Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

KALLE KIVILUOMA

Ohjelmakoodin käännös- ja käyttöönottotyökalun suunnittelu ja toteutus

TIETOJENKÄSITTELYN TUTKINTO-OHJELMA
2023

TIIVISTELMÄ

Kiviluoma, Kalle: Ohjelmakoodin käännös- ja käyttöönottotyökalun suunnittelu ja toteutus
Opinnäytetyö, AMK
Tietojenkäsittelyn tutkinto-ohjelma
04 2023
Sivumäärä: 33

Opinnäytetyössä suunniteltiin ja toteutettiin ohjelmiston kehitysprosessin automatisointiin ja hallintaan tarkoitettu työkalu, jota hyödynnetään toimeksiantajan tuotteen kehitystyössä. Työssä käytiin läpi yleistä tietoa kohteena olevan ohjelmiston rakenteesta ja siinä käytettävistä teknologioista. Työn tavoitteena oli rakentaa suunnitelman pohjalta edellä mainittu työkalu, jonka työn toimeksiantaja oli tilannut. Työn tarkoituksena oli parantaa kehittäjän tehokkuutta ja tuottavuutta, sekä saavuttaa pitkäaikainen ajansäästöllinen hyöty ohjelmiston parissa työskenteleville kehittäjille, sekä työnantajalle.

Työkalu suunniteltiin laatimalla vaatimusmäärittely sovellukselle, pohtimalla mahdollisia reunaehtoja sen toiminnalle, sekä visualisoimalla käyttöliittymän alustava rakenne. Suunnittelussa otettiin huomioon ohjelmiston arkkitehtuuriin liittyvien riippuvuustietojen selvittäminen ja tämän automatisointi. Työhön liittyi toteutuksen aikataulutus ja resurssointi työnantajalle, sillä työkalu tehtiin työajalla.

Työkalun päätoiminnallisuus oli automatisoida aiemmin kehittäjän manuaalisesti tekemät toimenpiteet ohjelmakehityksen aikana, jotka vievät usein paljon aikaa itse työltä, eivätkä konkreettisesti liity itse kehitystyöhön. Työkalu automatisoi ohjelman rakentamisen ja tästä syntyvien komponenttien siirron kohdeohjelmiston vaatimiin sijainteihin, jotta kehittäjällä on mahdollisuus päästä näkemään tekemänsä muutokset mahdollisimman nopeasti ja näin minimoimalla keskeytykset työssä.

Avainsanat: automaatio, ohjelmistokehitys, ohjelmistosuunnittelu, vaatimusmäärittely

Abstract

Kiviluoma, Kalle: Design and implementation of program code compilation and deployment tool
Bachelor's thesis
Business Information Systems
04 2023
Number of pages: 33

In the thesis, a tool for automating and managing the software development process was designed and implemented, which is utilized in the development work of the client's product. The work also covered general information about the structure of the targeted software and the technologies used in it. The goal of the work was to build the tool based on the design, which had been commissioned by the client of the thesis. The purpose of the thesis was to improve the efficiency and productivity of the developer, as well as to achieve a long-term time-saving benefit for the developers working with the software, as well as for the employer.

The tool was designed by drawing up a requirement specification for the application, considering possible boundary conditions for its operation, and visualizing the preliminary structure of the user interface. The design considered the need to resolve dependency data related to the architecture of the software and its automation. The work involved scheduling and resourcing the implementation of the tool for the employer, as the tool was made during working hours.

The main functionality of the tool was to automate the actions previously performed manually by the developer during program development, which often takes a lot of time from the work itself and is not concretely related to the development process itself. The tool automates the building of the program and the transfer of resulting artifacts to the locations required by the target software, so that the developer can see the changes they have made as quickly as possible and thus minimize interruptions in work.

Keywords: automation, software development, software design, requirement specification

SISÄLLYS

SYMBOLI- JA LYHENNELUETTELO

CLI	Command Line Interface. Rajapinta, jonka avulla käyttäjä voi olla vuorovaikutuksessa sovelluksen kanssa.
EAR	Enterprise Application Archive. Sovellusarkisto, joka tyypillisesti sisältää WAR ja EJB(JAR)-arkistoja.
JAR	Java Archive. Sovellusarkisto, joka sisältää sovelluksen luokat, resurssit ja metatiedot.
JVM	Java Virtual Machine on abstrakti tietokone, jonka päätehtäviin kuuluu tulkata luokkatiedostoja (.class) koneluettavaan muotoon.
Maven	Maven on työkalu, joka perustuu ”project object model” (POM) konseptiin, mahdollistaen projektiin liittyvien tapahtumien suorittamisen ja hallinnan.
WAR	Web Application Archive. Sovellusarkisto, joka sisältää staattisia web-tiedostoja, kuten JSP- tai CSS-tiedostoja JAR-arkistossa.
WildFly	Nykyisin RedHatin ylläpitämä avoimen lähdekoodin sovelluspalvelin. Tunnettu aiemmin nimellä JBoss AS/JBoss.

1 JOHDANTO

Ohjelmointityötä tehdessä tulee vastaan tilanteita, joissa toivoo asioiden toimivan sujuvammin ja nopeammin. Yksi näistä asioista on ohjelmakoodin kääntämisen ja käyttöönoton joustamattomuus. Esimerkiksi tässä opinnäytetyössä käsiteltävän Javan lähdekoodi on käännettävä luokkatiedostoksi ennen ohjelman suoritusta. JVM lataa, tarkistaa ja suorittaa tämän luokkatiedoston, eli käytännössä tulkitsee sen koneluettavaan muotoon (IBM Cloud Education, 2021). Käytännössä tämä tarkoittaa sitä, että Java on sekä käännettävä että tulkittava kieli. Varsinaiset tulkittavat kielet eivät vaadi lainkaan koodin kääntämistä tavu- tai konekieleksi toimiakseen, vaan se tapahtuu ajonaikaisesti. Käännetyillä kielillä rakennetut ohjelmat ovat usein tehokkaampia kuin tulkittavat kielet, johtuen tulkittavien kielten tarvitsemasta kääntämisestä ohjelman ajon aikana. Tämä kuluttaa työaseman resursseja ohjelman käynnissä olon aikana. (Baeldung, 2023.)

Käännetyn kielen haitat ilmenevät kehitysvaiheessa – jokaisen ohjelmaan/ohjelmistoon tehdyn muutoksen jälkeen lähdekoodi on käännettävä, ja tästä syntyvät tiedostot mahdollisesti siirrettävä ohjelman vaatimaan paikkaan projektin rakenteen mukaan. Tämä on todella aikaa vievää ja tämän opinnäytetyön tavoitteena on suunnitella ratkaisu, joka minimoi tähän prosessiin kuluvan ajan automatisoimalla ennen manuaalisesti tapahtuneita asioita.

Tässä opinnäytetyössä käydään läpi perusasioita käännettävästä ohjelmakoodista, sen vaatimasta kääntämisestä ja käyttöönotosta, ja näiden aiheuttamasta ongelmakentästä. Lisäksi tarkastellaan käännettävän ohjelmakoodin eroavaisuuksia tulkittavaan ohjelmakoodiin. Sivutaan myös projektinhallintajärjestelmä Mavenia, sekä tutustutaan konttialustapalvelu Dockerin toiminnallisuuteen perustasolla. Tämän jälkeen käydään läpi toteutettavan työkalun konkreettinen toteutussuunnitelma.

2 YLEISTÄ

2.1 Ohjelmakoodi

2.1.1 Käännettävä ja tulkittava ohjelmointikieli

Käännettävät ohjelmointikoodikielet, kuten COBOL, PL/I, C/C++, tulee kääntää kääntäjällä joko kone-, tai tavukieleksi. Lopputuloksena on hyvin tehokasta koodia, jota on mahdollista suorittaa lukemattomia kertoja vaatimatta uudelleenkääntämistä. Tämä tarkoittaa sitä, että ohjelmakoodin kääntämiseen vaaditaan resursseja vain yhden kerran jokaisen tehdyn muutoksen jälkeen. Tämän jälkeen ohjelma on valmis suoritusta varten. (IBM, n.d.)

Tulkittavien ohjelmointikielien - kuten Python, JavaScript, PHP ja Ruby - lähdekoodeja ei etukäteen käännetä konekieleksi. Tämä tapahtuu ajonaikaisesti rivi kerrallaan. Tässä etuna on joustavuus ajonaikaisissa toiminnoissa, kuten vianetsinnässä ja lähdekoodiin tehtävissä muutoksissa. Haittapuolena kuitenkin ajonaikainen rasite, joka johtuu tulkkaajan vaatimista resursseista ja tästä johtuva hitaus. Tulkittavat ohjelmointikielet eivät siis sovellu kovin hyvin suuren kokoluokan projekteihin, jotka vaativat jo itsessään tietokoneelta paljon resursseja. (Limón, 2022.)

Opinnäytetyössä kohteena oleva kieli Java taas mielletään molempiin tyyppeihin, sekä käännettyyn että tulkittavaan kieleen. Tämä johtuu siitä, että kääntäjä kääntää lähdekoodin ensin luokkatiedostoiksi (.class). Nämä luokkatiedostot sisältävät tavukoodia, joka ei vielä ole koneluettavaa. Vasta JVM tulkitsee luokkatiedoston koneluettavaan muotoon. Tässä hyötynä on se, että sama koodi on käytettävissä millä tahansa alustalla, jolla on tuki JVM:lle. Tämä mahdollistaa ohjelman suorittamisen riippumatta ympäristöstä ja sen konfiguraatioista. (JavaTpoint, n.d.) Tässä työssä keskitytään Java-kieleen ja sen ominaisuuksiin ison finanssi- ja vakuutusalan ohjelmiston kehityksessä.

