KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

Jussi Hukkanen

REST API:N OPTIMOINTI JA PÄIVITYS OPENAPI 3:EEN

Opinnäytetyö
Joulukuu 2018

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Joulukuu 2018
Tietojenkäsittelyn koulutusohjelma

Tikkarinne 9
80200 JOENSUU
+358 13 260 600 (vaihde)

Tekijä
Jussi Hukkanen

Nimeke
REST API:n optimointi ja päivitys OpenAPI 3:een

Toimeksiantaja
Nasta Technologies Oy

Tiivistelmä

REST-arkkitehtuurimallin mukaiset ohjelmointirajapinnat ovat verkkosovellusten ja pilvipalvelujen keskeisiä rakennuspalikoita. Tässä opinnäytetyössä paranneltiin kiinteistöomaisuuden hallintaan tarkoitetun Assetti-pilvipalvelun REST-rajapinnan teknistä toteutusta. Rajapinta oli OpenAPI Specification 2:n mukainen ja prosessin yhtenä toimenpiteenä se päivitettiin vastaamaan OpenAPI:n versiota 3. Parannustoimenpiteiden tavoitteena oli tehdä rajapinnan laajentamisesta helpompaa ohjelmistokehittäjälle.

Työssä tutustuttiin OpenAPI 3:n uusiin ominaisuuksiin ja tärkeimpiin eroihin verrattuna edelliseen versioon. Lisäksi tutkittiin erilaisia työkaluja rajapintakuvauksen muuntamiseen OpenAPI 2:sta 3:een ja ohjelmakoodin automaattiseen tuottamiseen kuvauksen pohjalta. Assetti API optimoitiin helpommin hallittavaksi jakamalla rajapintakuvaus osiin, selkeyttämällä hakemistorakennetta ja poistamalla päällekkäistä koodia. Tämän jälkeen selvitettiin, millaisia muutoksia rajapinnan toteutuksessa tarvittaisiin, jotta se ei enää olisi riippuvainen omista erillisistä tietuerakenteistaan vaan voisi paremmin hyödyntää samaa koodia kuin Assetin muut osat.

Opinnäytetyön tuloksista ilmeni, ettei OpenAPI 3:een päivittäminen yksin tuottanut Assetti API:n tapauksessa merkittäviä hyötyjä. Sen sijaan muut optimointitoimet tekivät rajapinnan ylläpidosta ja laajentamisesta aiempaa helpompaa. Etenkin rajapinnan omien tietuerakenteiden karsiminen nähtiin tärkeänä parannuksena. Opinnäytetyön yhteydessä toteutettiin myös testaustyökalu, jonka avulla rajapinnan toimivuus voitiin varmistaa muutostöiden aikana ja niiden jälkeen.

Kieli
suomi

Sivuja 39
Liitteitä 2
Liitesivumäärä 13

Asiasanat
verkkosovellus, API, REST, OpenAPI

Karelia AMMATTIKORKEAKOULU

THESIS
December 2018
Degree Programme in Business Information Technology

Tikkarinne 9
FI 80200 JOENSUU
FINLAND
Tel. +350 13 260 600 (switchboard)

Author
Jussi Hukkanen

Title
Optimising a REST API and Upgrading to OpenAPI 3

Commissioned by
Nasta Technologies Oy

Abstract

Application programming interfaces based on the REST architecture model are basic building blocks of modern web services. In this thesis, different technical improvements were made to the REST API provided by Assetti, a cloud service for managing property investments. The original version adhered to OpenAPI Specification 2, and as part of the process, it was upgraded to be based on OpenAPI 3. The goal of the improvements was to make expanding the API easier for software developers.

The new features of OpenAPI 3 were explored as well as the most significant differences from the previous version. Software options for converting API definitions from OpenAPI 2 to 3 and generating source code automatically were investigated. Assetti API was optimised for easier maintainability by splitting the definition into several files, using a clear directory structure and reducing duplicate code. Further investigation was conducted about the possibility of utilising more of the existing code base of Assetti in the API, so that the API would not require its own separate modelling of database entities.

The results of the thesis indicated that Assetti API did not benefit greatly from upgrading to OpenAPI 3 alone. However, the other measures taken made it more easily maintainable and expandable. The solution found for discarding the API's own database entity models had a particularly positive impact. A dedicated testing tool was written as part of the thesis and this was used to verify the correct operation of the API both during and after the optimisation process.

Language
Finnish

Pages: 39
Appendices: 2
Pages of Appendices: 13

Keywords
web application, API, REST, OpenAPI

Sisältö

Liitteet

Lyhenteet

API — Application Programming Interface (suom. ohjelmointirajapinta) määrittelee, miten sovellusten välinen tiedonsiirto tapahtuu tai miten ohjelmistokirjastoa käytetään ohjelmistokehitystyössä (Gazarov 2016).

CRUD — Create, Retrieve, Update and Delete, neljä tietokannan perustoimintoa eli tietueiden luonti, haku, päivitys ja poisto (Heller 2007).

DTO — Data Transfer Object, olio-ohjelmoinnissa tiedonsiirtoa varten rakennettu luokka (Fowler 2003).

HTTP — Hypertext Transfer Protocol, World Wide Webissä tapahtuvaan tiedonsiirtoon käytetty protokolla (Fielding, Gettys, Mogul, Frystyk, Masinter, Leach & Berners-Lee 1999).

JSON — JavaScript Object Notation, tietorakenteiden kuvaamiseen ja tiedonsiirtoon tarkoitettu syntaksi, jota sekä ihmisten että tietokoneiden on helppo lukea (Json.org 2018).

JWT — JSON Web Token, verkkosovellusten hyödyntämä tunnistautumismenetelmä, jossa käyttäjän autentikointiin tarvittavat tiedot esitetään kompaktin merkkijonon muodossa (Stecky-Efantis 2016).

OAI — OpenAPI Initiative, Linux Foundationin alaisuuteen vuonna 2015 perustettu yhteenliittymä, joka ylläpitää OpenAPI Specificationia (OAI 2018).

OAS — OpenAPI Specification, eniten käytetty avoimen lähdekoodin ratkaisu REST-rajapintojen toteutukseen. OAS tunnettiin ennen nimellä Swagger Specification. (OAI 2018.)

OAS 2 — OpenAPI Specification 2.0.

OAS 3 — OpenAPI Specification 3.0.0.

REST — Representational State Transfer, tietoverkoissa toimivien sovellusten suunnitteluun tarkoitettu arkkitehtuurimalli, jossa tiedonsiirtoon käytetään HTTP-protokollaa (Elkstein 2008).

REST API — REST-mallin mukainen ohjelmistorajapinta.

URI — Uniform Resource Identifier, verkossa olevan resurssin tunniste (The World Wide Web Consortium 2004).

URL — Uniform Resource Locator on URI, joka määrittelee verkossa olevan resurssin sen sijainnin ja käsittelytavan kautta (The World Wide Web Consortium 2001).

UUID — Universally Unique Identifier, 128 bittiä pitkä universaali tunniste, jota käyttämällä esimerkiksi tietokannan tietueet tai tietoverkon resurssit voidaan yksilöidä niin, ettei sama UUID käytännössä voi esiintyä kahta kertaa (International Telecommunication Union 2018).

XML — Extensible Markup Language, kieli, jolla kuvataan tietoverkossa käsiteltävien tietueiden rakennetta. XML:n elementtejä ei ole määritelty valmiiksi, vaan ne voidaan keksiä sovellusalueen ja tilanteen mukaan. (Myer 2005.)

YAML YAML Ain’t Markup Language, datan serialisointikieli, joka pyrkii helppolukuisuuteen ja tehokkuuteen (Yaml.org 2006).

1 Johdanto

Web-palvelujen toteutuksessa Representational State Transfer eli REST on 2000-luvun edetessä vakiintunut *de facto* -ratkaisuksi. Se on Roy Thomas Fieldingin väitöskirjassaan vuonna 2000 esittelemä arkkitehtuurimalli, jossa tiedonsiirto tapahtuu HTTP-protokollaa hyödyntäen (Elkstein 2008). Tavanomaiset CRUD-operaatiot, joilla tietokannan tietueita luodaan, haetaan, muokataan ja poistetaan, suoritetaan REST-mallissa käyttämällä HTTP:n POST-, GET-, PUT- ja DELETE-verbejä (Heller 2007).

Linux Foundationin alaisen OpenAPI Initiativen hallinnoima OpenAPI Specification (OAS) määrittelee tavan kuvailla REST-mallin mukaisia järjestelmärajapintoja. OAS-järjestelmäkuvaukset kirjoitetaan JSON- tai YAML-muotoon, joten ne ovat helposti luettavissa ja muokattavissa (OAI 2017a). Palvelin- ja asiakassovellusten koodin runko voidaan generoida OAS-kuvauksen pohjalta automaattisesti käyttämällä esimerkiksi OpenAPI Generator -ohjelmaa, joka kykenee tuottamaan ohjelmakoodia kymmenillä eri ohjelmointikielillä (OpenAPI Tools 2018a). Monet järjestelmät tarjoavat julkisen OAS-muotoisen API-kuvauksen, ja automaattinen koodin generointi helpottaa tällaisten järjestelmien kanssa kommunikoivien asiakassovellusten kehittämistä. Julkisia rajapintakuvauksia on koottu hakemistoihin kuten APIs.guruun. (APIs.guru 2017.)

Tässä opinnäytetyössä tehtiin erilaisia parannuksia kiinteistöomaisuuden hallintaan tarkoitetun Assetti-pilvipalvelun OAS-pohjaiseen API:hin. Uusia toimintoja ei lisätty eikä muutoksia tehty API:n ulospäin näkyviin osiin, vaan työ rajattiin koskemaan API:n teknistä toteutustapaa, jonka optimoinnilla pyrittiin helpottamaan uusien toimintojen lisäämistä myöhemmin.

Työssä haettiin vastauksia seuraaviin kysymyksiin:

1. Millä keinoilla Assetti API:n teknistä toteutusta voitaisiin parannella, jotta sen ylläpito ja uusien toimintojen lisääminen olisi kehittäjien kannalta helpompaa?
    a. Miten OAS 2:sta OAS 3:een päivittäminen hyödyttäisi ylläpidettävyyttä?
    b. Millaisia muutoksia tarvittaisiin, jotta API voisi hyödyntää Assetin koodissa käytettyjä tietueluokkia?
2. Miten varmistetaan, että Assetti API on optimointien jälkeenkin täydellisesti yhteensopiva olemassa olevien asiakassovellusten kanssa?

Tämän tekstin toinen luku luo katsauksen OAS 3:n uusiin ominaisuuksiin, minkä jälkeen kuvaillaan opinnäytetyön käytännön osuus. Työ toteutettiin kahdessa vaiheessa. Luvussa 3 kuvataan ensimmäistä vaihetta, johon edellä mainittu kysymys 1a liittyy ja jossa API-kuvaus päivitettiin OAS 3:n mukaiseksi. Tämän jälkeen aloitettiin toinen vaihe, josta kerrotaan luvussa 4. Siinä etsittiin vastausta kysymykseen 1b. Kysymys 2 kulki taustalla mukana opinnäytetyöprosessin alusta loppuun.

2 OpenAPI 2.0 ja 3.0.0

OAS:n ensimmäinen versio kehitettiin vuonna 2010 Wordnik-nimisen yrityksen sisäisenä projektina ja vuodesta 2015 alkaen sen kehitystyöstä on vastannut useiden yritysten ja muiden toimijoiden muodostama OpenAPI Initiative (OAI 2018). Vuonna 2017 julkaistiin uusin 3.0.0-versio (OAI 2017b) ja tässä luvussa käydään läpi joitakin tämän version uusista ominaisuuksista, jotka ovat tämän opinnäytetyön kannalta keskeisiä. Tarkoituksena ei siis ole tehdä kattavaa selvitystä kaikesta siitä, mikä uudessa versiossa on muuttunut. Olemassa olevia OAS 2 -muotoisia API-kuvauksia on mahdollista muuntaa OAS 3 -muotoon käyttämällä esimerkiksi Mermade Softwaren kehittämää muunnostyökalua (Mermade Software 2018).

2.1 Nimeämis- ja versiointikäytäntö

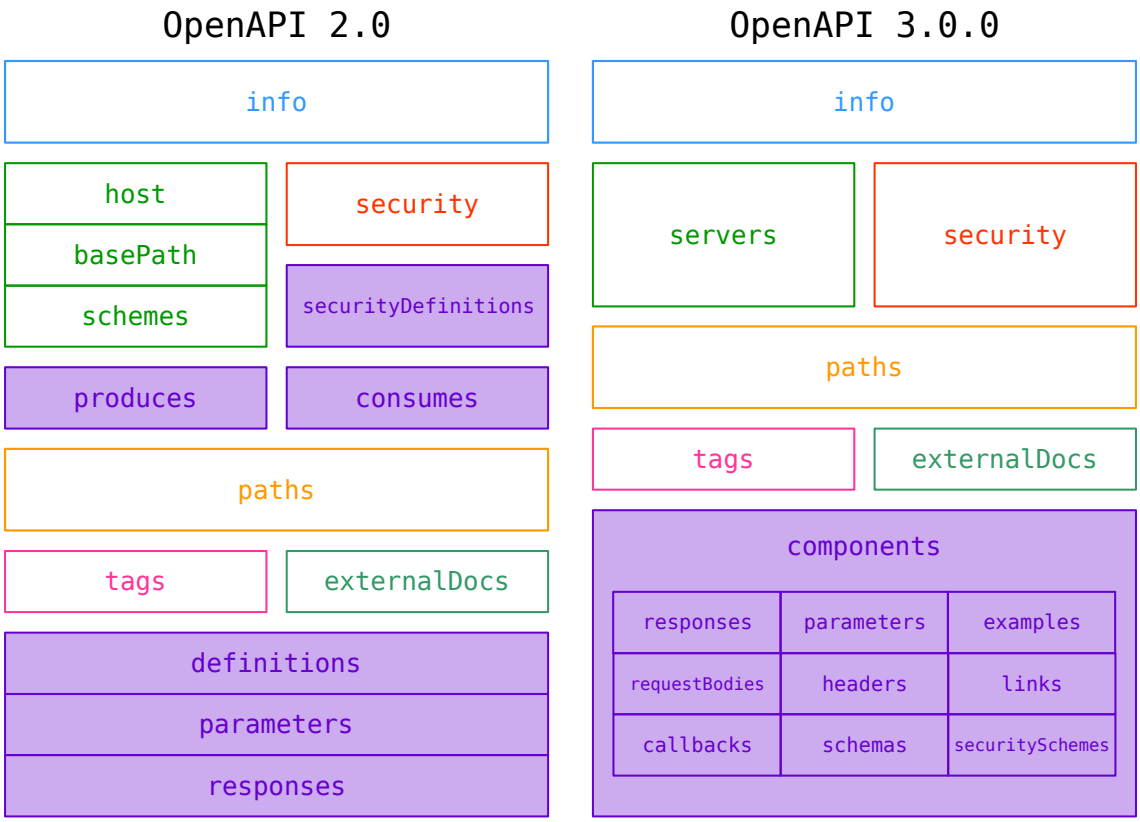
OpenAPI kehitettiin alun perin Swagger-spesifikaation nimellä ja uudessa versiossa nimeksi on vakiintunut OpenAPI Specification (Koberger 2017). Swagger-nimellä viitataan nykyään vain SmartBear Softwaren Swagger-ohjelmistoon, jonka työkaluilla kehitetään, ylläpidetään ja dokumentoidaan OAS:n mukaisia API-kuvauksia (SmartBear Software 2018a).

