Jänönen Matti

Azure DevOps Git-versionhallintaominaisuuden integrointi Jira-ohjelmistoon

Insinööri (AMK)
Tieto- ja viestintätekniikka
Kevät 2021

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä(t): Jänönen Matti

Työn nimi: Azure DevOps Git-versionhallintaominaisuuden integrointi Jira-ohjelmistoon.

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: Azure, Forge, Git, Jira, Pilvipalvelu

Opinnäytetyön toimeksiantona oli tutkia ja kehittää esimerkkiratkaisu, kuinka Atlassian-yhtiön Jira-ohjelmiston pilvipalveluversion tehtävänäkymään saadaan linkit kyseistä tehtävää varten tehtyihin koodimuutoksiin, joissa on käytetty Azure DevOps -palvelun Git-versionhallintaominaisuutta. Lisäksi toimeksiantoon kuului kaupallisten ratkaisujen hyötyjen ja haittojen tutkiminen. Toimeksianto on saatu KajaPro Oy -yhtiöltä.

Työn toteutusta varten tutustuttiin Atlassian-yhtiön tarjoamiin sovelluskehitysalustoihin, ohjelmointikehyksiin ja työkaluihin. Lisäksi oli tarpeellista tutkia Azure DevOps -ohjelmiston toiminnallisuuksia. Esimerkkiratkaisua varten luotiin kaksi toteutusvaihtoehtoa, jonka jälkeen valittiin toteutustapa ja kehitystyökalut.

Esimerkkiratkaisussa käytettiin Forge-sovelluskehitysalustaa ja toteutustapana mallia, missä tiedot koodimuutoksista lähetetään Azure DevOps -palvelusta Forge-alustalla kehitetylle laajennukselle. Tiedot tallennetaan käyttäen Forge-alustan Storage API -ominaisuutta, josta ne myös haetaan Jira-ohjelmiston tehtävänäkymään.

Kehitystyön lopputulos on toimiva ratkaisu, jossa koodimuutosten linkit on saatu kohdennettua oikeille tehtävänäkymille. Kehitettävää jäi tapaan, miten tiedot tallennetaan, sekä myös käyttöliittymään olisi ollut hyvä tehdä muutos. Kaupallisten ratkaisujen osalta on luotu kustannusarvio, sekä tutkittu käyttöönoton helppoutta.

Esimerkkiratkaisu havainnollistaa, miten Atlassian-yhtiön Jira-ohjelmiston pilvipalveluversiolle voidaan kehittää laajennus. Lisäksi ratkaisua voi pienen jatkokehityksen jälkeen käyttää yrityksen infrastruktuurissa.

KAMK • University of Applied Sciences

Abstract

Author(s): Jänönen Matti

Title of the Publication: Integrating Azure DevOps Git feature to Jira Software.

Degree Title: Bachelor of Engineering

Keywords: Azure, Cloud, Forge, Git, Jira

The assignment of the thesis was to study and develop an example solution for how to get links to the issue view of the Atlassian Jira software for the code changes made for that task, which have been made with the Git version control feature of the Azure DevOps service. In addition, the assignment included researching the pros and cons of commercial solutions. The assignment was received from KajaPro Oy.

For the implementation of the work, the application development platforms, programming frameworks and tools developed by Atlassian were studied. In addition, it was necessary to study the functionalities of Azure DevOps software. For the example solution, two implementation options were created, after which the implementation method and development tools were selected.

The example solution used the Forge application development platform and implemented a model where information about code changes is sent from Azure DevOps to an extension developed on the Forge platform. The data is stored using the Storage API feature of the Forge platform, from where it is also retrieved to the task view of the Jira software.

Result of the development work is a workable solution in which the links to the code changes have been targeted to the correct task views. There was room for improvement in the way the data is stored, and it would also have been good to make a change to the user interface. A cost estimate has been created for commercial solutions, and the ease of implementation has been studied.

The example solution illustrates how to develop plug-in to the Atlassian Jira Software application. After a little further development, this solution can be used in the company's infrastructure

Sisällys

Symboliluettelo

AWS Lambda	Palvelimeton laskentapalvelu koodin suorittamiseen
FaaS	(Function as a Service) Pilvipalvelumuoto, jossa palveluntarjoaja hallitsee fyysiset palvelinlaitteistot sekä virtuaalikoneet, käyttöjärjestelmät ja verkkopalvelinohjelmistot
Git	Versionhallintajärjestelmä
Json	(JavaScript Object Notation) Yksinkertainen avoimen standardin tiedostomuoto tiedonvälitykseen.
Node.js	Asynkroninen tapahtumavetoinen ajoympäristö, joka sisältää kaiken tarvittavan JavaScript-kielellä kirjoitetun ohjelman suorittamiseen palvelimella
REST	(Representational State Transfer) HTTP-protokollaan perustuva arkkitehtuurimalli
URL	(Uniform Resource Locator) Merkkijono tiedon sijainnille

1 Johdanto

Opinnäytetyön aihe on saatu toimeksiantona KajaPro Oy -yhtiöltä. KajaPro Oy on vuonna 2004 perustettu yhtiö, joka tarjoaa ohjelmistotuotannon alihankintapalveluita ohjelmistojen kehittämiseen. Kotipaikkana toimii Kajaani, mutta yrityksellä on myös toimipiste Oulussa.

Tarkoituksena opinnäytetyössä on tutkia, miten Jira-ohjelmiston tehtävänäkymään saadaan lisättyä linkit Azure DevOps Git-versionhallinnan yhdistettyihin koodimuutoksiin (engl. commits). Ajatuksena on, että jos koodimuutoksen viesti sisältää Jira-ohjelmiston tehtäväavaimen, linkki kyseiseen koodimuutokseen lisätään siihen tehtävänäkymään, jonka avain on koodimuutoksen viestissä.

Integroinnin avulla on helpompi seurata, mitä kyseiseen työtehtävään on jo tehty. Kehittäjien ei tarvitse etsiä Azure DevOps-palvelusta kehitystyön senhetkistä tilannetta. Eli kehittäjät saavat paremman yleiskuvan työtehtävän tilanteesta nopeammin. Lisäksi integraation avulla voidaan nähdä koko työhön liittyvä ketju aina eepoksesta (engl. epic) yksittäiseen koodimuutokseen, mikä auttaa projektien johtoa työtehtävien kokonaiskuvan hahmottamisessa työtehtäviin liittyen.

Toimeksiantoon kuuluu myös kehitystyönä esimerkkiratkaisu. Lisäksi tutkitaan kaupallisten ratkaisujen hyötyjä ja haittoja.

Tutkimus ja kehitystyö suoritetaan Azure DevOps- ja Jira-ohjelmistojen pilvipalveluversioille.

2 DevOps-ympäristö

Koska toimeksianto tehdään DevOps-ympäristöön, tässä luvussa selvennetään, mitä tarkoittaa termi DevOps sekä esitellään opinnäytetyössä käytettävään ympäristöön liittyvät ohjelmistot ja järjestelmät.

2.1 Mitä DevOps tarkoittaa?

DevOps on kulttuuri, jonka keskiössä on jatkuvat käytännöt. Ajatuksena on käyttää toimintamallia, jossa tavoitteena on saada tuote julkaistua mahdollisimman nopeasti, mutta samalla pitää huolta, että tuote on myös korkealaatuinen. Automaatiota käytetään usein tuotteen laadun varmistamiseen sekä prosessin sujuvuuteen. [1.]

Toimintamallissa on myös ajatuksena hyötyä tuotteen julkaisusta mahdollisimman paljon. Tämä tarkoittaa sitä, että jos julkaisu viivästyy, siitä saavutetut hyödyt ovat vähäisemmät verrattuna sujuvaan prosessiin, jossa tuotteeseen tehdyt muutokset saadaan julkaistua jatkuvasti. Tällaista tapausta kutsutaan usein jatkuvaksi toimitukseksi. [2.] Olennainen osa DevOps-toimintamallia on myös jatkuva integrointi, joka on ohjelmistokehitysmenetelmä, jossa useiden tekijöiden koodimuutokset yhdistetään yhdeksi ohjelmistoprojektiksi tiimin yhteiseen arkistoon. Ennen integrointia käytetään automaattisia työkaluja uuden koodin oikeellisuuden vahvistamiseen. Jatkuvan integroinnin avulla virheet voidaan havaita hyvissä ajoin. [3.]

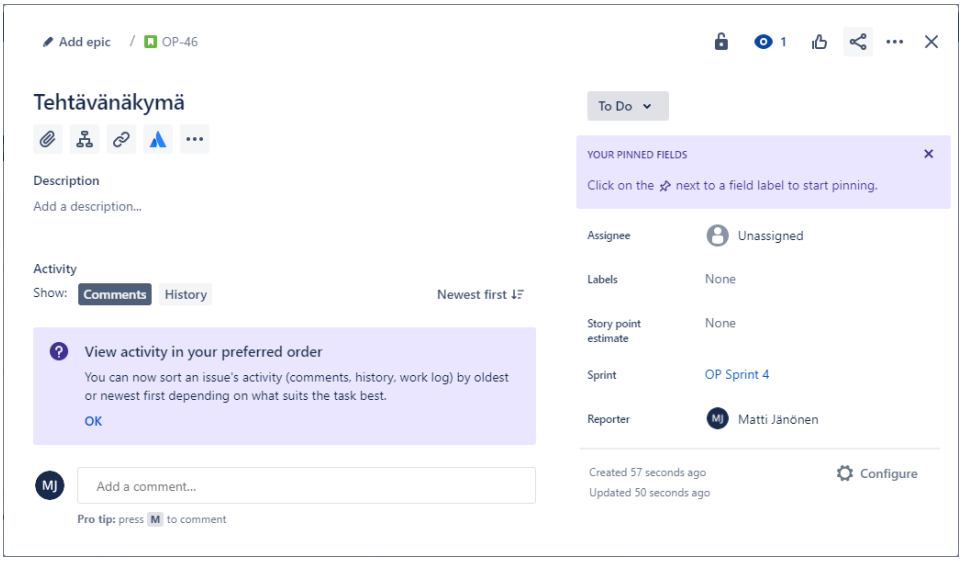
2.2 Git

Git on erittäin suosittu ja aktiivisesti ylläpidetty avoimen lähdekoodin versionhallintajärjestelmä, jonka alun perin kehitti Linus Torvalds vuonna 2005. Git-versionhallintaa käytetään kaupallisissa ja avoimen lähdekoodin projekteissa. Git on hajautettu versionhallintajärjestelmä, eli kehittäjien työkopio koodista on myös arkisto, joka voi sisältää kaiken muutoshistorian. Arkistossa olevat objektit on suojattu salausalgoritmilla, joka suojaa muutoshistoriaa vahingossa tapahtuvilta muutoksilta. Eli Git-järjestelmän avulla voidaan olla varmoja, että sisältöhistoria on aito. Lisäksi Git-järjestelmän suorituskyky on huomattavan hyvä. Esimerkiksi muutosten yhdistämiset on optimoitu suorituskyvyn kannalta. [4.]

