Haaga-Helia
ammattikorkeakoulu Oy

Visuaalisen regressiotestaustyökalun kehittäminen ja käyttömahdollisuudet

Mikko Nirkko

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2021

Tiivistelmä

Haaga-Helia ammattikorkeakoulu Oy

Tekijä(t)
Mikko Nirkko

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Raportin/Opinnäytetyön nimi
Visuaalisen regressiotestaustyökalun kehittäminen ja käyttömahdollisuudet

Sivu- ja liitesivumäärä
41+4

Opinnäytetyön tarkoituksena on tutkia visuaalisen regressiotestauksen käytettävyyttä toimeksiantajan jo olemassa olevien testaustoimenpiteiden tueksi. Tätä tarkoitusta varten kehitettiin verkkosovellus, jolla visuaalista regressiotestausta voi suorittaa selaimessa ilman tarvetta testausohjelmistojen asentamiselle.

Kuvakaappausten vertailuun perustuvaa sovellusta ei voida käyttää funktionaaliseen testaamiseen, eikä se sovellu myöskään testaamiseen sellaisten sovelluspäivitysten yhteydessä, jotka muuttavat tarkoituksellisesti sovelluksen tai sen sisällön ulkoasua. Kehitettävän työkalun ensisijainen käyttötarkoitus sen sijaan on testata regressiota sellaisten päivitysten yhteydessä, joiden mukana ei tule tarkoituksellisia muutoksia palvelun ulkoasuun. Yleisin esimerkki tällaisista päivityksistä on tietoturvapäivitykset palveluun, sen alustaan tai johonkin sen komponenteista.

Opinnäytetyötä varten kehitetty ART-Tool -niminen testaustyökalu toimii ottamalla kuvakaappauksia verkkosivustosta ennen ja jälkeen sivustoon tehtäviä versiopäivityksiä. Vanhan ja uuden version kuvakaappauksia verrataan toisiinsa, ja liialti toisistaan eroavista kuvapareista ilmoitetaan käyttäjälle, jotta tämä voi tarkistaa, mitkä elementit ovat muuttuneet, ja johtuuko muutokset regressiosta.

ART-Tool sisältää Pythonilla kehitetyn selainautomaatio-ohjelman, joka kerää kuvakaappaukset ja suorittaa niiden vertailun. Työkalua käytetään Spring-pohjaisella verkkosovelluksella.

Julkaisunsa jälkeen ART-Tool havaittiin käyttäjäkokemusten perusteella sen verran hyödylliseksi toimeksiantajan tarpeisiin, että se on otettu vakituiseksi osaksi regressiotestauksen toimenpiteitä versiopäivitysten yhteydessä, toiminnallisen testaamisen tueksi.

Asiasanat
Ohjelmistokehitys, Laadunvarmistus, Regressio, Verkkosovellukset

Sisällys

1 Johdanto

1.1 Toimeksianto ja lähtötilanne

Toimeksiantajana toimii Exove Oy. Exove toteuttaa ohjelmistoratkaisuja ja muita digitaalisia palveluja sekä julkisen, että yksityisen sektorin yrityksille. (Exove, 2021)

Toimeksiantoni perustui tarpeeseen luoda uusia tapoja testata regressiota automaattisesti, manuaalisen testauksen tukena. Tavoitteena oli uudistaa ja laajentaa nykyisiä testausprosesseja kasvavaan testaamisen tarpeeseen.

Lähestymiskulmaksi valittiin kehittää tätä varten oma testaustyökalu itse. Visuaalista regressiota oli hieman testattu Exovella jo joitakin vuosia aiemmin käyttämällä mm. erinäisiä skriptejä toiminnallisuuden toteuttamiseen. Näitä jo olemassa olevia ratkaisuja ajateltiin voitavan käyttää uuden työkalun pohjana. Kävi kuitenkin ilmi, että skripteissä käytetyt kirjastot ja metodit olivat sittemmin vanhentuneita, eikä niiden käyttäminen pohjana olisi juuri edistänyt kehitystyötä.

Ennen opinnäytetyön toimeksiantona tilatun työkalun luomista olin perehtynyt visuaaliseen regressiotestaamiseen jo hieman aiemmin, pohjana edellä mainitut skriptit. Kehitin vuonna 2019 ensimmäisen, raa'an prototyypin visuaaliseen regressiotestaamiseen. Tämän prototyypin perusteella voitiin todeta, että sen jatkotyöstäminen itsenäiseksi ohjelmistoksi voisi olla kannattavaa, käyttäen prototyyppiä pohjana. Prototyypin kehitysprosessi ja siinä havaitut haasteet olivat lopulta suuressa asemassa uuden työkalun vaatimusmäärittelyssä. Lisää prototyypistä luvussa 3.1.

Valmiita, jo olemassa olevia visuaalisen regression testaustyökaluja on useita (Tästä lisää luvussa 2.4). Tämä tiedostaen valittiin toimeksiannoksi silti kehittää täysin oma työkalu. Varta vasten kehitetyssä ohjelmassa voidaan varmistua, että kaikki ohjelman osat palvelevat toimeksiantajan tarvetta mahdollisimman tehokkaasti, kehitysprosessi on toimeksiantajalle edullinen toteuttaa ja ylläpitää ilman ylimääräisiä kuluja, ja edistää tekijänsä henkilökohtaista kehitystä ohjelmistokehittäjänä ja testaajana.

1.2 Raportissa käytettyä termistöä

Ajo
ART-Toolin yksittäinen kokonainen testaustoimenpide. Joko verrokkikuvakaappausten ottaminen, tai testattavien kuvien kerääminen + kuvaparien vertailu.

ART-Tool
ART-Tool, eli Automated Regression Testing Tool on opinnäytetyön toimeksiantona kehitetty testaustyökalu. Myöhemmin käytetään myös nimityksiä työkalu, sekä palvelu.

Binääritiedosto
Tiedosto, joka sisältää muuta, kuin tekstimuotoista dataa. (The Linux Information Project, 2006)

CD / continuous deployment
Automatisoitu uuden järjestelmäversion asentaminen haluttuun ajoympäristöön. (Verona, 2016)

CMS
Sisällönhallintajärjestelmä

Crawler
Opinnäytetyötä varten Python-ohjelmointikielellä kehitetty ohjelma, jolla suoritetaan ART-Toolin selaintoiminnot, sekä kuvakaappausten vertailu.

CSS
Cascading Style Sheets, yksinkertainen mekanismi tyylien, kuten fonttien, värien ja välistyksen lisäämiseen verkkodokumentteihin. (W3 Consortium, 2020)

DevOps
Toimintaperiaate, jossa suositaan ohjelmistokehityksen eri työtehtävien, vaiheiden ja osa-alueiden yhdentämistä, sekä niiden tekijöiden runsasta yhteistyötä. (Verona, 2016)

DBMS / Database Management System
Kokoelma toisiinsa liittyvää dataa, ja ohjelmisto, jolla sitä hallinnoidaan. DBMS:n ensisijainen tavoite on tarjota tehokas ja helppokäyttöinen tapa varastoida ja hallinnoida dataa. (Vidhya, 2016. 1.1)

Dashboard
Opinnäytetyötä varten kehitetty verkkosovellus, joka toimii ART-Toolin käyttöliittymänä.

Haavoittuvuus (Vulnerability, threat)
Vika tai heikkous järjestelmän suunnittelussa, implementaatiossa, tai sen käytössä ja hallinnassa, jota voidaan väärinkäyttää ohjelman tietoturvajärjestelyjen mitätöimiseksi. (Mueller, 2015)

Git
Avoimen lähdekoodin versionhallintajärjestelmä ohjelmistojen lähdekoodin hallintaan. (Git, 2021)

Git-repositorio
Yksittäisen projektin lähdekoodin varastointiin ja hallintaan osoitettu hakemisto git-järjestelmässä.

HTML
HTML, eli Hypertext Markup Language, on merkintäkieli, jota käytetään verkkosivujen rakenteen ja sisällön jäsentelemiseen. (Mozilla Developer Network, 2020)

ImageMagick
Avoimen lähdekoodin kuvankäsittelyohjelma. (ImageMagick, 2021)

Java
Alun perin Sun Microsystemsin julkaisema ohjelmointikieli ja ohjelmistoalusta.
(Oracle, 2021)

Jenkins
Avoimen lähdekoodin automaatiopalvelin, jolla voidaan automatisoida ohjelmiston rakentamiseen, testaamiseen tai asentamiseen liittyviä tehtäviä. (Jenkins User Documentation, 2021)

Nginx
Avoimen lähdekoodin ohjelmisto, jota voidaan käyttää mm. verkkopalvelimena, välimuistina, kuormantasaajana tai median suoratoistotyökaluna.
(Nginx, 2021)

NoSQL
Yleinen ilmaus tietokannoille, jotka eivät noudata relaatiotietokantajärjestelmien toimintaperiaatetta. (Tiwari, 2011)

Refaktorointi
Ohjelman lähdekoodin sisäisen rakenteen parantaminen alkuperäisen toiminnallisuuden säilyttäen. (Agile Alliance, 2020

Relaatiomalli
Malli, jossa tietokanta on kokoelma relaatioita. Relaatio itsessään on kokoelma (table) arvoja. Jokainen tällaisen tablen datarivi on oma kokoelmansa toisiinsa liittyviä data-arvoja.
(Guru99, 2021)

Relaatiotietokanta (tämän projektin kontekstissa)
Ohjelmisto, jolla voidaan varastoida ja hallinnoida dataa relaatiomallin mukaisesti.
(Date, 2008)

Robot Framework
Geneerinen avoimen lähdekoodin automaatio-ohjelmistokehys. Robot Framework on tehty ja toimii Pythonilla, mutta sen voi integroida lähes mihin tahansa alustaan ja sitä käytetään testiautomaatiossa sekä ohjelmistorobotiikassa. (Robot Framework, 2020.)

Ohjelmistokehys
Osittain abstraktiksi jätettyjä ohjelmistorunkoja, joita eri tavoin täydentämällä saadaan rakennettua kokonaisia uusia sovelluksia tai sovelluksen osia. (Helsingin Yliopisto, 2017)

PHP
Avoimen lähdekoodin skriptauskieli, joka sopii erityisesti verkkopalvelujen kehitykseen.
(php.net, 2021)

Python
Korkean tason ohjelmointikieli, joka sopii nopeaan ja useampia teknologioita yhdistelevään ohjelmistokehitykseen. (python.org, 2021)

Rajapinta, API
Järjestelmään määritelty tapa, jolla sen dataa tai ominaisuuksia voidaan operoida ilman tarvetta ymmärtää järjestelmän teknistä toteutusta yksityiskohtaisesti.
(Preibisch, 2018)

Sisällönhallintajärjestelmä / CMS (tämän projektin kontekstissa)
Ohjelmisto, jonka avulla voidaan luoda ja muokata verkkopalvelun sisältöä. Tämän lisäksi sisällönhallintajärjestelmällä hallitaan mm. sisällön talletukseen ja julkaisuun liittyvät toimenpiteet. Eri toiminnot voidaan olla jaoteltu erillisiin komponentteihin, mutta selkeyden vuoksi puhutaan usein yhtenäisestä sisällönhallintajärjestelmästä. (Barker 2016, 1)

Sisältö (tämän projektin kontekstissa)
Verkkopalveluun toteutettu tietyn aiheen tai aiheet käsittävä sivu. Esimerkiksi artikkeli verkkolehdessä tai blogikirjoitus.

Skripti
Lista komennoista, jotka voidaan ajaa ohjelmassa tai muussa ajoympäristössä. (TechTerms, 2021)

SQL
Standardoitu kieli, jonka avulla voidaan tarkastella ja muuttaa tietokannan sisältämää dataa. (Guru99)

Tietokanta (tämän projektin kontekstissa)
Kokoelma dataa, ja varsinkin sen hallinnointiin käytettävä ohjelmisto (ks. DBMS)

Verkkosovellus
Ohjelma, jota suoritetaan verkkopalvelimella. Verkkosovellusta käytetään verkkoselaimen kautta. (TechTerms, 2021)

WCAG 2.1
Web Content Accessibility Guidelines, versio 2.1. Kokoelma ohjeita ja kriteerejä (luokiteltu vaatimustasoihin A, AA & AAA), joiden avulla verkkopalvelu voidaan toteuttaa saavutettavaksi, eli mahdolliseksi ymmärtää ja operoida fyysisistä tai kognitiivisista haasteista huolimatta. (W3C Consortium, 2021)

2 Projektin kannalta oleellisia konsepteja (tietoperusta)

2.1 Regressiotestaus ohjelmistokehityksessä

Regressiotestauksella pyritään selvittämään, onko ohjelmistoon toteutetut muutokset aiheuttaneet tai tuoneet esiin vikoja ohjelmiston niissä osissa, joita ei ole tarkoituksenmukaisesti muutettu (ISTQB, 2018).

Regressiotestauksessa luodaan kokoelma testejä, joiden tarkoituksena on osoittaa, että ohjelmisto toimii odotetulla tavalla. Samat testit suoritetaan lukuisia kertoja kehitysprosessin aikana, aina kun ohjelmistoon tehdään muutoksia. Usein tämä testien toistuvuus tekee niistä sopivia kohteita automaatiolle (Hambling, Morgan, Samaroo, 2010. 2.4).

Patton (2005, 4.1.1) jakaa testauksen tyypit korkeimmalla tasolla kahteen kategoriaan seuraavasti: Black Box -testaus, ja White Box -testaus. Black Box -testaus tarkoittaa sitä, että testausta suorittava henkilö on tietoinen siitä, millaista lopputulosta tulisi odottaa testattavan ohjelmiston toiminnasta. Testaaja ei voi sen sijaan tarkastella testattavan ohjelmiston koodia tai muita sisäisiä komponentteja selvittääkseen taustalla toimivaa logiikkaa, vaan toimivuuden arviointi tapahtuu puhtaasti lähtötilanteen ja lopputuloksen perusteella. Vastaavasti White Box -testauksessa testausta suorittava henkilö pystyy tarkastelemaan testattavan ohjelmiston rakennetta ja toimintalogiikkaa. Tästä saatavalla informaatiolla voidaan tehdä päätellä todennäköisiä ongelmakohtia etukäteen, ja hyödyntää sitä testauksen suunnittelussa.

Testaamisen tehokkuuden ja hallittavuuden parantamiseen on kehitetty lukuisia testaustyökaluja eri testaustoimenpiteisiin. Testaustyökalut voivat olla testattavan ohjelmiston kehittäjien tai testaajien itse kehittämiä. Osa työkaluista sen sijaan on ulkopuolisen kehittäjän tai yrityksen toteuttamia, joiden käyttöön saatetaan myydä lisenssejä. Työkalun käyttötarkoitus ei itsessään määrittele, onko se tyypillisesti kehittäjien itse tekemä, vai valmis kolmannen osapuolen ratkaisu (Hambling ym, 2010. 6.1).

Tyypillinen verkkosovellus voidaan jakaa toiminnallisuuksiensa puolesta kolmeen kerrokseen. Ensimmäinen kerros, Presentation layer, sisältää sovelluksen visuaalisen esityksen ja käyttöliittymän. Esimerkkejä Presentation layer -kerrokseen kohdistuvista testauskohteista on käyttöliittymäelementtien asettuminen toisiinsa nähden, fontit ja värit sekä yhteensopivuus eri selainten kanssa. Tyypillisesti näiden arviointiin tarvitaan ihminen suorittamaan lopullisen arvion, joka nostaa reilusti työmäärävaatimuksia (Myers, Badgett, Sandler 2012. 201–203).