2.1.2 Java ohjelmointikielenä

Java on historiansa aikana vakiinnuttanut asemansa johtavana ja luotettavana kielenä varsinkin isoa määrää dataa käsittelevissä sovelluksissa ja ohjelmistoissa. Sen etuina esimerkiksi uusia ohjelmistohankkeita ajatellessa ovat muun muassa vahva tyypitys, käyttöjärjestelmäriippumattomuus, monikäyttöisyys, skaalautuvuus ja luotettavuus. (IBM Cloud Education, n.d).

Java on periytynyt C ja C++ kielistä, jotka ovat yhdet tärkeimmistä koskaan kehitellyistä ohjelmointikielistä. Tämän vuoksi niillä on paljon samankaltaisuuksia syntaksin ja olioparadigman lisäksi. Kaikkia näitä kieliä on suunniteltu, testattu ja paranneltu niitä käyttävien ammattilaisten toimesta. Tämä on mahdollistanut tarpeiden mukaisten ominaisuuksien lisäykset ja parannukset. (Schildt, 2019, luku 1, kohta ”Java's Lineage: C and C++”)

Javaa on kuitenkin jatkojalostettu merkittävästi liiketoimintahankkeita ajatellen Enterprise Editionin (Java EE) muodossa. Antonio Goncalves (2018, luku 1, kohta ”Understanding Java EE”) kertoo Java EE:n merkittävyydestä liiketoimintamalleja tukevana alustana. Tässä luvussa hän mainitsee muun muassa rajapinnat Java Transaction API (JTA), Java Message Service (JMS) sekä Java Persistence API (JPA). Edellä mainitut rajapinnat on suunniteltu yritysjärjestelmien tarpeet huomioiden. Nämä helpottavat, nopeuttavat ja suoraviivaistavat ohjelmiston kehitysprosessia huomattavasti. Työssä kohteena oleva ohjelmisto perustuu Java EE:iin.

2.2 Maven

Opinnäytetyössä kohteena olevan ohjelmiston hallintaan käytetään projektinhallintatyökalua nimeltä Maven. Mavenin tarkoituksena on mahdollistaa käyttäjilleen Java-kehitysympäristön tilan hahmottaminen mahdollisimman lyhyessä ajassa. Keinot tämän tarkoituksen toteuttamiseksi ovat projektin rakennusprosessin helpottaminen, rakennusjärjestelmän yhtenäistäminen, projektitietojen laadukkuus, sekä kannustaminen parempiin kehityskäytäntöihin.

(Apache Maven, 2016.) Mavenin toiminta pohjautuu moduulien juurihakemistoissa oleviin pom.xml-tiedostoihin (kuva 1), jotka sisältävät tietoa moduulin riippuvuuksista, lähdehakemistosta, lisäosista ja muista konfiguraatioista. (Turing, n.d.).

