Timo Katajamäki

RÄÄTÄLÖIDYN MIKROPALVELUKOKONAISUUDEN TOTEUTTAMINEN PILVIPALVELUNA

Tietotekniikan koulutusohjelma
2017

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

RÄÄTÄLÖIDYN MIKROPALVELUKOKONAISUUDEN TOTEUTTAMINEN PILVIPALVELUNA

Katajamäki, Timo
Satakunnan ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Tammikuu 2017
Ohjaaja: Ekholm, Ari
Sivumäärä: 29

Asiasanat: sovellus, python, ohjelmointirajapinta, autentikointi, rest

---

Kiinan mobiilisovellusmarkkinoille pääsy vaatii Kiinalaisen julkaisuyhtiön, jolla on lisenssi mobiilipelien julkaisuun. Lisenssihankinnan luomat haasteet eväävät monelta mobiilipelitalolta mahdollisuuden julkaista omissa nimissään pelejä Kiinan markkinoilla. Lisenssihaasteiden lisäksi useimmmat sovelluskauppapaikat vaativat omien SDK:idensa integroimista julkaistaviin peleihin, mikä luo omat haasteensa ylläpidollisiin tehtävin.

Jotta edellisen vaatimukset täyttyvät, MyGamez on luonut MySDK-ratkaisun, joka pitää sisällään useita eri SDK:ita. Ratkaisun ympärille on lisäksi kehitetty palvelinympäristö, joka sisältää joukon erilaisia palveluita. Opinnäytetyössä selvitettiin toimintapainotteisen toimintatutkimuksen menetelmällä MySDK-tuotteen ympärille kehittyvän mikropalvelukokonaisuuden hallintaa ja tuotteistamista.

Opinnäytetyössä määriteltiin ja kehitettiin järjestelmä mikropalveluista koostuvien ympäristöjen hallintaan. Järjestelmä koostuu REST-rajapinnasta sekä sitä kuluttavista käyttöliittymistä. Yhtenä isona kokonaisuutena oli toteuttaa loppukäyttäjien keskitetty hallinta sekä nykyaikaisten standardien mukainen käyttäjän tunnistaminen.

Opinnäytetyö osoitti, että REST-rajapinta soveltuu hyvin tietojen hallintaan. Työtä voidaan siten käyttää esimerkkinä toimintatutkimuksen soveltamisesta ohjelmistoprojektissa, jonka tavoitteena on etsiä ja kehittää tehokkaampia ratkaisuja. Lopputuloksena on selkeä ja helposti laajennettavissa oleva järjestelmä, joka toteuttaa vaatimusmäärittelyn.

IMPLEMENTING A CUSTOMIZED MICROSERVICE ENVIRONMENT AS A CLOUD SERVICE

Katajamäki, Timo
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Information Technology
January 2017
Supervisor: Ekholm, Ari
Number of pages: 29

Keywords: application, python, api, authentication, rest

Access to China's mobile apps market requires a Chinese company which holds a license to publish mobile games in China. The challenges in acquiring the publishing license makes it difficult for foreign mobile game producers to release their games on the Chinese mobile apps markets. Additionally, many of the app stores in China require integration of their own SDKs into the games, which present upkeep challenges.

To solve these problems, MyGamez has created MySDK solution, which handles all different SDK requirements that app stores have. There is also a backend service infrastructure built around the MySDK. In this thesis, action research was used to find and create a solution for managing and productionizing all the microservices being built around MySDK.

This thesis defines a system to manage separate environments that comprise of microservices. The system includes a RESTful API and the clients which consume the API. Other large part of the thesis was to create a system for managing end users centrally, and provide a modern way to authorize user access.

The thesis showed that RESTful APIs are a good way to manage information. Thus the thesis can be used as an example of a successful use of action research in a software project. The end result is a clean and easily extendable system, which meets the requirement specifications.

SISÄLLYS

LYHENTEET JA TERMIT

API
Application Programming Interface, Ohjelmointirajapinta, määrittelee menetelmät tietojen syöttämiseen sekä tuottamiseen.

Autentikointi
Käyttäjän identiteetin todentaminen esimerkiksi salasanan avulla.

Edustapalvelin
Reverse proxy, on tyypillisesti julkiseen internettiin näkyvä palvelinohjelma joka ohjaa HTTP-kyselyjä määriteltyjen sääntöjen mukaisesti toisille palvelinohjelmille.

Eväste
Verkkopalvelimen käyttäjän koneelle asettamia avain-arvo –pareja, tyypillisesti käytössä selaimissa.

HTTPS
HTTP Secure, HTTP sekä TLS/SSL-protokollien yhdistelmä jonka avulla verkkoliikenne suojataan asiakasohjelman ja palvelimen välillä.

HTTP-tilakoodi
Suoritettaessa HTTP-pyynnön, palvelimelta tulevassa vastauksessa on numeerinen standardoitu tilakoodi joka kertoo vastauksen tuloksen (onnistunut, epäonnistunut...).

JDBC
Java Database Connectivity, on Java-ohjelmointikielessä rajapinta relaatiotietokannan käyttämiseen.

JSON
JavaScript Object Notation, RFC4627:n mukainen formaatti tiedon esittämiseen avain-arvo –pareina.

JWK

JSON Web Key, JSON-muodossa esitetty kryptografinen avain.

JWS
JSON Web Signature, digitaalisesti allekirjoitettua JSON muotoista tietoa.

JWT
JSON Web Token, JSON-muotoinen käyttäjätunniste joka sisältää eri tietoja käyttäjästä.

Muotoiluohjelma
Templating engine, ohjelma joka renderoi mallipohjasta ja avain-arvo –pareista mallipohjan mukaisen dokumentin.

MariaDB
Avoimen lähdekoodin relaatiotietokantajärjestelmä.

Mikropalvelu
Perinteisen monoliittisen sovelluksen vastakohta, mikropalvelun tehtävänä on toteuttaa itsenäisenä ja tilattomana palveluna pienemmän toimintokokonaisuuden.

OAuth 2.0
Avoin standardi käyttäjien tunnistautumiseen web-palveluille.