Toinen kerros, Business layer, sisältää sovelluksen toiminnallisuudet, kuten esimerkiksi käyttäjien tunnistautumisen ja käyttöön liittyvät toiminnot ja transaktiot. Näiden testaamista varten kannattaa laatia yksityiskohtaiset testisuunnitelmat ja -metodit. Esimerkiksi testatessa verkkokauppaa, kuuluu testaustoimenpiteisiin valikoiman selaaminen, ostoskorin täyttäminen ja lopullisen maksutoimenpiteen suorittaminen. Testattavien askelten tunnistaminen voi olla haastavaa riippuen testattavasta järjestelmästä, ja tarkkaa suunnitelmaa voi harvoin kierrättää toisesta projektista (Myers, 2012. 205).

Kolmas kerros, Data layer, sisältää ohjelmiston käyttämän, ja / tai käyttäjältään keräämän datan. Tyypillisesti tämä tarkoittaa sovelluksen dataa hallinnoivaa ja varastoivaa tietokantajärjestelmää. Testattavia asioita ovat mm. tietokannan nopeus, datan eheys ja palautumiskyky virhetilasta (Myers, 2012. 208–209).

2.2 Sisällönhallintajärjestelmän ylläpidossa tehtävät versiopäivitykset ja niiden seurauksena tyypillisesti havaittavat regressiot

Sisällönhallintajärjestelmän tarkoitus on luoda ja hallita suuria määriä verkkopohjaista sisältöä niin, ettei sivustojen sivujen HMTL-rakennetta tarvitse luoda käsin. Tämä mahdollistaa useiden sisällönsyöttäjien toiminnan saman verkkopalvelun alla ilman, että sivuston ulkoasu ja brändi pirstaloituu (Bradford, Ndubisi, Nelson 2006, 5).

Toimeksiantajan tyypillisessä ohjelmistoprojektissa ohjelmistoon liittyvät työtehtävät eivät pääty palvelun julkaisuhetkeen, vaan projekti siirtyy tukivaiheeseen. Sisällönhallintajärjestelmän tapauksessa tämä tarkoittaa mm. mahdollisten bugien korjaamista, uusien tai olemassa olevien toiminnallisuuksien jatkokehitystä ja järjestelmän sekä sen komponenttien, eli kolmannen osapuolen toteuttamien toiminnallisuuksien ylläpitoa. Komponenttien ylläpitäminen merkitsee useimmiten niihin kohdistuvia versiopäivityksiä. Yleisimmät motivaattorit versiopäivityksiin liittyvät yleensä joko siihen, että kyseisten komponenttien ylläpito niiden alkuperäisten kehittäjien toimesta päättyy nykyiselle versiolle, tai komponentissa on havaittu tietoturva-aukko, jonka uusi versiopäivitys korjaa. Näissä tapauksissa on välttämätöntä päivittää järjestelmä tai sen komponentti välittömästi tuettuun ja tietoturvalliseen versioon.

Versiopäivitysten yhteydessä ilmenee silloin tällöin regressioita. Yleisin regression aiheuttaja päivitystoimenpiteen jälkeen on se, että päivitetty komponentti tai sen osa ei ole enää täysin yhteensopiva sovelluksen muun koodin kanssa. Käyttöliittymässä tämä ilmenee usein ulkoasultaan virheellisinä, tai kokonaan puuttuvina käyttöliittymäelementteinä.

Yleensä visuaaliset viat johtuvat suoraan järjestelmän ulkoasua määrittelevien osien virheistä, mutta virhe voi ilmetä myös muuttuneena sisältönä tai toiminnon lopputuloksena, jonka on aiheuttanut varsinaisesti ulkoasukoodiin liittymätön osuus.

2.3 Visuaalinen regressiotestaus

Nimensä mukaisesti visuaalinen regressio on tahaton muutos ohjelmiston visuaalisessa ilmeessä siihen tehtyjen muutosten jälkeen. Verkkosovellusten tapauksessa tämä tarkoittaa käyttöliittymän visuaalisten elementtien asetteluun tai graafiseen ulkoasuun liittyviä seikkoja. Visuaalinen elementti voi olla esimerkiksi kuva, sivu tai näiden yksittäinen osa. Ulkoasun validoinnissa huomioonotettavia asioita ovat mm. fontit, värit, koko ja sijainti (Software Testing Help, 2021).

Yksinkertaisimmillaan visuaalinen regressiotestaaminen on käyttöliittymän silmämääräistä arviointia ohjelmistoon kohdistettujen muutosten jälkeen. Manuaalisen testaamisen etu on se, että ihminen voi tarkastella käyttöliittymää subjektiivisesti, ja havaita todennäköisiä ongelmakohtia käyttämällä ohjelmistoa samankaltaisesti, kuin ohjelmiston varsinaiset käyttäjät. Testausmetodeja voi olla esimerkiksi käyttöliittymästä otettujen kuvakaappausten vertailu. Ensimmäinen kuvakaappaus otetaan käyttöliittymästä sen ollessa halutussa tilassa. Tätä kuvakaappausta käytetään vertailukohtana testauksessa. Muutosten tekemisen jälkeen otetaan uusi kuvakaappaus, jota verrataan ensimmäiseen kuvakaappaukseen visuaalisten muutosten kartoittamiseksi (Software Testing Help, 2021).

Visuaalinen regressiotestaus kuitenkin voidaan automatisoida sitä varten kehitetyillä työkaluilla. Lähes poikkeuksetta nämä työkalut perustuvat kuvakaappausten vertailuun. Työkalut voivat olla joko ohjelmistopaketteja tai kirjastoja, joita kehittäjät integroivat osaksi ohjelmiston koodia tai kehitysympäristöä, tai sellaisenaan valmiita palveluja (Skrypny, 2019).

Automaattisen visuaalisen regressiotestauksen hyödyt liittyvät sen tehokkuuteen. Automaattisia testejä voidaan toistaa nopeasti, joka soveltuu hyvin nopeatempoiseen käyttöliittymäkehitykseen. Toisin kuin ihminen, automaatio pystyy arvioimaan kuvakaappausten välisiä muutoksia pikselintarkasti, ja havaita hienovaraisempia muutoksia, jotka saattavat jäädä ihmiseltä huomaamatta. Automaattiset testit pystyvät muodostamaan havaittuja eroavaisuuksia esitteleviä testausraportteja testaamisen jälkeen, joita kehitystiimin on helppo tulkita (Software testing help. 2021). Kuvakaappauksia sisältävän raportin tulkitseminen voi myös usein olla yksinkertaisempaa, kuin mahdollisten ongelmakohtien havainnointi suoraan ohjelmiston koodista (Murdoch, 2019).

Automaattisen visuaalisen regressiotestauksen puutteet liittyvät usein sen kyvyttömyyteen erotella muutosten tarkoitusta. Jos kriteerinä on ainoastaan kuvissa keskenään samassa paikassa sijaitsevien pikselien väriarvot, syntyy tilanteita, joissa tarkoituksenmukainen eroavaisuus tulkitaan regressioksi. Tällaisia tilanteita on esimerkiksi animoidut tai muuten interaktiiviset elementit tai muutokset sivun informatiivisessa sisällössä. Yksi tapa pyrkiä olemattoman regression havainnoinnin vähentämiseen on lisätä sallittujen eroavien pikselien suhdetta muuttumattomiin pikseleihin. Tämä metodi ei kuitenkaan kerro koko totuutta, koska ulkoasultaan ja sisällön määrältään pelkistetymmät palvelut sisältävät prosentuaalisesti vähemmän kohtia, joissa todennäköisesti havaitaan muuttuneita pikseleitä (Taylor, 2019).

Eroavaisuuksia havaitessa täytyy ihmisen arvioida automaation tuottama raportti. Tämän perusteella voidaan arvioida, oliko kyseessä korjauksia vaativa virhetila ohjelmiston toiminnassa, vai virheellisesti tulkittu testi. Jos havaitut muutokset ovat tarkoituksenmukaisia, voidaan vertailukohtana käytettävät kuvakaappaukset päivittää uutena perustilana (Software Testing Help, 2021).

2.4 Olemassa olevia visuaalisen regression testaustyökaluja

Visuaalisen regression testaustyökalut jakautuvat karkeasti ohjelmistopaketeiksi, jotka asennetaan ja määritetään lähdekoodin tasolla, sekä valmiiksi, sellaisenaan käytettäviksi ulkopuolisiksi palveluiksi. Luvuissa 2.4.1 & 2.4.2 on valikoitu esimerkkejä näistä palveluista.

2.4.1 Ohjelmistopakettina tarjottavia

Kaikille mainituille ohjelmistopaketeille yhteistä on niiden perimmäinen toimintaperiaate: Visuaalisen regression tunnistamiseksi kerätään ja vertaillaan kuvakaappauksia ohjelmistosta sen eri tiloissa. Kaikki palvelut ovat avoimen lähdekoodin ohjelmistoja.

Wraith

Wraith on toteutettu Ruby-ohjelmointikielellä. Wraith tarjoaa saman näkymän kahden eri ajankohdan vertailun lisäksi mahdollisuuden suorittaa vertailua kahden eri ympäristön, kuten saman palvelun tuotantoympäristön ja testausympäristön välillä. Wraith vertailee kuvakaappauksia keskenään, laatii havainnekuvat havaituista eroavaisuuksista, ja kokoaa tulokset jäsenneltyyn raporttiin (BBC, 2019).

Needle

Python-ohjelmointikielellä toteutettu Needle tarjoaa kuvakaappausten vertailun lisäksi mahdollisuuden vertailla palvelun CSS-määrityksiä ja HTML-elementtien sijaintia toisiinsa nähden (Needle, 2021).

Gatling

Ruby-ohjelmointikielellä toteutettu Gatling mahdollistaa sivun elementtien vertailun erillisinä vertailuina koko sivun kerralla vertaamisen sijaan (Rotbart, 2013).

Aye Spy

Javascript-kielellä toteutettu Aye Spy tarjoaa lukuisia ominaisuuksia tavallisen vertailun lisäksi:

- Mobiililaitteiden emulointi
- Amazonin tarjoaman S3 -pilvitallennustilan integrointi kuvakaappausten säilömiseen.
- Sivulatauksen yhteydessä ajettavien skriptien luominen, jos palvelun käyttöliittymää täytyy operoida sopivan käyttöliittymän tilan saavuttamiseksi ennen kuvakaappauksen ottamista.
- Todennäköisten vääriä positiivisia aiheuttavien elementtien tunnistaminen ja näkyvistä poistaminen ennen kuvakaappauksen ottamista.

(News UK, 2020)

2.4.2 Esimerkkejä valmiista visuaalisen regression testaustyökaluista

Diffy

Diffy on visuaalisen regression testaamiseen käytettävä palvelu. Diffy tarjoaa käyttäjälle tavallisen kuvakaappausten pikselien vertailun lisäksi lukuisia lisäominaisuuksia. Diffyn käyttämä algoritmi eroavaisuuksien tunnistamiseen osaa havaita, milloin sisältö on lähinnä liikkunut hieman pystysuunnassa, joka vähentää väärien positiivisten määrää. Muita metodeja vertailun tarkkuuden parantamiseksi on mm. peittää tiedettävästi muuttuvia elementtejä, kuten mainosbannereita pois kuvakaappauksista, sekä ajaa omia skriptejä ennen kuvan ottamista halutun tilan saavuttamiseksi. Diffy on integroitavissa lukuisiin eri järjestelmiin, joista oleellisimmaksi on nostettu sisällönhallintajärjestelmät Drupal ja Wordpress. Diffy sisältää ilmaisen 14 päivän kokeiluvaiheen, mutta sen jälkeen sen käyttö on maksullista. Diffyn hinnoittelu on jaoteltu eri luokkiin, joissa ratkaiseva ero on testattavien sivujen yhteismäärä. Rajaamatonta käyttöä tarjotaan enterprise-tason yrityksille, mutta näitä ratkaisuja ei ole suoraan hinnoiteltu, vaan jokainen sopimus neuvotellaan tarjouksen perusteella erikseen (Diffy, 2021).

Percy

Browserstack-yrityksen tarjoama Percy on visuaalisen regression havaitsemiseen luotu testauspalvelu. Kuvakaappausvertailun tueksi Percy tarjoaa mahdollisuutta käyttää eri selaimia ja laitteita testaamisen monipuolisuuden parantamiseksi, hyödyntäen Browserstackin virtuaalisia laiteresursseja. Kuvakaappausten luotettavuuden parantamiseksi sivua voidaan vertailla eri leveyksillä ja animaatioita voidaan pysäyttää. Percy on integroitavissa lukuisiin eri kehitysympäristöihin ja on yhteensopiva eri verkkoteknologioiden kanssa. Percy mainostaa myös olevansa GDPR-kelpoinen palvelu. Toisin kuin Diffy, Percy tarjoaa ilmaisen version palvelustaan ilman kokeiluaikaa. Kuukausittaisten kuvakaappausmäärien ja joidenkin ominaisuuksien käyttöä on rajoitettu ilmaisessa versiossa. Ilmaisen version 5000 kuvakaappausta kuukaudessa voidaan korottaa 25000:en Professional-paketissa. Myös enterprise-tason ratkaisua tarjotaan, jonka hinta räätälöidään asiakkaan tarpeen mukaan (Browserstack, 2021).

Applitools Eyes

Applitoolsin kehittämä Applitools Eyes on visuaalisen regression havainnointiin luotu palvelu. Applitools mainitsee valtikseen sen, että se käyttää tekoälyä eroavaisuuksien paikantamiseen. Tekoälyn käyttäminen on Applitoolsin mukaan huomattavasti nopeampaa, kuin perinteinen toiminnallinen testaus tai visuaalinen testaus, varsinkin Applitoolsin tarjoamassa Ultra Fast -ympäristössä, joka nopeuttaa testausprosessin keston minuuteista sekunteihin. Applitools Eyes on integroitavissa lukuisiin eri kehitysympäristöihin, ja tarjoaa myös mahdollisuutta arvioita lähdekoodia käyttöliittymän lisäksi. Applitools tarjoaa Eyes-palvelustaan ilmaisen version, jolla 1 käyttäjä voi testata 100:a näkymää kuukaudessa. Tämän lisäksi löytyy vähemmän rajoitettuja maksullisia lisenssejä (Applitools, 2021).

3 Työkalun suunnittelu

3.1 Aiemman prototyypin kehitysprosessi ja siinä havaitut rajallisuudet

ART-Toolin pohjana toimi syksyllä 2019 kehitetty verkkosivustojen visuaalisen regressiotestaustyökalun ensimmäinen toimiva versio (myöh. prototyyppi).
Prototyyppi toimi seuraavasti:

- Testattavan sivuston osoite syötetään ohjelmalle.
- Ohjelma kerää sivuston näkymistä kuvakaappaukset, ja tallentaa ne myöhemmin suoritettavaa vertailua varten.
- Ohjelma suoritetaan uudestaan sen jälkeen, kun testattava sivusto on päivitetty.
- Uudet kuvakaappaukset kerätään, ja niitä verrataan vanhoihin
- Jos kuvissa havaitaan eroja, merkitään näkymä muuttuneeksi.
- Lopuksi luodaan jäsennelty raportti, johon kootaan kaikki muuttuneet näkymät tarkempaa tarkastelua varten.