Nimenvaihdoksesta johtuen OAI suosittelee, että API-kuvauksen sisältävän tiedoston nimenä olisi tiedostomuodosta riippuen joko *openapi.json* tai *openapi.yaml* (OAI 2017c). Tämä eroaa aiempien versioiden käytännöstä, jossa kuvaustiedoston nimi oli yleensä *swagger.json* tai *swagger.yaml*.

OAS:n 2.x-versioperheeseen ei ole koskaan julkaistu päivitysversioita (kuten 2.1), mutta jos niin olisi tehty, käytössä ollut melko umpimähkäinen versionumerointitapa ei olisi pystynyt yksiselitteisesti ilmaisemaan, onko uusi versio yhteensopiva vanhan kanssa. Versiossa 3.0.0 numeroinnista on tehty johdonmukaisempaa ottamalla käyttöön niin sanottu semanttinen versiointi, jossa versionumero on muotoa *”major.minor.patch”*. (Miller 2016.) Ensimmäinen osa eli *major* muuttuu vain silloin, kun spesifikaatiosta julkaistaan uusi, aiempien kanssa epäyhteensopiva versio. Keskimmäistä eli *minor*-osaa korotetaan, kun spesifikaatioon lisätään uusia ominaisuuksia yhteensopivuuden kärsimättä. Kolmas osa eli *patch* kasvaa yhdellä silloin, kun esimerkiksi spesifikaation sanamuotoja korjataan. (Ralphson 2017a.)

2.2 Muutokset tiedostorakenteessa

OAS 3:n mukainen JSON- tai YAML-muotoinen API-kuvaustiedosto on päällisin puolin melko samannäköinen kuin vastaava OAS 2:n mukainen versio, mutta monet niistä elementeistä, joista kuvaus koostuu, ovat vaihtaneet nimeä, sisällön muotoilua tai molempia (Ralphson 2017a). Kuvassa 1 on esitetty OAS 2:n ja OAS 3:n rakenteelliset erot.



Kuva 1. OpenAPI 2.0:n ja 3.0.0:n erot. Toisiaan vastaavat elementit on värikoodattu. Ne OAS 2:n juuritason elementit, jotka OAS 3:ssa on koottu *components*-elementin sisään, on merkitty violetilla. (Koberger 2017.)

OAS 2 salli API:n osoitteen määrittämisen vain palvelinnimen, juuriosoitteen ja protokollan perusteella, ja nämä kaikki olivat kuvaustiedostossa juuritason elementtejä (*host*, *basePath* ja *schemes*). OAS 3:ssa vastaavat tiedot esitetään erillisen palvelinmallin muodossa, ja näitä malleja voi olla useita. (Miller 2016.) Ne luetellaan listamuodossa *servers*-elementin sisällä, joka sijaitsee joko kuvaustiedoston juuritasolla tai tarpeen mukaan yksittäisten toimintojen kuvausosioissa. API:n eri toimintoihin tulevat kutsut on näin mahdollista ohjata eri palvelimille. API:n URI:n ei myöskään ole pakko olla staattinen, vaan se voidaan esittää kaavan muodossa ja se voi sisältää vaihtoehtoisia osia. (Koberger 2017.) Kuvissa 2a ja 2b on esimerkit saman API:n palvelinmäärityksestä OAS 2:n ja 3:n muodossa.

```yaml
swagger: '2.0'
info:
  title: Authentiq Connect API
  version: '1.0'
  description: >
    Authentiq Connect OAuth 2.0 and OpenID Connect API reference.

    Learn about [Authentiq ID](https://www.authentiq.com/) or check out the
    [Authentiq Connect](https://developers.authentiq.com) developer
    documentation.
  termsOfService: 'https://www.authentiq.com/tos/'
  contact:
    name: Team Authentiq
    url: 'https://www.authentiq.com/'
    email: hello@authentiq.com
host: connect.authentiq.io
basePath: /
schemes:
  - https
consumes:
  - application/x-www-form-urlencoded
  - application/json
produces:
  - application/x-www-form-urlencoded
  - application/json
  - application/problem+json
  - text/html
```

Kuva 2a. Authentiq Connect API:n kuvauksen alkuosa OAS 2 -muodossa. (Ralphson 2017b.)

```yaml
openapi: 3.0.0-RC1
servers:
  - url: 'https://connect.authentiq.io/'
info:
  title: Authentiq Connect API
  version: '1.0'
  description: >
    Authentiq Connect OAuth 2.0 and OpenID Connect API reference.

    Learn about [Authentiq ID](https://www.authentiq.com/) or check out the
    [Authentiq Connect](https://developers.authentiq.com) developer
    documentation.
  termsOfService: 'https://www.authentiq.com/tos/'
  contact:
    name: Team Authentiq
    url: 'https://www.authentiq.com/'
    email: hello@authentiq.com
```

Kuva 2b. Authentiq Connect API:n kuvauksen alkuosa OAS 3 -muodossa. (Ralphson 2017c.)

Yksi tärkeimpiä OAS 3 -kuvauksen juuritason uusia elementtejä on *components*. Sen sisään on koottu useita muita elementtejä, jotka OAS 2:ssa olivat juuritasolla erillään, sekä tällaisia elementtejä merkitykseltään korvaavia uusia elementtejä. Tällä helpotetaan API-kuvauksen sisältämien objektien uudelleenkäytettävyyttä. Yksi keskeisimmistä elementeistä *components*:ssa on *schemas*, joka vastaa OAS 2:n juuritason *definitions*-elementtiä. Siinä määritellään API-kutsuissa lähetettävät ja palautettavat tietorakenteet eli skeemat. Varsinaisten skeemamääritysten rakenne on pitkälti samanlainen molemmissa versioissa, kuten kuvissa 3a ja 3b on esitetty. API-kutsujen vastaukset voidaan muotoilla *responses*-elementissä samalla syntaksilla, jota skeemamäärityksissä käytetään. (Koberger 2017.)

```
definitions:
  Token:
    description: |
      Successful token response
    required:
      - token_type
    properties:
      token_type:
        type: string
      access_token:
        description: The access token issued by the authorization server.
        type: string
      id_token:
        description: ID Token value associated with the authenticated session.
        type: string
      refresh_token:
        description: 'The refresh token issued to the client, if any.'
        type: string
      expires_in:
        description: The lifetime in seconds of the access token.
        type: integer
        format: int32
      expires_at:
        description: The time the access token will expire in seconds since epoch.
        type: integer
        format: int64
      scope:
        description: The scope of the granted tokens.
        type: string
  OAuth2Error:
    description: |
      Error Response defined as in Section 5.2 of OAuth 2.0 [RFC6749].
    required:
      - error
    properties:
      error:
        type: string
      error_description:
        type: string
```

Kuva 3a. Authentiq Connect API:n OAS 2 -muotoisen kuvauksen *definitions*-elementin alkuosa (Ralphson 2017b).

```
components:
  schemas:
    Token:
      description: |
        Successful token response
      required:
        - token_type
      properties:
        token_type:
          type: string
        access_token:
          description: The access token issued by the authorization server.
          type: string
        id_token:
          description: ID Token value associated with the authenticated session.
          type: string
        refresh_token:
          description: 'The refresh token issued to the client, if any.'
          type: string
        expires_in:
          description: The lifetime in seconds of the access token.
          type: integer
          format: int32
        expires_at:
          description: The time the access token will expire in seconds since epoch.
          type: integer
          format: int64
        scope:
          description: The scope of the granted tokens.
          type: string
    OAuth2Error:
      description: |
        Error Response defined as in Section 5.2 of OAuth 2.0 [RFC6749].
      required:
        - error
      properties:
        error:
          type: string
        error_description:
          type: string
```

Kuva 3b. Authentiq Connect API:n OAS 3 -muotoisen kuvauksen *schemas*-elementin alkuosa *components*-elementissä (Ralphson 2017c).

2.3 API-kutsut ja vastaukset

OAS 2 mahdollistaa tiedon välittämisen API-kutsussa useilla eri tavoilla. REST-mallin yleinen käytäntö on, että kutsun URI ja sen lopussa olevat parametrit kertovat, mitä halutaan tehdä ja miten. Esimerkiksi GET-kutsu osoitteeseen */users/10042?format=yaml&locale=EN* palauttaisi käyttäjän numero 10042 tiedot englanninkielisenä YAML-tiedostona. Kutsussa voidaan käyttää myös erikseen räätälöityjä HTTP-otsikkoja. (OAI 2017d.)

Kaikki edellä mainitut voitiin OAS 2:ssa luetella API-kutsun *parameters*-elementissä, mutta sen sisällä oli määritettävä myös POST- ja PUT-kutsuissa tarvittava HTTP-viestin runko tai web-lomakkeella lähetettävä data ikään kuin parametrina muiden joukossa. Lisäksi OAS 2 salli kutakin kutsua kohden vain yhden sisältötyypin, kuten JSON, XML tai pelkkä teksti. Näistä rajoituksista johtuen OAS 3 tarjoaa API-kutsun rungon kuvaamiseen erityisen *requestBody*-elementin. API-toiminnon vastaanottaman datan muoto voidaan määritellä sen sisäisessä *content*-elementissä jokaiselle sisältötyypille erikseen silloin, kun toiminnon halutaan tukevan useampaa kuin yhtä sisältötyyppiä. Myös lomakedata toimii *requestBody*:n sisältötyyppinä. (Koberger 2017.) Kuvissa 4a ja 4b on esimerkki POST-verbillä lähetettävän lomakedatan käsittelystä OAS 2:ssa ja 3:ssa.

```
/token:
  post:
    summary: Obtain an ID Token
    description: >
      Exchange en authorization code for an ID Token or Access Token.


      This endpoint supports both `client_secret_post` and
      `client_secret_basic` authenticatino methods, as specified by the
      client's `token_endpoint_auth_method`.


      See also:
        - [OIDC Token Endpoint](http://openid.net/specs/openid-connect-core-1_0.html#TokenRequest)
        - [OIDC Successful Token response](http://openid.net/specs/openid-connect-core-1_0.html#TokenResponse)
        - [OIDC Token Error response](http://openid.net/specs/openid-connect-core-1_0.html#TokenError)
    tags:
      - Authentication
    operationId: token
    consumes:
      - application/x-www-form-urlencoded
    parameters:
      - name: Authorization
        in: header
        description: |
          HTTP Basic authorization header.
        required: false
        type: string
      - name: client_id
        in: formData
        description: |
          The registered client ID.
        required: true
        type: string
      - name: client_secret
        in: formData
        description: |
          The registered client ID secret.
        required: true
        type: string
        format: password
      - name: grant_type
        in: formData
        description: |
          The authorization grant type, must be `authorization_code`.
        required: true
        type: string
      - name: code
        in: formData
        description: >
          The authorization code previously obtained from the Authentication
          endpoint.
        required: true
        type: string
      - name: redirect_uri
        in: formData
        description: >
          The redirect URL that was used previously with the Authentication
          endpoint.
        required: true
        type: string
    produces:
      - application/json
    responses:
      '200':
        $ref: '#/responses/Token'
      '400':
        $ref: '#/responses/OAuth2Error'
      '401':
        $ref: '#/responses/OAuth2Error'
    externalDocs:
      description: OpenID Token Endpoint
      url: 'http://openid.net/specs/openid-connect-core-1_0.html#TokenEndpoint'
```

Kuva 4a. Osa Authentiq Connect API:n kuvauksen *paths*-elementistä OAS 2 -muodossa. Lähetettävän viestin rakenne kuvaillaan *formData*-muotoisilla parametreilla. (Ralphson 2017b.)

```
/token:
  post:
    summary: Obtain an ID Token
    description: >
      Exchange en authorization code for an ID Token or Access Token.


      This endpoint supports both `client_secret_post` and
      `client_secret_basic` authenticatino methods, as specified by the
      client's `token_endpoint_auth_method`.


      See also:
        - [OIDC Token Endpoint](http://openid.net/specs/openid-connect-core-1_0.html#TokenRequest)
        - [OIDC Successful Token response](http://openid.net/specs/openid-connect-core-1_0.html#TokenResponse)
        - [OIDC Token Error response](http://openid.net/specs/openid-connect-core-1_0.html#TokenError)
    tags:
      - Authentication
    operationId: token
    parameters:
      - name: Authorization
        in: header
        description: |
          HTTP Basic authorization header.
        required: false
        schema:
          type: string
    responses:
      '200':
        $ref: '#/components/responses/Token'
      '400':
        $ref: '#/components/responses/OAuth2Error'
      '401':
        $ref: '#/components/responses/OAuth2Error'
    externalDocs:
      description: OpenID Token Endpoint
      url: 'http://openid.net/specs/openid-connect-core-1_0.html#TokenEndpoint'
    requestBody:
      content:
        application/x-www-form-urlencoded:
          schema:
            type: object
            properties:
              client_id:
                description: |
                  The registered client ID.
                type: string
              client_secret:
                description: |
                  The registered client ID secret.
                type: string
                format: password
              grant_type:
                description: |
                  The authorization grant type, must be `authorization_code`.
                type: string
              code:
                description: >
                  The authorization code previously obtained from the
                  Authentication endpoint.
                type: string
              redirect_uri:
                description: >
                  The redirect URL that was used previously with the
                  Authentication endpoint.
                type: string
            required:
              - client_id
```

Kuva 4b. Osa Authentiq Connect API:n kuvauksen *paths*-elementistä OAS 3 -muodossa. Lähetettävän viestin rakenne kuvaillaan kokonaan *requestBody*-osiossa. (Ralphson 2017c.)