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>example-project</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>example-project</name>
  <url>http://example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies>
    <!-- Example dependency -->
    <dependency>
      <groupId>com.example.dependency</groupId>
      <artifactId>dependency-module</artifactId>
      <version>1.0.0</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>3.1.0</version>
      </plugin>
    </plugins>
  </build>
</project>
```

Kuva 1. pom.xml-tiedoston rakenne

Tärkeässä osassa Mavenin toimintaa ovat ohjelmakomponentit (engl. artifact, myöhemmin ”komponentti”). Komponentti on Mavenin oman terminologian mukaan Maven-projektin rakennusprosessin tulos. Tämä tulos voi olla JAR, WAR, tai mikä muu tahansa suoritettava tiedosto. (Baeldung, 2022). Tässä työssä käytetään termiä ”komponentti” Mavenin tuottamista sovellusarkistoista ja termiä ”pom/pom.xml” komponenttien määrittelytiedostoista.

Kohdeohjelmistossa jokainen Mavenilla tapahtuva projektin rakennusprosessi tuottaa komponentin. Toisin sanoen jokainen kehittäjän tekemä Java-lähdekoodinmuutos päätyy lopulta komponentiksi. Komponentti voi olla:

- Käyttöönottokomponentti, joka on suoraan yhteensopiva WildFly-sovelluspalvelimen käsiteltäväksi
- Komponentti, joka on sisällytettynä yhteen tai useampaan muuhun komponenttiin, mukaan lukien käyttöönottokomponentteihin

Työssä käyttöönottokomponentteja ja muita komponentteja käsitellään eri tavoilla ja näistä kerrotaan lisää suunnitelma- ja toteutusvaiheessa.

Maven eroaa toisesta yleisesti käytössä olevasta projektinhallintatyökalu ANT:sta (Another Neat Tool) tehokkuudellaan. ANT:n käyttötarkoituksena on projektin rakentamisen ja käyttöönoton hallinta. Maven mahdollistaa näiden ominaisuuksien lisäksi muun muassa komponenttien, riippuvuustietojen, sekä muiden tiedostojen hallinnan ja automaattisen lataamisen määritellyistä tietovarastosta, joka tekee siitä paljon kyvykkäämmän työkalun. (Turing, n.d.) Maven on keskeisessä roolissa osana kohdeohjelmiston kehitysprosessia, sillä jokainen ohjelmistossa oleva moduuli käännetään ja paketoidaan komponenteiksi Mavenin avulla.

Riippuvuustiedoilla tarkoitetaan ulkoista moduulia tai kirjastoa, josta projekti tai moduuli on riippuvainen, jotta se voisi toimia halutulla tavalla. Maven projekteissa riippuvuudet on määritelty pom.xml-tiedostossa. Maven tarjoaa myös hallinnan myös transitiivisiin riippuvuuksiin, joka tarkoittaa riippuvuuden omia riippuvuuksia. (The Apache Software Foundation, 2022).

2.3 Docker

Docker on yleisesti käytössä oleva avoimen lähdekoodin konttipalvelualusta, jota käytetään sovellusten käyttöönoton ja toimituksien suoraviivaistamiseen. Dockerin toiminta perustuu sovellusten siirrettävyyteen ja omavaraisuuteen, joka käytännössä tarkoittaa sitä, että kontitettua sovellusta voidaan käyttää

missä tahansa laitteessa tai pilvessä, riippumatta käyttäjän järjestelmästä tai konfiguraatiosta. (Microsoft Learn, 2022.)

Dockerin avaintermejä ovat image ja kontti. Image on tiedosto, joka sisältää kaikki sovelluksen ajamiseen tarvittavat tiedostot. Näihin voi kuulua - sovelluksen arkkitehtuurin mukaan - esimerkiksi käyttöjärjestelmä, sovelluspalvelin, asetukset, sekä tietenkin itse sovellus. Kontti on sovellusyksikkö, joka on rakennettu imagen pohjalta. (Wallenius, 2022.)

Kohdeohjelmiston kehityksen aikaisessa prosessissa ollaan kiinnostuneita käytännössä vain kontista, joka on instanssi itse sovelluksesta. Sovelluksen lähdekoodiin tehtyjen muutosten jälkeen sovelluksen komponentteja siirretään Docker-konttiin testausta varten. Kehittäjän tekemät testatut muutokset varastoidaan GitLab-versionhallintapalveluun, jonka jälkeen myös image päivittyy automaattisesti Jenkins-automaatiopalvelun toimesta, jonka jälkeen tämän imagen pohjalta käynnistetyssä kontissa on nähtävillä käyttäjän tekemät muutokset.

2.4 WildFly

WildFly on avoimen lähdekoodin Java sovelluspalvelin, joka toteuttaa Java EE:n määritykset ja joka on suunniteltu tekemään suurien sovellusten käyttöönoton yksinkertaiseksi ja tehokkaaksi. WildFly on tehokas, ja sopii tämän ominaisuutensa vuoksi hyvin suurien ja monimutkaisten sovelluksien palvelimeksi. WildFly on myös hyvin kevyt palvelin, joka helpottaa ohjelman ajoa halvemman hintatason laitteistollakin. (Telang, n.d.)

Kohdeohjelmistossa WildFly on keskeisessä osassa kehitysprosessia, sillä lähes kaikki lähdekoodiin tehdyt muutokset päätyvät palvelimelle komponenttien muodossa. WildFly käsittelee uudet ja muuttuneet komponentit, jonka jälkeen kehittäjä näkee tekemänsä muutokset sovelluksessa. Tämä käyttöönottoprosessi tapahtuu automaattisesti, eikä useimmiten vaadi erillistä palvelimen

uudelleenkäynnistystä. Uudelleenkäynnistys tulee aiheelliseksi ajoittain käyttöönottoprosessin epäonnistuttua. Tässä tapauksessa käyttäjän on saatava epäonnistumisesta huomautus, jotta hän voi tehdä tarvittavat korjaukset virheen poistamiseksi. Epäonnistuneet käyttöönotot voidaan tunnistaa WildFlyn käyttöönottohakemiston komponenttien niin kutsutuista marker-tiedostoista. Marker-tiedosto kertovat komponenttien käyttöönoton statuksen ja sijaitsee käyttöönottohakemistossa. Nämä tiedostot ovat komponenttikohtaisia ja jokaisella käyttöönottokomponentilla on oma saman niminen marker-tiedosto, jonka pääte ilmaisee kyseisen komponentin statuksen. Marker-tiedostopäätteitä on olemassa useita, mutta suunnitelmaa varten tarpeellisia marker-tiedostoja ovat ”.isdeploying”, ”.failed” sekä ”.deployed”. ”.failed” pääte kertoo kyseisen komponentin käyttöönoton epäonnistuneen, ”.isdeploying” ilmaisee käyttöönottoprosessin olevan käynnissä ja ”.deployed” on merkki siitä, että komponentti on onnistuneesti suoriutunut käyttöönottoprosessista (RedHat Inc, 2016). Marker-tiedostojen hyödyntämisestä kerrotaan lisää luvussa 4.4.