OIDC
OpenID Connect, OAuth 2.0:n päälle rakennettu tunnistautumiskehys.

OP
OpenID Provider, OAuth 2.0 tunnistautumispalvelin joka kykenee tunnistamaan käyttäjän sekä tarjoamaan RP:lle tietoja käyttäjästä.

ORM
Object Relational Mapper, työkalu joka abstraktoi tietokantakuvauksen ohjelmassa muodostettuun olioon.

PaaS

Platform as a Service, palvelualustan tarjoaminen ulkoisesti tuotettuna.

Python

Korkean tason tulkattu, dynaaminen ohjelmointikieli.

RDBMS

Relaatiotietokantajärjestelmä.

REST

Representational State Transfer, HTTP-protokollaa käyttävä arkkitehtuurimalli ohjelmointirajapintojen luomiseen.

RP

Relying Party, OpenID Connect -asiakasohjelma joka vaatii loppukäyttäjän tunnistetiedot OP:lta.

(De-) Serialisointi

Tarkoittaa ohjelmassa tiedon muuntamista ennalta määritellyn tietorakenteen pohjalta ohjelman sisällä eläväksi olioksi, tai olion muuntamista tiedoksi.

SaaS

Software as a Service, ohjelmiston tarjoaminen ulkoisesti tuotettuna palveluna.

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena oli toteuttaa järjestelmä helpottamaan useammasta pienemmästä mikropalvelusta koostuvien ympäristöjen hallinnointia keskitetysti. Opinnäytetyössä käytetään toimintapainotteisen toimintatutkimuksen menetelmää järjestelmän luomiseen.

MyGamez on Kiinan android-markkinoilla toimiva mobiilipelijulkaisija, jonka pääsääntöisiä asiakkaita ovat länsimaalaiset mobiilipelitalot. Jotta yhtiö voisi julkaista mobiilipelin Kiinassa, tulee sillä olla Kiinalainen yhtiö, jolle on myönnetty lisenssi mobiilipelien julkaisuun. Tästä suljetusta luonteesta johtuen monen pelitalon realistinen vaihtoehto on etsiä julkaisukumppani, joka toimii jo Kiinan markkinoilla.

Kiinan valtio on estänyt Googlen palvelujen käytön maassaan. Tästä syystä Kiinan android-markkinoilla on yhteensä yli 200 sovelluskauppapaikkaa meille tutumman, Google Playn sijasta. Näitä sovelluskauppapaikkoja hallinnoivat eri operaattorit, laitevalmistajat ja muut yhtiöt. Näistä kauppapaikoista useimmilla on lisäksi omia vaatimuksiaan: yhtenä hyvin tyypillisenä vaatimuksena esimerkiksi kauppapaikan oman SDK:n integrointi julkaistaviin peleihin.

Tätä taakkaa helpottamaan MyGamez on luonut MySDK-ratkaisun, joka integroi useita eri SDK:ita itseensä ja tarjoaa yksinkertaisen rajapinnan pelikehittäjälle.

Opinnäytetyön tarkoituksena on luoda puitteet ”MyGamez as a Service” –tyyppiseen ratkaisuun, joka on osaltaan SaaS ja PaaS –hybridi. Alustalla toivotaan tavoitettavan asiakkaita, joilla on hyvin vahvat immateriaalioikeudet ja brändit.

2 OPINNÄYTETYÖN LÄHTÖKOHDAT

2.1 Vanha palvelinympäristö

Yhtiön käytössä oleva järjestelmä on vuosien aikana paisunut vaikeasti ylläpidettäväksi. Järjestelmässä ei ole tällä hetkellä selkeää rajapintaa ja käyttöliittymää ei ole eritelty toiminnallisuuksita.

2.2 Uusi järjestelmä

Kehitteillä oleva uusi mikropalveluista koostuva ympäristö luo standardoidut rajapinnat eri palveluille. Tämä helpottaa palvelujen välistä tiedonvaihtoa sekä käyttöliittymän kehittämistä.

Mikropalvelu-arkkitehtuuri kuitenkin tuo mukanaan omia haasteitaan, joihin toteutuksessa pureuduttiin. Yhtenä haasteena on palvelukohtaisten asetusten määritteleminen sekä ympäristön koostaminen.

2.3 Opinnäytetyön vaatimusmäärittely

MyGamezilla oli selkeät vaatimukset järjestelmälle:

1. Järjestelmän pitää kyetä koostamaan ja hallitsemaan useampia mikropalveluja.
2. Järjestelmän koostamille palveluille on kyettävä tarjoamaan asetuksia.
3. REST-arkkitehtuurimallia mukaileva toteutus rajapinnalle.
4. Selaimella käytettävä käyttöliittymä.
5. Käyttäjien helppo hallittavuus keskitetysti.
6. Käyttäjille oikeuksien määrittely.
7. Palvelimen käyttöjärjestelmä on unix-kaltainen Debian linux, järjestelmän on tuettava tätä käyttöjärjestelmäympäristöä.

3 SUUNNITTELU

3.1 Käytettävät teknologiat

Järjestelmän implementoinnissa käytettävän ohjelmointikielen tulisi olla toteuttajalle mieluinen. Tutkimukset ovat osoittaneet tällä olevan vaikutusta tuottavuuteen sekä koodin laatuun (McConnell 2004, 62). Asiakkaalla ei ollut järjestelmän toteutuksessa käytettävän ohjelmointikielen suhteen toivomusta, joten henkilökohtaisten preferenssien myötä päädyin käyttämään mahdollisista ohjelmointikielistä Pythonia.

Pythonin ympärille on rakennettu hyvin kattava standardikirjasto, hyvät dokumentaatiot sekä kattava valikoima kypsiä ja tuotantoon sopivia ohjelmistokehyksiä kattamaan järjestelmän perustarpeet.