Prototyyppi oli toteutettu ensisijaisesti Robot Framework -ohjelmistokehyksellä. Teknologiavalinta perustui pohjimmiltaan siihen, että monet työtehtäväni siihen aikaan sisälsivät selainautomaatiotestausta käyttäen Robot Frameworkia. Alustan ensisijainen käyttötarkoitus selainautomaatiossa vaikutti siten sopivimmalta myös visuaaliseen regressiotestaukseen. Aiempi kokemus robot-testien luomisessa nopeuttaisi työkalun kehitystyötä, koska jo saatua osaamista voi hyödyntää kehitystyön tehostamiseksi.

Robot Framework ei kuitenkaan riittänyt ainoaksi ohjelmointikieleksi projektin tarpeisiin. Koodin luettavuutta ja yksinkertaisuutta painottavaa ohjelmistokehystä ei ole suunniteltu monimutkaiseen ohjelmalogiikkaan, vaan lähinnä joko/tai -mallin yksinkertaisiin ehtoihin. Tämänkaltaisten ominaisuuksien lisääminen ei olisi ollut mahdotonta, mutta kehitettäessä kävi ilmi, kuinka nopeasti koodi alkoi muuttua hankalaselkoisemmaksi ja hitaammaksi ajaa.

Robot Frameworkin vakiokirjastoja voi laajentaa omilla Python-skripteillä. Tätä ominaisuutta hyödyntäen tehtiin monimutkaisempi logiikka ohjelmaan, kuten testattaviin sivustoihin liittyvän informaation tallennus, sivustokartan määrittely ja raporttien luonti. Testiajon kulku, asetusten määrittely ja selaintoiminnot luotiin Robot Framework-muotoisesti.

ImageMagick valikoitui kuvien vertailuun ja luontiin siksi, että se sisälsi halutut ominaisuudet, oli yksinkertainen käyttää, ja komentorivipohjaisuutensa takia se sisälsi hyvän yhteensopivuuden käytettäväksi eri ohjelmointikielillä. ImageMagickista olisi ollut saatavilla versio, jota voi käyttää suoraan Python-kirjastona, mutta sen ylläpito oli lakannut jo paljon aiemmin, eikä se ollut enää täysin yhteensopiva uudempien ImageMagick -versioiden kanssa.

Prototyypin ensimmäinen toimintamalli oli, että sitä ajettaisiin paikallisesti komentoriviltä. Tämä lähestymistapa vaatisi kuitenkin asennustoimenpiteitä uudelle käyttäjälle. Ohjelman lähdekoodin lataaminen ei olisi riittänyt itsessään, vaan käyttäjän täytyi myös asentaa Python ja sen lisäosat, kuten Robot Framework ja ImageMagick omalle tietokoneelleen, jotta ohjelman kaikki komponentit toimivat.

Ratkaisu oli, että ohjelma täytyisi tarjota käyttäjille verkossa palveluna, jolloin erillisiä asennuksia ei tarvittaisi. Ottaen huomioon ohjelman yksinkertainen rakenne ja mittakaava, ei olisi ollut järkevää pystyttää kokonaista palvelinta yksinomaan sen tarjoamiseen.
Sen sijaan päädyin käyttämään valmiiksi saatavilla olevaa Jenkins-palvelinta ohjelman tarjoamiseen, koska sinne oli määritelty automaattisesti ajettavaksi aiempia testiautomaatioprojekteja. Tällöin ei tarvinnut rakentaa muuta, kuin uusi Jenkins-projekti.

Jenkinsin toimintaperiaate ensisijaisesti jatkuvan integraation ja kehityksen työkaluna asetti muutamia haasteita. Ohjelman tilaa ei voinut tallentaa muistiin ajon jälkeen, vaan kaikki täytyi tallentaa tiedostoina projektille varattuun workspace-hakemistoon. Ohjelmalla ei ollut myöskään minkäänlaista tietokantaa, vaan kaikki ohjelman luoma ja käyttämä informaatio tallentui yksittäiseen binääritiedostoon, jolla oli jatkuva riski korruptoitua, jos ohjelma olisi kaatunut muokatessa sitä. Ohjelma kaatuili todella usein, joten riski tiedon häviämiseen oli todellinen. Tämän tietorakenteen tuoman relaation puutteen vuoksi HTML-raportit täytyi myös luoda joka ajolla uudestaan, ja ainoastaan viimeisimmän ajon raportti säilytettiin kerrallaan.

Ajojen määrittäminen ei ollut myöskään yksinkertaisimmasta päästä, koska asetusten määrittäminen täytyi tapahtua asettamalla ajoparametrejä Jenkinsin käyttöliittymässä, joka lopulta laajeni käyttäjäpalautteen perusteella vaikeaselkoiseksi, kun lisäasetuksia kehitettiin.

Jenkins-projektit on suunniteltu ajettaviksi yksittäinen ajo kerrallaan per projekti. Tämän takia myöskään regressiotestaus-prototyyppiä ei voinut ajaa useilla sivustoilla samanaikaisesti, vaan ajot menivät jonojärjestelmään, josta ne käynnistyivät yksi kerrallaan. Ohjelma oli myös Jenkinsin omien ohjelmistoversioiden armoilla. Ohjelman täytyi toimia niillä alustojen ja kirjastojen versioilla, jotka oli asennettu Jenkins-palvelimelle, koska niiden päivittäminen saattoi rikkoa muita projekteja, ja vastaavasti muutaman kerran kävi niin, että Jenkins-palvelimen versiopäivitykset rikkoivat prototyypin.

3.2 ART-Toolin rakenne ja vaatimukset prototyypin puutteet huomioon ottaen

3.2.1 Käytettävät ohjelmointikielet- ja alustat

Robot Frameworkin tilalle valikoitui Pythonin käyttäminen sellaisenaan, koska Robot Frameworkin tärkeimpiä etuja test case -pohjaisen selkokielisen skriptauksen ja automaattisen raportoinnin suhteen ei voitu hyödyntää prototyypissä.

Koska Robot Framework on rakennettu Pythonin päälle, tuntui loogisimmalta vaihtaa suoraan Pythoniin, jolloin myös robot-skriptien tulkitsemiseen ja Pythoniksi kääntämiseen käytettävä prosessointiaika voitiin käyttää suoraan ohjelmalogiikan suorittamiseen. Selainautomaation pohjana käytettiin automaatio-ohjelmistokehys Seleniumin Python-toteutusta. Tämän avulla voidaan hallita verkkoselaimen toimintaa suoraan Python-koodista käsin. Tätä kuvakaappauksia ottavaa ja vertailevaa kokonaisuutta kutsutaan tässä raportissa myöhemmin nimellä crawler tai Python-crawler.

Koska Jenkinsiä ei käytettäisi enää ART-Toolin operoimiseen, täytyi siihen valita muu tapa. Lopulta parhaimmaksi tavaksi valikoitui rakentaa yksinkertainen verkkokäyttöliittymä ajojen hallintaan ja tulosten tarkasteluun.

Verkkokäyttöliittymän hyötyjä ART-Toolia ajatellen ovat seuraavat:

- Järjestelmän ulkoasu ja operointi on täysin muokattavissa, ja siitä voidaan tehdä mahdollisimman helppokäyttöinen ensikertalaiselle.
- Dynaamisilla näkymillä on helpompi hahmottaa informaatiota testattavien sivujen asetuksista ja ajojen tuloksista.
- Meneillään olevan ajon tilaa voi seurata reaaliajassa.
- Kuvakaappausten laatua voi tarkkailla, jotta voidaan varmistaa vertailun onnistuminen etukäteen.
- Raportoinnin toteuttaminen osaksi palvelua tarjoaa mahdollisuuksia rikkaampaan raportointiin verrattuna staattisiin HTML-dokumentteihin.

Verkkokäyttöliittymän alustaksi valikoitui Java, käyttäen Spring Boot -ohjelmistokehystä, sekä Thymeleaf-ohjelmistokehystä dynaamiseen käyttöliittymän luontiin. Valitsin nämä teknologiat siksi, että niillä pystyi tuottamaan halutun kaltaisen verkkosovelluksen. Lisäksi minulla oli kertynyt niistä eniten kokemusta erinäisistä opiskeluihin liittyvistä projekteista, jolloin kehitykseen varatun ajan voisi käyttää tehokkaasti ilman suurta tarvetta uuden opettelulle.

ART-Toolia tulisi voida käyttää useiden käyttäjien toimesta samanaikaisesti. Tämän toteuttamiseen hyödynnetään threading-toimintoa Pythonissa. Jokainen ajo käynnistää uuden Python-prosessin, threadin, jolle annetaan tarvittavat tiedot siitä, mitä toimintoja sen

tulee suorittaa ja millä asetuksilla. Thread terminoituu, kun se on saanut suoritettua funktiot päätökseen. Threadien luonti tekee myös ohjelmasta toimintavarmemman, koska mahdolliset ennakoimattomat virhetilanteet suorituksessa voivat kaataa vain threadin, muttei itse ohjelmaa.

Kuvavertailun pohjana voidaan hyödyntää prototyypissä määriteltyjä ImageMagick-komentoja sellaisenaan, joka oli osasyy päätökseen käyttää ImageMagickia myös ART-Toolissa.

3.2.2 Tiedon tallentaminen

Metatiedon, kuten testattavan sivuston nimen, sivustokartan ja testiajossa käytettävien asetusten tallennusta varten tulisi käyttöön MySQL-tietokanta korvaamaan aiemmin käytetyn binääritiedoston. Tietokannan hyödyntäminen mahdollistaa seuraavat hyödyt verrattuna prototyypin toimintamalliin:

- Koko tietokanta ei voi korruptoitua yksittäisen virheellisen testiajon seurauksena.
- Dataa voi lukea mikä tahansa ohjelma, joka on yhteensopiva MySQL-tietokantojen kanssa.
- Data voidaan tallentaa käyttäjäautentikaation taakse, joka lisää järjestelmän tietoturvallisuutta.
- Testiajoista voidaan helpommin tallentaa historiatietoja useammalta, kuin vain viimeisimmältä ajolta.
- MySQL:n käyttö mahdollistaa datan monipuolisen tallentamisen ja esittämisen informaationa myöhemmin.

NoSQL-pohjaista dokumenttitietokantaa, kuten MongoDB:tä harkittiin vaihtoehtoisena ratkaisuna, mutta MySQL:n tuoma relaatio koettiin tarpeellisemmaksi eheän datan luontiin, kuin NoSQL:n tuoma notkeus datan rakenteessa.

Tietokannan integrointiin Python-ohjelman kanssa valittiin tietokantojen hallintaan tehty Python-kirjasto Peewee. Valinnan perusteena oli sen kevyt koko ja helppokäyttöisyys, mutta riittävät toiminnallisuudet sovelluksen toteuttamista ajatellen.

3.2.3 Palvelinratkaisu

ART-Tool tarjottaisiin omalla virtuaalipalvelimella, jossa sillä on täysin muista palveluista eristyksissä oleva ympäristö. Tällöin voitiin varmistaa, että palvelimella on juuri halutut versiot tarvittavista teknologioista. Käytettävissäni oli työnantajan oma virtuaalipalvelimien hallintajärjestelmä, jossa voidaan provisioida ART-Toolia varten sopivan kokoinen virtuaalipalvelin, sisältäen Linux-pohjaisen CentOS 7 -käyttöjärjestelmän asennuksen. Palvelimelle on asennettu Nginx tarjoamaan itse käyttöliittymäsovellusta, ja siihen määritellään

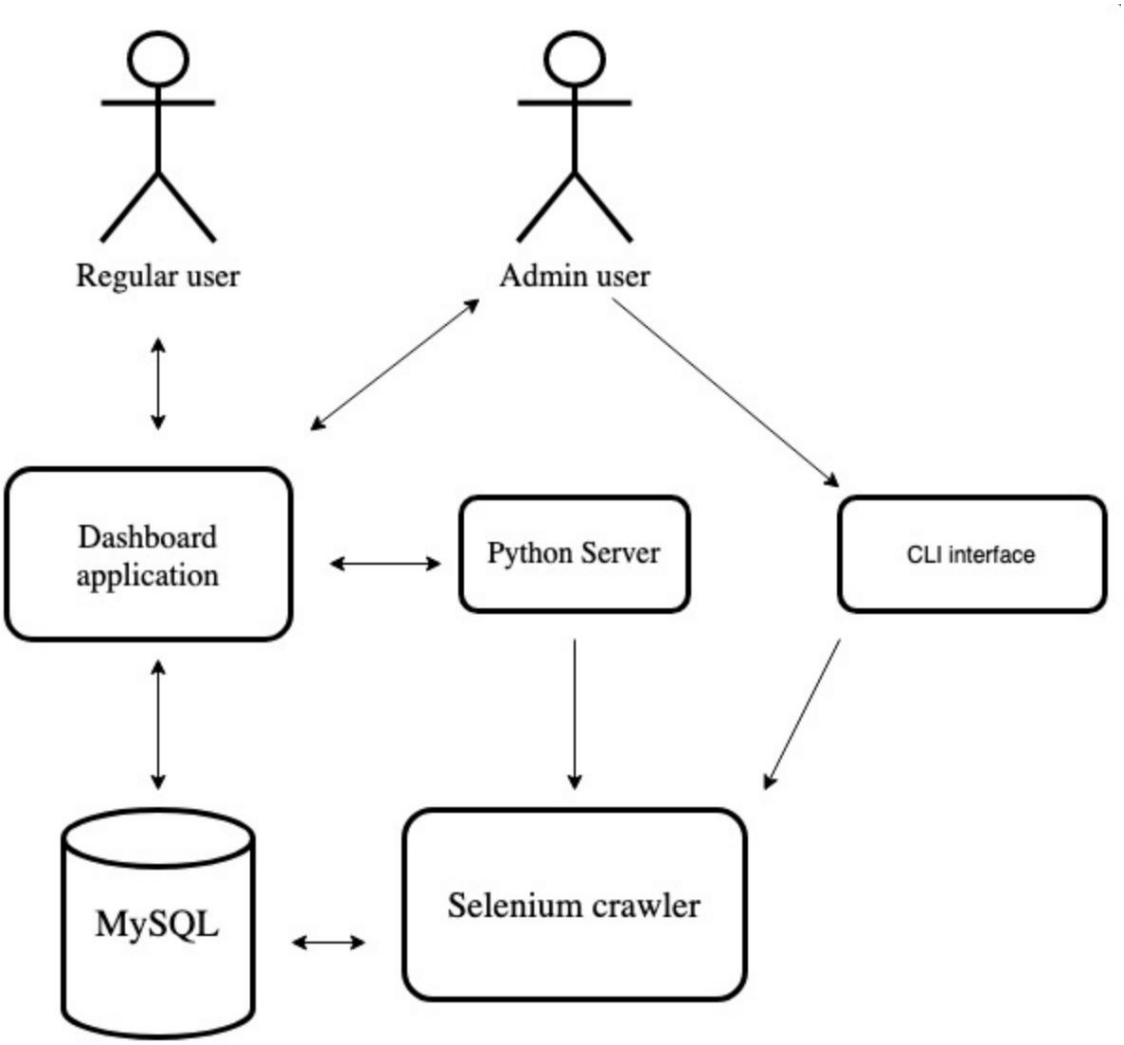
myös tarvittavat tietoturvallisuuteen liittyvät asetukset, kuten sallitut IP-osoitteet. Nginx:n etu oli myös se, että se ei tarvinnut muutoksia ohjelman lähdekoodiin, jos sen asetuksia haluttiin vaihtaa.