3 OHJELMISTON NYKYTILA

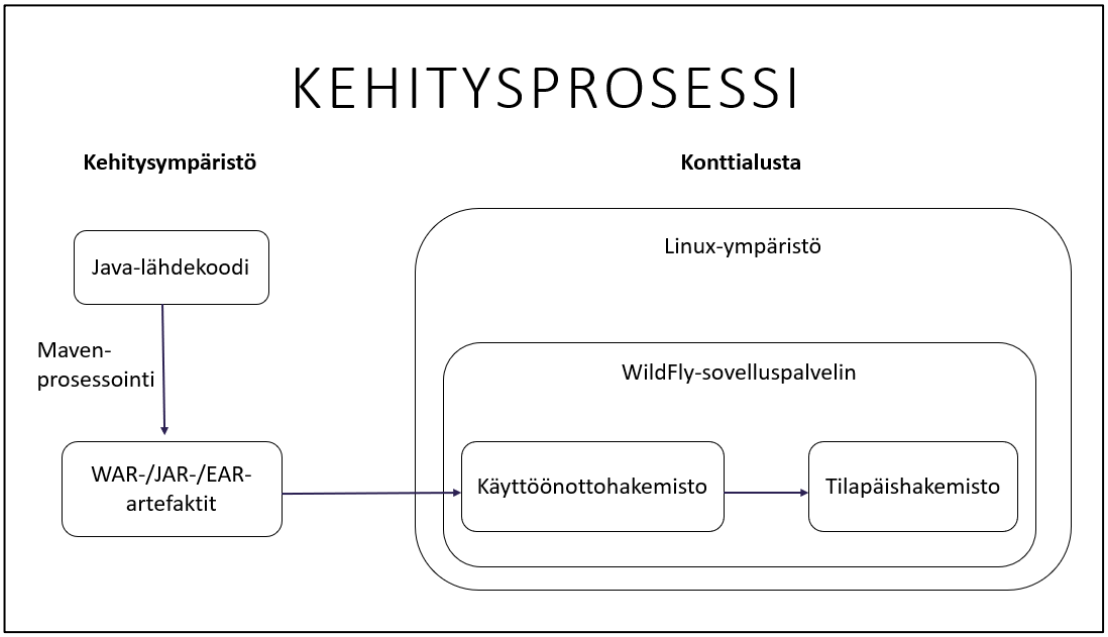
3.1 Ohjelmiston rakenne

Ohjelmistorakenteet eroavat tapauskohtaisesti, joten on vaikea määrittää yhtä ainoaa esimerkkiä, joka olisi pätevä kaikkiin mahdollisiin tilanteisiin. Tässä työssä käytetään rakennetta, joka vastaa työkalun käytön kohteena olevaa ohjelmistoa:

- Java EE -ohjelma, jonka hallintaan käytetään Mavenia
- WildFly-sovelluspalvelin
- Docker-kontti, joka sisältää sovelluspalvelimen ja Mavenin pakkaamat ja kääntämät käyttöönottotiedostot

WildFly-palvelimen sovelluspaketit, eli WAR- tai EAR-arkistot sijaitsevat Docker-kontissa sijainnissa $WILDFLY_HOME/standalone/deployments.

Tästä hakemistosta WildFly purkaa arkistoidut sovellustiedostot $WILDFLY_HOME/standalone/temp/vfs/deployments/ hakemistoon kuvion 1 osoittamalla tavalla. $WILDFLY_HOME tässä kontekstissa tarkoittaa WildFly-palvelimen oletuskotihakemistoa ja tämän tyylistä kirjoitusasua tullaan tässä työssä käyttämään jatkossa myös muiden vastaavien hakemistojen kohdalla.



Kuvio 1. Nykyinen kehitysprosessi

3.2 Ohjelmakehityksen nykyprosessi

Lähdekoodit on ensin käännettävä ja pakattava komponenteiksi, jotta Docker-kontin sovelluspalvelin osaa purkaa tämän tiedoston sisällön ohjelmiston vaatimiin paikkoihin. Tämä onnistuu ajamalla terminaalissa Maven komento mvn clean install työhakemiston ollessa moduulin juurihakemisto. Suorituksen jälkeen pakatut WAR-, EAR- tai JAR-komponentit löytyvät muutamia poikkeuksia lukuun ottamatta $WILDFLY_HOME/app/target hakemistosta. Poikkeukset ovat moduulikohtaisia.

Jos pakatun komponentin nimi vastaa jotain sovelluspalvelimen $WILDFLY_HOME/standalone/deployments hakemistosta löytyvistä komponenteista, korvataan vanha komponentti uudella SSH-yhteyden välityksellä. Tämän jälkeen WildFly käynnistää automaattisesti käyttöönottoprosessin, jotta

näemme kehittäjän tekemät muutokset. Tämä vie hetken aikaa ja saattaa vaatia palvelimen uudelleenkäynnistyksen.

Jos pakattu komponentti ei ole yksi sovelluspalvelimen $WILDFLY_HOME/standalone/deployments hakemistossa löytyvistä komponenteista, on kehittäjän käännettävä myös tämän paketin emomoduuli. Tämän jälkeen on myös suoritettava edellä mainitut toimenpiteet tämän moduulin kääntämisestä ja paketoinnista syntyvän komponentin kanssa. On mahdollista, että muutoksen kohteena oleva komponentti on riippuvuutena useassa eri moduulissa. Tässä tapauksessa kehittäjän on itse tiedettävä käsittelyä vaativat moduulit ja tämän mahdollinen räätälöintiprojektista löytyvä päämoduuli.

4 TOTEUTUSSUUNNITELMA

4.1 Vaatimukset ja rajoitteet

Asiakkuuksien erilaisuuksien takia ohjelmistot ja sen kehitysprosessit saattavat vaihdella. Tästä huolimatta voidaan määritellä työkalun vaatimukseksi ohjelman sujuva toiminta kaikissa mahdollisissa asiakkuuksissa, jotka vastaavat aiemmin kuvattua ohjelmiston rakennetta. Vaatimuksena on myös sovellusparametri- ja tietokantaskriptimuutosten automaattinen sisäänajo. Työkalun tulee toimia Linux- ja Windows-käyttöjärjestelmissä. Käyttäjälle tulee antaa mahdollisuus ilmoittaa mahdollisista ajonaikaisista virheistä.

Työkalun käyttäjältä vaaditaan Python-asennusta. Tämän lisäksi työkalu vaatii sekä tuote-, että räätälöintiprojektien sijaintia saman juurihakemiston alla.
WildFly ei tue muuttuneiden Java-luokkatiedostojen vientiä suoraan tilapäishakemistoon, joten VFS:ää käytettäessä ei voida vakuuttaa sen toimivuutta. Muuttuneiden Java-lähdekoodien tapauksessa on suositeltavampaa käyttää päämoduulien prosessointivaihtoehtoa.

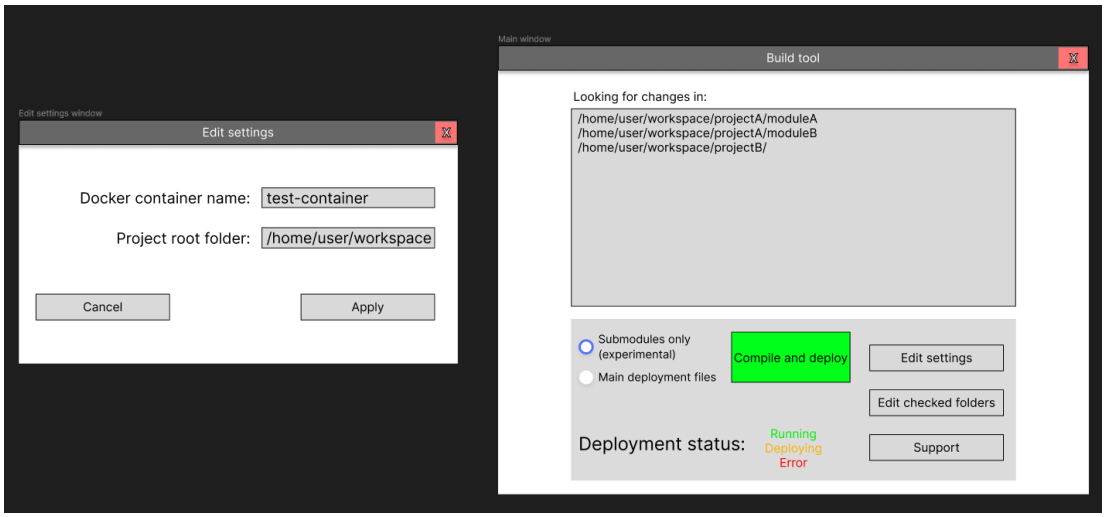
4.2 Käytettävät teknologiat ja kirjastot

Työkalun toteutuksessa tullaan käyttämään Python-ohjelmointikieltä ja ulkoista PyQt5-kirjastoa. Python-sovellus sisältää logiikan ja ohjelman varsinaisen toiminnan. PyQt5-kirjaston tarjoamat komponentit mahdollistavat ulkoasun luomisen. Pythonin kirjastoista tullaan hyödyntämään ainakin json-, subprocess- ja xml.etree.ElementTree-moduuleita. json-moduulin dump()-funktio serialisoi Python-objektin JSON-dokumentiksi ja load()-funktio vastaavasti deserialisoi JSON-dokumentin sisällön Python-objektiksi (Python, n.d.-a). Tämä moduuli vastaa käyttäjäasetusten ja muiden erillisten JSON-tiedostojen käsittelystä sovelluksen ajon aikana. subprocess-moduulin run()-funktio suorittaa vaadittavat terminaalikomennot ja ottaa tarvittaessa talteen näistä syntyvät tulosteet ja virheviestit (Python, n.d.-b). Kaikki työssä käytetyt Maven- ja Docker-komennot suoritetaan käyttäen tätä funktiota. xml.etree.ElementTree- moduulin tarkoituksena on erillisten XML-tiedostojen datan jäsentäminen ja luominen, jotta niitä olisi mahdollisimman yksinkertaista käsitellä Python-ohjelman suorituksen aikana (Python, n.d.-c). Tätä moduulia hyödynnetään pom.xml-tiedostojen käsittelyyn liittyvissä tarpeissa, kuten esimerkiksi komponenttien, ja emomoduulien nimien selvittämisessä.

Käyttäjäasetukset ja projektin riippuvuustiedot tullaan tallentamaan JSON-tiedostoiksi työkalun juurihakemistoon asennusohjelman aikana. Käyttäjäasetuksia on mahdollista muokata milloin vain ohjelman ajon aikana. Asennusohjelman aikana työkalu myös selvittää annettujen tietojen perusteella päämoduulien riippuvuustiedot, sekä Dockerin $WILDFLY_HOME/standalone/deployments -hakemiston komponenttien nimet. Nämä tiedot tallennetaan myös erillisiin JSON-tiedostoihin. Nämä tiedot ovat pääasiassa muuttumattomia, eikä työkalussa ole tarvetta mahdollistaa näiden tietojen muuttamista. Käyttäjä voi halutessaan poistaa käyttäjäasetukset sisältävän JSON-tiedoston, jolloin asennusohjelma selvittää nämä tiedot uudestaan.

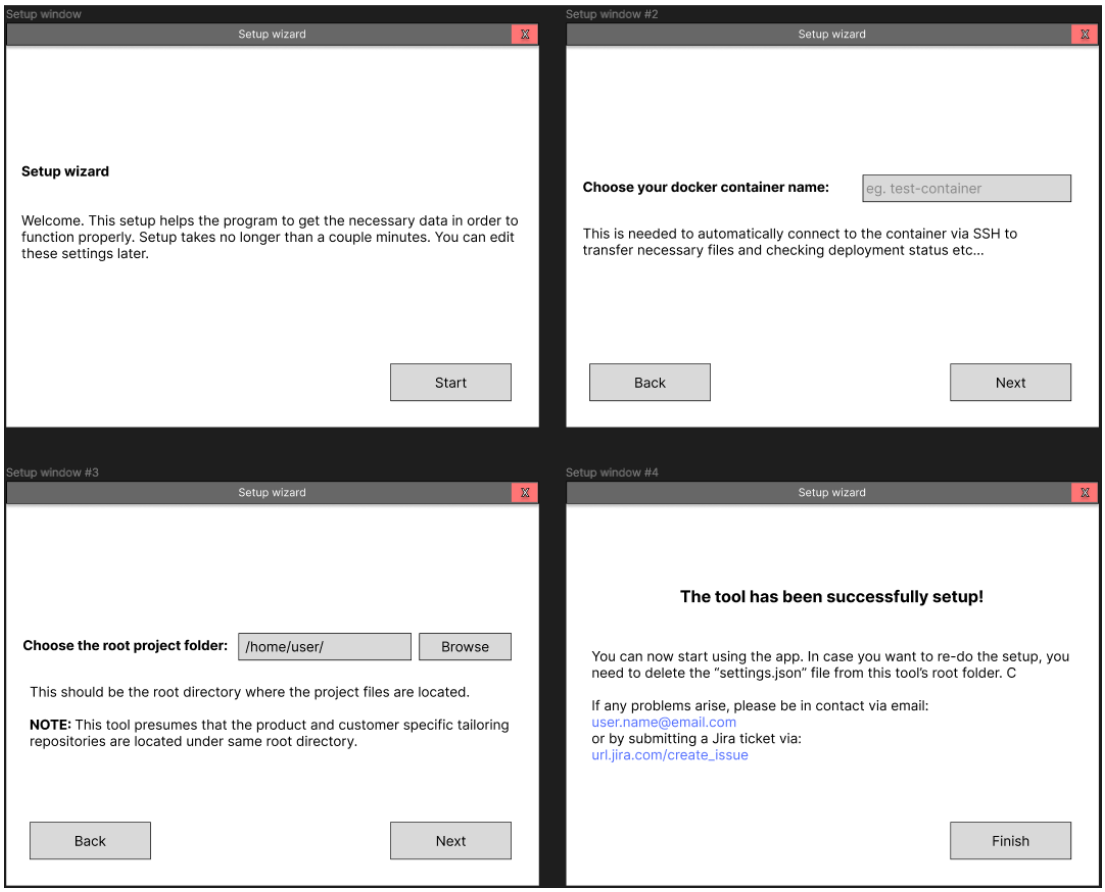
4.3 Käyttöliittymä