3.2 Järjestelmäarkkitehtuuri

Asiakkaan toivomuksena oli toteuttaa järjestelmä noudattaen ohjelmointirajapinta REST-arkkitehtuuria, jossa jokainen rajapintakysely on tilaton. Tällä tarkoitetaan kyselyitä, jotka ovat toisistaan täysin riippumattomia sekä pitävät sisällään kaiken tarvittavan tiedon yhden kyselyn suorittamiseen. REST-arkkitehtuurin pyrkimyksenä on yksinkertaistaa järjestelmää sekä helpottaa skaalaamista.

Rajapinta ja käyttöliittymä tulisivat olemaan omia itsenäisiä komponentteja järjestelmässä. Käyttäjähallinan osalta tutustuin eri avoimen lähdekoodin projekteihin, joista osaksi järjestelmää nousi Keycloak.

3.3 Käyttöliittymä

Vaatimuksia määritellessä käyttöliittymää koskevat vaatimukset olivat hyvin selkeitä: käyttöliittymän tulee olla selainpohjainen. Tämä ei ole vielä kovin rajaava vaade, joten päätin pitää asiat mahdollisimman yksinkertaisina käyttöliittymää suunnitellessa. Siten päädyin suunnittelussani käyttämään perinteisiä HTML5-, CSS-

ja JavaScript –teknologioita. Käyttöliittymän tyylittämiseen tulisin käyttämään Bootstrap CSS-kehystä, jonka avulla on helppo luoda mobiiliystävällisiä sivustoja.

Sivujen dynaaminen sisältö tultaisiin muodostamaan palvelinpuolella sivumalleista renderoimalla. Tällöin raskain taakka ei kohdistu heikompitehoisiin laitteisiin, joilla sivuja käytetään, sillä myös rajapintakyselyt ovat todella nopeita rajapinnan ja käyttöliittymän sijaitessa samalla palvelimella.

3.4 REST-rajapinta

REST-arkkitehtuurimallissa käytettäviä HTTP-metodeja ovat GET, POST, PATCH, PUT ja DELETE:

1. GET-metodia käytetään haluttaessa noutaa tietoja.
2. POST-metodi on tarkoitettu uusien tietojen luomiseen.
3. PUT-metodilla muokataan jo olemassaolevia tietoja.
4. DELETE-metodia käytetään poistettaessa tietoja.

Rajapinnan tulisi siis noudattaa näitä HTTP-metodeja. Rajapinnan hallinnoimia tietokokonaisuuksia voidaan kutsua resursseiksi. Selkeyden vuoksi päätin eriyttää resurssit rajapinnan linkkipoluissa. Esimerkiksi uuden resurssin luominen järjestelmään tapahtuisi lähettämällä POST-kyselyn polkuun “/resurssi”.

Rajapintaa suunnitellessa on hyvä yhtenäistää siirrettyjen tietojen formaatti. Nykyaikainen ja selkeä formaatti on JSON, joka on syrjäyttänyt XML:n rajapintojen yleisimpänä formaattina.

Jotta REST-arkkitehtuurimallin tilattomuus toteutuu, käyttäjien autentikointitiedot tulee lähettää jokaisen rajapintakyselyn mukana. Nykyaikaisena autentikointimenetelmänä voidaan pitää OAuth 2.0 –tunnisteita, jotka täyttävät tilattomuuden vaatimukset hyvin.

4 KÄYTETYT TEKNOLOGIAT

4.1 Python

Python ohjelmointikieli on yksi mailman suosituimmista ohjelmointikielistä. Sen merkittävimpiin ominaisuuksiin kuuluu koodin helppolukuisuus ja syntaksi, joka pyrkii maksimoimaan tuottavuuden. Pythonin etuna on myös alustariippumattomuus (Lutz 2014, 1). Python on korkean tason tulkattava ohjelmointikieli: sen sijaan, että koodi käännettäisiin kokonaan konekieliseksi, tulkin tehtävänä on kääntää lähdekoodi ennalta käännetyiksi konekielisiksi proseduureiksi.

Python on haarautunut kahteen kehityshaaraan, jotka ovat versio 2.x sekä 3.x. Tämä on osaltaan jakanut yhteisöä johtuen 3.x-haaran syntaksin yhteensopimattomuudesta 2.x-haaran syntaksin kanssa. Python 3 on julkaistu joulukuussa 2008, ja tätä työtä kirjoittaessani vanhempi Python 2 on edelleen suositumpi versio. Yleinen suositus on kuitenkin aloittaa uudet projektit Python 3:lla.

4.1.1 Flask

Flask on web-ohjelmistokehys, jonka tavoitteena on pitää ohjelmistokehyksen ydin mahdollisimman yksinkertaisena, ja näin tarjota ohjelmoijalle täyden valinnan-vapauden valita laajennuksista itselleen mieluisimmat vaihtoehdot toteuttamaan halutut toiminnot. Kuitenkin joitain valintoja on tehty ohjelmoijan puolesta, mutta nekin ovat helposti korvattavissa. Eräs tälläinen valinta on ollut käyttää Jinja2 – merkintäkieltä sivumallien luomiseen.

4.1.2 SQLAlchemy

SQLAlchemy on ORM, jonka avulla voidaan natiivilla python-koodilla kuvata tietorakenteita olioina. Yksi taulu voidaan esimerkiksi kuvata oliona, jolle voidaan määritellä tarpeiden mukaan konstruktori, metodeja sekä arvoja, jotka yhdistetään tietokannassa oleviin taulun sarakkeihin.

4.1.3 Gunicorn

Green Unicorn, eli gunicorn on Python WSGI (Web Server Gateway Interface) HTTP-palvelin. WSGI on spesifikaatio universaalille rajapinnalle web-palvelimen ja web-sovelluksen tai ohjelmistokehyksen välille.

Gunicornin tehtävänä on luoda ajettavasta web-ohjelmasta useita itsenäisiä aliohjelmia, joita se hallinnoi yhdellä isäntäprosessilla. Tämä on tarpeen, koska python on synkroninen kieli, jonka johdosta kukin aliohjelma kykenee palvelemaan vain yhtä asiakasta kerrallaan. Lisäprosessien luonti on palvelua skaalaava toimenpide, ja ennaltaehkäisee pullonkaulojen muodostumista.