API-toiminnot voivat OAS 3:ssa myös palauttaa erimuotoisia vastauksia riippuen siitä, mitä muotoja asiakassovellus tunnistaa. Tähän tarkoitukseen käytetään samankaltaista *content*-elementtiä kuin *requestBody*:ssa. (Ralphson 2017a.)

Tiedostojen lähettäminen on tehty OAS 3:ssa helpommaksi. Lomakedatan käyttö tähän tarkoitukseen on edelleen mahdollista ja jopa välttämätöntä silloin, kun halutaan lähettää useita tiedostoja yhdellä kertaa tai liittää metadataa tiedoston mukaan. Yksinkertaisemmissa käyttötapauksissa riittää, että *requestBody*:n *content*-elementissä määritetään yksi tai useampi sallittu tiedostomuoto. (SmartBear Software 2018b.)

3 Assetti API:n päivitys, ensimmäinen vaihe: OpenAPI 3.0.0

Tässä luvussa kuvataan opinnäytetyön käytännön osan ensimmäinen vaihe alkaen lähtötilanteen määrittelystä ja suunnitelluista parannuksista.

3.1 Lähtötilanne

Assetin REST API luotiin vuonna 2016. Se perustui OpenAPI 2.0 -määritykseen ja sen toteutuksessa käytettiin OAS:lle läheistä sukua olevaa Swagger-välineistöä. API:hin on tämän jälkeen lisätty uusia toimintoja, mutta OAS 2 -pohja on pysynyt muuttumattomana. Liitteessä 1 luetellaan API:n kaikki tämänhetkiset toiminnot.

API:n laajentamista on toisinaan hankaloittanut se, että API-kuvauksessa on melko paljon toistoa ja päällekkäisyyksiä. Luvuissa 3.5 ja 4 kuvataan toimenpiteitä, joilla toistoa pyrittiin vähentämään OAS 3:een päivittämisen jälkeen.

3.2 Prosessin simulointi

Assetti API:n resurssivalikoima sisältää muiden muassa kiinteistöjä, tontteja, huoneistoja, vuokrasopimuksia, yrityksiä ja yhteyshenkilöitä. Koska prosessia aloitettaessa oli epävarmaa, kuinka suuri työ koko API:n päivittämisessä tulisi olemaan, oli mielekästä soveltaa keskeisimpiä suunniteltuja päivitys- ja parannustoimenpiteitä ensin hyvin suppeaan versioon API:stä. Tähän testiversioon

sisällytettiin vain kiinteistöjen ja kiinteistösijoitussalkkujen tietoja koskevat toiminnot.

Kolmessa seuraavassa alaluvussa kuvatut toimenpiteet suoritettiin ensin testiversioon. Tämän jälkeen aloitettiin varsinaisen API:n päivitystyö. Sekä testiversion että kokonaisen version muunnostyö tapahtui nopeasti ja ilman yllätyksiä, vaikka jälkimmäinen oli noin 6000 riviä pitkä YAML-tiedosto. Tämä johtuu siitä, että OAS:n versiot 2.0 ja 3.0.0 ovat syntakseiltaan hyvin lähellä toisiaan (Ralphson 2017a).

3.3 Muunnos OAS 2:sta 3:een

API-määrityksen muuntamiseen OAS 2 -muodosta OAS 3:een testattiin kahta eri ohjelmaa, jotka ovat Mermade Softwaren kehittämä *swagger2openapi* ja LucyBot, Inc.:n *api-spec-converter*. Nämä molemmat perustuvat avoimeen lähdekoodiin. Niiden toiminnassa ei ole suuria eroja, koska viimeksi mainittu itse asiassa suorittaa varsinaisen muunnostyön *swagger2openapi*:ta käyttäen, kuten voidaan päätellä tutkimalla *api-spec-converter*:n lähdekoodia. Kun *api-spec-converter*:iä koekäytettiin, kävi ilmi, ettei se säilytä API-määrityksen elementtejä alkuperäisessä järjestyksessä. Tästä syystä konversiotyökaluksi valittiin *swagger2openapi*.

OAS 2 -pohjainen rajapintamääritys oli kirjoitettu kokonaan yhteen tiedostoon, jonka nimi oli *swagger.yaml*. OAI:n suositusten mukaisesti tiedostonimi vaihdettiin *openapi.yaml*:ksi OAS 3:een siirtymisen yhteydessä (OAI 2017c). API-kuvauksen muuntaminen OAS 3:een *swagger2openapi*:lla hoitui helposti ja nopeasti yhdellä komenolla, joka on esitetty kuvassa 5.

```
./oas-kit/packages/swagger2openapi/swagger2openapi.js --yaml
swagger.yaml >openapi.yaml
```

Kuva 5. Komento, jolla Assetti API:n kuvaus muunnettiin OAS 3 -muotoon.

3.4 Koodin generointi

Kun OAS:n mukainen API-määritys on kirjoitettu YAML- tai JSON-syntaksia käyttäen, seuraava vaihe on varsinaisen palvelimella ajettavan koodin generointi määrityksen pohjalta. Tämä tapahtuu yleensä osana järjestelmän käännösprosessia ja generoitu koodi jätetään versionhallinnan ulkopuolelle. Assetin API-koodin generointiin on aiemmin käytetty Swagger-perheeseen keskeisesti kuuluvaa Swagger Codegen -ohjelmaa, joka pystyy tuottamaan koodia useilla eri ohjelmointikielillä (SmartBear Software 2018c).

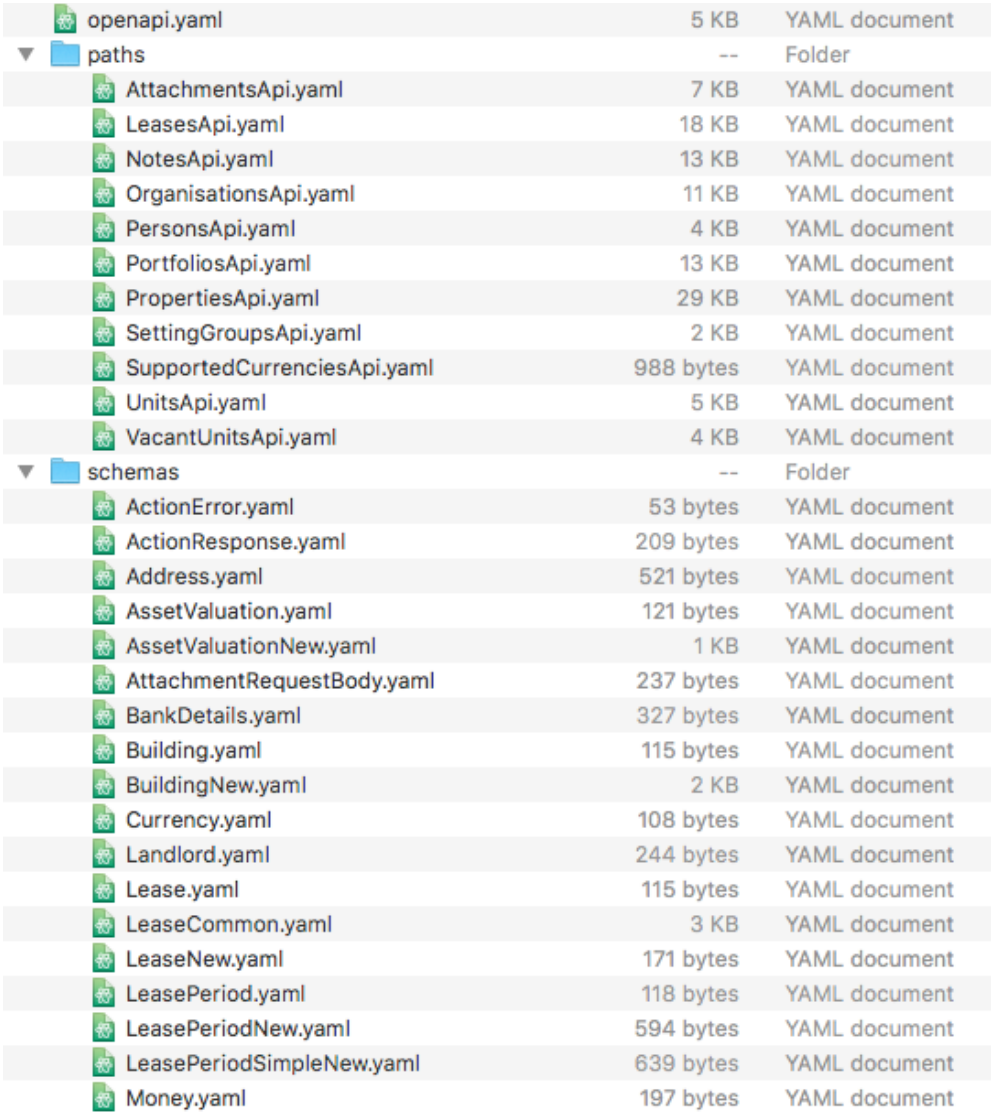
Opinnäytetyöprosessin aikana tuli pian selväksi, ettei Swagger Codegen ole enää hyvä valinta. Sen vakaat versiot eivät olleet yhteensopivia OAS 3:n kanssa eikä seuraavan vakaan 3.0.0-version julkaisupäivä ollut vielä tiedossa (SmartBear Software 2018d). Koodin generointia yritettiin kuitenkin Swagger Codegenin uusimmalla *”release candidate”* -versiolla 3.0.0-rc1. Se ei toiminut, koska sen generoimassa Java-koodissa funktioiden parametrit olivat eri järjestyksessä kuin ennen, mikä luultavasti johtui siitä, että OAS 3 erottaa *requestBody*-elementin API-kutsun parametreista. Generoidun koodin käyttö olisi näin ollen edellyttänyt muutoksia satoihin olemassa oleviin funktiokutsuihin. Java-kielisen palvelinkoodin generointi tapahtuu siten, että Swagger Codegen tuottaa joukon rajapintoja, joiden implementaatiot on kirjoitettava itse. Koska olemassa olevat implementaatiot eivät enää sopineet yhteen Swagger Codegen 3.0.0-rc1:n generoimien rajapintojen kanssa, oli etsittävä uusi ratkaisu.

OpenAPI Generator on vaihtoehtoinen työkalu API-koodin generoimiseen. Se on kehitetty Swagger Codegen 2.4.0:n pohjalta ja se pyrkii ensisijaisesti varmistamaan OAS 3 -yhteensopivuuden. Siinä, missä SmartBear Software omistaa Swagger Codegenin, on OpenAPI Generator kehittäjäyhteisön ylläpitämä. (OpenAPI Tools 2018b.) Assetin uuden OAS 3 -pohjaisen API-koodin generoinnista OpenAPI Generator suoriutui huomattavasti Swagger Codegeniä paremmin, koska se osasi käsitellä uutta API-kuvausta aivan kuin Swagger Codegen käsitteli vanhaa. Rajapintojen funktiomääritysten parametrijärjestys pysyi samana kuin ennen. Yhden hidasteen kuitenkin aiheutti OpenAPI Generatorista löytynyt bugi, jonka vuoksi API-kutsujen parametreille ei voinut määrittää teksti-

muotoisia oletusarvoja, koska generoitu koodi ei ollut validia (OpenAPI Tools 2018c). Oletusarvot poistettiin käytöstä, kunnes bugi korjattiin OpenAPI Generatorin uudemmassa versiossa (OpenAPI Tools 2018d).

3.5 Optimointi

Jotta API:n laajentaminen ja ylläpito saataisiin helpommaksi, rajapintamäärityksen sisältävä pitkä *openapi.yaml*-tiedosto paloiteltiin osiin. Uusi versio on käytännössä tiivistetty listaksi URI-muotoisia osoitteita, joihin API-kutsuja voidaan lähettää, ja jokaisen osoitteen kohdalla viitataan yhteen *paths*-alihakemistosta löytyvään YAML-tiedostoon. Nämä tiedostot on nimetty samojen tunnisteiden mukaan, joiden perusteella Assetti API:n graafinen käyttöliittymä jaottelee API-toiminnot kategorioihin (ks. luku 3.6). Tiedostojen nimiä ovat esim. *NotesApi.yaml* ja *VacantUnitsApi.yaml* ja ne sisältävät kuhunkin kategoriaan kuuluvien toimintojen kuvaukset. API-kutsuissa lähetettävien ja vastaanotettavien tietueiden määritykset eli skeemat eriytettiin *schemas*-alihakemistoon. Näitä ovat esim. *Unit.yaml* ja *ActionResponse.yaml*. Kuvassa 6 on esitetty paloitellun API-kuvauksen hakemistorakenne.



Kuva 6. Tiedostot, joista Assetti API:n kuvaus muodostuu. *schemas*-hakemiston sisällöstä on kuvassa vain noin kolmasosa.

Toinen merkittävä muutos API:n ylläpidon helpottamiseksi liittyi kaksinkertaisen koodin karsimiseen. Vanhassa versiossa lähes kaikki skeemat olivat kahtena kappaleena. Toista käytettiin uusien tietueiden luomiseen ja toista olemassa olevien tietueiden hakuun ja päivittämiseen. Näiden ainoa ero oli se, että ensin mainittuun sisältyi UUID eli universaali tunniste. Se on välttämätön yksittäisiä tietueita haettaessa, mutta asiakassovellus ei voi uutta tietuetta luodessaan lähettää tunnistetta, koska Assetti huolehtii sen luomisesta.