Ulkoasun suunnittelussa apuna on käytetty Figman tarjoamaa ilmaista verkossa toimivaa graafista suunnittelupalvelua, joka löytyy osoitteesta www.figma.com. Tämän avulla on saatu hahmotettua työkalun käyttöliittymän alustava rakenne kuvan 2 osoittamalla tavalla.



Kuva 2. Kuva hahmotellusta pääikkunasta ja asetuksien muokkaus näkymästä

Työkalu tulee olemaan ulkomuodoltaan hyvin pelkistetty ja sen on tarkoitus olla helppokäyttöinen ja suoraviivainen. Kun työkalu käynnistetään ensimmäisen kerran, tulee näkyviin kuvan 3 kaltainen asennusnäkymä, jonka tarkoituksena on pyytää käyttäjältä tarvittavat tiedot ohjelman toimivuuden takaamiseksi. Näitä tietoja ovat esimerkiksi Docker-kontin nimi ja projektin lähdekoodien juurihakemisto. Asennusohjelman suorittamisen jälkeen nämä tiedot tallennetaan työkalun juurihakemistoon. Asennusohjelman voi käynnistää uudelleen poistamalla kyseisen tiedoston ja valittuja asetuksia voi myös muuttaa käyttöliittymän ”Edit settings” -painikkeesta. Asennusohjelman aikana työkalu myös selvittää automaattisesti räätälöintiprojektin moduulien riippuvuudet, sekä WildFly-palvelimen käyttöönottokomponentit, jotka se löytää käyttäjän antamien tietojen perusteella.



Kuva 3. Kuva suunnitellusta asennusohjelmasta.

Käyttöliittymällä on painike ”Compile and deploy”, joka käynnistää työkalun varsinaisen toiminnallisuuden. Tässä on myös annettu käyttäjälle vaihtoehto suorittaa käännös ja käyttöönotto vain muutetuille alimoduuleille tai päämoduuleille. Alimoduuleita vietäessä on mahdollista, etteivät kaikki muutokset näy ennen ohjelman uudelleenkäynnistystä tai eivät näy lainkaan aiemmin mainittujen rajoitteiden takia. Tämän vuoksi tämän vaihtoehdon valitseminen on suositeltua vain ohjelman käyttöliittymään tehtävissä muutoksissa tai pienemmissä muutoksissa, jossa luokkien metodien rakennetta ei muuteta, eikä luokkia lisätä tai poisteta. Tästä rajoitteesta käyttäjälle annetaan selkeä visuaalinen tieto. Päämoduulien käännös ja käyttöönotto on suositeltua, jos tämänkaltaisia muutoksia on tehty.

4.4 Työkalun toiminta

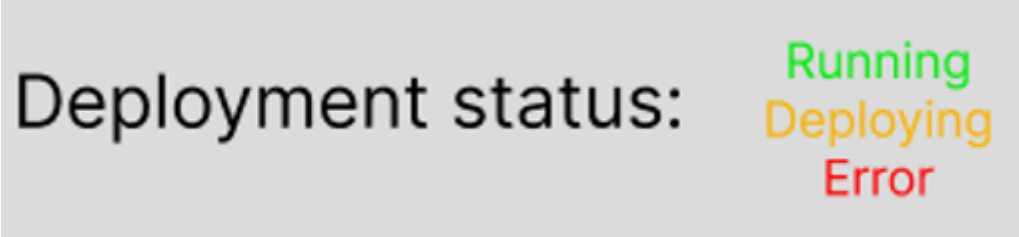
Työkalun varsinaisen toiminnallisuuden prosessi pyritään pitämään mahdollisimman yksinkertaisena. Kuten aiemmin mainittu, käyttäjällä on kaksi

vaihtoehtoa työkalun perustoimintaan liittyen: alimoduulien- tai päämoduulien kääntäminen ja käyttöönotto. Alimoduulien prosessointi on teoriatasolla hyvin suoraviivaista. Työkalu tutkii git-versionhallintaohjelman avulla valituista kansioista muuttuneita tiedostoja. Jos muuttuneita tiedostoja löytyy, työkalu tarkistaa, onko muuttuneen tiedoston kanssa samassa kansiossa pom.xml-nimistä tiedostoa. Jos ei ole, työkalu jatkaa etsintää ylemmistä kansoista niin kauan, kunnes pom.xml-tiedosto löytyy. Kun se on löytynyt, ajaa työkalu tässä hakemistossa mvn clean install –komennon, jonka lopputuloksena voidaan nähdä $MODULE/app/target -kansioon syntyvä komponentti. Työkalu ottaa myös talteen pom.xml-tiedostosta löytyvän komponentin tunnuksen, jota tarvitaan seuraavassa vaiheessa. Tämän jälkeen työkalu tutkii Docker-kontin sisältämän WildFlyn VFS-hakemiston kansioita ja vertaa niitä luomamme komponentin tiedostonimeen. Tämän jälkeen työkalu ylikirjaa VFS:ssä olevan kansion uudella komponentilla.

Päämoduulien vientiprosessi taas on hieman monimutkaisempi. Työkalun on tiedettävä päämoduulien paikalliset riippuvuudet prosessoidakseen tarvittavat moduulit. Kohdeohjelmistossa on niin kutsutut tuote- ja räätälöintiprojektit, joista räätälöintiprojekti sisältää käyttöönottokomponentit. Tällä projektilla taas on useita riippuvuuksia tuoteprojektin moduuleihin, joihin useimmiten muutokset tehdään. Räätälöintisovelluksen päämoduulien riippuvuudet saadaan selville ajamalla terminaalissa komento mvn dependency:tree. Komento on tarvetta ajaa vain kerran asennusohjelman yhteydessä. Syntyvästä tulosteesta pystytään päättelemään, mitkä räätälöintiprojektin päämoduulit on prosessoitava perustuen muutettujen tiedostojen komponenttien tunnuksiin. Tämän jälkeen työkalu siirtää päämoduulien kääntämisestä valmistuneen komponentin Docker- kontin $WILDFLY_HOME/standalone/deployments -hakemistoon, joka käynnistää käyttöönottoprosessin.

Käyttöönottoprosessin aikana työkalu osoittaa käyttöönoton statuksen – keltainen merkitsee sitä, että prosessi on kesken, vihreä sitä, että prosessi on valmis ja punainen sitä, että suorituksen aikana on tapahtunut virhe. Virhetilanteessa ohjelma antaa linkin virhelokiin. Käyttöönoton statuksen voi päätellä WildFlyn markkereista. Status on vihreä, kun jokainen käyttöönottohakemiston

markkeritiedosto sisältää tekstin ”.deployed”. Status on keltainen, kun yksi tai useampi markkeri sisältää tekstin ”.isdeploying” ja punainen, kun yksi tai useampi sisältää tekstin ”.failed” (kuva 4). Työkalun on siis iteroitava käyttöönottohakemuksen tiedostoja läpi niin kauan, kunnes status muuttuu vihreäksi tai punaiseksi.



Kuva 4. Käyttöönottoprosessin statuksen havainnollistaminen

Kaikkiin terminaalikomentoihin tullaan käyttämään Pythonin tarjoamaa subprocess moduulia, jonka moduulin run()-funktion avulla voidaan suorittaa terminaalikomentoja ja ottaa talteen näistä syntyvät tulosteet ja virheet. Terminaalikomentoja tarvitaan muun muassa komponenttien rakentamiseen Mavenin avulla, komponenttien siirtämiseen, sekä riippuvuustietojen selvittämiseen.

5 TOTEUTUS

5.1 Käyttöliittymä

Käyttöliittymän päänäkymän toteutukseen luotiin luokka nimeltä BuildHelperWindow, joka perii PyQt5 kirjaston QtWidgets-moduulin QtWidget-luokan. Tämä on toteutuksen pääikkuna, johon kaikki käyttöliittymän muut elementit myöhemmin sijoitetaan. QtWidgets-moduuli tarjoaa monia hyödyllisiä elementtejä ja seuraavaksi tutkimme näistä niitä, joita on työssä käytetty.