4.1.4 Jinja2

Jinja2 on merkintäkieli, jota käytetään pääsääntöisesti dynaamisten HTML-sivujen renderointiin palvelinpuolella. Jinja2-mallipohjat kääntyvät pythonkoodiksi, ja siksi sen syntaksi onkin pythonia mukaileva. Siitä löytyykin tuki eristetylle renderoinnille, joka parantaa tietoturvaa tapauksissa, joissa itse mallipohja voi olla dynaaminen.

4.1.5 Marshmallow

Marshmallow on kirjasto tietomallien serialisointiin, deserialisointiin sekä validointiin. Mallien määrittely voidaan tehdä normaalin luokkamäärittelyn mukaisesti periyttämällä malliluokka kirjastosta löytyvästä malliluokasta.
Marshmallow-SQLAlchemy -laajennoksen avulla on mahdollista määritellä tietomallin pohjaksi SQLAlchemy-malli.

4.1.6 Sphinx

Sphinx on reStructuredText (rst) merkkauskieltä käyttävä dokumentointityökalu, jonka avulla voidaan ohjelman lähdekoodiin upotetuista docstringeistä luoda

älykkäitä ja selkeitä dokumentaatioita. Httpdomain-laajennos tuo Sphinxiin tuen HTTP-pyyntöjen merkkaukseen.

4.1.7 pytest

Pytest on laajennoksia tukeva testauskehys, joka yksinkertaistaa testien kirjoittamisen. Siinä testifunktioissa arvojen varmentaminen tapahtuu “assert”-lauseella: esimerkiksi Python-kieleen integroidussa unittest-moduulissa arvojen varmentamisille on useita metodeja riippuen halutusta vertauksesta `assertEqual(muuttuja, "arvo")`.

Testit ovat mahdollista kirjoittaa kokonaisuuksia kattaviin tiedostoihin. Lisäksi kaikkia testejä koskevia määrityksiä voidaan luoda conftest.py-tiedostoon.

4.2 Keycloak

Keycloak on avoimen lähdekoodin järjestelmä käyttäjien hallitsemiseen, tunnistamiseen ja auktorisointiin. Järjestelmästä löytyy tuki LDAP sekä Active Directory –hakemistopalveluille, mahdollisuus luoda klustereita tasaamaan kuormitusta tai tarjoamaan parempaa ja luotettavampaa yhteyttä järjestelmään. Keycloak tarjoaa tuen projektin kannalta merkittävälle protokollalle, joka on OpenID Connect. (Keycloak projektin www-sivut 2016).

4.2.1 OpenID Connect

OpenID Connect on OAuth 2.0 protokollan ylle rakennettu käyttäjän tunnistamiskehys. OIDC-ympäristö koostuu OP:sta, RP:stä ja loppukäyttäjistä, jossa RP on riippuvainen OP:n käyttäjästä tuottamista varmennetuista tiedoista. Projektissa RP:n asemaa noudattaa rajapinta sekä sen käyttöliittymät.

OpenID Connectin tavanomaisen “Authorization Code Flow”in mukainen käyttäjän tunnistaminen:

1. RP valmistelee autentikointipyynnön joka sisältää halutut parametrit
2. RP lähettää valmistellun autentikointipyynnön OP:lle
3. OP pyytää käyttäjää tunnistautumaan
4. OP pyytää käyttäjältä suostumuksen tietojen luovuttamiseen RP:lle
5. OP ohjaa käyttäjän takaisin RP:lle tunnnistekoodin kera
6. RP suorittaa pyynnön OP:n tunnistepäätepisteelle tunnistekoodilla
7. RP vastaanottaa tunnisteet OP:lta ja validoi tunnisteet

OP:n myöntämät käyttäjätunnisteet koostuvat kolmesta komponentista: access tokenista, ID tokenista ja refresh tokenista.

4.2.2 Access Token

OAuth 2.0-standardin tunniste. Access tokenin rakennetta ei ole standardissa määritelty, mikä onkin toiminut kannustimena OpenID Connectin kehittämiselle.

4.2.3 ID Token

OpenID Connectin pääasiallinen laajennos OAuth 2.0-standardiin, joka mahdollistaa käyttäjien tunnistamisen ja auktorisoinnin. Tunnisteen esitysmuoto on allekirjoitettu JWT. (OpenID Connect 1.0 2014, 2.)

ID Tokenin tulee sisältää vähimmäisvaatimuksena viisi avain-arvo-paria

1. iss – Myöntäjä, HTTPS-muotoinen URL tunnisteen myöntäjän palveluun.
2. sub – Käyttäjäkoodi, staattinen koodi jonka myöntäjä määrittelee loppukäyttäjälle.
3. aud – Yleisö, ID Tokenia käyttävien tahojen tunnisteet pakollisena arvona “client_id”

4. exp – Vahenemishetki 1. aika, jonka jälkeen ID Tokenia ei tule hyväksyä käypänä tunnisteena.
5. iat – Myöntämishetki, aika jolloin ID Token on myönnetty.

Tyypillisesti tunnisteiden elinikä on lyhyt: esimerkiksi viisi minuuttia.

4.2.4 Refresh Token

Virkistystunnisteiden elinikä on muita tunnisteita pidempi (esimerkiksi 30 minuuttia). Niiden avulla voidaan noutaa tunnistuspalvelimelta uusia lyhytikäisiä tunnisteita ilman, että käyttäjää tarvitsee ohjata uudelleen tunnistuspalvelun sivulle.

5 SOVELLUKSEN TOTEUTUS

5.1 Keycloak

Toteutuksessa päädyttiin käyttämään Keycloak-ohjelmistoa käyttäjien hallintaan. Eri tasoisten oikeuksien määrittäminen käyttäjille tehtiin ryhmämäärityksillä. Tunnistusprotokollan käyttämän tunnistekomponentin sisältöön määriteltiin liitettäväksi käyttäjän ryhmätieto.