Koodin toistoa saatiin vähennettyä ottamalla tietueiden hakemisen ja päivittämisen osalta käyttöön komposiittiskeemat, jotka periytyvät vastaavista uusien tietueiden skeemoista mutta sisältävät myös UUID-kentät (OAI 2017e). Perintä ei

lukeudu OAS 3:n uusiin ominaisuuksiin, vaan sille olisi ollut tuki myös OAS 2:ssa (OAI 2017f). Kuvissa 7a ja 7b on esitetty skeematiedosto *AssetValuation-New.yaml*, jota käytetään kiinteistön arvon syöttämiseen, sekä sitä hyödyntävä komposiittiskeema *AssetValuation.yaml*, joka määrittelee esim. *GET /properties/{propertyUuid}/assetValuations* -kutsun palauttamien tietueiden muodon.

```yaml
type: object
required:
  - propertyUuid
  - valuationDate
properties:
  propertyUuid:
    type: string
    description: The property UUID.
  marketValue:
    $ref: ./Money.yaml
    description: 'The maket value, Max. Length decimal(18,2)'
  acquisitionPrice:
    $ref: ./Money.yaml
    description: 'The acquisition price, Max. Length decimal(18,2)'
  contributionValue:
    $ref: ./Money.yaml
    description: 'The contribution value, Max. Length decimal(18,2)'
  replacementValue:
    $ref: ./Money.yaml
    description: 'The replacement value, Max. Length decimal(18,2)'
  technicalValue:
    $ref: ./Money.yaml
    description: 'The technical value, Max. Length decimal(18,2)'
  externalValue:
    $ref: ./Money.yaml
    description: 'The external value, Max. Length decimal(18,2)'
  loanAllocation:
    $ref: ./Money.yaml
    description: 'The loan allocation, Max. Length decimal(18,2)'
  listingPrice:
    $ref: ./Money.yaml
    description: 'The listing price, Max. Length decimal(18,2)'
  exitValue:
    $ref: ./Money.yaml
    description: 'The exit Value, Max. Length decimal(18,2)'
  valuationDate:
    type: string
    format: date
    description: 'The valuation date, This value shuold be unique, yyyy-MM-dd'
```

Kuva 7a. *AssetValuationNew.yaml*-skeematiedosto.

```yaml
allOf:
- $ref: ./AssetValuationNew.yaml
- type: object
  required:
    - uuid
  properties:
    uuid:
      type: string
```

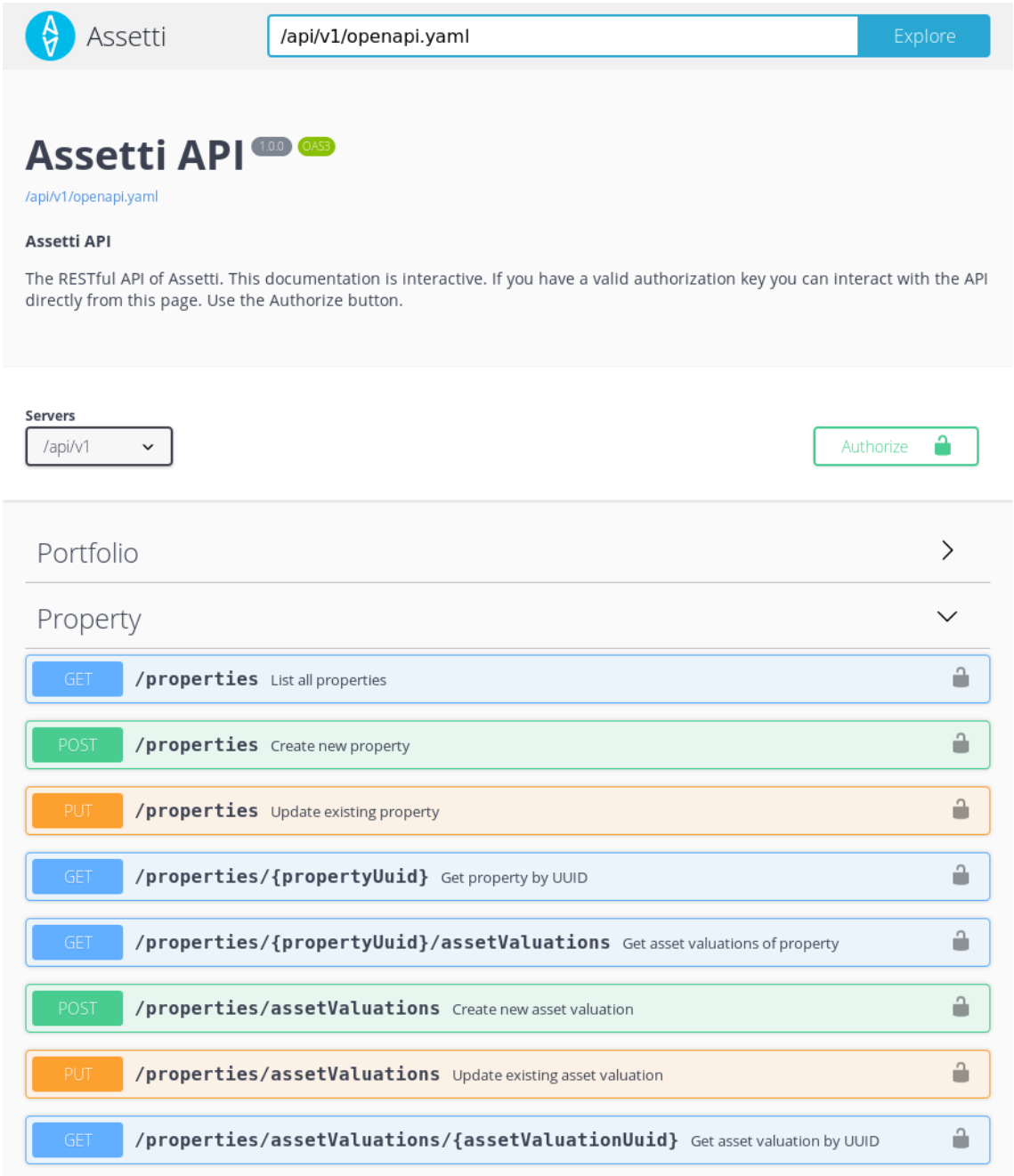
Kuva 7b. *AssetValuation.yaml*-skeematiedosto.

3.6 Graafinen käyttöliittymä: Swagger UI

Yksi Swagger-ohjelmiston hyödyllisimpiä osia on Swagger UI, jolla OAS-pohjaisen API:n käyttö on helpompaa kuin vaikkapa Unixin *curl*-komennolla. Swagger UI generoi selaimella käytettävän interaktiivisen dokumentaatiosivun suoraan API-määritystiedostojen pohjalta (SmartBear Software 2018e). Swagger UI -sivun kautta voidaan suorittaa API-kutsuja ja koota niissä lähetettävät tietueet sivulla esitettyjä malleja käyttäen niin, ettei JSON-olioita ole pakko kirjoittaa kokonaan itse. API-määritykseen voidaan sisällyttää kutsujen, tietueiden ja niiden kenttien kuvaustekstit, jolloin ne tulevat näkyviksi osana Swagger UI -sivua.

Assetti on myös jo pitkään tarjonnut oman API:nsä käyttäjille Swagger UI -sivun. API-kutsujen suorittaminen sen kautta on mahdollista Assetissa generoidun JWT-muotoisen API-avaimen avulla. Koska Assetin käyttämässä Swagger UI:n versiossa 2.1.5 ei ollut tukea OAS 3:lle, se vaihdettiin uusimpaan 3.17.4-versioon (SmartBear Software 2018f).

Uuteen versioon päivittämisen yhteydessä menetettiin vanhan Swagger UI -sivun ulkoasuun tehdyt räätälöinnit, joilla se oli saatu vastaamaan Assetin web-käyttöliittymän visuaalista ilmettä. Näin ollen päivityksen jälkeen oli korvattava Swagger UI:n oletuksena käyttämä värimalli ja fontti sekä osa grafiikoista, mutta koska versio 3 ei ole taaksepäin yhteensopiva aiempien versioiden kanssa, oli Swagger UI:n lähdekoodi kloonattava GitHub-repositoriosta ja Assetin visuaaliseen ilmeeseen sopiva versio käännettävä koodista itse (SmartBear Software 2017). Valmis Swagger UI -sivu esitetään kuvassa 8.



Kuva 8. Swagger UI 3.17.4:n tuottama Assetti API:n käyttöliittymä ja osa kiinteistöjä koskevista API-toiminnoista.

API-määrityksen jako useaan tiedostoon edellytti myös koodimuutoksia, jotta Swagger UI -sivu saatiin toimimaan. Sivua ei generoida varsinaisen API-koodin pohjalta, vaan Swagger UI on staattinen JavaScript-ohjelma, joka lataa palvelimella sijaitsevan JSON- tai YAML-muotoisen API-määrityksen ja kokoaa sen perusteella sivun sisällön itse. Tämä tapahtuu aina, kun Swagger UI -sivu avataan selaimessa, ja tämä toimintatapa on esitetty kuvassa 9.

Kuva 9. Swagger UI:n toimintatapa.

Assetin koodi estää palvelimeen kohdistuvat GET-pyynnöt, joihin ei sisälly validia JWT-autentikointiavainta, ja koska Swagger UI ei autentikoi itseään, koodissa on erikseen sallittu *swagger.yaml*:n eli API-määritystiedoston luku ilman autentikointia. Tähän kohtaan koodissa oli vaihdettava tiedostonimi *openapi.yaml*, minkä lisäksi järjestelyä oli laajennettava, jotta Swagger UI pääsi lataamaan myös uusien *paths*- ja *schemas*-alihakemistojen sisältämät tiedostot.

3.7 Testaus

OAS 3 -päivityksen yhteydessä laadittiin erityinen testaustyökalu, jota käyttämällä voitiin varmistua siitä, että API toimii halutulla tavalla myös päivityksen jälkeen. Tämä työkalu on Unix-pohjaisissa ympäristöissä ajettavaksi tarkoitettu Bash-skripti, joka sai nimen *fetch.sh*. Se käy läpi lähes kaikki API:n toiminnot ja tallentaa niistä saadut vastaukset JSON-tiedostoiksi. Vain liitetiedostojen lähettämistä ei testata, koska jokainen testiajo tallentaisi palvelimelle useita tarpeettomia tiedostoja, joiden poisto ei ole tällä erää mahdollista API:n kautta. Näin ollen API:n liitetiedosto-operaatioiden toimivuus on varmistettava manuaalisen testauksen kautta.

Testiohjelman yksittäinen ajo voidaan kohdistaa mille tahansa Assetin käyttämistä palvelimista. Sitä voidaan myös käyttää kehitystyön apuna paikallisella tietokoneella käynnissä olevan Assetin version testaamiseen. Testiohjelman koodi on liitteessä 2.

Testauksessa käytettiin yhtä Assetin testipalvelimista. Sinne kopioitiin toiselta palvelimelta esimerkkikiinteistöjä, -vuokrasopimuksia, -rakennuksia ja muuta aineistoa sisältävä ympäristö, johon vain testauksen suorittajalle annettiin pääsyoikeus. Ympäristön datasta otettiin varmuuskopio ja palvelimelle asennettiin tuolla hetkellä tuotantokäytössä ollut Assetin versio, johon sisältyi alkuperäinen OAS 2 -pohjainen API. Tämän jälkeen ajettiin *fetch.sh* ja tallennettiin sen API:ltä vastaanottama data myöhempää vertailua varten.

Kun uusi OAS 3 -pohjainen API oli yhdistetty versionhallinnan päähaaraan ja tämä Assetin versio asennettu testipalvelimelle, *fetch.sh* ajettiin uudelleen. Saadut tulokset eivät kaikilta osin vastanneet ensimmäisellä ajolla saatuja. Kun asiaa tutkittiin, ongelmien syyksi osoittautui kaksi erillistä bugia versionhallinnan päähaarassa. Nämä saatiin korjattua hyvin nopeasti, ja seuraavana päivänä suoritettu uusi testiajo palautti toivotut vastaukset. Nyt ainoa ero ensimmäisen ajon tuloksiin nähtiin vuokrattavana olevien huoneistojen lukumäärässä, joka oli kasvanut yhdellä, koska yksi tietokannassa oleva vuokrasopimus oli päättynyt ensimmäisen ajon jälkeen.

4 Assetti API:n päivitys, toinen vaihe: rakennemuutos

Jo opinnäytetyön aloituksen aikaan Assetin kehitystiimissä ehdotettiin erästä parannuskeinoa, jolla API:n ylläpitoa ja tulevaa laajentamista saataisiin helpotettua merkittävästi. Tämä keino on erillisistä API-skeemoista luopuminen. Ne tuovat tarpeetonta päällekkäisyyttä koodiin, koska Assetti käyttää tietokantaentiteettien hallintaan niin sanottuja DTO-luokkia, ja jokaista API:n tunnistamaa entiteettiä kohden on DTO:n lisäksi oltava erillinen skeemansa. Sekä OAS 3 että koodia tuottavat Swagger Codegen ja OpenAPI Generator tukevat myös API-määrityksen ulkopuolelta tuotavien luokkamääritysten käyttämistä API-skeemoina (SmartBear Software 2018g, OpenAPI Tools 2018e). Näin ollen yhdeksi opinnäytetyön tavoitteeksi otettiin sen selvittäminen, voisiko olemassa olevat DTO:t valjastaa API:n käyttöön, sekä mahdollisesti yhtä DTO-luokkaa hyödyntävän *proof of concept* -ratkaisumallin toteutus.

Ennen kuin DTO-ratkaisua oli mahdollista edes kokeilla, täytyi Assetin lähdekoodia järjestellä jonkin verran uudelleen. Koodi jakautuu useaan moduuliin, ja DTO:t sijaitsivat moduulissa, joka ei ollut API-koodin käytettävissä. Tästä syystä DTO:t ja niiden hallinnassa käytetyt välimuistipuskurit siirrettiin toiseen moduuliin, mikä vei oman aikansa. Tällä muutoksella ei ole vaikutusta Assetin muuhun toimintaan, vaan se pikemminkin osaltaan helpottaa koodin rakenteen hahmottamista ja kokonaisuuden hallintaa kehittäjän näkökulmasta.

4.1 Swagger UI ja skeemaesimerkit

DTO-luokkien käyttöönotto API-skeemojen tilalla toi oman haasteensa Swagger UI -sivun käyttöön. Se osaa automaattisesti näyttää YAML- tai JSON-tiedostoista luetut skeemojen rakenteet, jotta käyttäjä osaa ensinnäkin rakentaa API:lle lähetettävät tietueet oikein ja toisaalta tietää ennalta, millaisessa muodossa API palauttaa dataa. Koska uudessa ratkaisussa skeematiedostot puuttuvat eikä Swagger UI ole edes tietoinen DTO-luokkien olemassaolosta, se ei pysty esittämään käyttäjälle mitään mallia tietueiden rakenteesta.