2.3 Atlassian Jira

Atlassian-yhtiön kehittämää Jira-ohjelmistoa voidaan kuvailla työnhallintatyökaluksi, joka sopii monenlaisten työtehtävien hallintaan. Esimerkiksi ohjelmistolla voidaan hallita sovelluskehitykseen tai testitapauksiin liittyviä työtehtäviä. Ohjelmisto tukee myös ketteriä menetelmiä ja sisältää siihen tarkoitukseen Scrum- ja Kanban-taulut.[5.]

Jira-ohjelmistossa tehtävänäkymä sisältää nimensä mukaisesti jonkinlaisen työtehtävän. Näkymän avulla voidaan paremmin hallita työtehtävään liittyviä asioita. Esimerkiksi sen avulla voidaan arvioida työmäärää, sekä näkymän avulla voidaan saada yleiskatsaus työtehtävään liittyvistä asioista. Jokainen tehtävä sisältää uniikin avaimen, joka toimii tunnisteena. Avain sijaitsee näkymän vasemmassa yläkulmassa. Kuvassa 1 on esitetty tehtävänäkymä. [6.]



Kuva 1. Tehtävänäkymä.

Tehtävänäkymä on olennainen osa toimeksiantoa, koska juuri tähän näkymään tulee sisällyttää linkit koodimuutoksiin.

2.4 Microsoft Azure DevOps

Azure DevOps on ohjelmisto, joka tarjoaa tiimeille palveluja esimerkiksi versionhallintaan, CI/CD-putkien automatisointiin ja työnhallintaan. Ohjelmiston palvelut ovat kohdennettu DevOps-toimintamallia käyttävien tiimien tarpeisiin. Palvelu on saatavilla pilvessä Azure DevOps Services-palveluna tai paikallisesti Azure DevOps Server-palvelimen avulla. Azure DevOps sisältää seuraavat palvelut: Azure Repos on palvelu versionhallintaan, Azure Pipelines on palvelu jatkuvaan integrointiin ja toimitukseen, Azure Boards on työnhallintatyökalu, Azure Test Plans on palvelu sovellusten testaamiseen ja Azure Artifacts on palvelu, jolla voidaan integroida pakettien jakaminen putkiin. [7.]

Palveluiden lisäksi Azure DevOps sisältää Webhook-ominaisuuden, jonka avulla voidaan lähettää tietoja Json-muodossa palveluun, jolla on käytössä julkinen IP-osoite. Webhook voidaan määrittää lähettämään tietoja esimerkiksi tilanteessa, kun koodimuutos yhdistetään johonkin haaraan.[8.] Webhook on olennainen osa opinnäytetyön esimerkkiratkaisun toteutusta.

3 Käytetyt tekniikat

Luvun tarkoitus on esitellä Atlassian-yhtiön tarjoamat sovelluskehitysalustat ja ohjelmointikehykset, joilla voidaan luoda laajennuksia Jira-pilvialustalle ja olennaiset tekniikat liittyen opinnäytetyön toteutukseen. Opinnäytetyön esimerkkiratkaisussa on käytetty Forge-sovelluskehitysalustaa, joten luvussa keskitytään enemmän kyseiseen alustaan, mutta myös Connect ohjelmointikehys on ollut tarkastelun kohteena ennen lopullista toteutustavan valintaa, joten on tarkoituksenmukaista esitellä myös Connect tässä luvussa.

3.1 Virtualisointi

Forge-sovelluskehityksessä voidaan käyttää automaattista koodin kääntöä ja toimitusta. Tällöin tulee olla asennettuna Docker, ja lisäksi on suositeltavaa asentaa Windows Subsystem for Linux, josta WSL 2 -versio, jotta vältytään tiedostojärjestelmäongelmilta, kun käytetään Forge CLI -komentoja. [9.]

3.1.1 Windows Subsystem for Linux

Windows Subsystem for Linux mahdollistaa kehittäjien ajaa GNU/Linux-ympäristö suoraan Windows käyttöjärjestelmässä ilman virtuaalikonetta tai kaksoiskäynnistystä. WSL 2 -version avulla voidaan suorittaa ELF64 Linux -binaareja Windowsissa. Version ensisijaiset tavoitteet ovat tiedostojärjestelmän suorituskyvyn lisääminen sekä järjestelmän täydellisen yhteensopivuuden lisääminen.[10.]

3.1.2 Docker

Docker on alusta, jonka avulla sovelluksia voidaan laittaa kontteihin ja ajaa niitä. Tämä mahdollistaa sovelluksen irrottamisen infrastruktuurista, koska kontit sisältävät kaiken, mitä sovelluksen suorittaminen vaatii. Docker-alusta mahdollistaa kehittäjien toimia muuttumattomassa ympäristössä, mikä tekee kehitystyöstä suoraviivaisempaa. Esimerkiksi sovellusta ei tarvitse testata useissa eri ympäristöissä. [11.]

3.2 Valtuutus ja todentaminen

Yhteyden muodostaminen Azure- tai Jira-palveluun vaatii oikeuksien tarkistamista. Esimerkiksi yhteyden muodostamisessa Azure DevOps – palvelun REST API -ominaisuuteen, oikeudet tarkistetaan käyttäen Basic Authentication -todennustapaa, jossa salasanana käytetään Azure DevOps -palvelussa luotua henkilökohtaista tunnusta.

3.2.1 OAuth 2.0 -valtuutuskehys

OAuth 2.0 -valtuutuskehyksen avulla sovellukset voivat saada pääsyn kolmannen osapuolen HTTP-palvelujen käyttäjätileihin. Valtuutuskehys sisältää eri tapoja oikeuksien myöntämiseen, joista tyypillisimmät ovat: valtuutuskoodi (engl. Authorization Code), asiakastiedot (engl. Client Credentials), laitekoodi (engl. Device Code) ja päivitystunnus (engl. Refresh Token).[12.] [13.]

3.2.2 Tunnuspohjainen todennus

Tunnuspohjainen todennus (engl. Token-based authentication) toimii siten, että jokaiseen palvelimelle lähetettyyn pyyntöön liittyy allekirjoitettu tunnus, jonka aitous varmistetaan. Varmistamisen jälkeen palvelin vastaa pyyntöön.[14.]

3.2.3 Basic Authentication

Basic Hypertext Transfer Protocol on todennustapa, jossa lähetettävät tunnistetiedot koodataan Base64:llä. Tunnistetietoina käytetään käyttäjätunnusta ja salasanaa. [15.]

3.3 Jira Cloud -alusta

Jira Cloud -alusta on perusta kaikille Jira-tuotteille. Kehittäjille alusta mahdollistaa Jira-tuotteiden ominaisuuksien muokkauksen, laajentamisen tai uusien ominaisuuksien kehittämisen. Tämä voidaan toteuttaa luomalla sovelluksia, jotka integroidaan Jira-tuotteen kanssa. Sovelluskehitykseen Atlassian tarjoaa Forge-sovelluskehitysalustan sekä Connect-ohjelmointikehyksen. Molemmilla on mahdollista lisätä sisältöä suoraan Jira-tuotteen käyttöliittymään. Lisäksi on mahdollista käyttää Jiran REST API -ominaisuutta, vaikka sovellusta ei kehitettäisi edellä mainituilla sovelluskehitysalustoilla. Vaatimuksena on silloin käyttää OAuth 2.0 -valtuutuskehystä, kun REST API -pyynnöt tehdään.[16.]

3.4 Forge

Forge on sovelluskehitysalusta, joka on suunniteltu sovelluskehitykseen Atlassian yhtiön pilvialustalle. Forge toimii palvelimettomalla FaaS-alustalla, jonka tarjoaa AWS Lambda. Forge-alustan ajatuksena on, että kehittäjät voivat luoda sovelluksia ilman monimutkaista infrastruktuurin organisointia. [17.] Forge-alustan ajonaikainen järjestelmä sisältää joukon rajapintoja, jotka tarjoavat lisätoimintoja Forge-alustalle. Näillä rajapinnoilla sovellukset voivat olla vuorovaikutuksessa REST-päätepisteisiin sekä tallentaa tietoja. [18.]

Forge-alustaa kehitetään aktiivisesti, ja on hyvä ymmärtää, että osa päivityksistä voi rikkoa jo alustalle kehitetyn ohjelmiston [19]. Atlassian pitää vanhentuneet ominaisuudet kuusi kuukautta toiminnassa ennen niiden poistamista. Kehittäjät voivat siten valmistautua muutoksiin hyvissä ajoin. Osa Forge-alustan ominaisuuksista on vielä betavaiheessa, ja kuuden kuukauden sääntö ei koske näitä ominaisuuksia. [20.]

3.4.1 Forge CLI

Forge CLI on komentoriviliitäntä, joka auttaa hallitsemaan Forge-sovelluksia. Node.js ajoympäristöstä tulee olla asennettuna vähintään versio 10.0.0, jotta Forge CLI:n voi asentaa. Jos halutaan käyttää kaikkia Forge CLI:n komentoja, Node.js ajoympäristön lisäksi tulee asentaa Docker, josta vähintään versio 17.03. Forge CLI:n asennus voidaan suorittaa käyttäen `npm install -g`

`@forge/cli` komentoa, ja lista kaikista komennoista saadaan `forge --help` komennolla. [21.]

3.4.2 Manifest

Tiedosto manifest.yml on Forge-sovelluksen juuressa oleva tiedosto, jossa määritellään Forge-sovellus. Tiedosto lisätään automaattisesti, kun Forge-sovellus luodaan käyttäen Forge CLI -komentoa `forge create`. [22.]

3.4.3 Web trigger

Web trigger -ominaisuus mahdollistaa HTTP-pyyntöjen sitomisen Forge-sovelluksen funktioon. Esimerkiksi kolmannen osapuolen Webhook-ominaisuudesta voidaan lähettää POST pyyntö Web trigger-ominaisuudelle määritettyyn URL-sijaintiin. Tämän jälkeen funktio voi käsitellä pyyntöön liittyvät tiedot. [23.]

3.4.4 UI kit

UI kit helpottaa käyttöliittymien luontia Forge-sovelluksille. Työkalulla on tarkoitus luoda yksinkertaisia käyttöliittymäelementtejä. UI kit koostuu kolmesta pääkäsitteestä, komponentti (engl. component), koukku (engl. hook) ja tapahtumankäsittelijä (engl. event handler). Komponentit kuvaavat sovelluksen käyttöliittymän visuaaliset elementit, koukkujen avulla käyttöliittymästä saadaan dynaaminen, ja tapahtumankäsittelijä hallitsee tilanteita, joissa käyttäjä on vuorovaikutuksessa käyttöliittymän kanssa.[24.]