Keycloak edellyttää ennen tuotantokäyttöä seuraavat toimenpiteet:

1. Järjestelmän asennus.
2. Asetusten teko.
3. Tietokantaohjelmiston JDBC-ajurien lisäys.
4. Järjestelmän käynnistys.
5. Pääkäyttäjätunnuksen luonti.

5.1.1 Asennus

Projektin sivuilta on ladattavissa paketti, jonka voi linux-ympäristössä purkaa esimerkiksi järjestelmän /opt-hakemiston alle. Paikka soveltuu hyvin sijoituskohteeksi ohjelmille, joita ei löydy tavanomaisesta järjestelmäasennuksesta valmiina. Tällä perusteella päädyin sijoittamaan ohjelman /opt/keycloak –hakemistoon.

5.1.2 Asetukset

Keycloakin asetukset on mahdollista tehdä tekstieditorilla suoraan asetustiedostoihin. Vaihtoehtoisesti asetukset voidaan tehdä järjestelmän mukana tulevalla komentokehotteella, jolla voidaan ottaa yhteys käynnissä olevaan instanssiin.

Keycloakin mukana toimitetaan Java-pohjainen relaatiotietokanta “H2”. Tätä tietokantaohjelmistoa ei kuitenkaan suositella käytettäväksi tuotantoympäristössä.
Sen pääsääntöisenä tarkoituksena on tarjota nopea käyttövalmius.

Asiakkaan relaatiotietokannaksi on valikoitunut MariaDB. Tätä varten Keycloakkiin tulee asentaa JDBC-ajuri, joka on ladattavissa MariaDB-projektin kotisivuilta.

Jotta komentokehotteen kautta voidaan konfiguroida asetuksia, tulee ensimmäiseksi käynnistää palvelu. Tämä tapahtuu suorittamalla /opt/keycloak/bin –hakemistosta löytyvän shell-skriptin.

Instannsin ollessa käynnissä voidaan suorittaa samasta hakemistosta löytyvä komentokehoteohjelma “jboss-cli.sh”. Komentokehotteessa tulee antaa ensimmäiseksi connect–komento, joka avaa yhteyden Keycloak-palveluun. Yhteyden ollessa muodostettuna, annetaan komennot:

```
module add --name=org.mariadb --resources=/tmp/mariadb-java-client-1.5.5.jar --dependencies=javax.api,javax.transaction.api

/subsystem=datasources/jdbc-driver=mariadb:add(driver-name="mariadb",driver-module-name="org.mariadb",driver-class-name=org.mariadb.jdbc.Driver)

data-source add --jndi-name=java:/jboss/datasources/KeycloakDS --name=KeycloakDS --connection-url=jdbc:mariadb://localhost/keycloak --driver-name=mariadb --user-name=kayttajanimi --password=salasana
```

Ensimmäinen komento lisää MariaDB JDBC –ajurin. Toinen komento määrittelee mariadb-ajuriprofiilin. Viimenen komento luo KeycloakDS –nimisen tietopankin, joka käyttää samalta koneelta löytyvää MariaDB tietokantaohjelmistoa, “keycloak” –nimistä tietokantaa ja aiemmin luotua “mariadb” –ajuria. Lopuksi komennossa määritellään tietokannan käyttäjätunnus ja salasana. Nämä ovat hyvä luoda palvelukohtaisesti hyvin rajoitetuilla oikeuksilla.

Seuraavaksi päädyin viimeistelemään asetukset tekstieditorilla. Asetuksista tulee poistaa referenssiasennuksen jälkeenjättämät KeycloakDS –tietopankkiasetukset, joissa tietokannaksi on määritelty H2.

Keycloakin ja selaimen välisen verkkoliikenteen arkaluontoisuuden ja tietoturvallisuuden vuoksi on hyvin suositeltavaa ottaa verkkoliikenteen salaus käyttöön. Asiakkaan järjestelmässä tämä käytännössä tarkoitti Keycloak-instanssin tarjoilemista ulkoverkkoon edustapalvelimen kautta, jossa on otettuna käyttöön SSL-salaus.

Ennen ulkopuolisen liikenteen sallimista instanssille asti, on syytä luoda tunnistatutumisjärjestelmään ensimmäinen pääkäyttäjätunnus. Tämä on vaatimuksena ennen kuin järjestelmää voi käyttää.

Käyttäjän luominen on mahdollista komentokehotteessa tai web-käyttöliittymässä. Toteutuksessa päädyttiin jälkimmäiseen tekniikkaan. Instanssin kuunnellessa yhteydenottoja palvelimen localhostissa, voidaan yhteys muodostaa esimerkiksi tunneloimalla liikenteen SSH-yhteyden kautta palvelimen localhostille. Tämä onnistuu Windows-ympäristössä esimerkiksi PuTTY-asiakasohjelmalla, tai Unix-ympäristössä shellissä ajettavalla ssh-asiakasohjelmalla:

```
ssh -L 8080:localhost:8080 username@remotehost_address
```

Käytettävältä tietokoneelta on tämän jälkeen mahdollista avata selaimella osoite ”http://localhost:8080/auth”, joka ohjautuu tunnelin kautta palvelimelle.

5.1.3 Käyttöönotto

Keycloakissa on mahdollista luoda useita ympäristöjä, jotka ovat toisistaan riippumattomia kokonaisuuksia. Ensimmäiseksi luotu pääkäyttäjä on sijoitettu “Master” –ympäristöön (Keycloakissa termi “Realm”), jonka tarkoituksena on hallita tunnistautumisjärjestelmän pääkäyttäjiä. Näillä on oikeus hallita kaikkia ympäristöjä.

Ensimmäisten toimenpiteiden joukossa on uuden ympäristön luonti, jonka piiriin voidaan rekisteröidä tunnistuspalvelun asiakasohjelmia (esim. web-sivusto).

Kullekin asiakasohjelmalle voidaan määritellä käyttäjärooleja, joita voidaan antaa rekisteröidyille loppukäyttäjille. Näillä voidaan helposti toteuttaa roolipohjainen oikeuksien hallinta asiakasohjelman loppukäyttäjille.