Tämän puutteen korjaamiseksi *proof of concept* -ratkaisussa kirjoitettiin *UnitDTO*-luokkaa vastaava YAML-muotoinen skeemaesimerkkitiedosto ja lisättiin API-kuvaukseen viittaus siihen. Tämä oli mahdollista, koska OAS 3 on tehnyt skeemaesimerkkien määrittämisen paljon aiempaa helpommaksi (Miller 2016). Kuvissa 10a ja 10b on esitetty syntaksi, jolla API-kuvauksessa viitataan GET- ja POST-kutsuihin liittyviin skeemaesimerkkeihin, ja kuvassa 10c nähdään Swagger UI:n niiden pohjalta automaattisesti renderöimä esimerkki *GET /units/UUID* -kutsun palauttamasta vastauksesta.

```yaml
responses:
  '200':
    description: The unit objects
    content:
      application/json:
        schema:
          title: Units
          type: array
          items:
            $ref: Unit
            example:
              allOf:
              - $ref: ./UUIDExample.yaml
              - $ref: ./UnitExample.yaml
```

Kuva 10a. YAML-muotoisen API-kuvauksen määrittelemä vastaus kutsulle *GET /units*, joka palauttaa listan tiloista. Skeemaesimerkki kootaan *example*-elementissä kahdesta tiedostosta. *UUIDExample.yaml* sisältää pelkän UUID-kentän, joka on kaikille skeemoille yhteinen.

```yaml
requestBody:
  content:
    application/json:
      schema:
        $ref: Unit
      example:
        allOf:
        - $ref: ./UnitExample.yaml
```

Kuva 10b. Viestirungon määritys *POST /units* -kutsulle, jolla uusi tila lisätään tietokantaan, sekä viittaus kutsuun liittyvään skeemaesimerkkiin.

```
{
  "uuid": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "propertyUuid": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "buildingUuid": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "plotUuid": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "unitExternalId": "max length 255",
  "description": "HTML, max length 3000",
  "floor": "max length 255",
  "location": "max length 255",
  "surfaceArea": "decimal",
  "netFloorArea": "decimal",
  "unitValue": "decimal",
  "ventilationTypeSetting": "For available values, see /settingGroups/LEASE_VENTILATION_TYPE/settings",
  "unitTypeSetting": "For available values, see /settingGroups/ASSET_CLASS_TYPE/settings",
  "coolingType": "true/false",
  "vatAppliedInitially": "true/false",
  "commonArea": "true/false",
  "activeStartDate": "yyyy-MM-dd",
  "activeEndDate": "yyyy-MM-dd"
}
```

Kuva 10c. Skeemaesimerkin JSON-muotoinen esitys Swagger UI -sivulla.

4.2 Serialisointi

Assetti API:n skeemat ovat kenttiensä osalta hyvin lähellä vastaavia DTO-luokkia ja monet ovat niiden kanssa täysin identtisiä. Keskeisin ero näiden välillä on tapa, jolla luokat linkitetään toisiinsa. Esimerkiksi kiinteistö on Assetin näkökulmasta kokonaisuus, johon voi kuulua useita tontteja ja rakennuksia, joten rakennusta kuvaavassa DTO:ssa ja skeemassa on tieto siitä, mihin kiinteistöön se kuuluu. DTO käyttää tähän tarkoitukseen tietokannassa olevaa tietuenumeroa, joka on kokonaisluku ja jonka tietotyyppi Java-kielessä on *Long*. Näiden numeroiden ei kuitenkaan haluta näkyvän käyttäjille, joten API-skeemat linkittyvät toisiinsa UUID-tunnisteiden avulla. Tästä muodostui suurin haaste DTO-luokkien käyttöönotossa.

Olio-ohjelmoinnissa serialisoinnilla tarkoitetaan olion esittämistä teksti- tai binääritiedostomuodossa, joka soveltuu tiettyyn tarkoitukseen, kuten tietokantaan tallentamiseen tai verkossa lähettämiseen. Serialisoinnissa syntyvä tavujen sarja on voitava myöhemmin kääntää takaisin alkuperäistä oliota vastaavaksi kopioksi. Tätä takaisin kääntämistä kutsutaan deserialisoinniksi. (Microsoft 2018.) REST API:t käsittelevät yleensä JSON-muotoon serialisoitua dataa ja niin tekee myös Assetti API.

Jackson-ohjelmistokirjasto, jota Assetti API hyödyntää, huolehtii lähetettävän JSON-datan serialisoinnista ja vastaanotettavan datan deserialisoinnista automaattisesti, mutta näitä prosesseja on mahdollista räätälöidä tilanteen ja tarpeen mukaan käyttämällä Javan annotaatioita (Saloranta 2016). DTO-luokkien käyttäminen edellyttikin muutoksia siihen tapaan, jolla edellä mainitut linkitykset olioiden välillä serialisoidaan. Linkitetyn olion tunnistenumeron serialisointi sellaisenaan olisi rikkonut Assetti API:n yhteensopivuuden olemassa olevien asiakassovellusten kanssa, joten numeron tilalle täytyi voida serialisoida sitä vastaava UUID. Esimerkki tästä nähdään listauksessa 1 ja kuvassa 11, jotka esittävät osan vuokrakauden tiedoista DTO-luokassa ja vastaavan osan JSON-muotoon serialisoidusta tietueesta, jollaisen API-kutsu *GET /leases/leasePeriods/UUID* palauttaa.

```
public class LeasePeriodDTO extends BaseDTO implements Serializable {
    private Long leaseId;
    private Date startDate;
    /* ... */
}
```

Listaus 1. Osa *LeasePeriodDTO*-luokasta, joka määrittää yksittäisen vuokrakauden. Se linkitetään tiettyyn vuokrasopimukseen *leaseId*-tunnistenumerokentän avulla.

```
{
    "uuid": "9071F490-0B6B-43E0-AAF4-4A0A19DCDE75",
    "leaseUuid": "6547CDCF-870E-4C84-8A46-0C9AEFC2A714",
    "startDate": "2016-05-01"
}
```

Kuva 11. Osa JSON-muotoon serialisoitua vuokrakausitietuetta, jossa *leaseId*-kentän tilalla on *leaseUuid*. Tietueen oma UUID periytyy *BaseDTO*-kantaluokasta.

Osana tätä opinnäytetyötä laadittiin malli serialisoinnin toteuttamiseksi. Se käyttää tiloja eli huoneistoja kuvaavaa *UnitDTO*-luokkaa korvaamaan API:n *Unit.yaml*- ja *UnitNew.yaml*-skeemat. Tilan tiedoissa on tunnistenumerolinkki kiinteistöön, rakennukseen ja tonttiin. Listauksessa 2 oleva Jacksonia hyödyntävän serialisointiluokan hahmotelma esittää toimintaperiaatteen, jolla tunnistenumeron sijasta voitaisiin serialisoida UUID-merkkijono. Tällainen serialisoija otetaan käyttöön DTO-luokan sisällä *@JsonSerialize*-annotaatiota käyttämällä (Paraschiv 2018).

```
public class PropertyIdToUuidSerializer extends StdSerializer<Long> {
    public void serialize(Long id, JsonGenerator gen) {
        PropertyDTO dto = PropertyCache.get(id);
        gen.writeString(dto == null ? null : dto.getUuid());
    }
}
```

Listaus 2. Räätälöidyn UUID-serialisoijan malli.

Listauksen 2 mukainen esimerkkikoodi ei toiminut käytännössä, vaikka siinä päällisin puolin oli kaikki tarvittava paikallaan. Ongelmia aiheutui Apache Shiro -kirjastosta, joka tarjoaa Java-sovelluksille toimintoja muun muassa käyttäjän autentikointiin, kryptografiaan ja istuntojen hallintaan (Hazlewood 2011) ja jota Assetti käyttää. Listauksessa 2 mainittu funktiokutsu *PropertyCache.get(id)*

hakee välimuistista kiinteistötietueen sen tunnistenumeron perusteella, ja oikean välimuistipuskurin löytääkseen funktio tarvitsee asiakasnumeron. Sitä ei kuitenkaan ole saatavilla siinä tilanteessa, jossa serialisointikoodi ajetaan. Kun API-kutsu otetaan vastaan, sen mukana tullut käyttäjän identiteetin varmistava JWT-avain tarkistetaan, ja jos avain hyväksytään, Shiro avaa istunnon kutsun prosessoinnin ajaksi. Näyttää siltä, että istunto joko suljetaan ennen kuin API:n lähettämä vastaus serialisoidaan, tai serialisointikoodi ajetaan eri ohjelmasäikeessä, johon Shiro-istunnon vaikutuspiiri ei ylety. Ongelman perimmäistä syytä ei kyetty selvittämään, joten serialisointi oli toteutettava kiertotietä käyttäen.

Uudessa ratkaisussa *UnitDTO*-luokka sisältää tekstikentät kiinteistön, rakennuksen ja tontin UUID:lle ja se noudattaa kolmea uutta rajapintaa, joihin kiinteistön, rakennuksen ja tontin linkityksen serialisointi on ulkoistettu. Nämä rajapinnat ovat *LinkedToPropertyCache* (esitetty listauksessa 3), *LinkedToBuildingCache* ja *LinkedToPlotCache* ja ne ovat keskenään hyvin samanlaiset. Kolmen erillisen rajapinnan sijaan parempi ratkaisu olisi ollut yhden geneerisen rajapinnan soveltaminen kaikkiin tapauksiin, mutta Assetin välimuistipuskurien arkkitehtuuriratkaisu ei sallinut tätä, eikä arkkitehtuurin vaihtaminen toisenlaiseen ollut mielekästä tämän opinnäytetyön puitteissa.

```
public interface LinkedToPropertyCache {

    default void setPropertyIdFromUuid(long customerId,
        Supplier<String> getter, Consumer<Long> setter) {
        String uuid = getter.get();
        if (uuid != null) {
            PropertyDTO property = PropertyCache.getByUuid(customerId,
                uuid);
            setter.accept(property == null ? null : property.getId());
        }
    }

    default void setPropertyUuidFromId(long customerId,
        Supplier<Long> getter, Consumer<String> setter) {
        Long id = getter.get();
        if (id != null) {
            PropertyDTO property = PropertyCache.get(customerId, id);
            setter.accept(property == null ? null :
                property.getUuid());
        }
    }

}
```

Listaus 3. *LinkedToPropertyCache*-rajapinta. Funktioiden parametrit *getter* ja *setter* ovat viittauksia rajapintaa noudattavan DTO-luokan sisältämään aksessoriin ja mutaattoriin.

Rajapintojen ansiosta esimerkiksi *UnitDTO*-instanssin kiinteistölinkityksenä toimiva UUID-kenttä voidaan päivittää API-kutsun vastausta rakennettaessa kutsumalla *setPropertyUuidFromId()*-funktiota tilanteessa, jossa asiakasnumero tunnetaan ja voidaan lähettää funktiolle *customerId*-parametrina. Vastaavasti kun API:n kautta luodaan uusi *UnitDTO*-tietue, sen sisältämä kiinteistö-UUID deserialisoidaan käyttämällä *setPropertyIdFromUuid()*-funktiota. Tällä tavoin vältetään se, että välimuistia yritettäisiin lukea Shiro-istunnon ulkopuolelta. Ratkaisun huono puoli on sen edellyttämä suurehko määrä ylimääräistä koodia useassa DTO-luokassa ja rajapinnassa.

Serialisointiratkaisun toimivuus testattiin *fetch.sh*-skriptillä. Ensimmäinen testiajo paljasti, ettei uusi API-koodi osannutkaan serialisoida eräitä tilan tiedoista puuttuvia kenttiä tyhjinä merkkijonoina kuten vanha versio teki, vaan ne jäivät vastauksesta kokonaan pois. Tällä ei välttämättä olisi vaikutusta asiakassovellusten toimintaan, mutta työssä haluttiin pyrkiä siihen, ettei uuden API-koodin palauttamissa vastauksissa olisi havaittavissa mitään eroa aiempaan verrattuna, joten nämä särmät hiottiin pois. Tämän jälkeen uudella testiajolla saatiin

tulokset, jotka olivat käytännössä identtiset viimeisimmän ilman DTO-luokkia tehdyn testin kanssa. Näin ollen API:n DTO-rakennemuutos voitiin todeta toimivaksi ja *proof of concept* yhdistettiin versionhallinnan päähaaraan.

5 Johtopäätökset

Miten OAS 2:sta OAS 3:een päivittäminen hyödyttäisi ylläpidettävyyttä? Assetti API:n OAS 2 -muotoisen kuvauksen päivittämisen OAS 3:een oli etukäteen ajateltu itsessään parantavan API:n helppokäyttöisyyttä ja laajennettavuutta kehittäjän kannalta. Päivityksen yhteydessä kävi kuitenkin ilmi, ettei Assetti API hyödy OAS 3:sta merkittävissä määrin, koska se ei hyödynnä kovinkaan monia niistä OAS:n ominaisuuksista, jotka ovat eniten muuttuneet uudessa kolmosversiossa. Tämä ei silti tarkoita, että päivittäminen olisi ollut turhaa. Sellaiselle kehittäjälle, jolle OpenAPI ei ole entuudestaan tuttu, Assetti API:n uusi kuvaus näyttäytyy helpommin ymmärrettävänä kuin vanha. Tästä on kiittäminen erityisesti OAS 3:n tapaa erottaa viestin runko sen parametreista omaan *requestBody*-elementtiinsä.

Tiedostojen lähettämiseen tarjoutuu OAS 3:ssa enemmän valinnanvapautta. Vaikka etukäteen arveltiin, että Assetti API:n tiedostonsiirtotoiminnon voisi näin ollen uudistaa, se päädyttiin lopulta pitämään entisessä mallissaan, joka perustuu lomakedatan kaltaisten *multipart/form-data*-viestien lähettämiseen. Syynä oli yksinkertaisesti se, että tiedoston mukana on voitava samalla kertaa lähettää sen kuvausteksti, ja ainoastaan lomakedataratkaisu mahdollistaa tällaisten moniosaisten viestien lähettämisen.

Opinnäytetyön ensimmäisen vaiheen, jossa API-kuvaus päivitettiin OAS 3 -muotoon, arveltiin vaativan kolme tai neljä viikkoa. Koska prosessin monimutkaisuutta ei ollut helppoa arvioida, päivitys aloitettiin harjoittelemalla sitä API-kuvauksen karsitulla testiversiolla. Lopulta koko tämä ensimmäinen vaihe harjoittelu mukaan lukien saatiin valmiiksi kahden ja puolen viikon aikana 90 tunnin työllä. Työ sujui siis oletettua helpommin, ja koska mitään kohtuuttoman suuria

ongelmia ei tullut vastaan, olisi päivitysprosessin harjoittelun voinut jälkikäteen arvioiden jättää väliinkin.