QLabel-elementti mahdollistaa tekstin tai kuvien esittämisen käyttöliittymällä (The Qt Company, n.d.-a). Työssä tätä elementtiä käytettiin käytännössä

kaikissa paikoissa, joissa tekstiä on tarve näyttää (kuvio 2, nuoli 2), poissulkien muiden elementtien sisäänrakennetut ominaisuudet, jotka mahdollistavat myös tekstin näyttämisen osana toiminnallisuutta.

QListWidget-elementin tarkoituksena on näyttää käyttöliittymällä lista esimerkiksi tekstiä tai kuvia (The Qt Company, n.d.-b). Työssä tätä on käytetty listaamaan projektihakemistot, joita työkalu käy läpi muutosten tarkistamiseksi. Käyttäjällä on mahdollisuus muuttaa listaa haluamakseen. Listan sisältöä käytetään myöhemmin prosessoinnin aikana (kuvio 2, nuoli 1).

QPushButton nimensä mukaisesti luo käyttöliittymälle klikattavan painikkeen, jonka perusteella voidaan suorittaa haluttuja toimintoja (The Qt Company, n.d.-c). Työssä näiden painikkeiden avulla on toteutettu kaikki toiminnot, jotka vaativat käyttäjän valintoja (kuvio 2, nuoli 3).

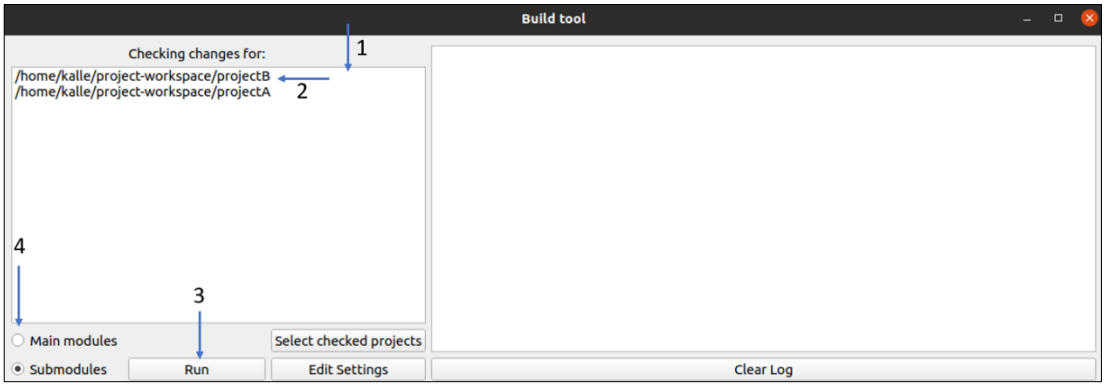
Kaikki käyttäjän valintoja vaativat paikat on toteutettu QRadioButton elementin avulla. Elementti sisältää joukon valintoja, jotka ovat valittavissa yksi kerrallaan. Valintoihin on mahdollista liittää tekstiä tai kuvia. Jokaisen valinnan kohdalla voidaan tutkia kyseisen valinnan tilaa isChecked()-funktion avulla, joka palauttaa True, jos se on valittuna. (The Qt Company, n.d.-c.) Tätä käytettiin työssä antamaan käyttäjälle mahdollisuus valita alimoduulien tai päämoduulien prosessoinnin väliltä, sekä käyttäjän tekemien valintojen kanssa (kuvio 2, nuoli 4).

QMovie on luokka, joka sijaitsee QtGui-moduulissa QtWidgets-moduulin sijaan. Tämän avulla käyttöliittymällä voidaan näyttää animoituja GIF tiedostoja tai muita animaatiota ilman ääniä. (The Qt Company, n.d.-d.) QMovie:ta on käytetty työkalussa näyttämään käyttöliittymällä GIF-animaatio, joka indikoi rakennusprosessin olevan kesken.

Työkalun päänäkymän responsiivisuus rakennusprosessin ollessa käynnissä saavutetaan käyttämällä QThread-luokkaa. Tämä ei varsinaisesti liity käyttöliittymän visuaaliseen ulkomuotoon. QThread-luokan avulla voidaan siirtää haluttu prosessi toiseen säikeeseen suorittamalla moveToThread(QThread)-

funktio (The Qt Company, n.d.-f). Tämä siirtää prosessin suorituksen irralliseksi käyttöliittymää suorittavasta säikeestä, jolloin käyttöliittymällä voidaan suorittaa muita toimia prosessoinnin ollessa kesken.

Elementtien asettelu toteutetaan QGridLayout-luokan toimesta, joka jakaa elementit riveihin ja kolumneihin (The Qt Company, n.d.-g). Pääasiassa työssä käytetyt elementit on aseteltu käyttäen tätä luokkaa.



Kuvio 2. Eri PyQt5 elementit käyttöliittymällä osoitettu numeroiduilla nuolilla

5.2 Toiminnallisuus

Toiminnallisuuden toteuttaminen on jaettu esiprosessointiin, prosessointiin, sekä jälkiprosessointiin. Tämä jaottelu helpottaa koodin luettavuutta ja pitää prosessin kulun helposti ymmärrettävänä. Seuraavissa kappaleissa kerrotaan lisää edellä mainituista vaiheista. Prosessoinnit tapahtuvat main.py-päätiedostosta irrallisessa build_functions.py-tiedostossa.

5.2.1 Esiprosessointi

Esiprosessointi alkaa jo asennusohjelman suorituksen aikana, jolloin käyttäjä antaa työkalulle tarvittavia tietoja. Käyttäjän on syötettävä projektin juurihakemiston polku, jonka avulla työkalu osaa tarvittaessa hakea sisältöä täällä sijaitsevien kansioiden tiedostoista. Toinen tieto, jonka käyttäjä syöttää, on

Docker-kontin nimi. Tämä on olennainen osa työkalun toimintaa, sillä tämä tarvitaan kaikkiin Docker CLI:n komentoihin. Kolmas tieto on WildFly sovelluspalvelimen nimi, jota tarvitaan käyttöönottoprosessissa. Lisäksi käyttäjä syöttää mahdollisen asiakkuusprojektin räätälöintiprojektin juurihakemiston, jonka jälkeen työkalu suorittaa tämän tiedon perusteella riippuvuustietojen selvityksen. Asennusohjelman lopussa työkalu etsii WildFly-palvelimen nimen perusteella kaikki pääohjelmakomponenttien nimet Docker-kontista. Kaikki nämä tiedot tallentuvat settings.json nimiseen tiedostoon, jota hyödynnetään projektin aikana useasti.

Esiprosessointiin luokitellaan myös ensimmäinen valmistava vaihe, kun käyttäjä painaa ”Run” nappia. Tämän vaiheen aikana build_functions.py-tiedoston luokkaan talletetaan seuraavat asiat settings.json-tiedostosta:

- Docker-kontin nimi
- WildFly-palvelimen nimi
- Räätälöintiprojektin juurihakemiston polku
- Työtilan juurihakemiston polku

Lisäksi esiprosessoinnin aikana näiden tietojen perusteella selvitetään ja talletetaan muuttujiin seuraavat asiat:

- VFS-tilapäishakemiston nimi
    - Tämä on tärkeää, sillä WildFly generoi uuden satunnaisen päätteen kansion nimen loppuun jokaisen uudelleenkäynnistyksen yhteydessä.
- VFS-tilapäishakemiston sisältö
- WildFly:n käyttöönottohakemiston polku

5.2.2 Prosessointi

Riippumatta käyttäjän valitsemasta prosessointityypistä työkalu käy ensimmäisenä läpi käyttäjän valitsemat kansiot ja tarkistaa ”git diff -w --name-only”-komennolla kaikki kyseisissä kansoissa tapahtuneet muutokset (kuva 5). Tämä komento palauttaa tiedon muuttuneista tiedostoista suhteessa

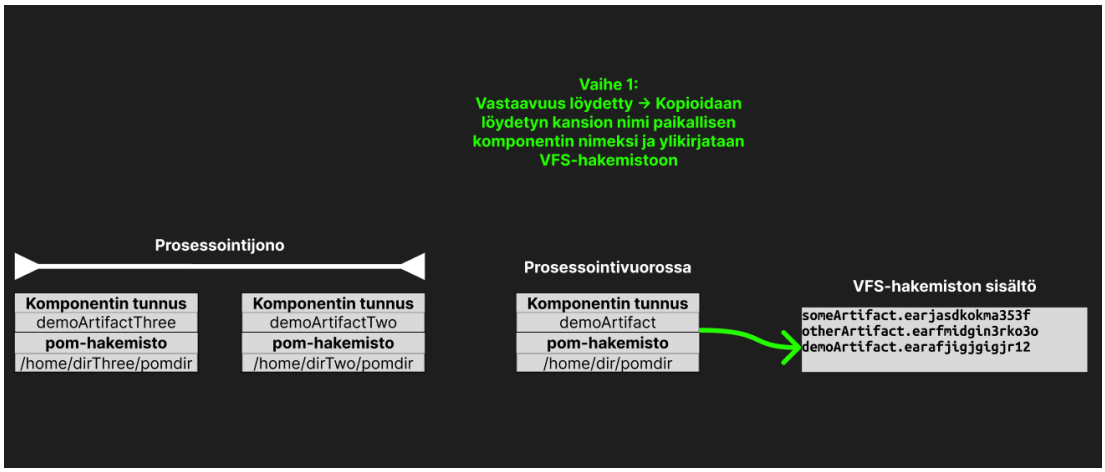
valmistelualueeseen. Työkalu ei siis ota huomioon jo varastoituja muutoksia. ”-w”-valinta ei ota huomioon muutoksia välilyönneissä ja ”--name-only”-valinta nimensä mukaisesti palauttaa vain tiedostojen nimet tarkempien muutostietojen sijaan (git-scm.com, n.d.). Tämän perusteella työkalu lähtee löydettyjen muutosten pohjalta etsimään kansio kerrallaan ylöspäin, kunnes se löytää pom.xml-tiedoston. Tämä tarkoittaa, että moduulin juurihakemisto on löytynyt. Työkalu ottaa talteen tämän kansion polun, sekä pom.xml:stä löytyvän yksilöllisen ”artifactId” tiedon talteen ja jatkaa prosessia näiden tietojen saamisen jälkeen seuraavaan vaiheeseen.