3.4.5 Rajoitukset

Forge-sovelluskehitysalusta sisältää rajoituksia, jotka on hyvä ottaa huomioon, jos laajennuksia kehitetään kyseisellä alustalla. Forge on Node.js-pohjainen, mutta seuraavat sisäänrakennetuista Node.js-moduuleista eivät ole tuettuja: async_hooks, child_process, cluster, constants, dgram,

dns, domain, http2, module, net, perf_hooks, readline, repl, sys, tls, trace_events, tty, v8, vm, worker_threads. [18.]

3.5 Connect

Connect on ohjelmointikehys Jira-, Confluence- ja Bitbucket-sovellusten pilvipalveluversioiden kehittämiseen. Connect-kehyksellä kehitettyihin sovelluksiin voi liittää vapaasti teknologioita, esimerkiksi tietokantoja. Koska Connect-sovellukset toimivat etäyhteydellä HTTP:n kautta, sovelluksen voi kirjoittaa millä ohjelmointikielellä tahansa. Atlassian on kehittänyt kaksi ohjelmointikehystä, jotka auttavat sovellusten kehityksessä. Näistä kehyksistä Atlassian-connect-express on Node.js-pohjainen, ja atlassian-connect-spring-boot on Java-pohjainen. Lisäksi kehittäjäyhteisö on luonut useita kehyksiä, esimerkiksi Python-ohjelmointikielelle on saatavilla Atlassian Python API. Kehittäjät voivat myös myydä Connect sovelluksia Atlassian Marketplacessa. [16.] [25.]

3.6 REST API

REST API mahdollistaa vuorovaikutuksen Jira-palvelun ja sovelluksen välillä. Rajapinnan avulla saadaan pääsy Jira-palvelun toimintoihin. Esimerkiksi voidaan hakea kommentit tietystä tehtävästä tai luoda palveluun uusi projekti. Yhteyden muodostaminen vaatii Connect-sovelluksilta JWT (JSON Web Token)-pohjaisen todennuksen, ja muilta sovelluksilta OAuth 2.0 -valtuutuksen. Lisäksi useimmat rajapinnan toiminnot vaativat oikeuksia toimiakseen, joten on pidettävä huoli siitä, että käyttäjällä, jonka nimissä pyyntö rajapintaan tehdään, on riittävät oikeudet. [26.]

4 Kaupalliset ratkaisut

Atlassian Marketplace on palvelu, jossa kehittäjät voivat myydä laajennuksia yrityksen tuotteille. Kyseisessä palvelussa on saatavilla laajennuksia, joilla voi toteuttaa Git-integraation Jira-ohjelmiston pilvipalveluversioon. Tähän opinnäytetyöhön on valittu kaksi kaupallista laajennusta tarkasteltavaksi. Varsinaista testausta ei suoriteta, vaan tarkastelu perustuu tekijöiden luomaan dokumentaatioon.

4.1 Azure Git Listener for Jira

Azure Git Listener for Jira on Mage Softwaren luoma laajennus, jolla voi Jira Cloud -sovelluksen tehtävänäkymässä nähdä linkitetyt Azure-palvelun tietovaraston (engl. repository) koodimuutokset, haarat (engl. branch) ja vetopyynnöt (engl. pull request). Lisäksi laajennuksella voi luoda uusia haaroja sekä vetopyyntöjä Azure-palvelun tietovarastoon. [27.]

Laajennuksen voi asentaa Atlassian Marketplace-palvelussa helposti nappia painamalla. Asennuksen jälkeen tuotteeseen tulee tehdä määritys, missä Azure-palvelun tietovarastot sidotaan Jira-palveluun. Käyttäjä voi käyttää OAuth-protokollaa tai tunnuspohjaista valtuutusmenetelmää linkityksen suorittamiseen. Laajennus osaa automaattisesti havaita kaikki tietovarastot, joihin käyttäjän Microsoft-tilillä on oikeudet ja näyttää ne määrityssivulla. Käyttäjä voi valita mitkä tietovarastot haluaa sitoa laajennukseen, ja näkee viimeisen ajankohdan, milloin laajennus on tarkistanut tietovaraston uusien tietojen osalta. Määrityksen jälkeen Jira-ohjelmiston tehtävänäkymän sivupaneelissa näkyy siihen sidotut koodimuutokset, haarat ja vetopyynnöt. Vaatimuksena käyttäjän on lisättävä Jira-tehtävänavain koodimuutosten viesteihin ja vetopyyntöjen otsikoihin sekä haarojen nimiin Azure DevOps -palvelussa. [28.]

Alla taulukossa 1 on [29] laajennuksen käyttöönoton kustannukset. Hinnat on muutettu euroiksi 1.3.2021 valuuttakurssin mukaisesti [30].

Käyttäjä määrä	Hinta/kuukaudessa	Hinta/vuodessa
1-10	0,83	9,92
50	41,32	495,87
100	82,64	991,74
1000	305,79	3669,42

Taulukko 1. Kustannukset

4.2 Git Integration for Jira

Git integration for Jira on BigBrassBand-yhtiön luoma laajennus, millä voi nähdä koodimuutokset Jira-palvelussa. Tämän lisäksi laajennuksella voidaan tehdä vetopyyntöjä ja hallita haaroja. Laajennus tukee kaikkia Git-servereitä. [31.]

Git Integration for Jira-laajennus luo linkin koodimuutoksen ja tietyn Jira-tehtävän välillä. Laajennus hakee tiedot Git-versionhallinnan tietovarastosta. Laajennus lisää tehtävänäkymään Git Roll Up- ja Git Commits -välilehdet. Asennus on helppo tehdä Atlassian Marketplace -palvelussa, mutta koska kyseessä on toiminnallisuuksiltaan kohtuullisen laaja kokonaisuus, tuotteen määrittämiseen pitää varata hieman aikaa. Määritys ei kuitenkaan ole vaikea toteuttaa.[32.]

Alla taulukossa 2 on [33] laajennuksen käyttöönoton kustannukset. Hinnat on muutettu euroiksi 1.3.2021 valuuttakurssin mukaisesti [30].

Käyttäjä määrä	Hinta/kuukaudessa	Hinta/vuodessa
1-10	8,26	99,17
50	82,64	991,74
100	165,29	1983,47
1000	394,63	4735,54

Taulukko 2. Kustannukset

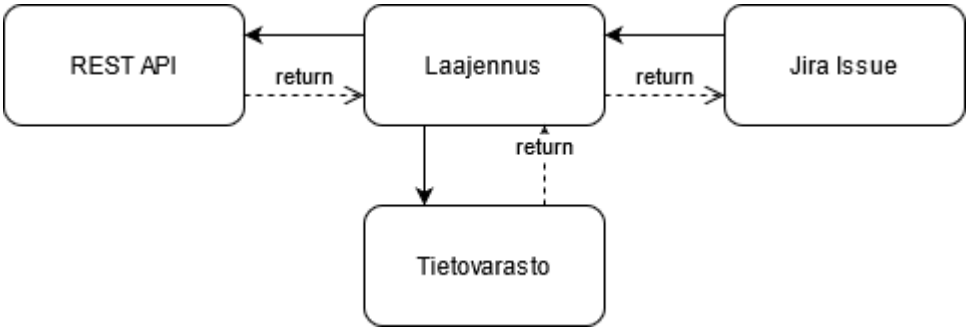
4.3 Kaupallisten ratkaisujen yhteenveto

Kaupallisissa ratkaisuissa käyttöönotto on nopea toteuttaa, ja käyttäjä saa useita ominaisuuksia käyttöönsä. Haittapuolena on tietenkin kustannukset, joiden kohtuullisuus käyttäjien itse päätettävissä. Lisäksi tuotteita ei voi laajentaa tai muokata. Esimerkiksi käyttäjä voisi haluta koodimuutosten viestit eri sijaintiin tehtävänäkymässä, mutta tämä ei onnistu ainakaan tarkastelun kohteena olleissa laajennuksissa.

5 Toteutusvaihtoehdot

Esimerkkiratkaisua varten on kehitetty kaksi toteutusvaihtoehtoa, joista on valittu toinen esimerkkiratkaisun toteutukseksi. Tämän luvun tarkoitus on esitellä vaihtoehdot, mitkä suunnitteluvaiheessa ovat olleet tarkastelun kohteena.

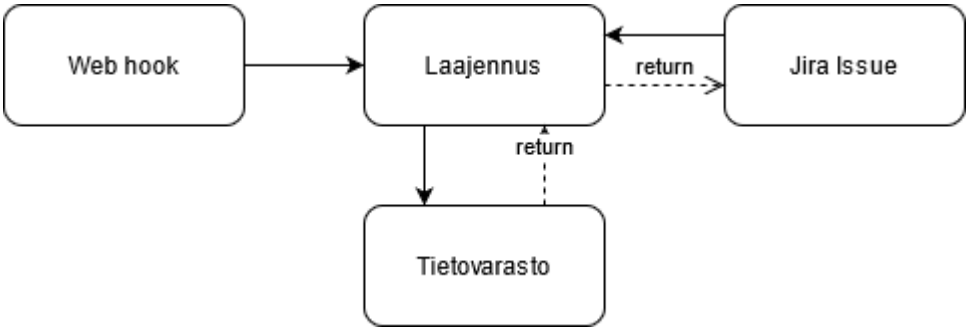
Ensimmäisessä vaihtoehdossa ajatuksena on hakea tarvittavat tiedot käyttäen Azure-palvelun REST API -rajapintaa. Eli laajennukselle määritetään Azure-projektit, joista tiedot haetaan. Tämän jälkeen lisätään tietokantaan tarvittavat tiedot jokaisesta koodimuutoksesta, joissa on Jira-tehtäväavain. Tämä prosessi pitää toistaa tietyin aikavälein, jotta tiedot saadaan päivitettyä, ja tietojen käsittelyyn pitää kehittää systeemi, joka lisää tietokantaan vain uudet tiedot. Tässä voidaan käyttää apuna koodimuutoksessa olevaa tunnusta, joka on uniikki jokaisella koodimuutoksella. Eli, jos koodimuutos sisältää avaimen, tarkistetaan, onko koodimuutoksen tunnus tietovarastossa, jonka jälkeen joko tallennetaan tai hylätään tiedot, riippuen tarkastuksen tuloksesta. Lopuksi tiedot haetaan tietokannasta tehtävänäkymään. Kuvassa 2 on esitetty toimintaperiaate kaaviona.