OAS 3 -päivityksen yhteydessä tehdyt rakenteelliset optimoinnit auttoivat omalta osaltaan tekemään API:n kehitystyöstä aiempaa helpompaa. Koska API-kuvaus on nyt jaettu useaan YAML-tiedostoon, jotka on nimetty loogisella tavalla, jokaisen yksittäisen operaation määritelmä löytyy vaivattomasti. Perintää hyödyntävien komposiittiskeemojen ansiosta taas samanmuotoista tietuerakennetta ei tarvitse kuvailla kahdesti. Nämä molemmat toimenpiteet olisivat toisaalta olleet mahdollisia myös OAS 2:n puitteissa, joten ne olisi aivan hyvin voitu tehdä jo aiemminkin.

Millaisia muutoksia tarvittaisiin, jotta API voisi hyödyntää Assetin koodissa käytettyjä tietueluokkia? API:n skeemojen korvaaminen olemassa olevilla DTO-luokilla helpottaa API-työskentelyä kehittäjän kannalta vielä enemmän kuin OAS 3:een siirtyminen. Näkyvin muutos on kaksinkertaisen koodin väheneminen. Alusta alkaen oli kuitenkin selvää, ettei tätä muutosta olisi kovin helppoa toteuttaa. Ensinnäkin koodia oli järjesteltävä uudelleen niin, että DTO-luokat saatiin tuotua näkyviksi myös API-koodille.

Alustavaa kartoitusta DTO-luokkien käyttämiseksi API:ssä tehtiin jo opinnäytetyön aloituksen yhteydessä OAS 3 -päivitystä suunniteltaessa. Jo tuolloin huomattiin tietuenumeroiden ja UUID-tunnisteiden aiheuttama ongelma. Tapa, jolla tämä ongelma päädyttiin myöhemmin ratkaisemaan, on hyvin lähellä yhtä niistä ratkaisumalleista, joita tuolloin alkuvaiheessa hahmoteltiin. Prosessin aikana löydettiin automatisoituun JSON-serialisointiin perustuva ratkaisu, joka olisi ollut täydellinen, jos Shiro-kirjaston avaama käyttäjäistunto olisi saatu ulottumaan serialisointikoodiin. Tämä rajoitus lykkäsi työn valmistumista jonkin verran, mutta toisaalta se opetti paljon hyödyllistä tietoa Shirosta. Myös serialisointiin käytettävä Jackson-kirjasto tuli opinnäytetyön myötä tutuksi.

Vaikka DTO-luokat tekevätkin YAML-muotoiset API-skeemaluokat tarpeettomiksi, ei ennen prosessin aloittamista huomioitu, että Swagger UI:ta varten tarvitaan erillisiä käsin kirjoitettuja skeemaesimerkkejä DTO-luokkien vastinpareik-

si. Ne on pidettävä ajan tasalla, jos DTO:t muuttuvat. Tästä syystä YAML-tiedostojen määrä ei rakennemuutoksen myötä vähenekään aivan niin paljon kuin oli toivottu. Toisaalta skeemaesimerkit saattaisi olla mahdollista generoida automaattisesti DTO-luokkien pohjalta, mutta tätä ei toistaiseksi ole tutkittu.

Jälkeenpäin voidaan sanoa, ettei opinnäytetyön toinen vaihe eli DTO-luokkien käytön edellyttämä rakennemuutos ollut merkittävästi vaikeampi kuin etukäteen oli oletettu. Se vaati kymmenen viikon aikana yhteensä noin 240 tunnin työmäärän, eikä lopputulos ole ihanteellinen, mutta siitä on saatu hyvä pohja jatkokehitykselle. Opinnäytetyön valmistumisen jälkeen Assetin kehitystiimi on alkanut tutkia mahdollista vaihtoehtoista toteutustapaa DTO-luokkien väliseen linkitykseen, ja se voisi toimiessaan edesauttaa myös API:n serialisointiongelman ratkaisua.

Miten varmistetaan, että Assetti API on optimointien jälkeenkin täydellisesti yhteensopiva olemassa olevien asiakassovellusten kanssa? Optimointiprosessin ei haluttu vaikuttavan millään tavoin siihen, missä muodossa API lähettää ja vastaanottaa dataa. Ainoa sallittu ulospäin näkyvä muutos on Swagger UI -sivun päivitetty ulkoasu. API:n ulkoinen muuttumattomuus varmistettiin järjestelmällisellä testaamisella, jota varten laadittiin *fetch.sh*-testiskripti. Vertaamalla erityisesti sen suorittamien GET-kutsujen palauttamaa JSON-dataa ennen OAS 3:een päivittämistä ja sen jälkeen kyettiin toteamaan, etteivät optimointitoimenpiteet olleet vaarantaneet yhteensopivuutta asiakassovellusten kannalta. Samaa menetelmää hyödynnettiin opinnäytetyön toisessa vaiheessa, kun API ohjelmoitiin serialisoimaan suoraan DTO-luokista lukemansa tiedot tarkalleen aiemman version tuottamia JSON-olioita vastaaviksi.

Testiskripti ei silti ole täydellinen. Ainoa tavoite sitä kirjoitettaessa oli sen varmistaminen, että OAS 3 -päivitys onnistui, eikä sitä ollut alun perin tarkoitus julkaista missään. Tästä syystä se on toteutettu melko umpimähkäisesti. Sitä voitaisiin kuitenkin pienen jatkokehittelyn jälkeen soveltaa myös API:n automaattiseen testaukseen. Asiasta on keskusteltu Assetin kehitystiimissä, mutta skriptin jatkokehitys ei kuulu tämän opinnäytetyön piiriin. Nykyisen version keskeisin

puute testiautomaation kannalta katsottuna on se, ettei se itse arvioi saamiensa vastausten oikeellisuutta eikä osaa verrata niitä mihinkään oletettuihin tuloksiin.

Toinen ongelma, joka olisi ratkaistava ennen *fetch.sh*:n valjastamista testiautomaation välineeksi, on se, että sen jokainen ajo luo tietokantaan uusia tietueita, jotka puolestaan näkyvät myöhempien ajojen hakutuloksissa. Palvelinympäristöä, johon testiajot kohdistuvat, ei voi käyttää mihinkään muuhun, ja se täytyy palauttaa varmuuskopiosta aina ennen testausta, mikä ei ole mahdollista *fetch.sh*:n itsensä kautta. Skriptin kolmas puute liittyy sillä luotujen tietueiden verraten niukkaan sisältöön. Niissä on pakollisten kenttien lisäksi vain pari muuta, ja luotettavampien testitulosten saamisen kannalta olisi perusteltua lähettää tietueita, joissa on dataa jokaisessa kentässä. Skripti ei myöskään testaa virheellisesti muodostettujen tietueiden lähettämistä.

Lähteet

APIs.guru. 2017. Browse APIs. https://apis.guru/browse-apis/. 13.9.2018.

Elkstein, M. 2008. Learn REST: A Tutorial: 1. What is REST? http://rest.elkstein.org/2008/02/what-is-rest.html. 5.9.2018.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. 1999. Hypertext Transfer Protocol - HTTP/1.1. https://tools.ietf.org/html/rfc2616. 6.9.2018.

Fowler, M. 2003. Data Transfer Object. https://martinfowler.com/eaaCatalog/dataTransferObject.html. 27.9.2018.

Gazarov, P. 2016. What is an API? In English, Please. https://medium.freecodecamp.org/what-is-an-api-in-english-please-b880a3214a82. 22.7.2018.

Hazlewood, L. 2011. Application Security With Apache Shiro. https://www.infoq.com/articles/apache-shiro. 11.11.2018.

Heller, M. 2007. REST and CRUD: The Impedance Mismatch. https://www.infoworld.com/article/2640739/. 13.9.2018.

International Telecommunication Union. 2018. Universally Unique Identifiers (UUIDs). https://www.itu.int/en/ITU-T/asn1/Pages/UUID/uuids.aspx. 28.10.2018.

Json.org. 2018. Introducing JSON. http://json.org/. 19.7.2018.

Koberger, G. 2017. A Visual Guide to What's New in Swagger 3.0. https://blog.readme.io/an-example-filled-guide-to-swagger-3-2/. 30.8.2018.

Mermade Software. 2018. swagger2openapi. https://github.com/Mermade/oas-kit/tree/master/packages/swagger2openapi. 15.9.2018.

Microsoft. 2018. Serialization (C#). https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/. 28.10.2018.

Miller, D. 2016. TDC: Structural Improvements: Explaining the 3.0 spec, Part 2. https://www.openapis.org/news/blogs/2016/10/tdc-structural-improvements-explaining-30-spec-part-2. 30.8.2018.

Myer, T. 2005. A Really, Really, Really Good Introduction to XML. https://www.sitepoint.com/really-good-introduction-xml/. 11.10.2018.

OAI. 2017a. OpenAPI Specification 3.0.0: Format. https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md#format. 18.8.2018.

OAI. 2017b. The OAI Announces the OpenAPI Specification 3.0.0. https://www.openapis.org/blog/2017/07/26/the-oai-announces-the-openapi-specification-3-0-0. 13.9.2018.

OAI. 2017c. OpenAPI Specification 3.0.0: Document Structure. https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md#document-structure. 18.8.2018.

OAI. 2017d. OpenAPI Specification 2.0: Parameter Object. https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md#parameter-object. 31.8.2018.

OAI. 2017e. OpenAPI Specification 3.0.0: Composition and Inheritance (Polymorphism). https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md#composition-and-inheritance-polymorphism. 19.8.2018.

OAI. 2017f. OpenAPI Specification 2.0: Composition and Inheritance (Polymorphism). https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md#composition-and-inheritance-polymorphism. 19.8.2018.

OAI. 2018. FAQ. https://www.openapis.org/faq. 30.8.2018.

OpenAPI Tools. 2018a. OpenAPI Generator. https://github.com/openapitools/openapi-generator. 13.9.2018.

OpenAPI Tools. 2018b. OpenAPI Generator: Question and Answer. https://github.com/OpenAPITools/openapi-generator/blob/master/docs/qna.md. 4.8.2018.

OpenAPI Tools. 2018c. [Java] Default Values for String-type Query Parameters are Generated With Two Sets of Quotes. https://github.com/OpenAPITools/openapi-generator/issues/540. 18.8.2018.

OpenAPI Tools. 2018d. OpenAPI Generator: Release 3.2.2. https://github.com/OpenAPITools/openapi-generator/releases/tag/v3.2.2. 24.8.2018.

OpenAPI Tools. 2018e. OpenAPI Generator: Bringing Your Own Models. https://github.com/OpenAPITools/openapi-generator/blob/master/docs/customization.md#bringing-your-own-models. 11.10.2018.

Paraschiv, E. 2018. Jackson Annotation Examples. https://www.baeldung.com/jackson-annotations. 28.10.2018.

Ralphson, M. 2017a. Comparing OpenAPI/Swagger 2.0 and 3.0.0-rc1. https://dev.to/mikeralphson/comparing-openapiswagger-20-and-300-rc1. 22.7.2018.

Ralphson, M. 2017b. Authentiq.Io Swagger 2.0 API Definition. https://gist.github.com/MikeRalphson/0c42c2778ea7a7a6fc0b5d59b4d7868e. 13.9.2018.

Ralphson, M. 2017c. Example of Converted Swagger 2.0 API Definition. https://gist.github.com/MikeRalphson/72e8172bddfa4de7330c9e218b95401f. 13.9.2018.

Saloranta, T. 2016. Jackson Annotations. https://github.com/FasterXML/jackson-annotations/wiki/Jackson-Annotations. 28.10.2018.

SmartBear Software. 2017. How to Customize Swagger UI 3.0.14 to Add Your Own Logo and Color Combination. https://github.com/swagger-api/swagger-ui/issues/3335#issuecomment-313476924. 10.10.2018.

SmartBear Software. 2018a. Swagger Open Source Tools. https://swagger.io/tools/open-source/. 30.8.2018.

SmartBear Software. 2018b. File Upload. https://swagger.io/docs/specification/describing-request-body/file-upload/. 31.8.2018.

SmartBear Software. 2018c. Swagger Code Generator: Overview. https://github.com/swagger-api/swagger-codegen#overview. 4.8.2018.

SmartBear Software. 2018d. Swagger Code Generator: Compatibility. https://github.com/swagger-api/swagger-codegen#compatibility. 4.8.2018.

SmartBear Software. 2018e. Swagger UI. https://swagger.io/tools/swagger-ui/. 19.8.2018.

SmartBear Software. 2018f. Swagger UI: Compatibility. https://github.com/swagger-api/swagger-ui#compatibility. 19.8.2018.

SmartBear Software. 2018g. Swagger Code Generator: Bringing Your Own Models. https://github.com/swagger-api/swagger-codegen#bringing-your-own-models. 11.10.2018.

Stecky-Efantis, M. 2016. 5 Easy Steps to Understanding JSON Web Tokens (JWT). https://medium.com/vandium-software/5-easy-steps-to-understanding-json-web-tokens-jwt-1164c0adfcec. 11.10.2018.

The World Wide Web Consortium. 2001. URIs, URLs, and URNs: Clarifications and Recommendations 1.0. https://www.w3.org/TR/uri-clarification/. 4.9.2018.

The World Wide Web Consortium. 2004. Architecture of the World Wide Web, Volume One: Identification. https://www.w3.org/TR/webarch/#identification. 4.9.2018.

Yaml.org. 2006. YAML Ain’t Markup Language. http://yaml.org/about.html. 22.7.2018.