3.2.4 Projektinhallinta

ART-Toolia varten luodaan uusi projekti työnantajan projektinhallintajärjestelmä Jiraan. Jiran avulla jokaisesta erillisestä työtehtävästä tehdään uusi työtehtävä, tiketti, joihin tallentuu myös tuntimerkkauksissa niille osoitettu ajankäyttö. Käyttämällä projektinhallintajärjestelmää voidaan seurata kehitystyön vaiheita tarkemmin, ja hahmottaa kokonaiskuvaa yksittäisten toimenpiteiden tarkkuudella. Jira generoi projektin nimen perusteella lyhyemmän projektitunnisteen, joka liitetään osaksi jokaista projektille merkittyä tehtävää. Jira-projekti luotiin nimellä Automated Regression Testing Tool, jonka projektitunniste lyhentyi muotoon ARTT. Tästä tunnisteesta johdettiin myös nimi ART-Tool.

ART-Toolin lähdekoodi sijaitsee kahdessa erillisessä Git-repositoriossa. Yksi repositorio on varattu käyttöliittymän web-sovellukseen ja toinen crawlerin Python-koodille.
Tämän raportin lisäksi muille kehittäjille merkityksellinen tekninen dokumentaatio tallennetaan tekstidokumenttina osaksi lähdekoodia. Työkalun käyttöliittymään toteutetaan mahdollisuus antaa palautetta lomakkeella suoraan ohjelmassa itsessään, sekä lisäksi mahdollisuus valita testiajon jälkeen, tuliko työkalun käytöstä heille hyötyä tapauskohtaisesti. Tätä dataa kerätään ensisijaisesti käytettäväksi lähdemateriaalina tähän raporttiin.

3.2.5 ART-Toolin komponenttien kommunikaatio keskenään



Kuva 1. ART-Toolin osien kommunikointi keskenään

Kuvakaappausten keräys ja vertailu suoritetaan omassa sovelluksessaan, ja käyttöliittymä on täysin erillinen kokonaisuus. Tämä ratkaisu valittiin siksi, että tällöin molempia järjestelmiä voidaan kehittää ilman, että ne vaikuttavat välttämättä toistensa toimintaan. Esimerkiksi käyttöliittymän ulkoasua voidaan muokata samalla, kun järjestelmä vertailee kuvatiedostoja. Vastaavasti muutoksia kuvakaappausten ottamiseen liittyvässä logiikassa voidaan testata nopeasti, ilman tarvetta käynnistää web-sovellusta uudestaan.

Sekä web-käyttöliittymä, että crawler käyttävät samaa MySQL-tietokantaa ensisijaisena tiedon lähteenään. Kaikki asetukset, sivustojen tiedot, ajojen lopputulokset, tiedostojen polut ja käyttölokit tallentuvat tietokantaan. Tämän datan perusteella käyttöliittymä luo näkymät testattavien sivustojen tarkastelua ja hallinnointia varten, ja crawler saa tarvittavan kontekstin kuvakaappausten ottamiseen ja vertailuun.

Käyttöliittymän ja crawlerin välille rakennetaan yksinkertainen Python-pohjainen palvelin, johon on luotu pääsy kaikkiin tarpeellisiin crawlerin metodeihin. Tämä palvelin hyväksyy vain paikallisia pyyntöjä, jotka tulevat käyttöliittymän verkkosovelluksesta. Palvelinratkaisuun päädyttiin, koska tämä mahdollistaa sen, että crawlerin ja käyttöliittymän sijainnillä pääpalvelimen tiedostojärjestelmässä ei ole väliä. Ainoastaan crawlerin verkko-osoite täytyy määrittää käyttöliittymäsovellukseen. Pyydetty toimenpide määrittyy crawlerille lähetetyn pyynnön osoitteessa. Lisätiedot, kuten testattavan sivuston id ja asetukset, lähetetään osana pyyntöä. Palvelin vie nämä tiedot eteenpäin crawlerille, ja niiden käsittely hoidetaan vasta crawlerissa, jolloin palvelimen koodin tarvitsee ainoastaan ohjata data oikeaan osoitteeseen.

4 ART-Toolin kehitysprosessi

4.1 Crawler

4.1.1 Crawlerin toimintaperiaate

Crawlerin toimintalogiikan ydin pohjautuu vahvasti prototyypissä määritettyyn reittiin:

- Tarkista, onko sivusto olemassa
- Tarkista, onko sivustosta otettu jo kuvakaappauksia
- Kerää kuvakaappaukset sivuston eri näkymistä, joko pohjakuviksi tai pohjia vasten verrattaviksi verrokkikuviksi
- Vertaile kuvakaappauksia keskenään, jos kuvatiedostoissa ei havaittu vikoja tai puutteita.

Crawlerin ensimmäiset versiot kehitettiin ennen käyttöliittymää, joten sitä täytyi pystyä operoimaan myös komentoriviltä. Toiminnallisuudet kehitettiin kuitenkin alusta alkaen niin, että niiden integroiminen osaksi tulevaa Python-palvelinta olisi mahdollisimman helppoa.

Ensimmäinen vaihe oli ohjelmoida komentorivin parametrien käsittely. Ainoa vaadittu syöte käyttäjältä on sivuston URL-osoite, tai pelkkä domain-nimi. Lisäksi voidaan antaa vapaaehtoinen parametri, joka määrittää ajon tarkoituksen manuaalisesti. Annettu sivuston osoite käsitellään ohjelman haluamaan muotoon automaattisesti, joka tarkoittaa sitä, että käyttäjän ei tarvitse huolehtia syötteen tarkasta muodosta itse.

Seuraavaksi ohjelma tarkistaa tietokannasta, löytyykö annetulla osoitteella jo dataa. Jos ei löydy, luodaan uusi datarivi sivustolle, ja käynnistetään ensimmäisten, referenssinä toimivien kuvakaappausten keräys.

Ennen kuvakaappausten ottamista täytyy tietää, millaisia sivuja ja näkymiä sivustolla on. Tätä varten hyödynnetään jälleen prototyypin toimintalogiikkaa:

- Mennään testattavan sivuston etusivulle.
- Haetaan etusivun html-rakenteesta jokainen linkki.
- Löydettyjen linkkien osoitteet kerätään listaan
- Listasta poistetaan tyhjät tai toistuvat osoitteet, sekä sivuston ulkopuolelle vievät osoitteet.
- Lisäksi täytyi luoda uusi vaihe, jossa crawler lähettää pyynnön jokaiseen jäljellä olevaan osoitteeseen siltä varalta, että kyseessä on uudelleenohjauksen sisältävä linkki, joka kuitenkin päällisin puolin vaikuttaa vievän sivuston domain-nimen sisältävään osoitteeseen.
    - Esimerkiksi ”sivu1.com/kalenteri” saattaa olla linkki, joka viekin kolmannen osapuolen kalenteriohjelmiston sivulle ”kalenteriohjelma.com”, jolla ei ole mitään tekemistä testattavan sivuston kanssa.
    - Tämä osoitteen odottamaton muutos aiheutti myös välillä crawlerin yllättävän kaatumisen.

- Kun lopullinen lista sivuista on kerätty, luodaan niistä omat datarivinsä, jotka tallennetaan tietokantaan. Peewee tekee SQL-lauseet automaattisesti, eikä niitä tarvitse itse määritellä.

Kun lista sivuista on koottu, siirrytään kuvakaappausten keräysvaiheeseen.

Jokaista sivulistalla olevaa sivua kohden tehdään seuraavat toimenpiteet:

- Mennään sivulle selaimella.
- Lasketaan sivun kokonaiskorkeus käyttäen JavaScriptiä.
- Jos korkeus on korkeampi, kuin 1080 pikseliä, muutetaan selainikkunan korkeus sivun korkuiseksi, jotta koko sivu tallentuu kuvakaappaukseen.
- Generoidaan nimi uudelle hakemistolle, jolle kyseiseen sivuun liittyvät kuvatiedostot tallennetaan.
- Kuvakaappaus tallennetaan png-kuvatiedostona sille luotuun hakemistoon.
- Juuri luodusta png-tiedostosta generoidaan ImageMagickilla myös pienikokoinen thumbnail-kuva käyttöliittymää varten.
- Sivun otsikko ja hakemiston nimi tallennetaan tietokantaan myöhempää käyttöä varten.

Kun kyseessä on pohjakuvien keräys, päättyy crawlerin toiminta tähän. Sivusto asetetaan ”idle” -tilaan, odottamaan verrokkikuvien keräämistä.

Kun on aika kerätä verrokkikuvat, noudatetaan edellä mainitut askeleet kuvien ottamiseen muutamalla eroavaisuudella:

- Ennen keräämisen aloittamista, käydään läpi kaikki vierailtavien sivujen hakemistot, ja tarkistetaan, että niistä löytyy ehjät kuvakaappaukset.
- Puutteita havaittaessa kerätään uudet pohjakuvat tilalle, ja vertailuvaihe peruuntuu tällöin.
- Koska sivustolla vierailtavien näkymien osoitteet ovat jo tiedossa, ei niitä tarvitse luoda uudestaan, vaan kerääminen voi alkaa heti.
- Kuvat tallennetaan omaan alihakemistoon, jotta ne eivät ylikirjoita jo otettuja pohjakuvia.

Kun kaikista testattavista näkymistä löytyy sekä pohja-, että verrokkikuva, aloitetaan vertailuvaihe, joka etenee seuraavasti:

- Käyttämällä ImageMagickia, lasketaan kuvaparin välillä eroavien pikselien prosentuaalinen määrä.
- Jos eroavien määrä ylittää tietyn raja-arvon, tulkitaan näkymän muuttuneen selvästi kuvien ottoaikojen välillä. Eron pysyessä alle tämän rajan, siirrytään seuraavaan kuvapariin.
- Raja-arvon ylittäneestä kuvasta koostetaan havainnekuva, jossa kaikki eroavat pikselit värjätään punaiseksi. Tätä kuvaa voidaan hyödyntää myöhemmin tarkastellessa vertailun tuloksia.
- Vertailun lopputulos tallennetaan osaksi kyseisen sivun tietoja tietokannassa, numeroarvona 1 (eroja havaittiin yli raja-arvon) tai 0 (raja-arvo ei ylittynyt).

Vertailun päätyttyä sivusto palautetaan idle-tilaan ja crawler-prosessi päättyy.

4.1.2 Crawlerin kehityksen aikana havaitut haasteet ja niiden ratkaisu

Crawlerin suorituskyky heikkeni ensin hyvin nopeasti, jos sillä ajettiin useampaa keräilyä tai vertailua samanaikaisesti. Tämä johtui puhtaasti siitä, että ART-Toolille provisiodut palvelinresurssit eivät olleet kovin runsaita, ja ne täytyi jakaa kaikkien toimintojen kesken. Jos yhtäaikaisia ajoja oli enemmän kuin kolme, oli todennäköistä, että vähintään yksi niistä keskeytyisi liian pitkän keston tai muistin puutteen takia.

Ratkaisu löytyi ulkoistamalla selaintoiminnot ja kuvakaappausten keräys toimeksiantajan omalle Selenium-palvelimelle, jossa oli mahdollista suorittaa samankaltaista selainautomaatiota puhtaasti sille varatuilla resursseilla. Selainten hallinta ja sivustojen HTML-rakenteen arvioiminen tapahtuu ART-Toolin palvelimen ulkopuolella, jolloin crawler-prosessien täytyy ainoastaan odottaa lopputulosten valmistumista. Tarvittaessa myös paljon suorituskykyä vaativa kuvien vertailuprosessi voidaan ulkoistaa esimerkiksi pilviresursseille, mutta toistaiseksi sitä ei ole toteutettu.

Useimmat testatut sivustot sisältävät uudella selainsessiolla ponnahdusvalikon, jossa täytyy valita, salliiko sivustolle mahdollisuuden tallentaa evästeitä käyttäjän selaimeen. Nämä valikot ilmestyvät yleensä liukumalla näkyviin alareunasta, jonka jälkeen ne jäävät osittain muun sisällön päälle, ellei käyttäjä valitse haluamaansa asetusta evästeiden suhteen. Tätä varten täytyi kehittää yleispätevä logiikka, joka pystyisi paikantamaan evästeiden hyväksyvän painikkeen sijainnin HTML-rakenteessa, sekä klikkaamaan sitä, jotta ikkuna häviäisi näkyvistä kuvien keräämisen ajaksi. Tätä tarkoitusta varten lopulta kehitettiin neljä vaihtoehtoista paikannustapaa, joista suoritetaan jokainen yksi kerrallaan, kunnes painike löytyy. Painikkeen paikantamisen ja klikkaamisen jälkeen odotetaan, että ikkuna on kokonaan hävinnyt ruudusta, jonka jälkeen voidaan kerätä ruutukaappauksia ilman näköesteitä.

Kuvia vertaillessa syntyy silloin tällöin vääriä positiivisia lopputuloksia, koska sivustoilla saattaa olla esimerkiksi satunnaisesti valikoituvia kuvaelementtejä tai animoituja käyttöliittymäelementtejä, kuten karuselleja. Kaikista yleisin turhien eroavaisuuksien aiheuttaja oli kuitenkin ns. lazy loading -metodia hyödyntävät kuva- ja muut elementit, joissa kuvatiedostot ladataan vasta itse sivun latautumisen jälkeen. Selenium ei osaa seurata näiden elementtien latautumista, vaan tulkitsee sivun olevan jo täysin latautunut siinä vaiheessa, kun kaikki muut elementit rakenteessa ovat latautuneet. Tämän takia lukuisissa kuvakaappauksissa oli eroja, koska joskus kuvat olivat ehtineet latautua lopulliseen muotoonsa, mutta välillä esimerkiksi verkkoyhteyden nopeusmuutosten takia osa oli vielä lataamatta.

Tämän ratkaisemiseksi kehitettiin vapaaehtoinen mahdollisuus lisätä ylimääräinen odotustauko sivulatauksen yhteyteen, jotta sivuston media on todennäköisemmin jo ehtinyt latautua lopulliseen muotoonsa. Tämä ratkaisu vähensi oleellisesti vääriä positiivisia tuloksia, jotka liittyivät selvästi lazy loading -elementteihin.

4.2 Dashboard-sovellus

4.2.1 Konsepti ja kehityksen aloitus

ART-Toolin hallintaan sopivimmaksi tavaksi havaittiin web-pohjainen käyttöliittymä. Pääasialliseksi teknologiaksi valikoitui Java, ja varsinkin Spring Boot -sovelluskehys.Sovelluksen raamit generoitiin automaattisesti hyödyntäen Spring Initializr -palvelua. Palveluun syötettiin halutut versiot, paketinhallinta sekä halutut lisäosat. Valmis ohjelmapohja ladattiin sitten pakattuna zip-tiedostona, ja tuotiin kehitysympäristöön.

Ensimmäinen kehitysvaihe oli luoda tietokantaan crawleria varten luodun rakenteen määrittely myös osaksi Java-sovellusta. Jokainen datatyyppi, kuten sivuston tietorivien määritykset toisinnettiin Java-natiiveiksi objekteiksi, jotka vastasivat ominaisuuksiltaan jo kerättyä dataa. Jokaiselle objektille luotiin myös oma interface, jonka avulla dashboard pystyy helposti hakemaan ja muokkaamaan datarivejä. Tietokannan data on täysin yhteensopivaa crawlerin ja dashboardin välillä, eikä sitä tarvitse erikseen kääntää tai prosessoida. Spring Boot toteuttaa SQL-integraation automaattisesti objektien määrittelyn jälkeen. Tämä nopeuttaa kehitysprosessia reilusti, koska kehitettäessä ei tarvitse luoda SQL-lausekkeita tai tietorakenteita manuaalisesti.