Kuva 2. Toteutusvaihtoehto yksi.

Tätä toteutustapaa voidaan pitää kohtuullisen raskaana vaihtoehtona, jos projekteja ja niissä olevia koodimuutoksia on paljon. Hyvänä puolena voidaan pitää, että kaikki tarvittavat tiedot saadaan varmasti lisättyä tietokantaan, sillä vaikka laajennus olisi suljettuna jonkin aikaa uudet tiedot saadaan aina haettua.

Toinen vaihtoehto on käyttää Azure-palvelun webhook-toimintoa, mikä voidaan määrittää lähettämään tiedot laajennukselle aina, kun koodimuutos tai vetopyyntö yhdistetään johonkin haaraan. Laajennus lisää tiedot tietovarastoon, jos koodimuutoksen viestissä on avain, jonka jälkeen ne voidaan hakea sieltä tehtävänäkymään. Kuvassa 3 on esitetty toimintaperiaate kaaviona.



Kuva 3. Toteutusvaihtoehto kaksi.

Tämän mallin heikkoutena voidaan pitää tilannetta, jossa jokin hetkellinen toimintahäiriö ilmenee esimerkiksi Azure-palvelussa, ja webhook ei lähetä tietoja laajennukselle. Se tarkoittaa, että tarvittavia tietoja ei ikinä saada tallennettua tietovarastoon. Tämä malli on kuitenkin kevyempi kuin edellinen, koska tietoja ei tarvitse prosessoida lähellekään niin paljoa.

Molemmissa toteutusvaihtoehdoissa täytyy muokata Jiran-tehtävänäkymää, joten Atlassian-yhtiön tukemia sovelluskehitysalustoja tai ohjelmointikehyksiä voidaan pitää tarpeellisina, jotta laajennuksen voi kehittää onnistuneesti. On myös hyvä huomioida, jos käytetään webhook-ominaisuutta Azure-palvelun pilvipalveluversiossa, ja ohjelmointikehyksenä käytetään Connectia, laajennukselle tarvitaan julkinen IP-osoite. Forge-alustalla voidaan käyttää web trigger -ominaisuutta, minkä ansiosta erillinen julkinen IP-osoite laajennukselle ei ole tarpeellinen. Lisäksi tietokannan lisääminen Forge-alustalle ei ole mahdollista. Alustalle on sen sijaan saatavilla Storage API ja Properties API, joita voi käyttää tietojen tallentamiseen, mutta ne ovat ominaisuuksiltaan rajoittuneita. Alustan valinnassa vaikuttavia tekijöitä ovat lisäksi Connect-ohjelmointikehyksen vapaus, sillä kehykselle voi vapaasti liittää haluamiaan kirjastoja ja tietokantoja, mutta toisaalta Atlassian on luonut Forge-alustalle FaaS-tyyppisen pilvipalvelu muodon, jonka avulla sovellusta ei tarvitse itse isännöidä. Sovelluksen voi toimittaa Atlassian-yhtiön palvelimelle, joka hoitaa sovelluksen isännöinnin.

6 Esimerkkiratkaisu

Vaikka Forge-sovelluskehitysalusta on vielä osittain beeta vaiheessa ja sisältää rajoituksia, esimerkkiratkaisu on toteutettu käyttämällä kyseistä alustaa. Syy valintaan on, että Forge-laajennukset toimivat ilman erillistä palvelinta käyttäen FaaS-tyyppistä pilvipalvelumuotoa, joten laajennusta ei tarvitse itse isännöidä, mikä vähentää infrastruktuurin määrää. Lisäksi Forge-alustan web trigger -ominaisuus mahdollistaa tietojen lähettämisen laajennukselle suoraan Azure DevOps -palvelun Webhook-ominaisuudesta ilman laajennukselle erikseen määritettyä julkista IP-osoitetta.

Ratkaisumallina käytetään tapaa, jossa Azure DevOps -palvelusta lähetetään webhook-ominaisuutta käyttäen tiedot laajennukselle, jossa tiedot tallennetaan käyttäen Forge-alustan Storage API-resurssin tietosäiliötä. Tiedot Jira-ohjelmiston tehtävänäkymään saadaan haettua samasta tietosäiliöstä. Jotta linkitys Azuren ja Jiran välillä onnistuu, on koodimuutosten viesteihin sisällytettävä Jira-tehtäväavain.

Yhtenä vaikuttavana tekijänä ratkaisumallin valintaan on mallin konsepti, jota voi käyttää myös, jos haluaa jonkin toisen palvelun webhook-ominaisuudesta lähettää tietoja Forge-sovellukselle johonkin näkymään. Tämä konsepti toimii hyvin sellaisissa tapauksissa, vaikka tietojenkäsittely joudutaan kehittämään uudelleen vastaamaan uusiin vaatimuksiin.

6.1 Ympäristön toteutus

Jos halutaan automaattinen koodin kääntö ja toimitus pilvialustalle, on kehittäjän asennettava Docker ja WSL2 backend, muussa tapauksessa Node.js-ajoympäristö ja Forge CLI -asennukset riittävät. Tällöin kehittäjän tulee koodimuutoksen jälkeen käyttää `forge deploy` komentoa, jotta muutokset käännetään ja toimitetaan pilvialustalle. Automaattinen koodin kääntö ja toimitus saadaan päälle `forge tunnel` komennolla.

Lisäksi on luotava Atlassian API -tunnus, joka voidaan luoda kirjautumalla sivulle https://id.atlassian.com/manage/api-tokens. Sivulla annetaan tunnukselle nimi, jonka jälkeen tunnus voidaan luoda. Tunnuksen avulla voidaan kirjautua Forge CLI -ominaisuuteen käyttämällä komentoa `forge login`.

6.2 Määritykset

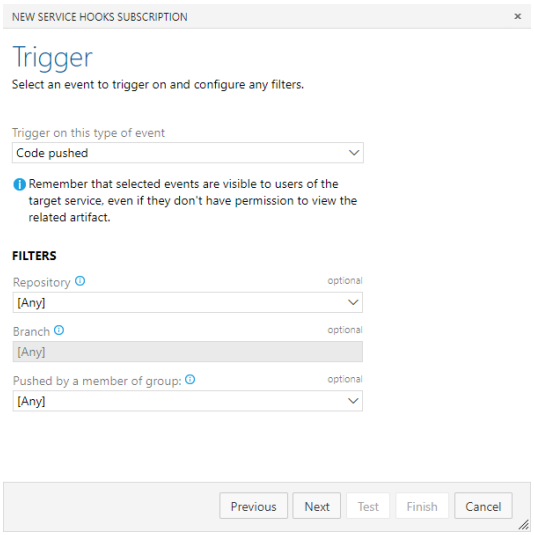
Luvussa käsitellään määritykset web trigger- ja webhook -ominaisuuksille. Määritykset täytyy tehdä myös esimerkkiratkaisun käyttöönoton yhteydessä, ellei toisin ole mainittu.

6.2.1 Web trigger

Esimerkkiratkaisu vaatii web trigger -ominaisuuden käyttöönoton. On myös muistettava, että jokainen Forge-sovelluksen asennus vaatii uniikin web triggerin asennuksen. Sovelluksen manifest.yml-tiedostoon on lisättävä määritys web trigger -moduulille ennen kuin asennus onnistuu. Esimerkkiratkaisun manifest.yml-tiedostossa määritys on tehty, joten käyttöönoton yhteydessä manifest.yml-tiedostoon ei tarvitse tehdä muutoksia. Määritys myös sitoo funktion käsittelemään kaikki pyynnöt, jotka web triggerille luotuun URL-sijaintiin tehdään. Tämän jälkeen web triggerin voi luoda käyttämällä Forge CLI -komentoa `forge webtrigger`, jonka perään tulee lisätä sovelluksen Id, johon web trigger luodaan. Id arvo saadaan `forge install:list` komennolla, josta valitaan haluttu asennus.

6.2.2 Webhook

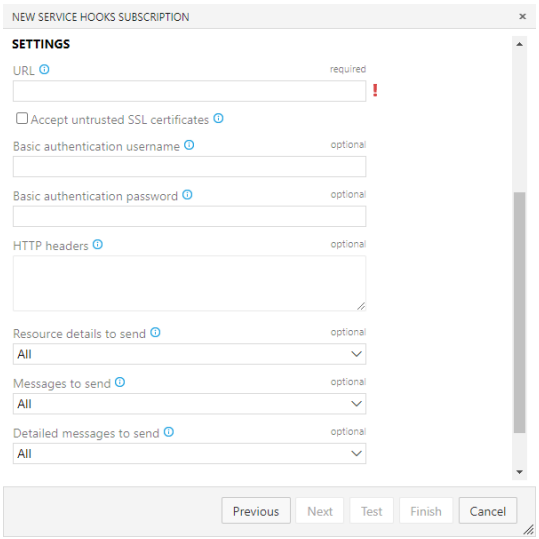
Toteutuksessa käytetään Azure-palvelun Webhook-ominaisuutta, joka tulee määrittää lähettämään POST pyyntö, joko tilanteessa missä uusi koodimuutos on yhdistetty johonkin haaraan tai uusi vetopyyntö on onnistuneesti toteutunut. Webhook-määritykset löytyvät Azure DevOps -palvelussa Project settings -valikon takaa, jonka jälkeen tulee valita Service hooks. Tämän jälkeen voidaan uudesta valikosta valita Webhook. Kuvassa 4 on esitetty näkymä asetuksista, kun valintana on koodimuutoksen yhdistäminen. Näkymän kohdassa Trigger on this type of event voidaan valita Code pushed, joka tarkoittaa koodimuutoksen yhdistämistä tai pull request merged, joka tarkoittaa vetopyyntöä. Lisäksi voidaan valita tietovarasto ja haara, jos halutaan POST-pyynnön lähetyksen tapahtuvan vain tietyissä tilanteissa. Jos valitaan vetopyyntö, tulee lisäksi valita Merge result kohtaan Merge successful, mikä tarkoittaa pyynnön lähetystä tilanteessa, missä vetopyyntö yhdistetään johonkin haaraan.



Kuva 4. Webhook-ominaisuuden laukaisu asetukset

Koska kuvassa 4 on valittu koodimuutos, ei vetopyynnölle tarkoitettua Merge result -kohtaa ole näkyvissä.

Edellisten asetusten on lisäksi määritettävä URL-osoite sekä käyttäjänimi ja salasana. Käyttäjänimen ja salasanan voi itse päättää, mutta ne on oltava samat, kuin laajennuksen configs.js-tiedostossa olevat `webtriggerlogin`- ja `webtriggerpassword` -arvot. URL-osoite tulee olla Forge-alustan web trigger -ominaisuudelle luotu URL. Kuvassa 5 on esitetty näkymä asetuksista URL sijainnille, salasanalle ja käyttäjätunnukselle.