Assetti API:n toiminnot

Kategoria/URI	Verbi	Kuvaus
Salkut		
`/portfolios`	GET	Hae kaikki salkut
	POST	Luo uusi salkku
	PUT	Muokkaa salkkua
`/portfolios/`*`UUID`*	GET	Hae salkku UUID:llä
`/portfolios/`*`UUID`*`/ownershipShares`	GET	Hae salkun omistusosuudet
`/portfolios/ownershipShares`	POST	Luo salkulle uusi omistusosuus
	PUT	Muokkaa salkun omistusosuutta
`/portfolios/ownershipShares/`*`UUID`*	GET	Hae salkun omistusosuus UUID:llä
`/portfolios/`*`UUID`*`/properties`	GET	Hae salkun kiinteistöt
`/portfolios/`*`UUID`*`/properties/`*`PropertyUUID`*	DELETE	Poista kiinteistö salkusta
	POST	Lisää kiinteistö salkkuun
Kiinteistöt		
`/properties`	GET	Hae kaikki kiinteistöt
	POST	Luo uusi kiinteistö
	PUT	Muokkaa kiinteistöä
`/properties/`*`UUID`*	GET	Hae kiinteistö UUID:llä
`/properties/`*`UUID`*`/assetValuations`	GET	Hae kiinteistön arvioinnit
`/properties/assetValuations`	POST	Lisää uusi arviointi
	PUT	Muokkaa arviointia
`/properties/assetValuations/`*`UUID`*	GET	Hae arviointi UUID:llä
`/properties/`*`UUID`*`/ownershipShares`	GET	Hae kiinteistön omistusosuudet
`/properties/ownershipShares`	GET	Hae kaikkien kiinteistöjen omistusosuudet
	POST	Luo kiinteistölle uusi omistusosuus
	PUT	Muokkaa kiinteistön omistusosuutta
`/properties/ownershipShares/`*`UUID`*	GET	Hae kiinteistön omistusosuus UUID:llä
`/properties/`*`UUID`*`/buildings`	GET	Hae kiinteistön rakennukset
`/properties/buildings`	GET	Hae kaikki rakennukset
	POST	Luo uusi rakennus
	PUT	Muokkaa rakennusta
`/properties/buildings/`*`UUID`*	GET	Hae rakennus UUID:llä
`/properties/`*`UUID`*`/plots`	GET	Hae kiinteistön tontit
`/properties/plots`	GET	Hae kaikki tontit
	POST	Luo uusi tontti
	PUT	Muokkaa tonttia
`/properties/plots/`*`UUID`*	GET	Hae tontti UUID:llä
`/properties/`*`UUID`*`/units`	GET	Hae kiinteistöön kuuluvat tilat

Assetti API:n toiminnot

Kategoria/URI	Verbi	Kuvaus
Viestit		
`/properties/`*`UUID`*`/notes`	GET	Hae kiinteistön viestit
`/units/`*`UUID`*`/notes`	GET	Hae tilan viestit
`/persons/`*`UUID`*`/notes`	GET	Hae henkilön viestit
`/organisations/`*`UUID`*`/notes`	GET	Hae organisaation viestit
`/leases/`*`UUID`*`/notes`	GET	Hae vuokrasopimuksen viestit
`/notes`	GET	Hae kaikki viestit vastauksineen
	POST	Lisää uusi viesti
	PUT	Muokkaa viestiä
`/notes/`*`UUID`*	GET	Hae viesti UUID:llä vastauksineen
`/notes/replies`	POST	Lisää vastaus viestiin
	PUT	Muokkaa vastausta
Liitetiedostot		
`/properties/`*`UUID`*`/attachments`	POST	Lisää liitetiedosto kiinteistöön
`/portfolios/`*`UUID`*`/attachments`	POST	Lisää liitetiedosto salkkuun
`/units/`*`UUID`*`/attachments`	POST	Lisää liitetiedosto tilaan
`/leases/`*`UUID`*`/attachments`	POST	Lisää liitetiedosto sopimukseen
Tilat		
`/units`	GET	Hae kaikki tilat
	POST	Luo uusi tila
	PUT	Muokkaa tilaa
`/units/`*`UUID`*	GET	Hae tila UUID:llä
	DELETE	Poista tila
Yhteyshenkilöt ja organisaatiot		
`/persons`	GET	Hae kaikki henkilöt
	POST	Luo uusi henkilö
	PUT	Muokkaa henkilöä
`/persons/`*`UUID`*	GET	Hae henkilö UUID:llä
`/organisations`	GET	Hae kaikki organisaatiot
	POST	Luo uusi organisaatio
	PUT	Muokkaa organisaatiota
`/organisations/`*`UUID`*	GET	Hae organisaatio UUID:llä
`/organisations/`*`UUID`*`/persons`	GET	Hae organisaation henkilöt
`/organisations/persons`	GET	Hae kaikkien organisaatioiden henkilöt
	POST	Luo uusi henkilö organisaatioon
	PUT	Muokkaa organisaation henkilöä
`/organisations/persons/`*`UUID`*	GET	Hae organisaation henkilö UUID:llä

Assetti API:n toiminnot

Kategoria/URI	Verbi	Kuvaus
Vuokrasopimukset		
`/leases`	GET	Hae kaikki sopimukset
	POST	Luo uusi sopimus
	PUT	Muokkaa sopimusta
`/leases/`*`UUID`*	GET	Hae sopimus UUID:llä
	DELETE	Poista sopimus
`/leases/`*`UUID`*`/leasePeriods`	GET	Hae sopimuksen vuokrakaudet
`/leases/leasePeriods`	GET	Hae kaikki vuokrakaudet
	POST	Luo uusi vuokrakausi
	PUT	Muokkaa vuokrakautta
`/leases/leasePeriods/`*`UUID`*	GET	Hae vuokrakausi UUID:llä
`/leases/`*`UUID`*`/leasePeriods/`*`PeriodUUID`*`/unitRents`	GET	Hae vuokrakauden vuokrat
`/leases/leasePeriods/unitRents/`*`UUID`*	GET	Hae vuokra UUID:llä
`/leases/leasePeriods/unitRents`	GET	Hae kaikki vuokrat
	POST	Luo uusi vuokra
	PUT	Muokkaa vuokraa
Valuutat		
`/supportedCurrencies`	GET	Hae tuettujen valuuttojen lista
Asetukset		
`/settingGroups`	GET	Hae asetuskategoriat
`/settingGroups/`*`GROUP`*`/settings`	GET	Hae kategoriaan kuuluvat asetukset
Vuokrattavat tilat		
`/vacantUnits`	GET	Hae kaikki vapaat tilat
`/vacantUnits/`*`UUID`*`/unitPictures`	GET	Hae tilan kuvat UUID:llä
`/vacantUnits/`*`UUID`*`/propertyPictures`	GET	Hae tilan kiinteistökuvat UUID:llä

Testaustyökalu *fetch.sh*

Rivinumerot on lisätty lukemisen helpottamiseksi eivätkä ne ole osa koodia.

```
 1 #!/bin/sh
 2
 3 # fetch.sh: A Bash script for testing the Assetti API.
 4 # For usage info, run 'fetch.sh -h'.
 5
 6 for a in "$@" ; do
 7     case "$a" in
 8         "-h" | "--help")
 9             echo "Usage: $0 [host]"
10             echo "host specifies the prefix of the target server."
11             echo "API requests are sent to '<host>.assetti.pro'."
12             echo "If no host is given, 'localhost:8080' is
   targeted."
13             exit 0
14             ;;
15     esac
16 done
17
18 # Make sure we have a valid setup
19
20 [ -z `which curl` ] && { echo "No 'curl' in path!" ; exit 1 ; }
21 [ -z `which sed` ] && { echo "No 'sed' in path!" ; exit 1 ; }
22 [ -z `which python3` ] && { echo "No 'python3' in path!" ; exit
   1 ; }
23 python3 -c 'import json.tool' 2>/dev/null || { echo "Python module
   'json.tool' not found!" ; exit 1 ; }
24
25 # Read host name from command line
26
27 if [ -n "$1" ] ; then
28     SERVER=$1
29     HOST=https://$SERVER.assetti.pro
30 else
31     SERVER=localhost
32     HOST=http://$SERVER:8080
33 fi
34 BASE=$HOST/api/v1
35 LIMIT=1000000
36
37 # Read API token
38
39 echo "Please paste your API token and press Enter."
40 read TOKEN
41
42 # Shortcuts for HTTP headers
43
44 H_ACCEPT="Accept: application/json"
45 H_AUTH="Authorization: Bearer $TOKEN"
46 H_CONTENT="Content-Type: application/json"
47
48 # Create a new subdirectory for responses
49
50 SUBDIR=$SERVER.`date "+%Y%m%d%H%M%S"`
51
52 mkdir $SUBDIR || { echo "Cannot create a subdirectory for JSON
```

Testaustyökalu *fetch.sh*

```
data!" ; exit 1 ; }
cd $SUBDIR

# echoFirst() prints the first value of key "$2" found in file "$1"

echoFirst () {
    grep -m1 "^        \"$2\":" $1 |cut -d'"' -f4
}

# echoFirstInside() is like echoFirst(), but looks for the key "$2" in element "$3"

echoFirstInside () {
    OFFSET=`grep -m1 -n "^    \"$3\":" $1 |cut -d':' -f1`
    [ -z "$OFFSET" ] && return 0
    LENGTH=`tail -n +$OFFSET $1 |grep -n -m1 "^    }" |cut -d':' -f1`
    tail -n +$OFFSET $1 |head -$LENGTH |grep -m1 "^        \"$2\":" $1 |cut -d'"' -f4
}

# init() sets up helper variables for doCurl()

init () {
    VERB="$1"
    ENDPOINT="$2"
    echo "$1 /$ENDPOINT ..."
    URL=$BASE/$ENDPOINT\?locale=EN\&limit=$LIMIT\&offset=0
    NO_UUID=`echo "$ENDPOINT" |sed -E "s/[[:xdigit:]]{8}-[[:xdigit:]]{4}-[[:xdigit:]]{4}-[[:xdigit:]]{4}-[[:xdigit:]]{12}/uuid/g"`
    NO_GROUP=`echo $NO_UUID |sed -E "s/[A-Z_]{2,}/group/g"`
    OUTPUT=$SERVER.$VERB.`echo $NO_GROUP |sed s,/,-,g`.json
}

# doCurl() performs a request:
#   $1 = HTTP verb
#   $2 = URI
#   $3 = request body (if $1 is POST or PUT)

doCurl () {
    init $1 $2
    TMPFILE=.curl.tmp
    if [ -n "$3" ] ; then
        curl -s -X $VERB -H "$H_ACCEPT" -H "$H_AUTH" -H "$H_CONTENT" $URL -d "$3" >$TMPFILE
    else
        curl -s -X $VERB -H "$H_ACCEPT" -H "$H_AUTH" $URL >$TMPFILE
    fi
    if [ `du -b "$TMPFILE" |cut -f 1` -le 0 ] ; then
        echo "=======> ERROR! Empty response from 'curl'"
        touch $OUTPUT
    else
        if [ "`cat $TMPFILE |head -c 1`" = "<" ] ; then
            STATUS=`cat $TMPFILE |tr '<>' '\n' |grep -m1 -B1 /title |head -1`
            if [ -z "$STATUS" ] ; then
```

Testaustyökalu *fetch.sh*

```
102                 echo "=======> ERROR! 'curl' responded:"
103                 cat $TMPFILE
104                 echo ""
105             else
106                 echo "=======> ERROR! HTML <title>
    was: \"$STATUS\""
107             fi
108             touch $OUTPUT
109         else
110             cat $TMPFILE |python3 -m json.tool --sort-keys
    >$OUTPUT
111         fi
112     fi
113     rm -f $TMPFILE
114 }
115
116 # fetch() performs a GET request to URI "$1"
117
118 fetch () {
119     doCurl GET $1
120     FIRST_UUID=`echoFirst $OUTPUT uuid`
121     [ -z "$FIRST_UUID" ] && {
122         FIRST_GROUPNAME=`echoFirst $OUTPUT groupName`
123         FIRST_UNITUUID=`echoFirst $OUTPUT unitUuid`
124     }
125 }
126
127 # create() performs a POST request to URI "$1" with body "$2"
128
129 create () {
130     doCurl POST $1 "$2"
131     NEW_UUID=`echoFirstInside $OUTPUT uuid newRecord`
132     [ -z "$NEW_UUID" ] && echo "=======> ERROR! `echoFirstInside
    $OUTPUT message error`"
133 }
134
135 # update() performs a PUT request to URI "$1" with body "$2"
136
137 update () {
138     doCurl PUT $1 "$2"
139     OLD_UUID=`echoFirstInside $OUTPUT uuid updatedRecord`
140     [ -z "$OLD_UUID" ] && echo "=======> ERROR! `echoFirstInside
    $OUTPUT message error`"
141 }
142
143 # Helpful string constants for building JSON request bodies
144
145 U='"uuid":'
146 NAME_1='"name": "API Test"'
147 MSG_CREATE='<p>Made by <code>fetch.sh</code></p>'
148 MSG_UPDATE='<p><em>MODIFIED</em> by <code>fetch.sh</code></p>'
149 DESC_1='"description": "'$MSG_CREATE'"'
150 DESC_2='"description": "'$MSG_UPDATE'"'
151
152 ################
153 ################ API CALLS
154 ################
155
156 # Portfolios
```

Testaustyökalu *fetch.sh*

```
157
158 fetch portfolios
159 portfolioUuid=$FIRST_UUID
160 [ -n "$portfolioUuid" ] && {
161     fetch portfolios/$portfolioUuid
162     fetch portfolios/$portfolioUuid/properties
163     fetch portfolios/$portfolioUuid/ownershipShares
164     portfolioOwnershipShareUuid=$FIRST_UUID
165     [ -n "$portfolioOwnershipShareUuid" ] && {
166         fetch portfolios/ownershipShares/
    $portfolioOwnershipShareUuid
167     }
168 }
169
170 PORTFOLIO_TYPE='"portfolioTypeSetting": "Portfolio"'
171
172 create portfolios "{ $NAME_1, $DESC_1, $PORTFOLIO_TYPE }"
173 newPortfolioUuid=$NEW_UUID
174 [ -n "$newPortfolioUuid" ] && {
175     update portfolios "{ $U \"$newPortfolioUuid\", $NAME_1,
    $DESC_2, $PORTFOLIO_TYPE }"
176 }
177
178 # Properties
179
180 fetch properties
181 propertyUuid=$FIRST_UUID
182 [ -n "$propertyUuid" ] && {
183     fetch properties/$propertyUuid
184     fetch properties/$propertyUuid/ownershipShares
185     fetch properties/$propertyUuid/buildings
186     fetch properties/$propertyUuid/plots
187     fetch properties/$propertyUuid/units
188     fetch properties/$propertyUuid/notes
189     fetch properties/$propertyUuid/assetValuations
190     assetValuationUuid=$FIRST_UUID
191     [ -n "$assetValuationUuid" ] && {
192         fetch properties/assetValuations/$assetValuationUuid
193     }
194 }
195
196 COST_CENTER='"costCenterId": "APITestCC"'
197 CITY='Lappeenranta'
198 COUNTRY='FI'
199 ZIP_A='53850'
200 ZIP_B='53851'
201 STREET_A='Laserkatu 6'
202 STREET_B='Laserkatu 6 B 18'
203 ADDRESS_A='{ "city": "'$CITY'", "country": "'$COUNTRY'",
    "postCode": "'$ZIP_A'", "streetAddress": "'$STREET_A'" }'
204 ADDRESS_B='{ "city": "'$CITY'", "country": "'$COUNTRY'",
    "postCode": "'$ZIP_B'", "streetAddress": "'$STREET_B'" }'
205 PROP_ADDRESS='"mainPropertyAddress": '$ADDRESS_A
206
207 create properties "{ $NAME_1, $DESC_1, $COST_CENTER, $PROP_ADDRESS
    }"
208 newPropertyUuid=$NEW_UUID
209 [ -n "$newPropertyUuid" ] && {
210     update properties "{ $U \"$newPropertyUuid\", $NAME_1,
```

