistetaan, että oikeanmuotoinen aineisto päätyy tietokantaan, ja vääränmuotoinen aineisto hylätään virheilmoituksin.

5.14 Atlas–syötin

Koska Atlas–syötin toteutetaan varsinaisen järjestelmän ulkoisena komponenttina, ei sen testaamisprioriteetti ole kaikista korkein. Lisäksi kaikki aineisto, jota syöttimellä tullaan käsittelemään on jo olemassa, eikä uutta Atlas–muodossa olevaa aineistoa pitäisi enää tulla. Näiden seikkojen vuoksi Atlas–syöttimen testaus tapahtuu suoraviivaisesti syöttimen käyttämisellä. Ajon jälkeen tarkistetaan syöttimen tietokantaan laittamat havainnot ja tarkistetaan siirrettyjen tietojen oikeellisuus.

Myös tiedostot, jotka Atlas–syötin hylkää tarkistetaan pistokokein. Tällä pyritään varmistaan, ettei syötin hylkää oikean muotoisia tiedostoja.

5.15 Rekisteröintipyynnön täyttäminen

Rekisteröintipyynnön tekemisen testitapaukset löytyvät integraatiotestaus-luvusta alaotsikon registerationRequest alta. Kuitenkin siten, että versionhallintaan liittyviä asioita ei testata validointitestauksen yhteydessä.

5.16 Käyttäjien hyväksyminen tai hylkääminen

Testataan hyväksymällä ja hylkäämällä rekisteröintipyyntö. Hyväksyntä ja hylkääminen tapahtuu nappia painamalla.

5.17 Valokäyrän poistaminen

Valokäyrän poistamista testataan ensinnäkin tapauksessa, jolloin käyttäjä ei ole ylläpitäjä eikä ole lisännyt valokäyrää järjestelmään. Tällöin poiston ei tulisi onnistua. Tämän jälkeen testataan tapaus, jossa käyttäjä ei ole ylläpitäjä, mutta on syöttänyt poistettavan valokäyrän järjestelmään. Tällöin valokäyär pitäisi poistomerkitsemään, muttei kokonaan poistamaan. Lopuksi testataan, että ylläpitäjä voi sekä poistomerkitä, että poistaa valokäyriä järjestelmästä.