ART-Toolin käyttäminen vaatii sisäänkirjautumisen. Kuka tahansa voi luoda uuden käyttäjän kirjautumissivulta käsin, mutta palvelun verkko-osoitteeseen pääseminen on rajattu ainoastaan toimeksiantajan omille IP-osoitteille. Kirjautumissivun kautta luodut käyttäjät saavat perustason oikeudet. Näillä oikeuksilla voi rajattomasti operoida ja muokata testattavia sivuja, mutta muiden käyttäjätilien hallintaan ei peruskäyttäjät pääse.

4.2.2 Ulkoasu ja käyttökokemus

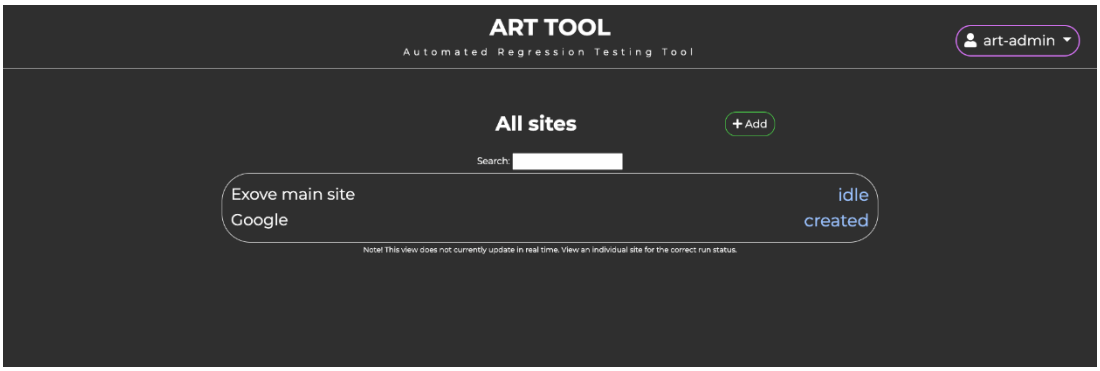
Tietorakenteiden jälkeen kehitys siirtyi käyttöliittymään itseensä. Käyttöliittymän ulkoasua ei suunniteltu etukäteen, vaan lopullinen design syntyi kokeilemalla erilaisia ratkaisuja. Lopputulos tähtää minimalistiseen ulkoasuun, jossa tummanharmaan taustan päällä on erivärisillä ääriviivoilla toteutettuja yksinkertaisia painikkeita.

Yksinkertaisen käyttökokemuksen saavuttamiseksi vältettiin liiallisen informaation näyttäminen kerrallaan yksittäisessä näkymässä. Tätä varten osa informaatiosta on piilotettu alihakemistoihin tai erikseen avautuviin haitarivalikoihin. Väreillä ja ikoneilla ilmaistaan sovelluksen tilaa, missä mahdollista ja luontevaa.

Käyttöliittymässä hyödynnetään toistuvia komponentteja kaikkialla, missä mahdollista, jotta käyttöliittymä pysyy varmasti yhdenmukaisena päivitysten yhteydessä, ja koodin ylläpito pysyy helppona.

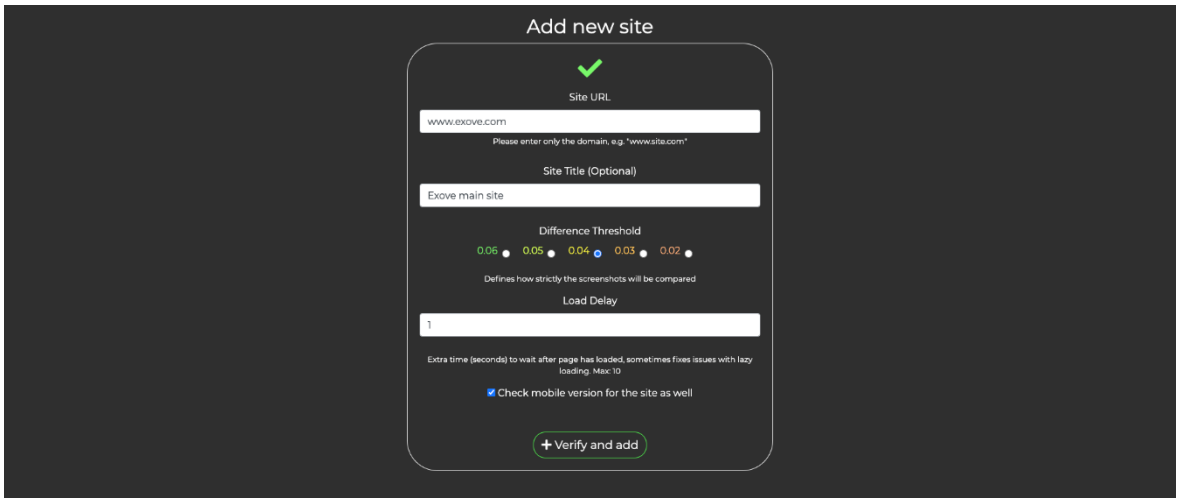
Paria poikkeuskohtaa lukuun ottamatta ART-Tool täyttää WCAG 2.1 saavutettavuusstandardin korkeimman AAA-tason. Tätä ei vaadittu saavutettavuuslain, eikä toimeksiantajan toimesta, mutta ART-Toolin kokoisessa sovelluksessa saavutettavuusparannusten tekeminen ei vienyt kauaa, ja saavutettavuus tarkoittaa usein myös sitä, että HTML-rakenne on hyvälaatuista. Suurin osa kriteeristön kohdasta täyttyi sellaisenaan valmiiksi, koska käyttöliittymän HTML-rakenne oli semanttisesti HTML-standardin mukaista ja tarpeeksi yksinkertaista. Käyttöliittymä on täysin käytettävissä myös pelkällä näppäimistösyötteellä. Visuaalisen regressiotestauksen näkökykyä vaativan luonteen vuoksi työkalun käyttäminen vaatii käyttäjää kuitenkin kykenevän näkemään vertailutulosten ymmärtämiseksi.

4.2.3 Näkymät



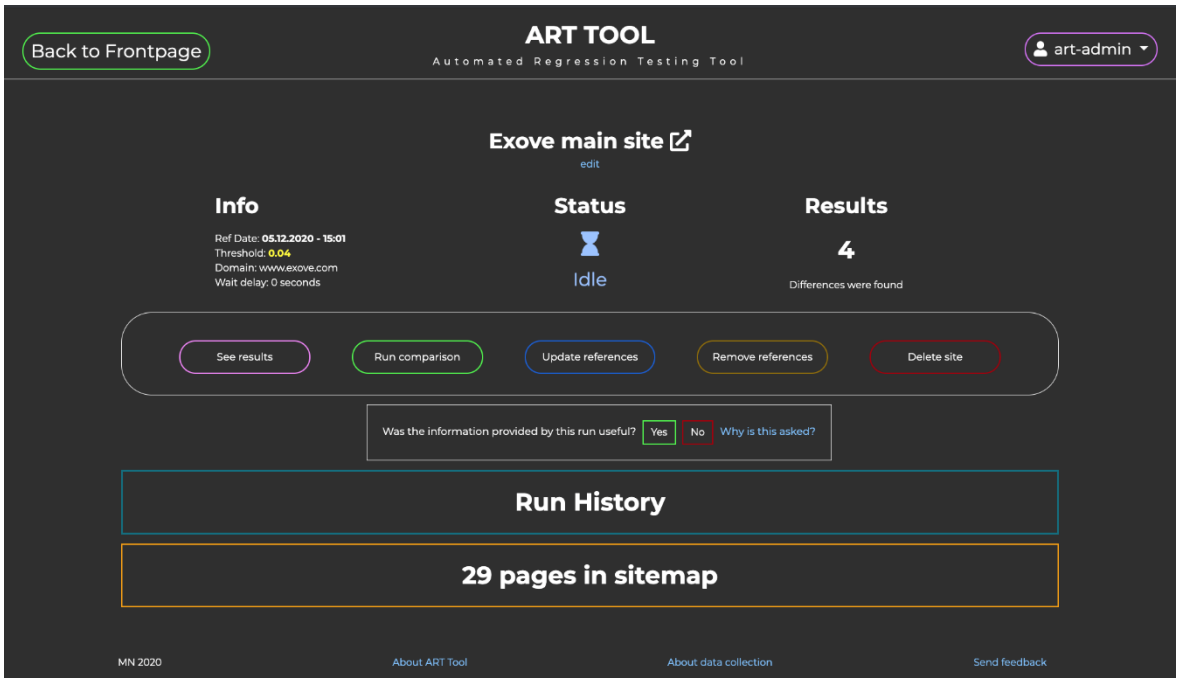
Kuva 2. ART-Toolin päävalikko

ART-Toolin ensimmäinen näkymä sisäänkirjautumisen jälkeen on sovelluksen päävalikko, josta nähdään kaikki lisätyt sivustot. Mahdolliset toimenpiteet tässä näkymässä on joko siirtyä lisäämään uusi testattava sivusto, tai valita yksi olemassa oleva sivusto, ja tarkastella sitä.



Kuva 3. ART-Toolin lomake uuden testattavan sivuston lisäämiseen

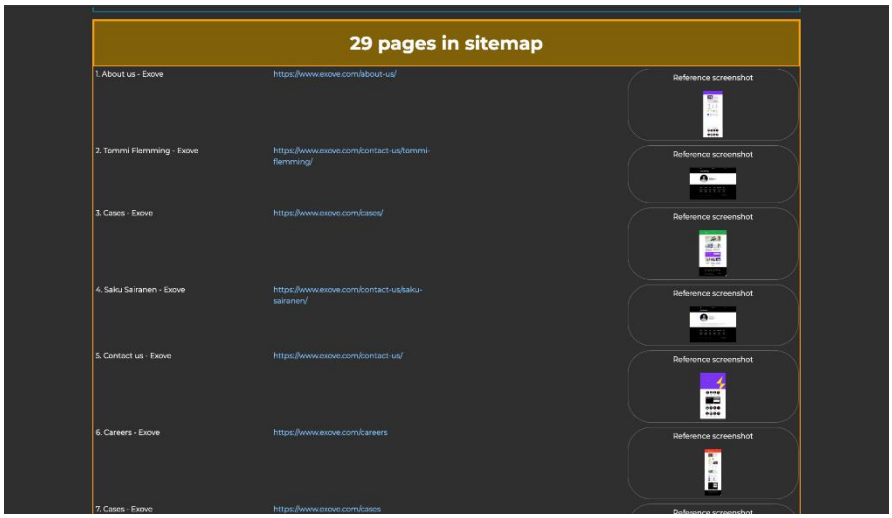
Sivuston lisääminen on toteutettu lomakemaisessa näkymässä. Ainoa pakollinen syöte on itse testattavan sivuston verkko-osoite. Osoitteen rakenne muutetaan automaattisesti haluttuun muotoon, ja sen jälkeen palvelu testaa automaattisesti, että annettu osoite on olemassa. Osoitteen lisäksi voi antaa myös erillisen nimen sivustolle, jos pelkästä url-osoitteesta ei ole selvää, mikä palvelu on kyseessä. Muutamia testiajon kulkuun liittyviä asetuksia voi myös syöttää jo tässä vaiheessa, kuten crawlerin tahtia ja mobiilinäkymien sisällyttämistä.



Kuva 4. ART-Toolin hallintapaneeli www.exove.com -sivuston testaamiseen.

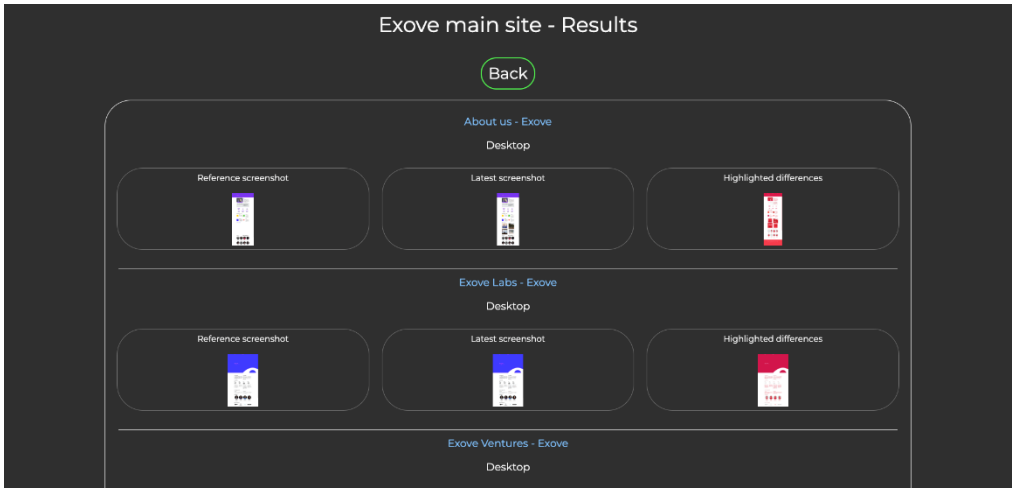
Kun haluttu sivusto päävalikosta on valittu, siirrytään sivuston omaan valikkonäkymään. Tämän näkymän kautta voidaan käynnistää testiajoja, tarkkailla ajon kulkua, sekä arvioida

ajon tuloksia. Valittavia toimintoja näytetään käyttäjälle vain sen verran, mitä on mahdollista tehdä. Varhaisemmassa versiossa kaikki painikkeet ja informaatio oli koko ajan nähtävillä, joka teki käyttöliittymästä vaikeaselkoisen, varsinkin uudelle käyttäjälle. Esimerkiksi juuri lisätylle sivustolle näytetään lähinnä mahdollisuudet lisätä ensimmäiset referenssikuvat sivustolle, tai vaihtoehtoisesti poistaa sivu työkalusta.



Kuva 5. Kerättyjen kuvakaappausten esikatselu

Kun referenssikuvat on lisätty, ilmestyy näkymään uusi pudotusvalikko, jonka avaamalla voidaan tarkastella, mitä näkymiä työkalu valitsi testattaviksi näkymiksi. Tämän lisäksi on mahdollista tarkastella kerättyjä ruutukaappauksia, jotta voidaan varmistaa sivuston olevan halutussa tilassa kuvakaappauksissa. Jos kuvissa ilmenee puutteita esimerkiksi lazy loadingin takia, voi vielä muuttaa ajon asetuksia ja hakea uudet kuvat, ennen kuin sivusto on päivitetty ja uusien referenssien hakeminen on mahdotonta.



Kuva 6. Kuvakaappausten vertailusta koottu testiraportti.

Kun kuvakaappausten vertailu on valmistunut, ilmoitetaan käyttäjälle, montako eroavaisuutta testiajo löysi. Jos yhtäkään eroavaisuutta ei löytynyt, ilmoitetaan tämä suoraan hallintavalikossa ja testausprosessi on sivuston osalta valmis. Jos eroavaisuuksia löytyi, aukeaa valikkoon uusi painike, jonka kautta pääsee tarkastelemaan sivunäkymä kerrallaan, missä eroavaisuudet tapahtuivat.

Jokaisen sivunäkymän alla on suora linkki näkymään, jotta voidaan helposti käydä myös itse sivustolla tarkastelemassa, miltä näkymä näyttää käyttäjän omassa selaimessa. Tämän lisäksi on suora pääsy kolmeen kuvaan:

1. Näkymän referenssinä toiminut kuvakaappaus
2. Uusi kuvakaappaus, joka erosi referenssistä
3. Erillinen havainnekuva, jossa kaikki poikkeavat pikselit on värjätty punaiseksi

4.2.4 Dashboardin kehityksessä havaitut haasteet ja ratkaisut

Kehitysvaiheessa dashboardissa ei ollut reaaliaikaista tapaa seurata testiajon kulkua. Ajon vaihe päivittyi tietokantaan reaaliaikaisesti crawlerin toimesta, mutta dashboardissa näkymä täytyi päivittää manuaalisesti. Seurannan ajantasaisuus on hyvin tärkeää työkalun tehokkaaseen käyttöön, joten se täytyi implementoida jollain tavalla.