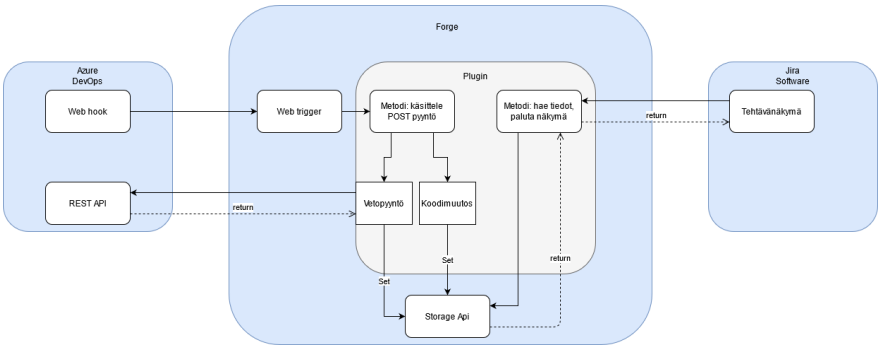
Kuva 5. Webhook asetukset

6.3 Toimintaperiaate

Azure Webhook -ominaisuudesta lähetetään POST-pyyntö, Forge-alustan web trigger-ominaisuudelle luotuun URL-sijaintiin. Kaikki pyynnöt, jotka saapuvat tähän sijaintiin, on sidottu laajennuksen funktioon, joka käsittelee pyynnön. Tiedot ovat erilaiset riippuen käsitelläänkö koodimuutoksen yhdistämistä johonkin haaraan, tai jos vetopyyntö on onnistuneesti toteutunut. Käytännössä tämä tarkoittaa, että vetopyyntö ei sisällä riittävästi tietoja koodimuutoksista, jonka seurauksena tiedot koodimuutoksista joudutaan hakemaan Azure DevOps -palvelun REST API:sta. Jotta tiedot voidaan hakea REST API:sta on Azure-palveluun luotava PAT (engl. personal access token) eli henkilökohtainen käyttöoikeuksien tunnus. On hyvä muistaa, että kaikki tietojen haut tehdään sen käyttäjän nimissä, jolle tunnus on luotu, joten käyttöoikeuksien tulee olla riittävät, että haku on mahdollista suorittaa onnistuneesti. Kun tiedot koodimuutoksista, jotka sisältävät tehtäväavaimen on saatu, tallennetaan ne käyttäen Jiran Storage API -ominaisuutta.

Yksi Storage APIn rajoitteista on, että sieltä voi maksimissaan hakea vain kaksikymmentä tallennettua kohdetta kerrallaan. Käytännössä se tarkoittaa, että tehtävänäkymään tällä toteutuksella voidaan saada maksimissaan sen verran linkkejä näkyviin. Esimerkkiratkaisussa maksimimäärä on pudotettu yhdeksääntoista, koska vanhin haettu viesti poistetaan automaattisesti, jotta tilaa saadaan uudemmille linkeille. Ongelman voi ratkaista tallentamalla tiedot kaikista koodimuutoksista, jotka on tehty tietylle Jira-tehtävälle taulukkoon, jonka jälkeen tarvitsee vain hakea yksi tallennettu kohde Jira-tehtävää kohden.

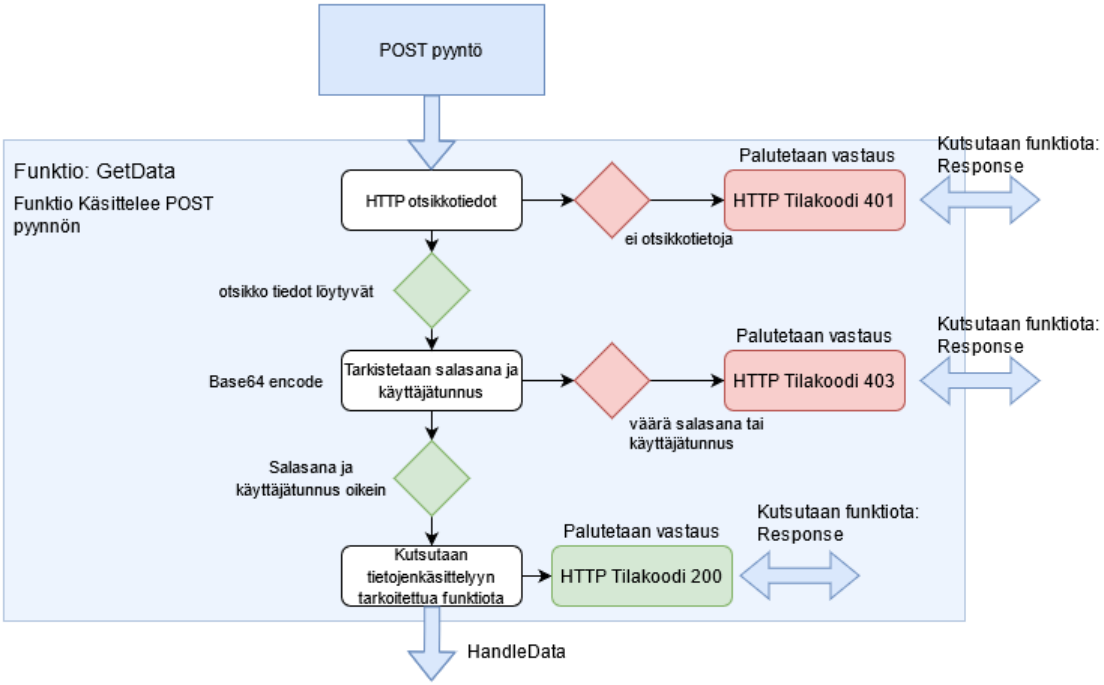
Kun tehtävänäkymä avataan Jira-palvelussa laajennuksen, funktio hakee tiedot tietovarastosta järjestää ne luontiajan mukaiseen järjestykseen, ja palauttaa taulukon tiedoista käyttäen UI Kit -komponentteja. Kuvassa 6 kaavio esimerkkiratkaisun toimintaperiaatteesta.



Kuva 6. Esimerkkiratkaisun toimintaperiaate

6.4 Tietojenkäsittelyn kuvaus

Funktio GetData ottaa tiedot vastaan, jotka on lähetetty Azure DevOps -palvelun Webhook-ominaisuudesta. Kuvassa 7 esitetään funktion toimintaperiaate kaaviona.



Kuva 7. Funktio GetData kaaviona

Funktio ensin tarkistaa sisältääkö POST-pyyntö otsikkotiedot. Jos otsikkotiedot puuttuvat palautetaan vastaus HTTP-tilakoodilla 401, joka tarkoittaa, että pyyntö ei sisällä otsikkotietoja.

```
//Tarkistetaan sisältääkö pyyntö otsikkotiedot
if (!request.headers.authorization ||
!request.headers.authorization.indexOf('Basic ') === -1)
{
      //Lähetetään vastaus, joka muodostetaan Response-funktiossa
      return Response('', 401, 'Missing Authorization Header');
}
```

Apuna käytetään funktiota Response, jolle annetaan parametreinä tiedot body-sisällöstä, tilakoodi ja viesti. Koska vastauksen mukana ei ole tarpeellista lähettää body-sisältöä, parametri voi olla tyhjä.

```
function Response(body, statusCode, statusText)
{
      //Vastauksen rakenne Json-muodossa
      var response =
                  {
                    body,
                    headers: {
                        'Content-Type': ['application/json'],
                    },
                    statusCode,
                    statusText
                  }
      //Palutetaan vastaus
      return response;
}
```

Funktio Response luo vastauksen, joka palautetaan kohteeseen, josta POST-pyyntö lähetettiin.

Jos otsikkotiedot löytyvät, käytetään base64-työkalua purkamaan otsikkotietojen koodauksen, jotta saadaan haettua käyttäjätunnus ja salasana.

Käyttäjätunnusta ja salasanaa verrataan config.js-tiedostossa oleviin arvoihin `webtriggerlogin` ja `webtriggerpassword`. Jos arvot poikkeavat, lähetetään vastaus tilakoodilla 403, joka ilmaisee väärää salasanaa tai käyttäjätunnusta. Jos vertailun tuloksena arvot ovat samat, lähetetään vastaus tilakoodilla 200, joka tarkoittaa pyynnön onnistumista, ja kutsutaan handleData-funktiota. Parametrinä annetaan saatu pyyntö.

Funktio handleData nimensä mukaisesti käsittelee tiedot liittyen POST-pyyntöön, jotka saadaan pyynnön body-sisällöstä. Funktio luo ensin Json-objektin body-sisällöstä, jonka jälkeen tarkistetaan, onko kyseessä vetopyyntö vai koodimuutos. Tieto saadaan Json-tiedoston `eventType` kohdasta, jossa `git.push` tarkoittaa koodimuutosta, ja `git.pull.request.merged` tarkoittaa vetopyyntöä. Vetopyynnön kohdalla tulee vielä tarkistaa, onko kyseessä juuri luotu vai onnistuneesti johonkin haaraan yhdistetty vetopyyntö. Tämä tieto löytyy Json-tiedoston `resource.status` kohdasta, jossa `completed` tarkoittaa onnistuneesti yhdistettyä vetopyyntöä. Jos kyseessä on koodimuutos, varmistetaan, että Json-tiedosto sisältää tiedot koodimuutoksesta, jos tietoja ei löydy, koodin suoritus loppuu. Tietojen löytyessä tarkistetaan sisältääkö koo-

dimuutoksen viesti Jira-tehtäväavaimia. Apuna käytetään funktiota getIssueKeys, johon parametrinä annetaan koodimuutoksen viesti. Viesti saadaan Json-tiedoston `commits`-taulukon `comment` kohdasta.

Avainten hakemisessa funktio getIssueKeys käyttää apuna säännöllistä lauseketta (engl. Regular Expression, Regex), jolla voidaan hakea viestistä tiettyjä kaavan mukaisia merkkijonoja. Kaavana käytetään `/\d+-[A-Z]+(?!-?[a-zA-Z]{1,10})/g` mallia, joka mahdollistaa Jira-tehtävänäkymän avaimen hakemisen tekstistä [34]. Vetopyyntö voi sisältää useita koodimuutoksia, ja yhdessä koodimuutoksen viestissä voi olla useita avaimia, joten löydetyt avaimet tallennetaan taulukkoon. Jos avaimia löytyy, palautetaan taulukko. Avainten hakuprosessi perustuu sivustolla [34] esitettyyn tapaan. Kuvassa 8 on esitetty avainten hakuprosessi.

```
function getIssueKeys(s)
{
  var jira_matcher = /\d+-[A-Z]+(?!-?[a-zA-Z]{1,10})/g;

  if(s != null)
  {
  s = s.toString();
  s = reverseString(s);
  var m = s.match(jira_matcher);
    if(m != null )
    {
      for (var i = 0; i < m.length; i++) {
      m[i] = reverseString(m[i]);
      }
    reverseString(m);
    return m;
    }
    else return '';
  }
  else return '';
}
```

Kuva 8. Avainten haku

Johtuen tavasta miten match-metodi toimii, joudutaan apuna käyttämään funktiota reverseString, joka muuttaa merkkijonon käänteiseen järjestykseen.