```
C:\Users\User\thesis [main +0 ~1 -0 !]> git diff -w --name-only
projectA/javaOne.java
C:\Users\User\thesis [main +0 ~1 -0 !]>
```

Kuva 5. Komentokehotteessa suoritettu komento muutosten selvittämiseksi

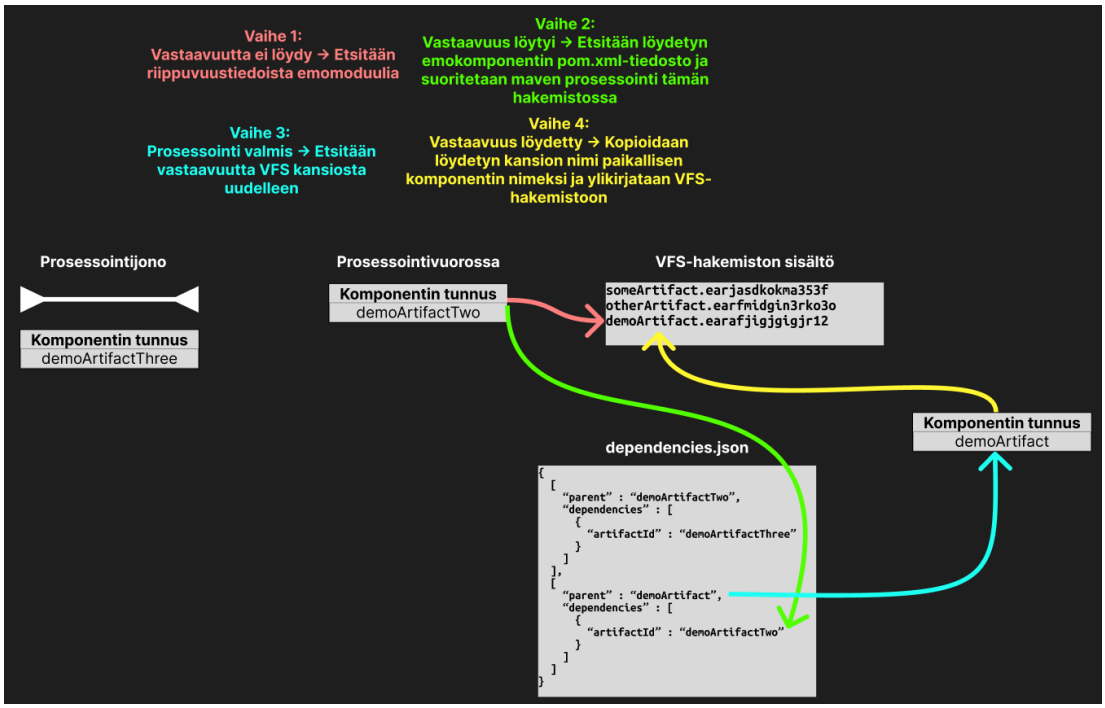
Seuraavassa vaiheessa työkalu ajaa käyttäjän valitseman Maven-komennon jokaiselle moduulille, jossa on tapahtunut muutoksia. Komento ajetaan vain kerran jokaista moduulia kohden, vaikka muutoksia olisi tehty saman moduulin alle useampia. Onnistuneiden Maven-rakennusprosessien tuloskomponenttien hakemisto, sekä komponentin tunnus talletetaan myöhempää käsittelyä varten muuttujaan. Epäonnistuneista prosessoinneista tallennetaan lokitiedosto, josta löytyy lisätietoa tapahtuneesta virheestä.

Käyttäjän valitsemattoman prosessointitavan perusteella seuraavat vaiheet eroavat toisistaan huomattavasti. Jos käyttäjä on valinnut prosessointitavaksi alimoduulien prosessoinnin, työkalu etsii Docker-kontin VFS-hakemistosta jokaisen prosessoidun komponentin tunnusta vastaavia kansioita. Jos vastaavuus löytyy, työkalu ylikirjaa löydetyn kansion sisällön prosessoidun artifaktin sisällöllä (kuvio 3).



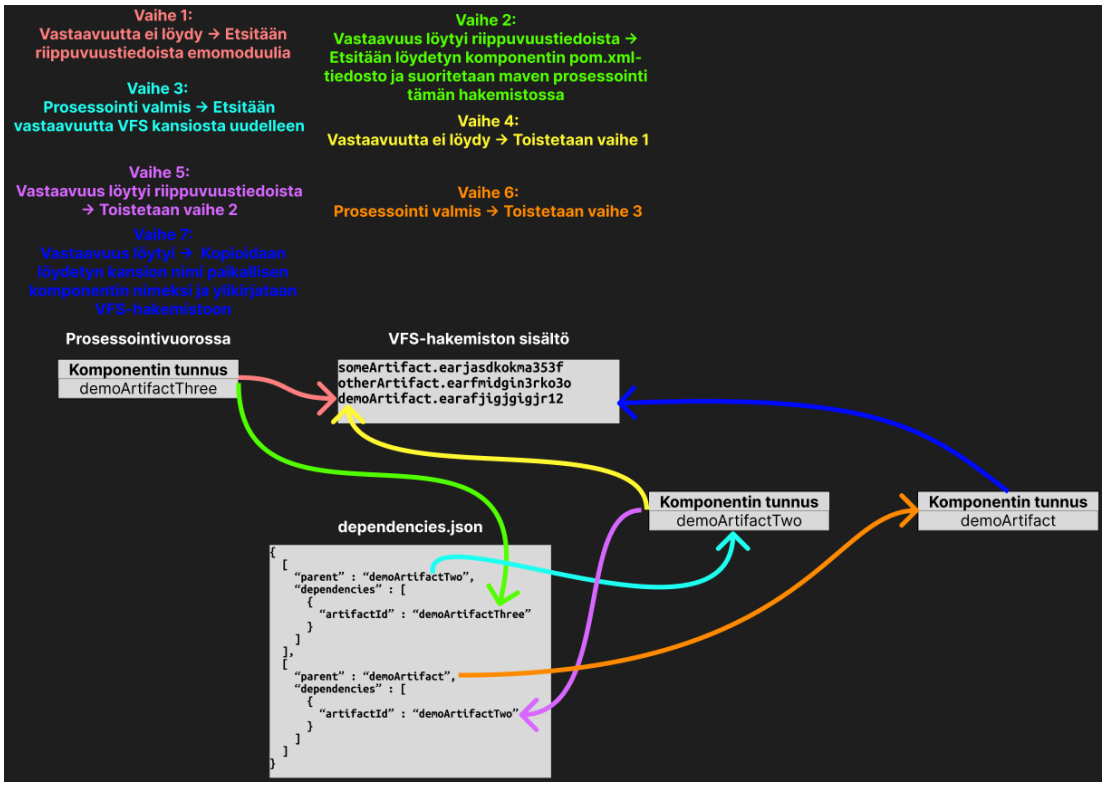
Kuvio 3. Alimoduulien prosessoinnin hahmotelma, jossa löydetty vastaavuus prosessoitavalle moduulille

Jos prosessoitua moduulia ei löydy VFS-hakemistosta, etsii työkalu tämän komponentin riippuvuustietojen perusteella tästä riippuvaa komponenttia ja yrittää prosessointia uudestaan (kuvio 4). Tämä tilanne on melko yleinen, jos muutoksia on tehty tuoteprojektin lähdekoodiin, sillä usein nämä komponentit on ylikirjattu räätälöintiprojektin komponenteilla. Tässä tapauksessa räätälöintiprojektin komponentti sisällyttää kyseisen tuoteprojektin komponentin. Jos komponentti on riippuvuutena useammassa kuin yhdessä moduulissa, käyttäjälle näytetään ikkuna, josta hän voi valita haluamansa komponentin prosessoinnin, jolloin prosessointia jatketaan valitun komponentin kanssa



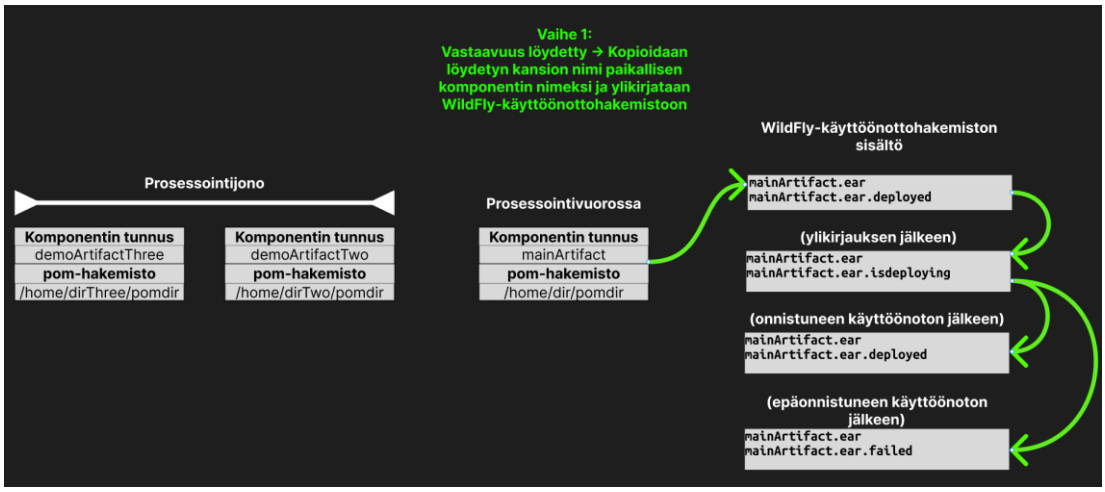
Kuvio 4. Alimoduulien prosessoinnin hahmotelma, jossa prosessoidulle moduulille ei löydy heti vastaavuutta VFS-hakemistosta

Eteen voi myös tulla tilanne, jossa vastaavuutta VFS-kansiosta ei löydy vielä tämänkään jälkeen. Tässä tapauksessa työkalu jatkaa riippuvuustietojen hakua niin kauan, kunnes vastaavuus löytyy (kuvio 5). Jos vastaavuutta ei loppujen lopuksi löydy lainkaan, prosessointi ohitetaan, ja tästä ilmoitetaan käyttäjälle.



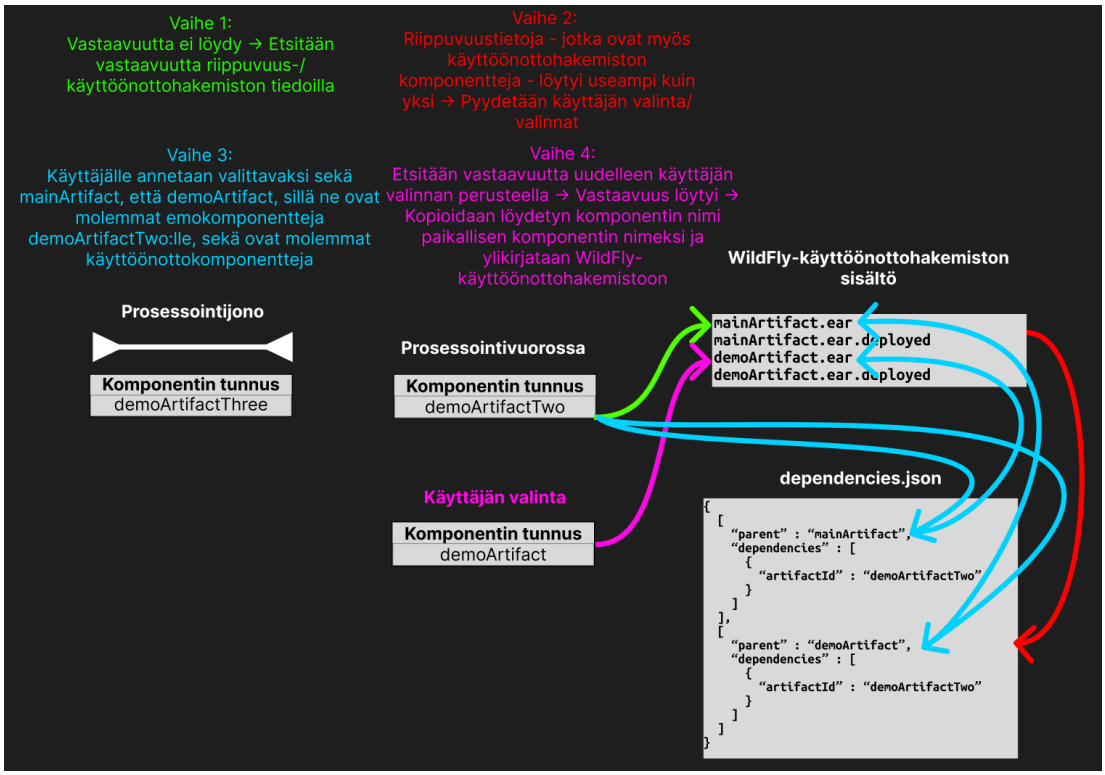
Kuvio 5. Alimoduulien prosessoinnin hahmotelma, jossa löydetylle emokomponentillekaan ei löydy vastaavuutta VFS hakemistosta

Jos käyttäjä on valinnut prosessointitavaksi päämoduulien prosessoinnin, työkalu tarkistaa, löytyykö prosessoidun komponentin nimeä vastaavaa tiedostoa WildFly:n käyttöönottohakemistosta. Jos vastaavuus löytyy, ylikirjataan löytynyt tiedosto prosessoidulla komponentilla (kuvio 6). Tämän jälkeen työkalu odottaa, kunnes käyttöönottohakemiston marker-tiedosto muuttuu ”.isdeploying”:sta ”.deployed”:ksi. Tämän jälkeen käyttäjälle ilmoitetaan prosessoinnin onnistumisesta. Jos tapahtuu niin, että marker-tiedosto muuttuu ”.failed”:ksi, käyttäjälle ilmoitetaan käyttöönoton epäonnistumisesta, jolloin hän voi tutkia lokitiedostojen avulla mahdollisia syitä virheelle.



Kuvio 6. Päämoduulien prosessoinnin hahmotelma, jossa prosessoidun komponentin nimellä löytyy vastaavuus WildFlyn-käyttöönottohakemistosta

Yleisempi tilanne on, ettei prosessoitua moduulia suoraan löydy käyttöönottohakemistosta. Tässä tilanteessa työkalu käy riippuvuustietoja läpi, kuten alimoduulien prosessoinnin yhteydessä. Eroavaisuutena on kuitenkin se, että jos riippuvuustiedoista löytyy vastaavuuksia useampi kuin yksi, käyttäjälle annetaan näistä vaihtoehdoista ne komponentit, jotka löytyvät WildFly:n käyttöönottohakemistosta. Tällä tavalla prosessia suoraviivaistetaan ja käyttäjän on helpompi valita tarpeisiinsa sopiva käyttöönottokomponentti prosessointia varten. Jos työkalu löytää vain yhden vastaavuuden emomoduuliksi riippuvuustiedoista, joka on myös yksi käyttöönottohakemiston komponenteista, suoritetaan kuvion 7 osoittamat toimenpiteet tälle komponentille. Jos yhtäkään riippuvuustiedoista löydetyistä emomoduuleista ei löydy käyttöönottohakemistosta, käyttäjälle valittavaksi kaikki vaihtoehdot löydetyistä riippuvuustiedoista. Tämän jälkeen tätä iteraatiota jatketaan niin kauan, kunnes kaikki muutokset tulevat käsitellyksi.



Kuvio 7. Päämoduulien prosessoinnin hahmotelma, jossa prosessoidun komponentin nimellä löytyy useampi vastaavuus WildFlyn-käyttöönottohakemistosta

Käyttäjällä on työkalun käyttöliittymällä mahdollisuus valita, onko muuttuneissa tiedostoissa mukana tietokantaskripti- tai sovellusparametrimuutoksia. Näiden prosessointi suoritetaan varsinaisten ohjelmakomponenttien prosessoinnin jälkeen. Työkalu yhdistää automaattisesti kaikki .sql-päätteiset tiedostot muuttuneista tiedostoista tietokantaskripteiksi, sekä kaikki .xml-tiedostopäätteiset tiedostot parametrimuutoksiksi. Nämä tiedostot siirretään Docker-kontin tilapäiskansioon ja suoritetaan tarvittavat terminaalikomennot SSH-yhteyden avulla. Näiden komentojen suorituksen jälkeen muutokset ovat heti käyttäjän nähtävillä.

6 TULOKSET JA POHDINTA

Työkalun toteuttaminen on helpottanut työn laatijan kehitysprosessia huomattavasti. Aivokuormitus on vähentynyt, koska työkalu osaa päätellä itse, mitkä komponentit tulee prosessoida – parhaimmassa tapauksessa vaatimatta käyttäjältä lainkaan syötettä. Myös koodimuutosten ulkopuoliseen prosessiin kuluva aika on ainakin puolittunut, jota voi pitää valtavana hyötynä tehokkuutta ajatellen. Työkalun käytöstä tullaan suorittamaan lyhyt pilottijakso pienellä otannalla, jotta saadaan arvokasta palautetta jatkokehitystä varten.