Ratkaisu toteutettiin Javascript-funktiolla, joka ladataan osana seurantanäkymää. Funktio tekee haun työkalun tietokantaan, käyttäen sitä varten luotua API-osoitetta. Jos haettu sivusto on aktiivisessa tilassa, alkaa funktio toistaa hakua sekunnin välein, päivittäen tuoreimman tilatiedon käyttöliittymään. Tätä jatkuu niin kauan, kunnes ajo on valmis ja sivusto ei ole enää aktiivinen, tai jos käyttäjä siirtyy toiseen näkymään. Ratkaisu ei ole täysin reaaliaikainen, mutta riittävä työkalun tarpeisiin. Esimerkiksi websocket-tekniikkaa käyttämällä voitaisiin saada aidosti reaaliaikainen ratkaisu, mutta sen toteuttaminen vaatisi liikaa resursseja verrattuna toiminnon tuomaan lisäarvoon.

Crawler kerää kaikki kuvakaappaukset osaksi sen omaa hakemistoa kovalevyllä. Täytyi kehittää tapa kertoa dashboardille, missä crawler kuvakaappauksineen sijaitsee tiedostojärjestelmässä, jotta kuvakaappaustiedostot voidaan tarjota verkon kautta loppukäyttäjille. Dashboardin omien hakemistojen käyttäminen tähän tarkoitukseen ei ollut mahdollista, koska dashboard pakataan ennen käyttöä binääritiedostoksi, joka ei sisällä navigoitavia kansiorakenteita tiedostojärjestelmässä. Ratkaisu oli luoda tiedostorajapinta dashboardiin. Tiedostorajapinnalla voitiin nimetä /screenshots/ -alipolku verkko-osoitteisiin, joka osoittaa tiettyyn sijaintiin palvelimen tiedostojärjestelmässä. Tällä tavalla /screenshots/ voitiin laittaa osoittamaan crawlerin imgs-alihakemistoon, joka sisältää itse kuvatiedostot. Osoitteen

olisi voinut kovakoodata olemaan tietyssä paikkaa tiedostojärjestelmässä, mutta tämä tarkoittaisi sitä, että Crawler täytyisi aina asentaa samaan paikkaan tiedostojärjestelmässä, joka voi olla mahdotonta tiettyjen palvelinympäristöjen konfiguraatioissa. Tämän ratkaisemiseksi dashboardiin lisättiin toiminto, joka pyytää crawlerilta sen sijainnin käynnistyessään, käyttäen piilotettua osoitetta sen rajapinnassa. Rajapinnasta saatu tiedostopolku asetetaan /screenshots/ -polun kohteeksi. Tällä ratkaisulla crawler voidaan asentaa mihin tahansa palvelimen tiedostorakenteessa, ja ainoa uusi vaatimus on se, että crawlerin palvelin täytyy olla käytössä, kun dashboardia käynnistetään.

4.2.5 Dashboardin päivittäminen automaattisesti

Dashboard-sovelluksen versiopäivitykset on täysin automatisoitu. Tämä on toteutettu luomalla CI/CD-putki lähdekoodista lopulliseen tuotteeseen.

Julkaisuputken ensimmäinen vaihe tapahtuu lähdekoodin hallintajärjestelmässä, Git-versionhallintajärjestelmään perustuvassa Bitbucketissa. Uudet toiminnallisuudet, bugikorjaukset ym. päivitykset kehitetään kloonaamalla viimeisin tuotantoversio koodista uuteen koodihaaraan, jossa muutokset tehdään. Kun muutokset ovat valmiit, yhdistetään uudet muutokset koodin tuotantoversioon, jolloin syntyy uusin tuotantoon vietävä versio.

Projektia varten tehtiin jatkuva julkaisuputki Jenkins-palvelimeen. Kun uusi versio dashboardin tuotantokoodista on luotu, ilmoittaa Bitbucket tästä Jenkinsille automaattisella webhook-pyynnöllä. Webhookin lähettämiselle voidaan asettaa lukuisia eri ehtoja, kuten esimerkiksi vain tietyn koodihaaran päivitykset tai hyväksytty koodiarvio. Dashboardin tapauksessa webhook-pyyntö lähetetään ainoastaan, kun tuotantohaaraan on onnistuneesti yhdistetty uutta koodia.

Dashboardin Jenkins-projekti aloittaa julkaisuprosessin saatuaan sille osoitetun webhook-pyynnön, tai jos käyttäjä käynnistää projektin manuaalisesti Jenkinsin käyttöliittymästä. Käynnistystavalla ei ole vaikutusta prosessin kulkuun.

Julkaisuprojektin ensimmäinen vaihe on kopioida uusin versio dashboardin lähdekoodista projektin hakemistoon palvelimella. Koska Java-sovellukset kootaan pakatuksi binääritiedostoksi ennen ajamista, täytyy dashboardille tehdä sama prosessi. Kokoamisprosessi voitaisiin toteuttaa joko Jenkinsissä, tai ART-Toolin palvelimella. Tässä projektissa sovellus kootaan jo Jenkinsissä, jotta ART-Toolin palvelimen rajallisia laitteistoresursseja ei kulu siihen.

Binääritiedostoa luodessa kokeillaan käynnistää tuoreeltaan pakattu versio sovelluksesta. Tässä vaiheessa ajetaan myös mahdolliset unit-testit, jos sellaisia on sovellukselle määritelty. Dashboard ei sisällä itse tehtyjä unit-testejä, koska niiden tekemisestä minulla ei ollut aikaisempaa kokemusta, ja täten niiden kehittäminen olisi vienyt paljon lisäaikaa.

Pakkausprosessin epäonnistuminen keskeyttää julkaisuprosessin, ja lähettää ilmoituksen projektin omalle kanavalle toimeksiantajan käyttämään viestintäpalveluun, Slackiin. ART-Toolin palvelimella sijaitseva versio dashboardista pysyy koskemattomana, joten epäonnistunut paketti ei voi korvata toimivaa versiota ja täten aiheuttaa keskeytystä palvelun käyttöön. Jos sovellus käynnistyy oikein ja testit menevät läpi pakkaamisen jälkeen, voidaan seuraavaksi siirtyä asentamaan juuri luotu binääripaketti palvelimelle.

Paketin asentamisen suorittaa Jenkins, luomalla yhteyden sovelluksen palvelimelle. Ensiksi se pysäyttää dashboardin käynnissä olevan prosessin. Kun prosessi on pysähtynyt, korvataan dashboardin binääriversio juuri luodulla, uusimmalla versiolla. Lopuksi dashboard käynnistetään, ja uusin versio ilmestyy käyttäjien saataville käynnistyksen päätteeksi. Lopuksi Jenkins ilmoittaa Slack-kanavalle, että asennusprosessi päättyi onnistuneesti. Jos Jenkins ei ole jostain syystä saatavilla, voidaan paketti rakentaa myös suoraan palvelimella lähdekoodista tarvittaessa.

Koko prosessi aina koodipäivityksestä uuden version käyttövalmiuteen kestää yleensä n. 40 sekuntia, josta n. 30 sekunnin ajan dashboard ei ole käytettävissä. Nämä luvut ovat huomattavasti pienempiä verrattuna manuaaliseen päivitysprosessiin, jossa kesti vähintään muutamia minuutteja. Lisäksi automatisoitu julkaisu minimoi mahdollisuuden inhimilliseen virheeseen, joka johtaisi potentiaalisesti pidempään korjausprosessiin ja palvelun saavuttamattomuuteen.

Dashboardin ja crawlerin täysi irrallisuus toisistaan tekee mahdolliseksi sen, että dashboardin ollessa saavuttamattomissa ei kuitenkaan crawlerin tekemät testiajot keskeydy.

4.3 Käyttötilastojen ja palautteen kerääminen

Työkalun käyttömäärien hahmottamiseksi ART-Tool kerää yksinkertaista dataa tietokantaan joka kerta, kun ajoja käynnistetään. Ajoista tallentuu seuraavat tiedot datariviksi:

- Päiväys
- Kesto
- Tyyppi (kuvakaappausten kerääminen referenssiksi vai vertailuun)
- Paljonko kuvia kerättiin
- Kuinka monta rajan yli eroavaa kuvaa löytyi vertailusta (jos vertailtiin)
- Sivusto, jota testattiin
- Käyttäjä, joka käynnisti ajon
- Oliko vertailun tuottamat tulokset hyödyllisiä (kyllä/ei)
    - Tämä kysytään käyttäjältä ajon jälkeen, ei ole pakko vastata

Käyttäjänimi tallennetaan lähinnä siksi, että käyttömääriä laskiessa voidaan helpommin tunnistaa ns. oikeat ajot, joiden tarkoitus on ollut oikeasti testata sivustoja, eikä työkalun toimintaa. Käytännössä tämä saavutetaan poistamalla laskuista ne ajot, jotka on ajettu ennen työkalun varsinaista julkaisupäivää, ja joissa toimijana on ollut kehityskäyttöön tarkoitettu testitunnus. Lisäksi käyttäjätunnuksen ei tarvitse sisältää käyttäjän oikeaa nimeä, joten halutessaan voi käyttäjä syöttää sellaisen käyttäjänimen, josta tätä ei voida tunnistaa.

Yksinkertaisen, kyllä tai ei -vaihtoehtoihin jakautuvan palautekyselyn taustalla oli tavoite luoda tapa antaa palautetta mahdollisimman matalalla kynnyksellä. Strategian taustalla on ajatus siitä, että käyttäjät todennäköisemmin antavat palautetta, jos sen antaminen on nopeaa ja yksinkertaista. Yksinkertaisen palautteen lisäksi on mahdollista antaa monipuolisempaa palautetta suoraan käyttöliittymään luodulla palautelomakkeella.

Datan keräämisestä kerrotaan myös käyttäjälle omalla ohjesivullaan dashboardissa. Jokainen sovelluksen tallentama tietotyyppi listataan, ja käyttäjätunnuksen tallentaminen mukaan tuloksiin perustellaan samoin, miten tässä dokumentissa.

5 ART-Toolin käyttöönotto ja sen tuoman arvon analysointi

5.1 Kehitystyön valmistuminen ja avaaminen käyttäjille

Ensimmäinen virallinen julkaistu ART-Toolista valmistui perjantaina 4. Syyskuuta 2020. Palvelu oli periaatteessa ollut saatavilla toimeksiantajan sisäverkossa aiemminkin, mutta sinne ei voinut luoda omaa käyttäjää, eikä työkalua voinut operoida kirjautumatta ensin sisään.

Julkaisupäivänä järjestettiin demotilaisuus, jossa ensin esiteltiin työkalun historiaa, rakennetta ja toimintaperiaatetta. Tämän jälkeen sen käyttöä demonstroitiin käytännössä. Tilaisuuteen lähetettiin avoin kutsu kaikille toimeksiantajan työntekijöille. Tilaisuuteen osallistui n. 20 henkilöä. Demotilaisuuden päätteeksi avautui mahdollisuus luoda omia käyttäjiä työkaluun, joten tässä vaiheessa palvelu oli kokonaisuudessaan julkaistu kohdeyleisölleen. Julkaisun yhteydessä julkaistiin myös käyttöohje työkalun oikeaoppiseen käyttöön.

5.2 Laajempi käyttöönotto ja sen haasteet

Ensimmäinen tilanne, jossa työkalu näki runsasta käyttöä, oli kun sisällönhallintajärjestelmä Drupaliin julkaistiin kriittinen tietoturvapäivitys, joka asennettiin pikimmiten kaikille sivustoille, jotka olisivat muuten jääneet haavoittuviksi. Päivityksiä asentavia kehittäjiä kehotettiin käyttämään ART-Toolia yhtenä työkaluna päivityksen tuoman mahdollisen regression havaitsemiseen. Tämä tarkoitti sitä, että testiajoja oli parhaimmillaan päällä n. kymmenellä eri sivustolla samaan aikaan. Kehitettäessä ART-Toolia ei testattu, kuin maksimissaan n. kolmella samanaikaisella ajolla. Tämä johtui siitä, että skaalautuvuutta arvioidessa oli oletus, että työkalua käyttäisi lähinnä laadunvarmistukseen erikoistuneet henkilöt, ja hekin harvoin. Sen sijaan nyt samanaikaisia käyttäjiä oli lähemmäs kaksikymmentä.

Lopputulos oli se, että ART-Tool meni täysin jumiin. Sen muisti ja prosessointiteho loppui kesken, joten kaikki testiajot joko kaatuivat, tai kestivät äärimmäisen kauan valmistua. Tästä ensimmäisestä käyttöaallosta ei siten saatu tuotettua huomattavaa arvoa loppukäyttäjille. Työkalun jatkokehityksen näkökulmasta sen sijaan tuli selväksi, että sovelluksen skaalautuvuutta tulisi tehostaa.

Tähän asti Crawler oli suorittanut kaikki selaustoimintonsa paikallisesti, käyttäen palvelimen omia resursseja. Tämä oli pääsyy sille, että ART-Tool ei kestänyt yllättävää työtaakkaa. Testiajojen hyytyminen väheni huomattavasti sen jälkeen, kun selainautomaatio ulkoistettiin erilliselle Selenium Grid -palvelimelle. Sittemmin ART-Tool on nähnyt aktiivista

käyttöä aina, kun kriittisiä, välittömästi asennettavia tietoturvapäivityksiä julkaistaan. Vaikka edelleen osa ajoista hyytyy silloin tällöin, on käyttökokemus ollut huomattavasti toimivampi keskimäärin.

5.3 Käyttäjäpalautteen perusteella hahmotetut jatkokehityskohteet

5.3.1 Mahdollisuus päättää, mitkä näkymät sivustolta testataan

Useampi työkalua kokeillut käyttäjä olisi kaivannut mahdollisuutta määrittää itse testattavat sivunäkymät. Varsinkin multisite-malliset sivustot, eli samalla palvelimella sijaitsevat yleensä domain-nimellä erotellut kokonaisuudet olisi haluttu lisätä samaan vertailuprojektiin. Toistaiseksi ART-Toolissa ei voi lainkaan määrittää manuaalisesti, mitä näkymiä testataan, vaan työkalu rakentaa sivulistan itse. Ehdotus on looginen, koska multisite-ympäristössä päivityksen tekeminen palvelimella vaikuttaa kaikkien siellä sijaitsevien sivustojen toimintaan joka tapauksessa, joten olisi loogista myös testata ne kerralla. Toiminnallisuus tullaan implementoimaan työkaluun, sisältäen myös mahdollisuuden rajata esimerkiksi vain tietyt polut osaksi testausta, jos halutaan nopeasti testata jotain tiettyä näkymää/näkymiä.

5.3.2 Ilmoitukset

Useamman kerran on myös kyselty mahdollisuudesta lisätä ilmoituksia ajon lopputuloksista, esimerkiksi sähköpostiin tai Slackiin. Tämä toiminnallisuus tekisi testaamisesta tehokkaampaa, koska testaajan ei tarvitse aktiivisesti seurata ajon edistymistä, vaan voi tehdä muita asioita, kunnes saa ilmoituksen lopputuloksen vahvistumisesta. Lisäksi toiminnallisuus on helppo toteuttaa, joten tästäkin syystä toiminnallisuus tullaan implementoimaan.