Jos avaimia ei löydy, koodin suoritus loppuu. Avainten löytyessä haetaan Json-tiedostosta viestin ja avainten lisäksi tieto luontiajasta, sekä luodaan linkki kyseiseen koodimuutokseen. Linkki joudutaan luomaan, koska Json-tiedoston tiedoista ei suoraan saada poimittua linkkiä. Linkki luodaan

yhdistämällä Json-tiedoston `resource.repository.remoteUrl` merkkijonoon `/commit/`, jonka jälkeen sijoitetaan koodimuutoksen tunniste, joka saadaan Json-tiedoston `resource.commits`-taulukon `commitId` kohdasta. Loppuun lisätään seuraava merkkijono.

`?refName=refs%2Fheads%2Fmaster`

Lopuksi tallennetaan tiedot käyttäen Storage API -ominaisuutta. Tietovarastoon tallennetaan viesti, linkki, luontiaika ja avaimet. Storage API vaatii tallennuksen yhteydessä avaimen, jonka avulla tiedot voidaan tunnistaa ja hakea. On muistettava, että jos tietojen tallennuksessa käytetään samaa avainta, entinen tieto korvataan uudella. Tämän takia tietojen tallennuksessa käytetään avaimena merkkijono sarjaa, joka alkaa löydetyllä Jira-tehtäväavaimella. Tämän jälkeen, lisätään merkit #1# ja loppuun koodimuutoksen tunniste. Näin voidaan tallentaa tiedot kirjoittamatta entisten tietojen päälle ja jokainen avain on uniikki.

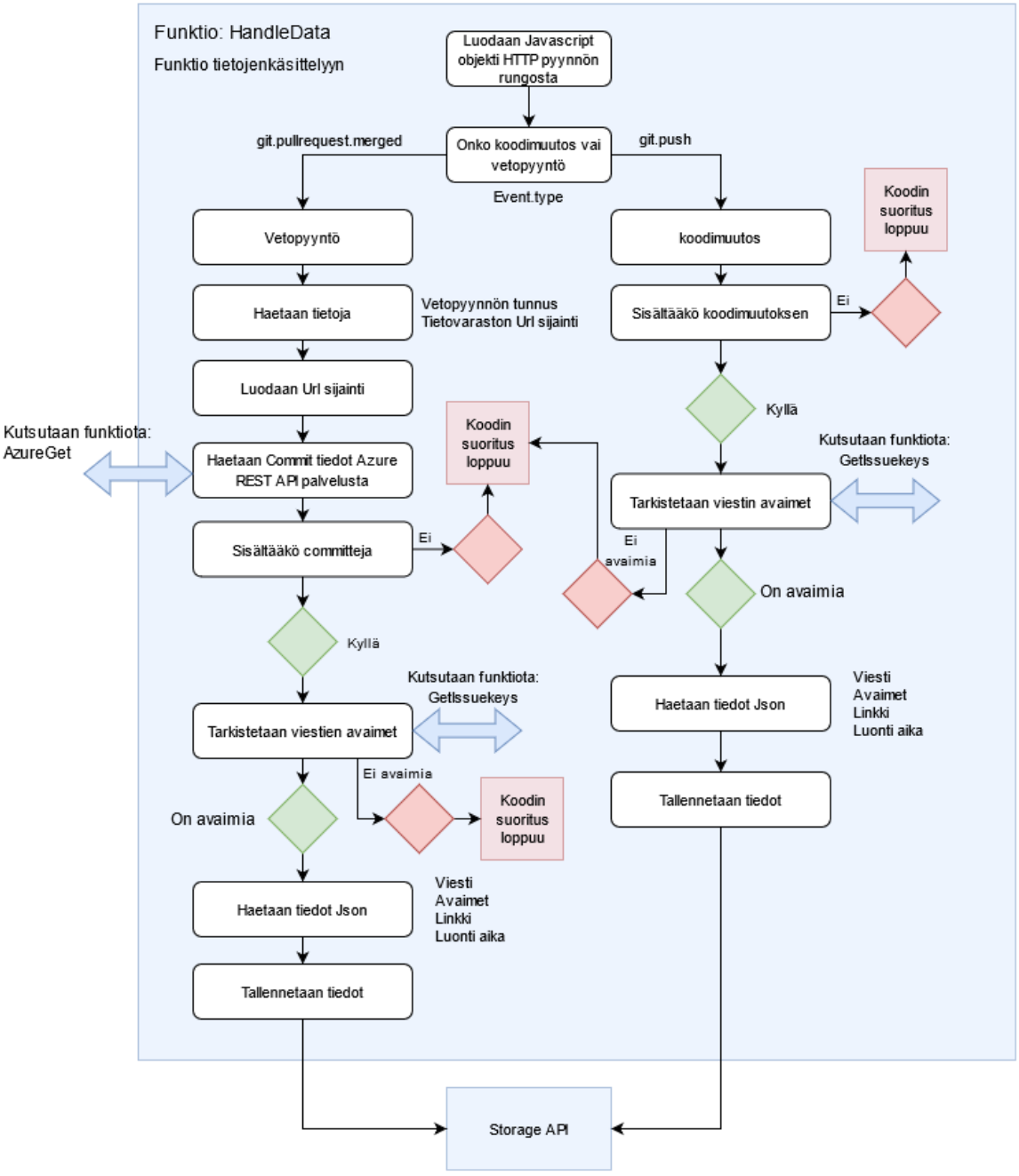
Vetopyyntöjen kohdalla prosessi on samankaltainen, mutta tiedot koodimuutoksista on haettava Azure DevOps -palvelun Rest API -ominaisuudesta, ja on muistettava, että vetopyyntö voi sisältää useita koodimuutoksia. Funktion azureGet avulla voidaan kyseiset tiedot hakea. Parametrinä funktioon annetaan URL sijainti, josta tiedot haetaan. URL sijainti saadaan rakennettua hakemalla ensin Json-tiedostosta `resource.repositoty.url` kohta, johon lisätään merkkijono `/pullRequests/`, jonka jälkeen lisätään Json-tiedostosta kohta `resource.pullRequestId`, ja loppuun lisätään merkkijono `/commits?api-version=6.0`.

Funktiossa azureGet koodataan tunnistetiedot käyttäen base64-työkalun koodaus toimintoa. Vaatimuksena tarvitaan Azure-palvelun PAT ja käyttäjänimi. Tietojen haussa käytetään Forge-alustan sisäänrakennettua fetch-metodia, jolle annetaan parametreina URL sijainti ja tiedot hakuun liittyvistä määrityksistä.

```
//Määritykset Json-muodossa
var options = {
  'method': 'GET',
  'hostname': 'dev.azure.com',
  'path': '',
  'headers': {
    'Authorization': 'Basic ' + Base64-tunnistetiedot,
    'Cookie': ''
  },
  'maxRedirects': 20
};
```

Esimerkkikoodi esittää fetch-metodille annettavat määritykset.

Kuvassa 9 on esitetty kaaviona tietojenkäsittelyprosessi.



Kuva 9. Funktio HandleData

Liitteessä 1 on esitetty koko tietojenkäsittelyprosessi kaaviona POST-pyynnöstä tietojen tallennukseen, jossa on mukana kaikki käytetyt funktiot.

6.5 Tietojen haku käyttöliittymään

Tietojen saamiseksi tehtävänäkymään laajennuksessa on käytetty App nimistä funktiota. Tiedostossa manifest.yml määritetään ohjelman pääfunktio, joka on nimeltään run. Funktio run kertoo ohjelmalle, mihin näkymään tiedot saatetaan näkyviin ja mikä funktio palauttaa tiedot. Esimerkkikoodissa esitetään tapa, miten App-funktio määritetään palauttamaan tiedot Jiran-tehtävänäkymään.

```
export const run = render(
  <IssuePanel>
    <App />
  </IssuePanel>
);
```

Funktiossa App kutsumalla UI Kit -ominaisuuden sisäänrakennettua funktiota useProductContext saadaan tietojen hakua varten Jiran-tehtäväavain siitä näkymästä, missä se on juuri avattu. Avaimen avulla voidaan suorittaa tietojen haku Storage API-ominaisuudesta. Alla esimerkkikoodi, kuinka avain saadaan haettua käyttäen useProductContext-funktiota.

```
const context = useProductContext();
context.platformContext.issueKey.toString()
```

Tietojen haun suorittamiseen Storage API-ominaisuudesta käytetään kyselyä, jossa etsitään avainta, joka alkaa jollakin merkkijonolla. Kyselyssä avain tarkoittaa tietojen tallennuksen yhteydessä annettua avainta, jonka avulla tiedot voidaan hakea. Koska tallennuksen yhteydessä avain on muodostettu tavalla, jossa ensin annetaan Jira-tehtäväavain, jonka jälkeen lisätään merkit `#1#`, ja lopuksi koodimuutoksen tunniste, kyselyn avain muodostuu Jira-tehtäväavaimesta, jonka perään lisätään merkit #1#. Tällä tavoin saadaan haettua oikeat tiedot. Lisäksi kyselyyn lisätään metodit, joilla ilmaistaan kyselylle, että haetaan useita kohteita, ja rajoitetaan hakumäärä kahteenkymmeneen, joka on maksimimäärä. Kysely perustuu sivustolla [35] esitettyyn tapaan.

```
storage.query().where(key, startsWith()).limit(20).getMany());
```

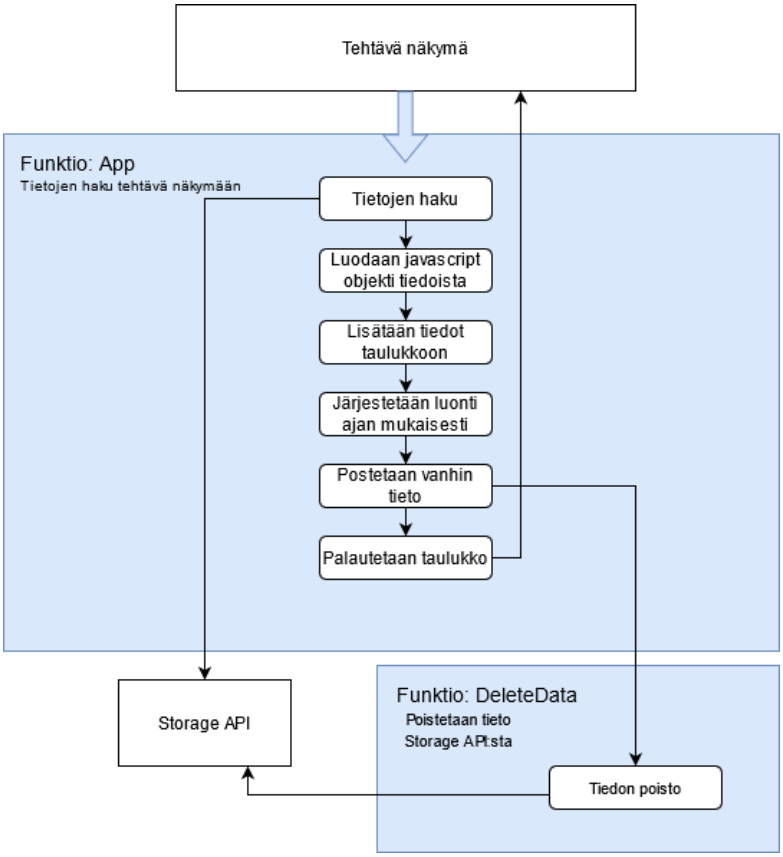
Koodin metodin startsWith parametriin tulee sijoittaa haussa käytetty avain.