Luotuun ympäristöön voidaan nyt rekisteröidä asiakasohjelma (RP). Toteutuksessa päädyttiin konfiguroimaan asiakasohjelman tyypiksi “confidential”, joka edellyttää OP:lle lähetetyissä tunnistepyynnöissä salaisen asiakaskoodin sisällyttämistä, koska muuten OP:lta ei voida saada tunnisteita. Asiakaskoodin voi käydä katsomassa Keycloakin web-käyttöliittymässä asiakasohjelman sivulta löytyvän “Credentials” –välilehden alta, koska asiakasohjelmaa konfiguroitaessa tarvitaan kyseinen koodi.

Keycloak asiakasohjelman tukiessa käyttäjien tunnistautumista selaimella, on syytä asettaa “redirect_uri” (päätepiste RP:llä) mahdollisimman tarkasti. Käyttäjän tunnistauduttua OP:lle, ohjataan loppukäyttäjä takaisin RP:n sivuille kyseiseen osoitteeseen.

5.2 Autentikointimoduuli

Moduulin tulisi toteuttaa “Authorization Code Flow”in mukainen tunnistautuminen web-käyttöliittymän osalta ja ainoastaan käyttäjätunnisteiden todentaminen ja oikeuksien tarkistaminen rajapinnassa.

Autentikointimoduulin runkona päätettiin käyttää Python Package Indexistä löytyvää oic-moduulia, joka toteuttaa OpenID Connect spesifikaation mukaisen toiminnan siltä osin kuin on projektin kannalta tarpeellista.

Autentikointimoduuli on rakennettu yhdeksi luokaksi, josta luodaan ilmentymä ohjelmassa. Moduuli tarvitsee joitain asetuksia toimiakseen:

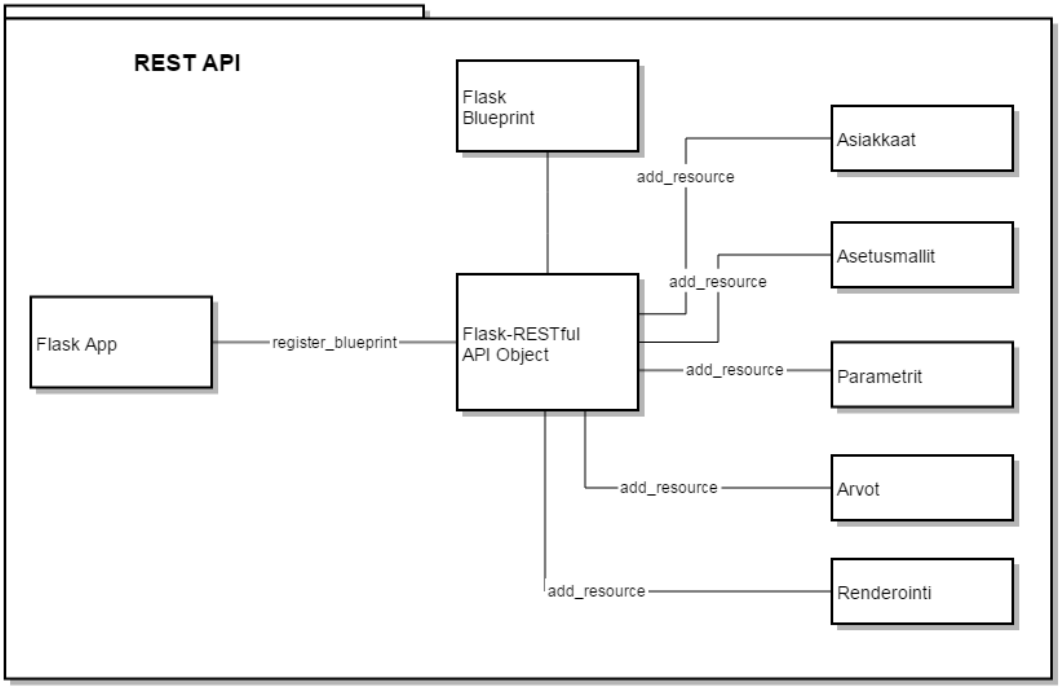
1. OP:lle rekisteröidyn asiakasohjelman uniikin id:n (client_id).
2. Asiakasohjelman koodin (client_secret).
3. Tunnistuspalvelimen myöntäjätunnisteen, joka on käytännössä linkki tunnistautumispalveluun. Keycloakin kohdalla linkki on muotoa “https://domain.tld/auth/realms/RealmName”.
4. Koska, käyttäjätunnisteet ovat epäsymmetrisellä salausperiaatteella allekirjoitettuja JSON-objekteja, tarvitaan salausmetodin julkinen avain joilla voidaan todentaa JSON-objektien koskemattomuus.

Salauksen julkisen avaimen nouto tapahtuu autentikointimoduulissa suorittamalla HTTP-kyselyn autentikointipalvelun päätepisteeseen. Vastauksena tulee JWK-objekti, josta muodostetaan avain.

Käyttöliittymää varten autentikointimoduuli tukee käyttäjätunnisteen noutamista HTTP-otsakkeen lisäksi (käytössä vain rajapinnassa) evästeistä, jotka ovat merkattu HttpOnly- sekä Secure-merkinnöillä. Käytännössä nämä tarkoittavat sitä, että evästeet lähetetään palvelimelle jokaisen HTTPS-pyynnön yhteydessä. Lisäksi HttpOnly-merkintä suojaa evästeeseen pääsyn selaimessa suoritettavilta skripteiltä.

5.3 Sovellusrajapinta

Rajapinnan toteuttamiseen käytin Flask-webkehystä, jonka toiminnallisuutta laajensin Flask-RESTful –paketilla.