Testaustyökalu *fetch.sh*

```
    $DESC_2, $COST_CENTER, $PROP_ADDRESS }"
211 }
212
213 fetch properties/ownershipShares
214 ownershipShareUuid=$FIRST_UUID
215 [ -n "$ownershipShareUuid" ] && {
216     fetch properties/ownershipShares/$ownershipShareUuid
217 }
218
219 # Buildings
220
221 fetch properties/buildings
222 buildingUuid=$FIRST_UUID
223 [ -n "$buildingUuid" ] && {
224     fetch properties/buildings/$buildingUuid
225 }
226
227 [ -n "$newPropertyUuid" ] && {
228
229     UUID='"propertyUuid": "'$newPropertyUuid'"'
230     BUILDING_ID='"buildingId": "APITestBuilding"'
231
232     create properties/buildings "{ $UUID, $DESC_1, $BUILDING_ID }"
233     newBuildingUuid=$NEW_UUID
234     [ -n "$newBuildingUuid" ] && {
235         update properties/buildings "{ $U \"$newBuildingUuid\",
    $UUID, $DESC_2, $BUILDING_ID }"
236     }
237 }
238
239 # Plots
240
241 fetch properties/plots
242 plotUuid=$FIRST_UUID
243 [ -n "$plotUuid" ] && {
244     fetch properties/plots/$plotUuid
245 }
246
247 [ -n "$newPropertyUuid" ] && {
248
249     UUID='"propertyUuid": "'$newPropertyUuid'"'
250     PLOT_NAME='"plotName": "APITestPlot"'
251     PLOT_ADDRESS='"plotAddress": '$ADDRESS_B
252
253     create properties/plots "{ $UUID, $DESC_1, $PLOT_NAME,
    $PLOT_ADDRESS }"
254     newPlotUuid=$NEW_UUID
255     [ -n "$newPlotUuid" ] && {
256         update properties/plots "{ $U \"$newPlotUuid\", $UUID,
    $DESC_2, $PLOT_NAME, $PLOT_ADDRESS }"
257     }
258 }
259
260 # Asset valuations
261
262 [ -n "$newPropertyUuid" ] && {
263
264     UUID='"propertyUuid": "'$newPropertyUuid'"'
265     DATE='"valuationDate": "2015-07-01"'
```

Testaustyökalu *fetch.sh*

```
266     VALUE_1='"marketValue": { "value": "1200000.00",
    "currencyCode": "USD" }'
267     VALUE_2='"marketValue": { "value": "1275000.00",
    "currencyCode": "USD" }'
268
269     create properties/assetValuations "{ $UUID, $DATE, $VALUE_1 }"
270     newValuationUuid=$NEW_UUID
271     [ -n "$newValuationUuid" ] && {
272         update properties/assetValuations
    "{ $U \"$newValuationUuid\", $UUID, $DATE, $VALUE_2 }"
273     }
274 }
275
276 # Units
277
278 fetch units
279 unitUuid=$FIRST_UUID
280 [ -n "$unitUuid" ] && {
281     fetch units/$unitUuid
282     fetch units/$unitUuid/notes
283 }
284
285 [ -n "$newPropertyUuid" ] && {
286
287     PROP_UUID='"propertyUuid": "'$newPropertyUuid'"'
288     UNIT_ID='"unitExternalId": "APITestUnit"'
289     UNIT_TYPE='"unitTypeSetting": "Apartment"'
290     VENT_TYPE='"ventilationTypeSetting": "Gravitational"'
291
292     create units "{ $PROP_UUID, $DESC_1, $UNIT_ID, $UNIT_TYPE,
    $VENT_TYPE }"
293     newUnitUuid=$NEW_UUID
294     [ -n "$newUnitUuid" ] && {
295         update units "{ $U \"$newUnitUuid\", $PROP_UUID, $DESC_2,
    $UNIT_ID, $UNIT_TYPE, $VENT_TYPE }"
296     }
297 }
298
299 # Contacts
300
301 fetch persons
302 personUuid=$FIRST_UUID
303 [ -n "$personUuid" ] && {
304     fetch persons/$personUuid
305     fetch persons/$personUuid/notes
306 }
307
308 fetch organisations
309 organisationUuid=$FIRST_UUID
310 [ -n "$organisationUuid" ] && {
311     fetch organisations/$organisationUuid
312     fetch organisations/$organisationUuid/persons
313     fetch organisations/$organisationUuid/notes
314 }
315
316 fetch organisations/persons
317 personUuid=$FIRST_UUID
318 [ -n "$personUuid" ] && {
319     fetch organisations/persons/$personUuid
```

Testaustyökalu *fetch.sh*

```
320 }
321
322 FIRST_NAME='"firstName": "API"'
323 LAST_NAME='"lastName": "Clone"'
324
325 create persons "{ $FIRST_NAME, $LAST_NAME, $DESC_1 }"
326 newPersonUuid=$NEW_UUID
327 tenantUuid=$newPersonUuid
328 ownerUuid=$newPersonUuid
329 ownerType='PRIVATE_INDIVIDUAL'
330 [ -n "$newPersonUuid" ] && {
331     update persons "{ $U \"$newPersonUuid\", $FIRST_NAME,
    $LAST_NAME, $DESC_2 }"
332 }
333
334 ORG_NAME='"name": "API Clones Inc."'
335 INDUSTRY='"industryTypeSetting": "Manufacturing"'
336
337 create organisations "{ $ORG_NAME, $INDUSTRY, $DESC_1 }"
338 newOrgUuid=$NEW_UUID
339 [ -n "$newOrgUuid" ] && {
340     update organisations "{ $U \"$newOrgUuid\", $ORG_NAME,
    $INDUSTRY, $DESC_2 }"
341     [ -z "$ownerUuid" ] && {
342         ownerUuid=$newOrgUuid
343         ownerType='ORGANISATION'
344     }
345 }
346
347 ORG_UUID='"organisationUuid": "'$newOrgUuid'"'
348 LAST_NAME_1='"lastName": "Corporate Clone"'
349 LAST_NAME_2='"lastName": "Clone of a Clone"'
350
351 create organisations/persons "{ $ORG_UUID, $FIRST_NAME,
    $LAST_NAME_1 }"
352 newOrgPersonUuid=$NEW_UUID
353 [ -n "$newOrgPersonUuid" ] && {
354     update organisations/persons "{ $U \"$newOrgPersonUuid\",
    $ORG_UUID, $FIRST_NAME, $LAST_NAME_2 }"
355 }
356
357 # Portfolio/property ownership shares
358
359 PERC='"percentage":'
360
361 [ -n "$newPortfolioUuid" ] && [ -n "$ownerUuid" ] && {
362
363     UUID='"portfolioUuid": "'$newPortfolioUuid'"'
364     OWNER='"owner": { "uuid": "'$ownerUuid'", "contactType":
    "'$ownerType'" }'
365
366     create portfolios/ownershipShares "{ $OWNER, $UUID, $PERC
    12 }"
367     shareUuid=$NEW_UUID
368     [ -n "$shareUuid" ] && {
369         update portfolios/ownershipShares "{ $U \"$shareUuid\",
    $OWNER, $UUID, $PERC 16 }"
370     }
371 }
```

Testaustyökalu *fetch.sh*

```
372 [ -n "$newPropertyUuid" ] && [ -n "$newOrgUuid" ] && {
373
374     UUID='"propertyUuid": "'$newPropertyUuid'"'
375     OWNER='"organisationUuid": "'$newOrgUuid'"'
376     MONTH='"propertyShareMonth": "2015-04-01"'
377
378     create properties/ownershipShares "{ $NAME_1, $DESC_1, $OWNER,
    $UUID, $MONTH, $PERC 15 }"
379     shareUuid=$NEW_UUID
380     [ -n "$shareUuid" ] && {
381         update properties/ownershipShares "{ $U \"$shareUuid\",
    $NAME_1, $DESC_2, $OWNER, $UUID, $MONTH, $PERC 18 }"
382     }
383 }
384
385 # Leases
386
387 fetch leases
388 leaseUuid=$FIRST_UUID
389 [ -n "$leaseUuid" ] && {
390     fetch leases/$leaseUuid
391     fetch leases/$leaseUuid/notes
392     fetch leases/$leaseUuid/leasePeriods
393     leasePeriodUuid=$FIRST_UUID
394     [ -n "$leasePeriodUuid" ] && {
395         fetch leases/leasePeriods/$leasePeriodUuid
396         fetch leases/$leaseUuid/leasePeriods/
    $leasePeriodUuid/unitRents
397         unitRentUuid=$FIRST_UUID
398         [ -n "$unitRentUuid" ] && {
399             fetch leases/leasePeriods/unitRents/$unitRentUuid
400         }
401     }
402 }
403
404 fetch leases/leasePeriods
405 fetch leases/leasePeriods/unitRents
406
407 [ -n "$newPropertyUuid" ] && [ -n "$newUnitUuid" ] && [ -n
    "$tenantUuid" ] && {
408
409     PROP_UUID='"propertyUuid": "'$newPropertyUuid'"'
410     UNIT_UUID='"unitUuid": "'$newUnitUuid'"'
411     TENANT='"tenant": { "uuid": "'$tenantUuid'", "contactType":
    "PRIVATE_INDIVIDUAL" }'
412     LEASE_ID='"contractId": "APITest1"'
413     LEASE_START='"leaseAgreementStart": "2015-02-01"'
414     LEASE_END='"leaseAgreementEnd": "2016-07-31"'
415     PERIOD='"leasePeriod": { "startDate": "2016-02-14" }'
416     RENT='"rentAmount": { "value": "640.00", "currencyCode": "EUR"
    }'
417     UNIT_RENT='"unitRent": { '$UNIT_UUID', '$RENT' }'
418
419     create leases "{ $PROP_UUID, $TENANT, $LEASE_ID, $LEASE_START,
    $LEASE_END, $PERIOD, $UNIT_RENT }"
420     newLeaseUuid=$NEW_UUID
421
422     [ -n "$newLeaseUuid" ] && {
423
```

Testaustyökalu *fetch.sh*

```
424         COMMENT='"leaseTermsComments": "'$MSG_UPDATE'"'
425
426         update leases "{ $U \"$newLeaseUuid\", $COMMENT,
    $PROP_UUID, $TENANT, $LEASE_ID, $LEASE_START, $LEASE_END }"
427         LEASE_UUID='"leaseUuid": "'$newLeaseUuid'"'
428         PERIOD_START='"startDate": "2016-05-01"'
429
430         create leases/leasePeriods "{ $LEASE_UUID,
    $PERIOD_START }"
431         newPeriodUuid=$NEW_UUID
432         [ -n "$newPeriodUuid" ] && {
433
434             PERIOD_START='"startDate": "2016-05-21"'
435             update leases/leasePeriods "{ $U \"$newPeriodUuid\",
    $LEASE_UUID, $PERIOD_START }"
436
437             PERIOD_UUID='"leasePeriodUuid": "'$newPeriodUuid'"'
438             RENT='"rentAmount": { "value": "695.00",
    "currencyCode": "EUR" }'
439             create leases/leasePeriods/unitRents "{ $PERIOD_UUID,
    $UNIT_UUID, $RENT }"
440             newUnitRentUuid=$NEW_UUID
441             [ -n "$newUnitRentUuid" ] && {
442
443                 RENT='"rentAmount": { "value": "700.00",
    "currencyCode": "EUR" }'
444
445                 update leases/leasePeriods/unitRents
    "{ $U \"$newUnitRentUuid\", $PERIOD_UUID, $UNIT_UUID, $RENT }"
446             }
447         }
448     }
449 }
450
451 # Currencies
452
453 fetch supportedCurrencies
454
455 # Settings
456
457 fetch settingGroups
458 groupName=$FIRST_GROUPNAME
459 [ -n "$groupName" ] && {
460     fetch settingGroups/$groupName/settings
461 }
462
463 # Notes
464
465 fetch notes
466 noteUuid=$FIRST_UUID
467 [ -n "$noteUuid" ] && {
468     fetch notes/$noteUuid
469 }
470
471 TITLE='"title": "API Test Note"'
472 BODY='"comment": "'$MSG_CREATE'"'
473 METADATA="$TITLE"
474
475 [ -n "$newPropertyUuid" ] && {
```

Testaustyökalu *fetch.sh*

```
476     PROP_UUID='"propertyUuid": "'$newPropertyUuid'"'
477     METADATA="$METADATA, $PROP_UUID"
478 }
479 
480 create notes "{ $BODY, $METADATA }"
481 newNoteUuid=$NEW_UUID
482 [ -n "$newNoteUuid" ] && {
483 
484     BODY='"comment": "'$MSG_UPDATE'"'
485 
486     update notes "{ $U \"$newNoteUuid\", $BODY, $METADATA }"
487 
488     NOTE_UUID='"noteUuid": "'$newNoteUuid'"'
489     BODY='"comment": "'$MSG_CREATE'"'
490 
491     create notes/replies "{ $NOTE_UUID, $BODY }"
492     newReplyUuid=$NEW_UUID
493     [ -n "$newReplyUuid" ] && {
494         BODY='"comment": "'$MSG_UPDATE'"'
495         update notes/replies "{ $U \"$newReplyUuid\", $NOTE_UUID,
    $BODY }"
496     }
497 }
498 
499 # Vacant units
500 
501 fetch vacantUnits
502 vacantUnitUuid=$FIRST_UNITUUID
503 [ -n "$vacantUnitUuid" ] && {
504     fetch vacantUnits/$vacantUnitUuid/unitPictures
505     fetch vacantUnits/$vacantUnitUuid/propertyPictures
506 }
507 
508 # Save MD5 checksums of all responses for easier comparison
509 # NOTE: the response received by create() will be different each
    time,
510 # because it contains the UUID of a new entity!
511 
512 [ -n `which md5sum` ] && md5sum `ls -d $SERVER.*.json`
    >$SERVER.md5sum
```