Suunnitelmassa asetetuista vaatimuksista saavutettiin suurin osa. Asiakasprojektiriippumattomuuden toteutumista ei pystytty työn aikana varmistamaan, johtuen NDA-sopimuksista, joiden allekirjoittamista vaaditaan asiakasprojektien kehitykseen. Myös käyttöjärjestelmäriippumattomuutta ei voitu varmistaa työn aikana, sillä työn laatijalla on ollut käytössään vain Linux-pohjainen käyttöjärjestelmä. Näiden toteutuminen varmistuu, kun työkalu otetaan yleisempään käyttöön.

Työn tavoitteena oli rakentaa suunnitelman pohjalta edellä mainittu työkalu, jonka työn toimeksiantaja oli tilannut. Työn tarkoituksena oli parantaa kehittäjän tehokkuutta ja tuottavuutta, sekä saavuttaa pitkäaikainen ajansäästöllinen hyöty ohjelmiston parissa työskenteleville kehittäjille, sekä työnantajalle.

Tavoitteeksi asetettu suunnitelman pohjalta rakennettu työkalu saatiin suoritettua. Tarkoitukseksi asetettu tehokkuuden ja tuottavuuden parantaminen ja ajansäästöllinen hyöty on konkreettisesti ja selvästi havaittavissa päivittäisessä kehitystyössä.

Jatkokehitysideoita työn aikana tuli vastaan muutamia. Käyttäjälle tulisi antaa parempi työkalun hallittavuus lisävalintojen avulla, joiden myötä voidaan tarjota työntekijän yksilöllisiin mieltymyksiin mukautuva työkalu. Lisäksi yhteensopivuutta eri asiakasprojektien kanssa voidaan optimoida, jotta työkalun

saumaton käyttö ei rajoitu vain tiettyyn osajoukkoon kehittäjistä. Vaikka prosessiin kuluva aika on huomattavasti vähentynyt, on mahdollista, että työkalun koodipohjaa voidaan optimoida merkittävästi, jotta pystytään edelleen minimoimaan ylimääräinen rakennusprosessiin haaskautuva työaika.

LÄHTEET

Baeldung. (27.2.2022). What Is an Apache Maven Artifact? Haettu 4.4.2023 osoitteesta https://www.baeldung.com/maven-artifact

Baeldung. (20.3.2023). Haettu 25.4.2023 osoitteesta: https://www.baeldung.com/cs/compiled-vs-interpreted-languages

Docker Inc. (n.d.). Docker overview. Haettu 24.2.2023 osoitteesta https://docs.docker.com/get-started/overview/

git-scm.com. (n.d.). git diff. Haettu 28.3.2023 osoitteesta: https://git-scm.com/docs/git-diff

Goncalves, A. (2013). Beginning Java EE 7 (1st ed. 2013.). Apress. Saatavilla osoitteesta https://doi.org/10.1007/978-1-4302-4627-5

IBM. (n.d.). Compiled versus interpreted languages. Haettu 17.12.2022 osoitteesta: https://www.ibm.com/docs/en/zos-basic-skills?topic=zos-compiled-versus-interpreted-languages

IBM Cloud Education. (30.6.2021). JVM vs. JRE vs. JDK: What's the Difference? Haettu 20.2.2023 osoitteesta https://www.ibm.com/cloud/blog/jvm-vs-jre-vs-jdk

IBM Cloud Education. (n.d.). What is Java? Haettu 20.1.2023 osoitteesta https://www.ibm.com/topics/java

JavaTpoint. (n.d.). Is Java Interpreted or Compiled. Haettu 13.12.2022 osoitteesta: https://www.javatpoint.com/is-java-interpreted-or-compiled

Limón B. (6.7.2022). Compiled vs interpreted language: Basics for beginning devs. Haettu 5.1.2023 osoitteesta: https://www.educative.io/blog/compiled-vs-interpreted-language

Microsoft Learn. (18.5.2022). What is Docker? Haettu 21.2.2023 osoitteesta https://learn.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-defined

Python, (n.d.-a). json — JSON encoder and decoder. Haettu 10.4.2022 osoitteesta: https://docs.python.org/3/library/json.html#module-json

Python, (n.d.-b). subprocess — Subprocess management. Haettu 10.4.2022 osoitteesta: https://docs.python.org/3/library/subprocess.html#module-subprocess

Python, (n.d.-c). xml.etree.ElementTree — The ElementTree XML API. Haettu 10.4.2022 osoitteesta: https://docs.python.org/3/library/xml.etree.elementtree.html

RedHat, Inc. (30.6.2016). WildFly 10 | Application deployment. Haettu 25.2.2023 osoitteesta https://docs.jboss.org/author/display/WFLY10/Application%20deployment.html

Schildt, H. (2019). Java: A beginner's guide (Eighth edition.). McGraw-Hill Education. Saatavilla osoitteesta: https://www.finna.fi/Record/samk.991415276905968?sid=2946210279

Telang T. (n.d.). What is WildFly application server? Haettu 18.2.2023 osoitteesta: https://www.educative.io/answers/what-is-the-wildfly-application-server

The Apache Software Foundation. (11.12.2022). POM Reference. Haettu 11.4.2023 osoitteesta: https://maven.apache.org/pom.html#pom-relationships

The Qt Company. (n.d.-a). QLabel Class. Haettu 19.3.2023 osoitteesta https://doc.qt.io/qt-5/qlabel.html

The Qt Company. (n.d.-b). QListWidget Class. Haettu 19.3.2023 osoitteesta https://doc.qt.io/qt-5/qlistwidget.html

The Qt Company. (n.d.-c). QPushButton Class. Haettu 19.3.2023 osoitteesta https://doc.qt.io/qt-5/qpushbutton.html

The Qt Company. (n.d.-d). QRadioButton Class. Haettu 19.3.2023 osoitteesta https://doc.qt.io/qt-5/qradiobutton.html

The Qt Company. (n.d.-e). QMovie Class. Haettu 19.3.2023 osoitteesta https://doc.qt.io/qt-5/qmovie.html

The Qt Company. (n.d.-f). QThread Class. Haettu 19.3.2023 osoitteesta https://doc.qt.io/qt-5/qthread.html

The Qt Company. (n.d.-g). QGridLayout Class. Haettu 19.3.2023 osoitteesta https://doc.qt.io/qt-5/qgridlayout.html

Turing. (n.d.). Apache Maven: What Is It? How Does This Work In Java Development? Haettu 21.2.2023 osoitteesta https://www.turing.com/kb/what-is-apache-maven#what-is-maven-in-java?

Wallenius N. (23.2.2022). Mikä on Docker ja mitä hyötyä siitä on? Haettu 21.2.2023 osoitteesta https://niklaswallenius.fi/mika-on-docker/