Kuva 2. Rajapintasovelluksen rakenne

Rajapinnalla on neljä asetusprofiilia:

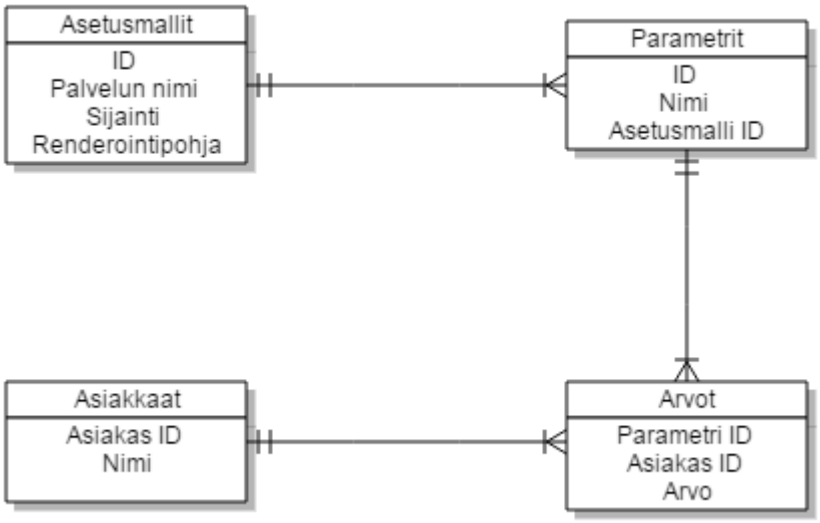
1. Tuotanto (vakaa, tuotantokäyttöä varten)
2. Epävakaa (epävakaa, uusien ominaisuuksien testausta varten)
3. Kehitys (kehitystyötä varten)
4. Testaus (yksikkötestausta varten)

Asetukset ovat normaaleja luokkamäärityksiä, jotka kaikki periytyvät yhteisestä asetusluokasta. Tämä asetusmalli helpottaa asetusten hallintaa ja mahdollistaa myös asetusprofiilikohtaisten funktioiden luomisen. Merkittävimpänä erotuksena profiileiden välillä on tietokannan yhteysmääritys. Kaikki profiilit käyttävät eri tietokantaa.

Käytettävä asetusprofiili luetaan palvelua käynnistettäessä ympäristömuuttujasta, jonka perusteella haluttavan profiilin luokasta luetaan asetukset luotuun Flask-ilmentymään.

5.3.1 Tietokantamallit

Mikropalvelujen asetuksia koskevat tietomallit ovat toteutettu seuraavasti.



Kuva 1. Tietokantamallit

Tietokantamallien kuvaus tehtiin SQLAlchemy objekteina ja kustakin mallista määriteltiin Marshmallow-malli, joiden avulla voidaan suorittaa tiedon serialisointi ja deserialisointi. Käytännössä tämä tarkoittaa käyttäjäsyötteen tulkintaa, sekä validointia ja tietokantaobjektien esittämistä rajapintakyselyiden vastauksina.

5.3.2 Resurssit

Rajapinnan resurssien jako tapahtui loogisten kokonaisuuksien mukaan, ja nämä kokonaisuudet ovat lähdekooditasolla hajautettu omiksi luokiksi Flask-RESTful – laajennosta hyödyntäen. Flask-RESTful antaa tavan kuvata resursseja luokkina, joiden metodeiksi määritellään HTTP-metodien nimet.

Luokkamäärittelyn jälkeen kukin luokka liitetään sille määriteltyyn päätepisteeseen.

5.3.3 Asiakasresurssi

Asiakasresurssilla hallitaan rakenteessa ylimpänä olevaa mallia, jotta jokaiselle asetusmallille määritellylle parametrille voidaan antaa arvoja tarvitaan sille omistaja (asiakas).

5.3.4 Asetusmalliresurssi

Asetusmalli koostuu nimestä, asetustiedoston sijainnista koostettavan järjestelmän levyjärjestelmässä ja renderointipohjasta. Renderointipohja on Jinja2-merkkauskielellä toteutettu.

5.3.5 Parametriresurssi

Parametreihin määritellään avain-arvo –parien avaimet, joiden relaationa on asetusmalli.

5.3.6 Arvoresurssi

Arvot määrittävät avain-arvo –parien arvot, relaatioina asetusmalli, parametric sekä asiakas.

5.3.7 Renderointiresurssi

Renderointi-päätepisteellä on ainoastaan yksi tuettu HTTP-metodi, GET. Tätä päätepistettä käytetään muodostamaan käyttövalmiita asetustiedostoja mikropalveluille. Päätepiste vaatii polussaan kaksi numeerista tunnistetta, joiden pohjalta asetus muodostetaan. Esimerkkipolku on "/render/1/1", jossa asiakkaan (id 1) asetusmalli (id 1) renderoidaan asiakkaalle määritellyiden arvojen mukaisesti

5.3.8 Testaus

Rajapinnan testaus tapahtuu pytestiä hyödyntäen, jossa conftest.py –nimiseen tiedostoon on mahdollista määritellä kaikkia testejä koskevia määrityksiä. Projektin rajapinnan testausta varten määrittelin kaksi funktiota "app" ja "client". Näille funktioille annoin pytest.fixture –dekoraattorin, jolle on mahdollista antaa "scope" avaimella eri arvoja. Tässä tapauksessa "app" sai arvoksi "session" (funktio suoritetaan testausperiodin aikana vain kerran) ja "client" sai arvokseen "function" (funktio suoritetaan jokaisen testifunktion aluksi).

App-funktion tehtävänä on luoda väliaikainen tietokanta tarvittavine taulurakenteineen, ja palauttaa rajapinnan ilmentymä jota vasten kaikki HTTP-kyselyt ajetaan.

Client-funktion ainoana tehtävänä on palauttaa rajapintailmentymästä "test_client" –metodin palautusarvo. Client-funktion parametriksi on määritelty "app", joka varmistaa sen, että app-funktio on suoritettu ennen kuin ensimmäistäkään HTTP-pyyntöä lähdetään suorittamaan.

Rajapinnan resurssitestaukset ovat määritelty omaan tiedostoonsa. Yhtä funktiota voidaan pitää yhtenä testinä. Kukin testausfunktio saa argumenttina aikaisemmin määritellyn client-funktion, jolla suoritetaan HTTP-kyselyt. Testauksessa käydään läpi kaikkien resurssien luomista, muokkaamista, noutamista ja poistamista. Vastauksista tarkistetaan arvojen oikeellisuus sekä HTTP-tilakoodi.