5.3.3 Mobiili

Ensimmäinen julkaistu versio ART-Toolista ei tukenut mobiilinäkymien testausta. Tämä koettiin sen verran oleelliseksi puutteeksi nykypäivän mobiilikäyttäjien määrän huomioon ottaen, että se toteutettiin pian julkaisun jälkeen, ja ehti sikäli myös mukaan tähän raporttiin. Työkalun itsensä käyttö on mahdollista myös mobiilissa, koska käyttöliittymä suunniteltiin responsiiviseksi. Tehokkaimman käyttökokemuksen saa kuitenkin käyttämällä työpöytäversiota.

5.3.4 Ajojen keskeyttäminen

Alun perin ART-Toolin ajoja pystyi keskeyttämään vain järjestelmänvalvoja-tason käyttäjällä. Tämä johtui siitä, että keskeytysprosessi tuhoaa myös jo kerätyt kuvakaappauksen ja sivukartan kokonaan, jotta voidaan varmistaa, ettei tietokantaan tai tiedostojärjestelmään jää virheellistä dataa. Koin tämän toimenpiteen sen verran radikaaliksi, että vähensin riskiä sille, että joku tuhoaisi tiedostonsa vahingossa. Kävi kuitenkin ilmi, että ajot saattavat silloin tällöin edelleen hyytyä, joka tarkoitti sitä, että ainoana järjestelmänvalvojana olin myös ainoa, joka pystyi keskeyttämään hyytyneitä ajoja. Tavallinen käyttäjä ei voi tätä tehdä, koska käyttöliittymä ei anna mahdollisuuksia operoida sivun asetuksia tai ajon tilaa silloin, kun ajo on päällä. Tarpeeksi monen tätä asiaa kysyvän yhteydenoton jälkeen avasin kaikille käyttäjille mahdollisuuden perua ajoja, mutta toiminnon seurauksista varoitetaan käyttäjää selvästi.

Lisäksi keskeytystoiminnon logiikkaa suunnitellaan uudestaan niin, että kaikkea dataa ei välttämättä menetä joka tilanteessa. Kuvakaappaukset voitaisiin kerätä ensin väliaikaisina tiedostoina, jotka korvaavat aiemmat kuvat vasta, kun kaikki kuvat on onnistuneesti kerätty. Tällöin keruuprosessin epäonnistuessa tai manuaalisesti keskeytettäessä ei menetetä alkuperäisiä kuvakaappauksia. Vertailuvaiheeseen edenneessä ajossa tämä mahdollistaisi myös sen, että käyttäjä voi halutessaan keskeyttää vertailun ja palata siihen myöhemmin ilman tarvetta kerätä kuvia uudelleen. Lisäksi käyttäjä voisi valita eroavaisuuksia havaittaessa, korvataanko pohjakuvat uusilla kuvilla, jos nämä edustavat paremmin sivuston haluttua tilaa.

5.4 ART-Toolin merkitys toimeksiantajan toiminnalle

ART-Tool on nykyään vakituinen osa toimeksiantajan testaustoimenpiteitä. Tietoturvapäivityksiä tekevät kehittäjät käyttävät sitä aktiivisesti, ja sen käyttöä on suositeltu kehittäjille suunnatuissa ohjeissa versiopäivityksiin. Lisäksi yleinen käyttökohde on PHP-päivitykset palvelimilla. Työkalu on saanut positiivista palautetta ominaisuuksiensa puolesta. Jopa siinä määrin, että työkalun ollessa huoltotoimenpiteiden takia käyttämättömissä, on sen poissaolo koettu oleelliseksi puutteeksi regression etsimisessä päivitysten tekemisen yhteydessä.

Koska työkalu ei ota kantaa sivuston kontekstiin tai toiminnallisuuksiin, vaan ainoastaan näkyviin pikseleihin black box -periaatteella, saattaa se herkästi merkata sellaisiakin näkymiä muuttuneiksi, joissa ei oikeasti ole mitään muuttunut. Tämä herkkyys on lopulta työkalun suurin vahvuus. Jos se ei löydä eroavaisuuksia sivuston versioista pikselitasolla, on se itsessään hyvä indikaattori siitä, että vähintään ulkoasunsa ja käyttöliittymäkomponenttiensa puolesta sivusto on pysynyt muuttumattomana päivitystoimenpiteiden jälkeen.

Työkalu ei poista manuaalisesti tehtävän, funktionaalisen testaamisen tarvetta. Palvelu ainoastaan lataa näkymät, mutta ei ole vuorovaikutuksessa sivuston toimintojen kanssa, poikkeuksena evästeiden hyväksyminen. Tämä on tiedostettu jo suunnitellessa työkalua, sekä sen julkaisun jälkeen. ART-Tool suunniteltiin tuomaan lisäarvoa nykyisiin toimintamalleihin, eikä korvaamaan mitään olemassa ollutta ratkaisua. Tämän tavoitteen työkalu on saavuttanut, koska se on lisätty vakituiseksi osaksi testaustoimenpiteitä, eikä se kuitenkaan syrjäyttänyt mitään aiempia prosesseja. Lisäksi työkalun passiivinen luonne tekee siitä välittömästi yhteensopivan lähes mille tahansa sivustolle, koska sivustokohtaista testikoodia ei tarvitse määrittää.

Työkalun käyttäminen ei aiheuta kuluja toimeksiantajalle, koska se on asennettu toimeksiantajan omalle sisäiselle palvelimelle, eikä sen käyttöön liity minkäänlaisia lisenssimaksuja. Toimeksiantaja omistaa myös palvelun lähdekoodin, joten sitä voidaan ylläpitää ja päivittää juuri toimeksiantajan omiin tarpeisiin sopivaksi, mikä ei ole aina mahdollista valmiissa, erityisesti suljetun lähdekoodin ohjelmassa. Itse rakennettu ratkaisu varmistaa myös sen, että luottamuksellinen data, kuten työn alla olevien sivustojen sijainnit ja tiedot, eivät tallennu kolmannen osapuolen tietokantoihin tai muihin tiedostojärjestelmiin.

Keskimääräinen ART-Toolin testausajon kesto on 7 minuuttia 15 sekuntia referenssikuvakaappausten keräämiseen, ja 10 minuuttia 12 sekuntia testattavien kuvakaappausten keräämiseen ja kuvaparien vertailuun.

Yhteensä työkalun käytöstä koituva aika testaajalle on siten n. 20 minuuttia, josta suurin osa on passiivista ajon valmistumisen odottamista. Työkalun käyttö ei siis vie paljon aikaa, mutta tuloksilla on käytännön käyttöarvoa.

Vertailutuloksiin johtaneita ajoja on joulukuuhun 2020 mennessä suoritettu yhteensä 208 kappaletta, mutta vain 57 niistä sisältää vastauksen kysymykseen siitä, oliko ajon tulos käyttäjän mielestä hyödyllinen. Vastauksista valtaosa, 49/57 kpl, on kuitenkin positiivisia, mikä vastaa myös vapaamuotoista käyttäjäpalautetta, mitä työkalusta on annettu Slackin välityksellä.

Lisää käyttötilastoja on liitteessä 4.

5.5 Projektista saatu kokemus ja arvio omasta toiminnasta

5.5.1 Suunnittelu

Tämä projekti oli ensimmäinen tuotantokäyttöön asti päätynyt projekti, jonka olen suunnitellut ja toteuttanut kokonaan itsenäisesti. Varsinkin tarkka vaatimusmäärittely osoittautui yllättävän haastavaksi, ja muutoksia suunnitelmiin tuli jatkuvasti kehityksen edetessä. Logiikan visualisoiminen kaavioiksi auttoi alkuun pääsemisessä, koska niitä seuraamalla oli perusrungon rakentaminen helpompaa. Datan rakenne muuttui huomattavasti toteutuksen aikana, mutta lopullista toteutusta ja suunnitelmia vertaamalla voidaan havaita, missä osuuksissa nähtiin eniten kehitystä.

Alun perin sovellus oli tarkoitus rakentaa Amazon Web Services (AWS) -pilviympäristöön. Tämä olisi minimoinut infrastruktuurin ylläpitämisen tarpeen, ja tehnyt skaalautumisesta mutkatonta. Kävi kuitenkin pian ilmi, että aloitusvaiheessa kokemattomuuteni pilviympäristöistä, sekä potentiaaliset käyttökulut olisivat voineet aiheuttaa myöhemmin ajallisia ja kenties rahallisiakin haasteita, ottaen huomioon, ettei projektilla ollut erillistä budjettia käytettävissään. Tarvittaessa ART-Tool voidaan migroida pilveen, jos ratkaisulla havaitaan tarpeeksi potentiaalista lisäarvoa.

Suurin suunnittelussa havaittu puute oli ajankäytön suunnitteleminen suhteessa projektin kehitystyön tarpeisiin. Alkuperäinen suunnitelma oli huomattavasti optimistisempi, kuin miten kehitys lopulta eteni. ART-Toolin kehitys eteni sovitusti vain silloin, jos minulla ei ollut asiakastöitä tehtävänä, jotka menivät tärkeysjärjestyksessä ART-Toolin edelle. Projektin kautta on kuitenkin kertynyt reilusti uutta kokemusta siitä, miten erinäisiä ohjelmatoimintoja voi toteuttaa, ja kuinka kauan niiden toteuttaminen kestää.

5.5.2 Kehitystyö

Pyrin käyttämään työkalun kehityksessä mahdollisimman hyviä käytäntöjä tehokkaan ja kestävän koodin suhteen. Koodia täytyi refaktoroida useita kertoja kehitysprosessin aikana, koska olin kehittäessä löytänyt parempia tapoja toteuttaa tiettyjä toimintoja, jotka olin jo aiemmin toteuttanut. Pyrin luomaan kaikista toteutettavista toiminnoista erilliset työtehtävät toimeksiantajan projektinhallintajärjestelmä Jiraan, mutta välillä saatoin kiireessä toteuttaa ominaisuuksia osana muita ominaisuuksia, joka tekee projektin koodin historian hahmottamisesta hankalampaa muille kehittäjille. Koodi olisi myös hyötynyt aktiivisesta vertaisarvioinnista, mutta tehokkuussyistä ja ohjelman kokeellisuuden takia päätin jättää arvioinnin järjestämättä.

Projekti oli ensimmäinen, johon olen luonut automatisoidun julkaisuputken. Putken luominen auttoi hahmottamaan hyviä ja huonoja menettelytapoja DevOps -kehitysmallissa, mistä tulee olemaan hyötyä myös tulevaisuudessa.

Työkalun käyttöliittymän graafisen ulkoasun toteuttaminen oli myös kohtalaisen suuri haaste, koska olin aiemmin lähinnä käyttänyt valmiita kirjastoja, kuten Bootstrapia graafisten elementtien määrittämiseen. Tässä projektissa loin kaiken grafiikan itse suoraan CSS:llä. Tämä lähestymistapa toi huomattavaa kokemusta niin CSS:n käytöstä, kuin käyttäjäystävällisen kokemuksen luomisesta visuaalisen esityksen keinoin.

5.5.3 Yhteenveto

Projekti osoittautui reilusti alkuperäistä suunnitelmaa työläämmäksi. Jiran datan perusteella projektin tekniseen kehitykseen on käytetty yhteensä n. 9 viikkoa. Tästä huolimatta lopullisiin tavoitteisiin päästiin, ja toimeksiantajan tarpeeseen vastattiin, joten olen tyytyväinen omaan työpanokseeni kokonaisuudessaan, pois lukien reilusti venynyt aikataulu.

Lähteet

Applitools, 2021. Applitoolsin virallinen sivusto. Näkymät:
Applitools Eyes, Automated Visual Testing. Luettavissa: https://applitools.com/products-eyes/. Luettu: 9.5.2021.
Pricing. Luettavissa: https://applitools.com/pricing/. Luettu: 9.5.2021.

Barker, D, 2016. Web Content Management, Luku 1, What Is a Content Management System? Luettavissa: http://bib.fi/824exA/global. Luettu: 3.5.2021

BBC, 2019. Wraithin lähdekoodi ja tekninen dokumentaatio. Github.
Luettavissa: https://github.com/bbc/wraith. Luettu: 6.5.2021

Bradford, Lee Eden; Ndubisi, Nelson Oly, 2006. Content Management Systems, sivu 5. Emerald Group Publishing. Luettavissa: https://haagahelia.finna.fi/Record/nelli21.1000000000242865. Luettu: 9.5.2021.

Browserstack, 2021. Percyn viralliset sivut. Näkymät:
Features. Luettavissa: https://www.browserstack.com/percy/features. Luettu: 9.5.2021.
Pricing. Luettavissa: https://www.browserstack.com/pricing?product=percy. Luettu: 9.5.2021.

Date, C. J., 2008. The relational Database Dictionary, Luku 19. Apress
Luettavissa: https://learning.oreilly.com/library/view/the-relational-database/9781430210412/ch19.html. Luettu: 3.5.2021

Diffy, 2021. Diffyn virallinen sivusto. Näkymät:
Etusivu, Luettavissa: https://diffy.website/. Luettu: 9.5.2021
Features, Luettavissa: https://diffy.website/features. Luettu: 9.5.2021
Pricing, Luettavissa: https://diffy.website/pricing. Luettu: 9.5.2021

Exove, 2021. Olemme yritys, joka uskoo digitaaliseen kasvuun. Luettavissa:
https://www.exove.com/fi/. Luettu: 9.5.2021

Jenkins 2021. Jenkins User Documentation
Luettavissa: https://www.jenkins.io/doc/. Luettu: 3.5.2021

Git 2021. Git.