Haetuista tiedoista luodaan Json-objekti, josta tiedot taulukoidaan. Taulukko järjestetään luontiajan mukaiseen järjestykseen, käyttäen sort-metodia. Jos taulukossa on soluja kaksikymmentä, kutsutaan DeleteData-funktiota, johon annetaan parametrinä luontiajaltaan vanhin avain. Tämän jälkeen tiedot palautetaan tehtävänäkymään käyttäen UI Kit -ominaisuuden komponentteja kuvassa 10 esitetyllä tavalla.

```
return (
    <Table>
        <Head>
            <Cell>
                <Text>Message</Text>
            </Cell>
            <Cell>
                <Text>Link</Text>
            </Cell>
        </Head>
        {arraytest.map(issue => (
        <Row>
            <Cell>
                <Text  format="markdown">{issue.message}</Text>
            </Cell>
            <Cell>
                <Text content={`**[Link to Azure](${issue.link})**`} format="markdown"/>
            </Cell>
        </Row>
    ))}
    </Table>
);
```

Kuva 10. Tietojen palautus

Kuvassa 11 on esitetty kaaviona koko tietojenkäsittelyprosessi, kuinka tiedot palautetaan tehtävänäkymään.



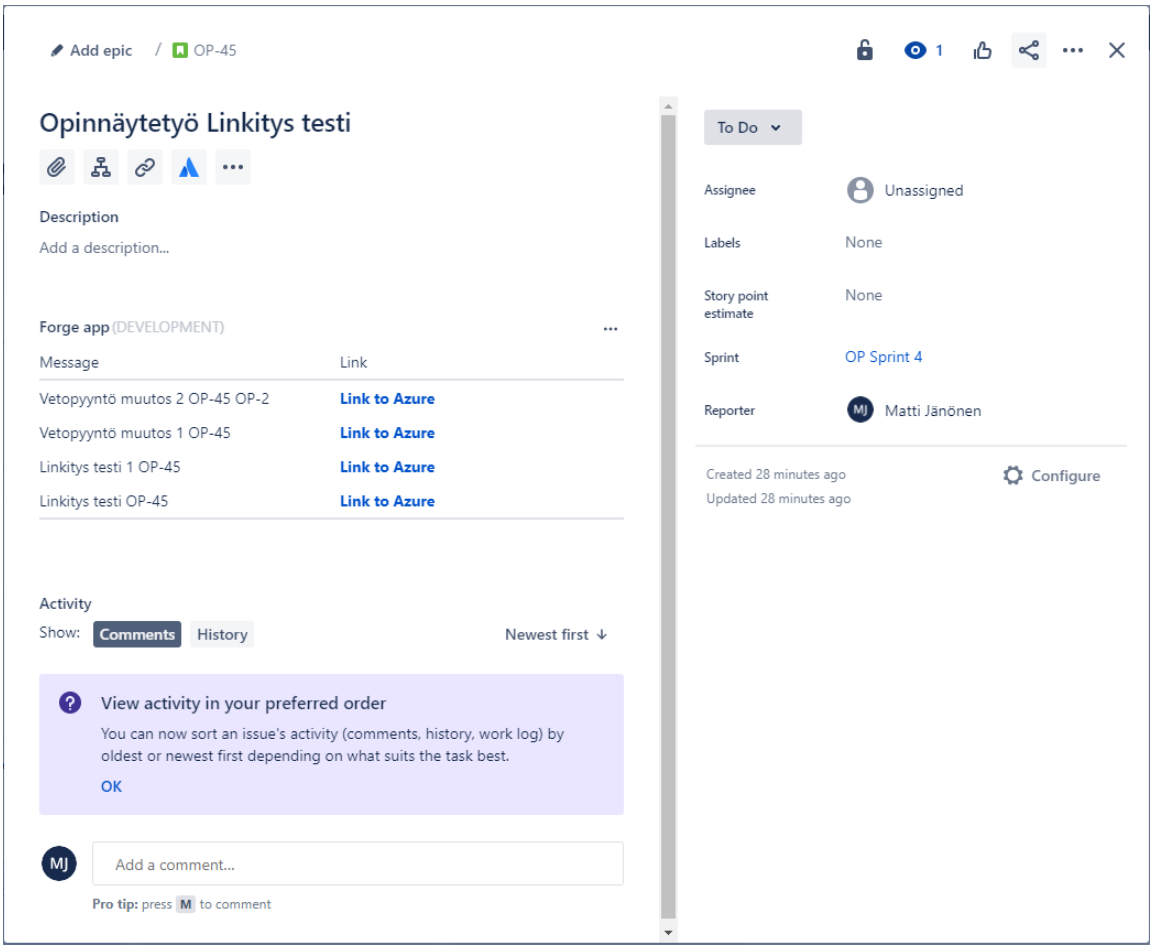
Kuva 11. Kaavio tietojen palautuksesta käyttöliittymään

Funktio DeleteData on testiluontoinen, ja ei varsinaisesti toimi oikealla tavalla App-funktion yhteydessä. Koska App-funktiota kutsutaan tehtävänäkymän avaamisen yhteydessä, voi olla tilanteita, joissa koodimuutoksia on lisätty Storage APIn tietovarastoon huomattava määrä ennen tehtävänäkymän avaamista, ja avaamisen yhteydessä poistetaan kahdestakymmenestä haetusta avaimesta luontiajaltaan vanhin. Tämä tarkoittaa, että tietovarastossa voi olla useita tietoja, joita ei olla haettu. Näistä tiedoista osa voi olla myös vanhempia kuin juuri poistettu tieto.

Jos tätä funktiota halutaan käyttää, tulisi tietojen poisto tehdä juuri ennen tietojen tallennusta. Tämä tarkoittaa, että ennen tallennusta tulee tiedot avaimista hakea ja järjestää luontiajan mukaiseen järjestykseen, jonka jälkeen kutsutaan DeleteData-funktiota. Parametrinä annetaan tietovarastoon tallennettu avain vanhimmasta tiedosta. Tämän jälkeen tallennus voidaan suorittaa.

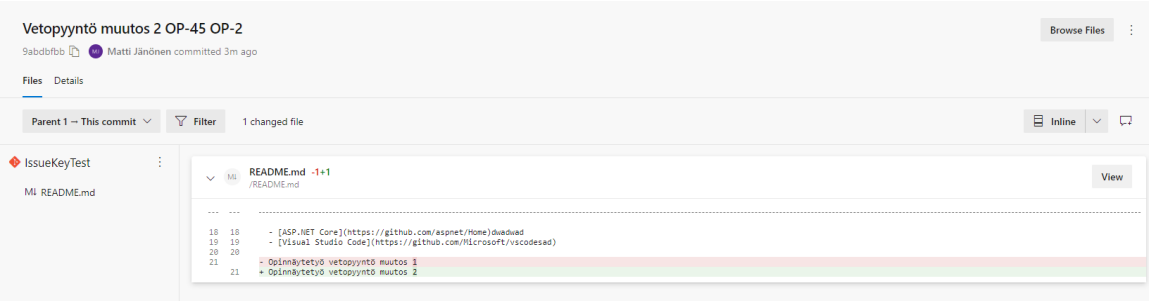
7 Lopputulos

Kehitystyön lopputuloksena linkit koodimuutoksiin saadaan esitettyä Jira-ohjelmiston tehtävänäkymässä. Kuvassa 12 on esitetty lopputulos laajennuksen käyttöliittymän ulkoasusta tehtävänäkymässä. Esimerkkiratkaisun käyttöliittymään on sisällytetty koodimuutosten viesti sekä linkki.



Kuva 12. Linkit ja viestit tehtävänäkymässä

Valitsemalla Link to Azure käyttäjä siirtyy selaimessa suoraan koodimuutokseen. Kuvassa 13 on esitetty koodimuutos, kun käyttöliittymän ylintä linkkiä painetaan. Tuloksena käyttäjä siirtyy selaimessa Azure DevOps -palvelun koodimuutokseen.



Kuva 13. Koodimuutos Azure DevOps -palvelussa

8 Yhteenveto

Työn tarkoituksena oli tutkia ja kehittää esimerkkiratkaisu, miten Jira-palvelun tehtävänäkymään saadaan linkit koodimuutoksiin ja osoittamaan ne oikeaan tehtävään. Lisäksi tutkittiin kaupallisten ratkaisujen hyötyjä ja haittoja.

Esimerkkiratkaisu on onnistunut, vaikka kehitettävää jäi tietojen tallennuksen osalta. Eli jatkokehityksenä olisi hyvä muuttaa toteutusta, miten tiedon tallennetaan. Tämä toki muuttaa lisäksi hieman tietojen haku prosessia tehtävänäkymään. Lisäksi käyttöliittymään voisi koodimuutoksen viestistä tehdä linkin. Esimerkkiratkaisu kuitenkin toimii, ja jatkokehitys ei vaadi huomattavia muutoksia koodiin.