Testausten päätyttyä luotu väliaikainen tietokanta poistetaan.

5.4 Sovellusrajapinnan käyttöliittymä

Työssä päädyttiin luomaan selainpohjainen käyttöliittymä tietojen hallitsemiseen. Tietojen muodostuessa pääosin eri palvelujen asetuksista, päädyttiin luomaan näiden palvelujen ympäristössä ajettava komentokehote-ohjelma. Näin saatiin yksinkertaistettua asetustiedostojen loppusijoittaminen.

5.4.1 Web-käyttöliittymä

Web-käyttöliittymän toteuttamiseen käytin Flask-webkehystä. Palvelun arkkitehtuuri on periaatteiltaan MVC-tyyppinen. Resurssien mallit ovat kuvattu html-lomake luokkina, joiden perusteella käsittelijä luo näkymiä.

Käyttöliittymään tunnistautuminen tapahtuu Keycloak-tunnistuspalvelun kautta, josta saadut tunnisteet tallennetaan selaimen evästeisiin. Web-pavelun näkymien tiedot noudetaan näitä käyttäjätunnisteita käyttäen rajapinnasta yksinkertaisella rajapinta-asiakasohjelmalla.

5.4.2 Komentokehote –asiakasohjelma

Järjestelmän tämän hetkinen merkittävin rooli on tarjota asetustiedostoja mikropalveluille, jotta näiden koostaminen olisi helppoa. Toteutuksessa luotiin komentokehoteohjelma, joka käyttää REST-rajapintaa. Asiakasohjelma on terminaalissa ajettava tekstipohjainen python-ohjelma, käynnistämisen jälkeen ohjelma alkaa lukemaan käyttäjän antamia komentoja, näitä ovat:

1. auth (noutaa autentikointipalvelusta käyttäjätunnisteet)
2. ls (ottaa argumenttina asiakkaat tai asetusmallit, listaa halutut resurssit)
3. strap (ottaa argumenttina numeerisen asiakas id:n, luo asetustiedostot)

Asiakasohjelma pitää sisällään omia jinja2-merkkauskielisiä malleja, joiden pohjalta luodaan lisäksi erilaisia tarvittavia skriptejä sekä tiedostoja järjestelmän koostamiseen.

5.4.3 Testaus

Rajapintaa käyttävän web-käyttöliittymän testaus on monivaiheisempaa kuin pelkän rajapinnan testaus. Projektissa testauskehykseksi valikoitui Selenium pytestin rinnalle, joka tarjoaa rajapinnan sitä tukevien selainten hallintaan. Toteutuksessa selaimeksi valikoitui avoimen lähdekoodin selain Chromium joka toimii pohjana Google Chrome selaimelle.

Käyttöliittymätestausta varten käydään seuraavat vaiheet läpi ennen testien ajamista:

1. Luodaan rajapinnasta väliaikainen instanssi aliohjelmana.
2. Luodaan käyttöliittymästä väliaikainen instanssi säikeeseen.
3. Luodaan virtuaalinen näyttö (ruutupuskuri).
4. Luodaan ilmentymä selaimesta joka käyttää virtuaalinäyttöä.

Selainilmentymää voidaan nyt ohjata testausfunktioissa noutamaan sivuja. Lisäksi sivuille voidaan syöttää näppäinpainalluksia sekä klikkauksia (elementtien valinta CSS-valitsimilla tai XPathilla).

Testeissä käydään läpi läpi kaikkien resurssien lisäys, poisto, muokkaus ja nouto. Koska kaikki testit suoritetaan selaimella, olisi myös mahdollista tallentaa kuvankaappauksia testauksen eri vaiheista. Projektissa ei kuitenkaan sen hyvin yksinkertaisen rakenteen vuoksi koettu tätä tarpeelliseksi.

Testien päätyttyä käynnistetyt instanssit sammutetaan, ja luotu väliaikainen tietokanta tuhotaan.

6 YHTEENVETO

Projektin tavoitteena oli yksinkertaistaa ja helpottaa mikropalvelujen hallintaa, jotta rajapinnan kautta olisi helppo koostaa palveluista suurempi kokonaisuus. Mielestäni tavoitteet täyttyivät ja projektista lisäksi järjestelmäarkkitehtuuri mahdollistaa uusien ominaisuuksien toteuttamisen helposti. Yhtenä tulevaisuuden suunnitelmana onkin kytkeä rajapintaan mikropalveluiden käynnistys/sammutus sekä päivittäminen. Tätä ei vielä lähdetty toteuttamaan, koska siirtymävaihe uuteen järjestelmäarkkitehtuuriin on vielä kesken.

Autentikointipalvelu Keycloak on ollut hyvin positiivinen kokemus sekä itselle että asiakkaalle. Yksi isoimmista ongelmista on ollut käyttäjien haastava hallinta, jonka Keycloak ratkaisi mainiosti. Todennäköisesti tulevaisuudessa autentikointipalvelusta tullaan toteuttamaan klusteroitu kokoonpano, jotta voidaan tarjota luotettavampi saatavuus yhtiön kiinalaisille työntekijöille.

Kokonaisuudesssa opinnäytetyö on ollu mielenkiintoinen ja opettavainen kokemus. Projekti osoitti toimintapainotteisen toimintatutkimuksen olevan hyvä menetelmä ohjelmistoprojekteissa. Kehitysideaoita on syntynyt niin työn aikana kuin sen jälkeenkin. Kehkeytyneistä ideoista on hyöytyä myös tulevaisuuden projekteissa.

LÄHTEET

McConnell, S. 2004. Code Complete. Redmond: Microsoft Press.

Keycloak projektin www-sivut. 2016. Viitattu 9.12.2016. http://www.keycloak.org/

Lutz, M. 2014. Python Pocket Reference, Fifth Edition. Sebastopol: O'Reilly Media, Inc.

OpenID Connect 1.0. 2014. Viitattu 9.12.2016. https://openid.net/specs/openid-connect-core-1_0.html