Kaupallisten ratkaisujen osalta, jos käyttäjämäärät ovat pieniä, kustannukset eivät olleet kovinkaan suuria kummankaan tutkinnan kohteena olleen laajennuksen kohdalla, mutta nousivat tuhansiin euroihin käyttäjämäärien kasvaessa. Kuitenkin esimerkkiratkaisun kaltaisen laajennuksen pystyy kehittämään kohtuullisessa ajassa. Voidaan arvioida, että käyttäjämäärien kasvaessa voisi olla edullisempaa yritykselle kehittää oma laajennus kaupallisen ratkaisun sijaan.

Lähteet

1. Soni M. Hands-on Azure DevOps. First Edition ed. India: BPB Publications; 2020.

2. Bass L, Weber I, Zhu L. DevOps A Software Architect's Perspective. : Addison-Wesley; 2015.

3. Atlassian. What is Continuous Integration? Saatavilla: https://www.atlassian.com/continuous-delivery/continuous-integration. Viitattu 24.2., 2021.

4. Atlassian. What is Git. Saatavilla: https://www.atlassian.com/git/tutorials/what-is-git. Viitattu 24.2., 2021.

5. Atlassian. What is Jira used for? Saatavilla: https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for. Viitattu 24.2., 2021.

6. Atlassian. What is an issue? Saatavilla: https://support.atlassian.com/jira-software-cloud/docs/what-is-an-issue/. Viitattu 5.4., 2021.

7. Microsoft. What is Azure DevOps? Saatavilla: https://docs.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops. Viitattu 24.2., 2021.

8. Microsoft. Webhooks. 2020; Saatavilla: https://docs.microsoft.com/en-us/azure/devops/service-hooks/services/webhooks?view=azure-devops. Viitattu 6.5., 2021.

9. Atlassian. Install Forge on Windows. Saatavilla: https://developer.atlassian.com/platform/forge/installing-forge-on-windows/. Viitattu 6.5., 2021.

10. Microsoft. What is the Windows Subsystem for Linux? Saatavilla: https://docs.microsoft.com/en-us/windows/wsl/about. Viitattu 2.3., 2021.

11. Docker Inc. Docker overview. Saatavilla: https://docs.docker.com/get-started/overview/. Viitattu 1.3., 2021.

12. Anicas M. An Introduction to OAuth 2. 2014; Saatavilla: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2. Viitattu 22.3., 2021.

13. OAuth 2.0. Saatavilla: https://oauth.net/2/. Viitattu 1.3., 2021.

14. auth0. Token Based Authentication. Saatavilla: https://auth0.com/learn/token-based-authentication-made-easy/. Viitattu 22.3., 2021.

15. Internet Engineering Task Force. The 'Basic' HTTP Authentication Scheme. 2015; Saatavilla: https://tools.ietf.org/html/rfc7617. Viitattu 22.3., 2021.

16. Atlassian. About Jira Cloud platform. Saatavilla: https://developer.atlassian.com/cloud/jira/platform/. Viitattu 27.2., 2021.

17. Atlassian. Introducing Forge, a new way to build and run apps for the Atlassian cloud. Saatavilla: https://www.atlassian.com/blog/announcements/introducing-forge. Viitattu 24.3., 2021.

18. Atlassian. Forge Runtime. Saatavilla: https://developer.atlassian.com/platform/forge/runtime-reference/. Viitattu 7.4., 2021.

19. Atlassian. About Forge. Available at: https://developer.atlassian.com/platform/forge/. Viitattu 24.3., 2021.

20. Atlassian. Forge deprecation policy. Saatavilla: https://developer.atlassian.com/platform/forge/deprecation-policy/. Viitattu 25.3., 2021.

21. npm. Forge CLI. Saatavilla: https://www.npmjs.com/package/@forge/cli. Viitattu 25.3., 2021.

22. Atlassian. Manifest. Saatavilla: https://developer.atlassian.com/platform/forge/manifest-reference/. Viitattu 24.3., 2021.

23. Atlassian. Web trigger events (beta). Saatavilla: https://developer.atlassian.com/platform/forge/events-reference/web-trigger/. Viitattu 1.3., 2021.

24. Atlassian. UI kit (beta). Saatavilla: https://developer.atlassian.com/platform/forge/ui-kit/. Viitattu 6.5., 2021.

25. Atlassian. Connect frameworks and tools. Saatavilla: https://developer.atlassian.com/cloud/jira/platform/connect-frameworks-and-tools/. Viitattu 6.4., 2021.

26. Atlassian. REST API. Saatavilla: https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/. Viitattu 7.4., 2021.

27. Mage Software. Azure Git Listener for Jira. Saatavilla: https://marketplace.atlassian.com/apps/1221787/azure-git-listener-for-jira?hosting=cloud&tab=overview. Viitattu 1.3., 2021.

28. Mage Software. Azure Git Listener for Jira Documentation. Saatavilla: https://www.mage-software.com/documentation. Viitattu 1.3., 2021.

29. Mage Software. Azure Git Listener for Jira Cloud Pricing. Saatavilla: https://marketplace.atlassian.com/apps/1221787/azure-git-listener-for-jira?hosting=cloud&tab=pricing. Viitattu 1.3., 2021.

30. Suomen Pankki. Valuuttakurssit. Saatavilla: https://www.suomenpankki.fi/fi/Tilastot/valuuttakurssit/. Viitattu 1.3., 2021.

31. BigBrassBand. Git Integration for Jira. Saatavilla: https://marketplace.atlassian.com/apps/4984/git-integration-for-jira?hosting=cloud&tab=overview. Viitattu 1.3., 2021.

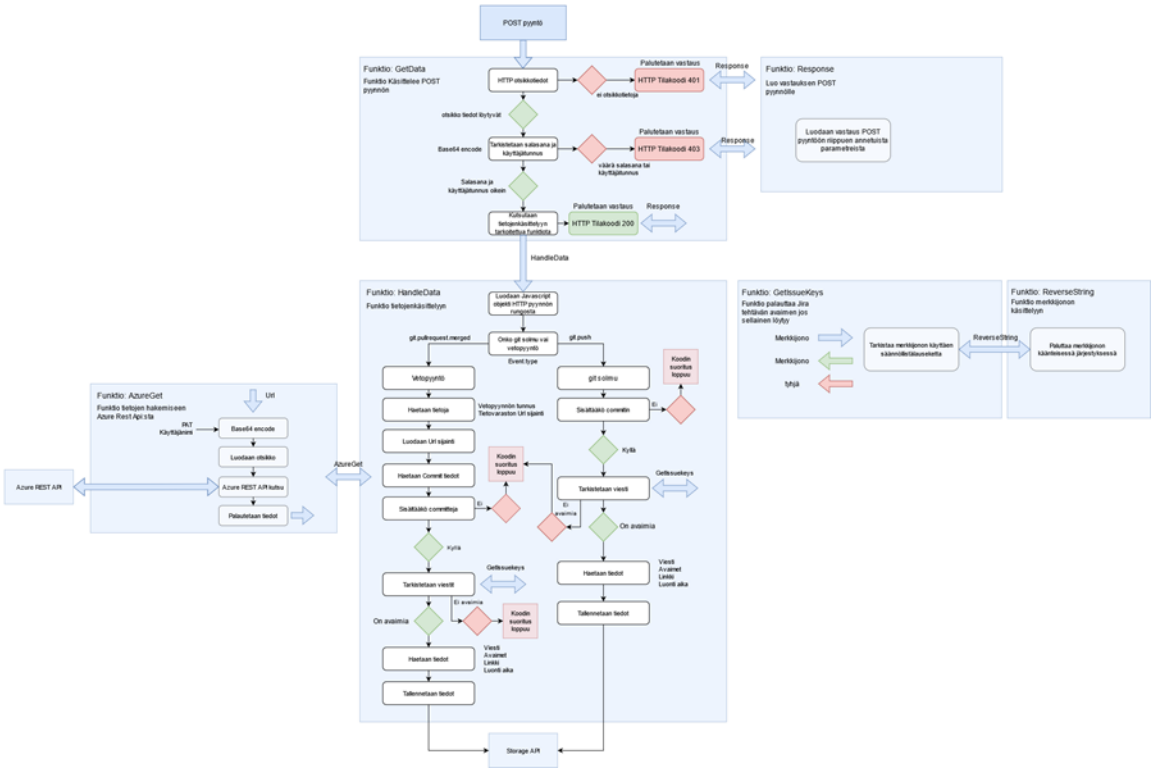
32. BigBrassBand. Git Integration for Jira Documentation. Saatavilla: https://bigbrassband.com/documentation.html. Viitattu 1.3.,2021.

33. BigBrassBand. Git Integration for Jira Cloud Pricing. Saatavilla: https://marketplace.atlassian.com/apps/4984/git-integration-for-jira?hosting=cloud&tab=pricing. Viitattu 1.3., 2021.

34. G__Sylvie_Davies__bit-booster_com_. Regex pattern to match JIRA issue key. Saatavilla at: https://community.atlassian.com/t5/Bitbucket-questions/Regex-pattern-to-match-JIRA-issue-key/qaq-p/233319. Viitattu 26.5., 2021.

35. Atlassian. Query. Saatavilla at: https://developer.atlassian.com/platform/forge/runtime-reference/storage-api-query/. Viitattu 6.5., 2021.

Liitteet


POST pyyntö
Funktio: GetData
Funktio Käsittelee POST pyynnön
HTTP otsikkotiedot
Palautetaan vastaus
HTTP Tilakoodi 401
Response
ei otsikkotietoja
Funktio: Response
Luo vastauksen POST pyynnölle
otsikko tiedot löytyvät
Base64 encode
Tarkistetaan salasana ja käyttäjätunnus
Palautetaan vastaus
HTTP Tilakoodi 403
Response
väärä salasana tai käyttäjätunnus
Salasana ja käyttäjätunnus oikein
Palautetaan vastaus
HTTP Tilakoodi 200
Response
HandleData
Funktio: HandleData
Funktio tietojenkäsittelyyn
Event type
git pullrequest.merged
git push
Vetopyyntö
Hae tietoja
Luodaan Uri sijainti
Haetaan Commit tiedot
Sisältääkö committeja
Kyllä
Ei
Tarkistetaan viestit
Ei avaimia
On avaimia
Haetaan tiedot
Tallennetaan tiedot
Storage API
Funktio: AzureGet
Funktio tietojen hakemiseen Azure Rest Api:sta
Url
PAT Käyttäjänimi
Base64 encode
Luodaan otsikko
Azure REST API kutsu
Palautetaan tiedot
Azure REST API
AzureGet
Funktio: GettissueKeys
Funktio palauttaa Jira tehtävän avaimen jos sellainen löytyy
Merkkijono
Merkkijono
tyhjä
Tarkistaa merkkijonon käyttäen käännöstaulusetelta
ReverseString
Funktio: ReverseString
Funktio merkkijonon käsittelyyn
Palauttaa merkkijonon käänteisessä järjestyksessä