Luettavissa: https://git-scm.com/. Luettu: 3.5.2021

Guru99, 2021. Relational Data Model in DBMS: Concepts, Constraints, Example.
Luettavissa: https://www.guru99.com/relational-data-model-dbms.html. Luettu: 3.5.2021

Guru99 2021. What is SQL? Learn SQL Basics, SQL Full Form & How to Use.
Luettavissa: https://www.guru99.com/what-is-sql.html. Luettu: 3.5.2021

Hambling, Brian; Morgan, Peter; Samaroo, Angelina; British Computer Society 2010. Software Testing. British Informatics Society Limited.
Luettavissa: https://haagahelia.finna.fi/Record/nelli21.2550000001179516. Luettu: 4.5.2021

Helsingin Yliopisto 2017. Ohjelmistoarkkitehtuurit, Luento 8. Dia 3.
Luettavissa: https://courses.helsinki.fi/sites/default/files/course-material/4522853/ohjelmistoarkkitehtuurit_17_8.pdf. Luettu: 4.12.2020

ImageMagick, 2021. ImageMagick – Convert, Edit, or Compose Digital Images.
Luettavissa: https://imagemagick.org/index.php. Luettu: 3.5.2021

ISTQB, 2021. ISTQB – Regression testing.
Luettavissa: https://glossary.istqb.org/app/en/term/regression-testing-1. Luettu: 4.5.2021

The Linux Information Project, 2006. Binary File Definition.
Luettavissa: http://www.linfo.org/binary_file.html. Luettu: 3.5.2021

Mozilla Developer Network 2020. HTML Basics.
Luettavissa: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics. Luettu: 4.12.2020

Mueller, J, 2015. Security for Web Developers. O'Reilly Media, Inc. Luku 1.
Luettavissa: https://haagahelia.finna.fi/Record/nelli21.3710000000514768. Luettu: 3.5.2021

Murdoch, Graham, 2019. Visual Regression Testing. Base Design.
Luettavissa: https://baseweb.design/blog/visual-regression-testing/. Luettu: 6.5.2021

Myers, Glenford J; Badgett, Tom; Sandler, Corey, 2012. Art of Software Testing. John Wiley & Sons, Inc.
Luettavissa: http://bib.fi/mgF2tA/global. Luettu: 4.5.2021

Needle, 2021. Needlen tekninen dokumentaatio. Readthedocs.io.
Luettavissa: https://needle.readthedocs.io/en/latest/. Luettu: 6.5.2021

News UK, 2020. Aye Spy:n tekninen dokumentaatio. Github.
Luettavissa: https://github.com/newsuk/ayespy. Luettu: 6.5.2021

Nginx, 2021. What is NGINX?
Luettavissa: https://www.nginx.com/resources/glossary/nginx/. Luettu: 3.5.2021

Oracle, 2021. What is Java technology and why do I need it?
Luettavissa: https://java.com/en/download/help/whatis_java.html. Luettu: 3.5.2021

Patton, Ron, 2021. Software Testing, Second Edition. Sams.
Luettavissa: https://haaga-helia.finna.fi/Record/nelli21.1000000000034840. Luettu: 4.5.2021

PHP, 2021. What is PHP?
Luettavissa: https://www.php.net/manual/en/intro-whatis.php. Luettu: 3.5.2021

Preibisch, S, 2018. API Development: a practical guide for business implementation success, luku 1. Apress.
Luettavissa: https://haagahelia.finna.fi/Record/3amk.271803. Luettu: 3.5.2021

Python Software Foundation, 2021. What is Python? Executive Summary.
Luettavissa: https://www.python.org/doc/essays/blurb/. Luettu: 3.5.2021

Robot Framework, 2020. Introduction.
Luettavissa: https://robotframework.org/#introduction. Luettu: 4.12.2020

Rotbart, Gabriel, 2013. Gatling, tekninen dokumentaatio. Github.
Luettavissa: https://github.com/gabrielrotbart/gatling. Luettu: 6.5.2021

Software Testing Help, 2021. Guide To Visual Regression Testing With Visual Testing Tools. Luettavissa: https://www.softwaretestinghelp.com/visual-validation-testing/. Luettu: 6.5.2021

Skrypny, Alex, 2019. What is visual regression? Salsa Digital.
Luettavissa: https://salsadigital.com.au/insights/what-is-visual-regression. Luettu: 6.5.2021

TechTerms, 2021. Script definition.
Luettavissa: https://techterms.com/definition/script. Luettu: 4.12.2020

TechTerms, 2021. Web application definition.
Luettavissa: https://techterms.com/definition/web_application. Luettu: 4.12.2020

Vidhya, V, 2016. Database Management Systems. Alpha Science International Ltd, Oxford. Luku 1.1.
Luettavissa: https://haagahelia.finna.fi/Record/nelli21.3790000000558171. Luettu: 3.5.2021

Taylor, Twain, 2019. What Is Automated Visual Regression Testing? Saucelabs.
Luettavissa: https://saucelabs.com/blog/what-is-automated-visual-regression-testing. Luettu: 6.5.2021

Tiwari, S, 2011. Professional NoSQL. Wrox. Luku 1
Luettavissa: https://haagahelia.finna.fi/Record/nelli21.2550000000052346. Luettu: 3.5.2021

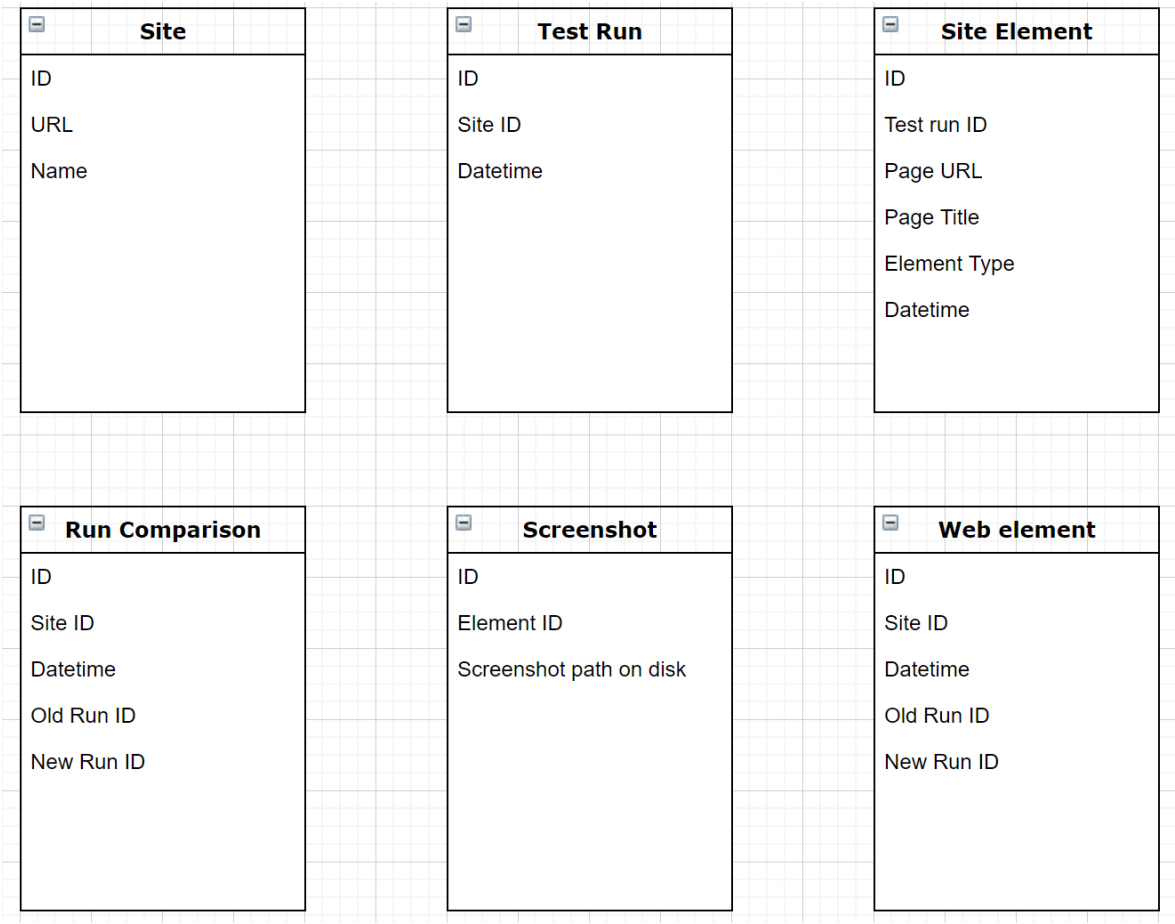
Verona, Joakim, 2016. Practical DevOps. Packt Publishing. Luvut 1.1 & 5.8.
Luettavissa: https://haagahelia.finna.fi/Record/nelli21.3710000000603772. Luettu: 3.5.2021

W3 Consortium 2020. Cascading style sheets.
Luettavissa: https://www.w3.org/Style/CSS/. Luettu: 4.12.2020

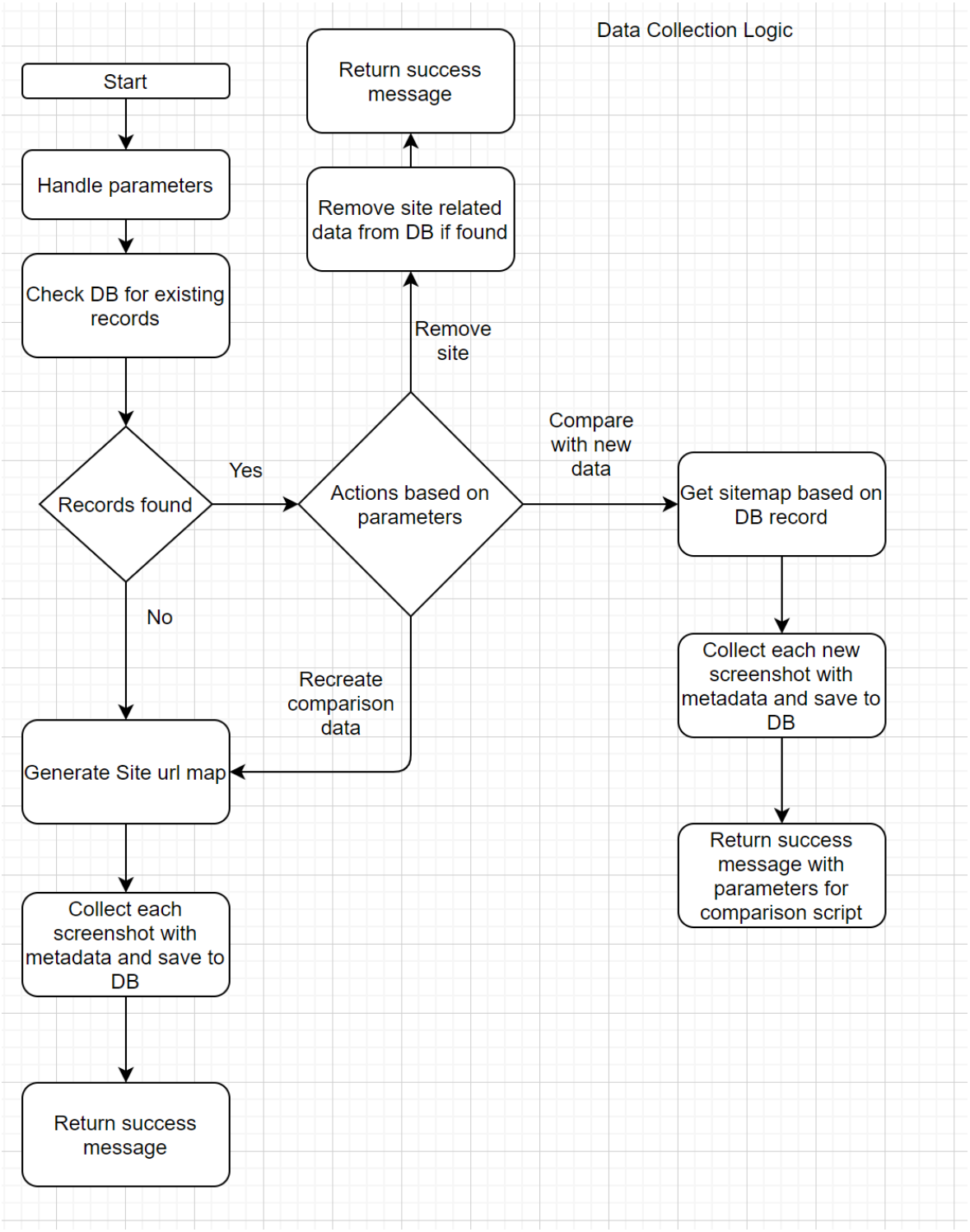
W3 Consortium 2021. Web Content Accessibility Guidelines (WCAG) Overview.
Luettavissa: https://www.w3.org/WAI/standards-guidelines/wcag/. Luettu: 3.5.2021

Liitteet

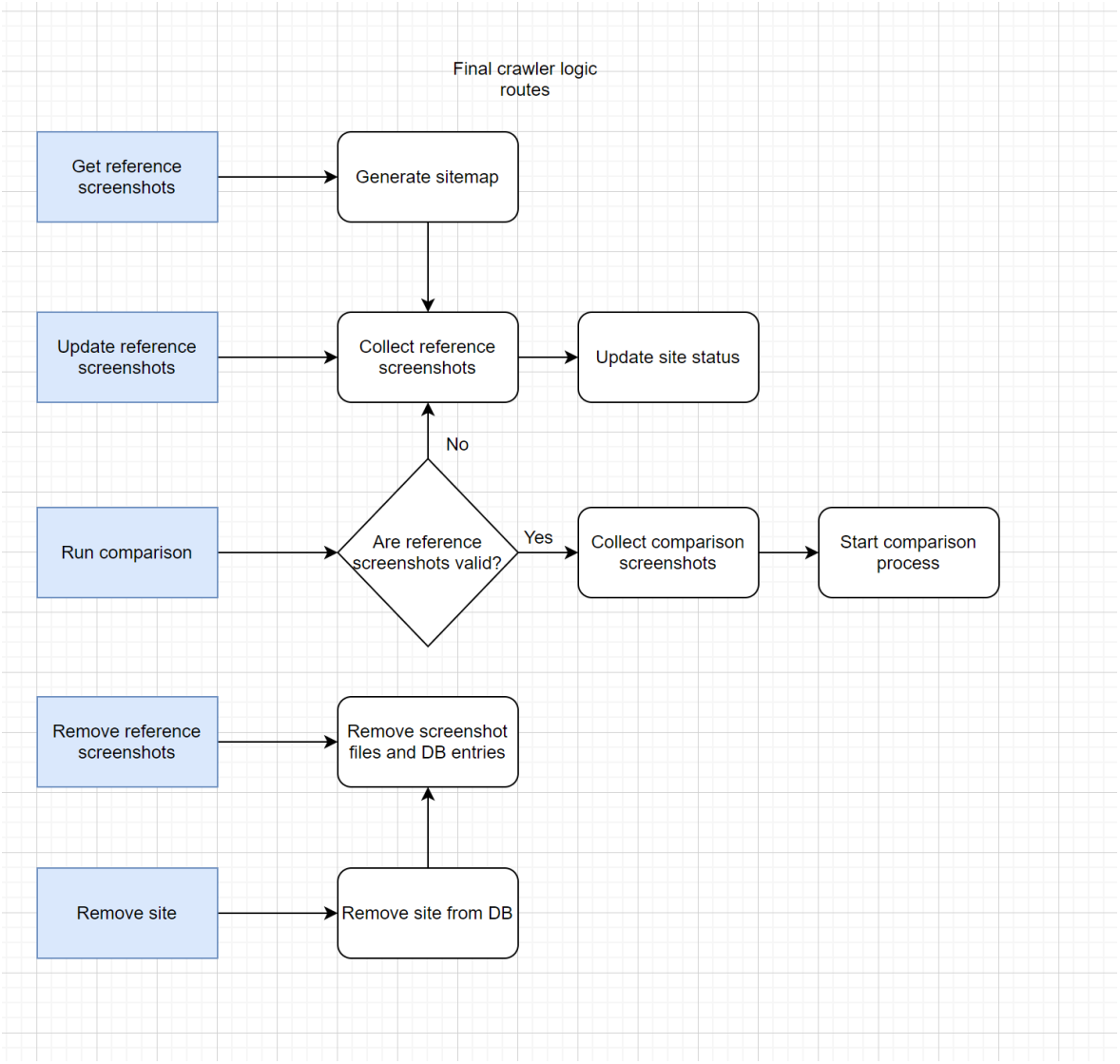
Liite 1. ART-Toolin tiedostorakenne ennen toteutusta



Liite 2. Kuvakaappausten keräämisen ja vertailun logiikkakaavio ennen toteutusta



Liite 3. Crawlerin lopullisen version eri ajokäskyt ja niiden mahdolliset reitit



Liite 4. ART-Toolin käyttötilastoja

Data ajalta 14.8.2020 – 6.12.2020

Loppuun asti onnistuneesti suoritettuja testiajoja on 459 kpl, joista:

4. 251 kpl on referenssikuvien keräämiseen
5. 208 kpl on testattavien kuvien + kuvaparien vertailuun

Kuvakaappauksia on kerätty yhteensä 22217 kpl

Keskimääräinen kuvakaappausmäärä per sivusto on 51

Työkalulla on testattu 85 eri verkkosovellusta.

Keskimääräinen referenssikuvien keräysajo on kestänyt 7 minuuttia ja 15 sekuntia, verrokkiajo 10 minuuttia 12 sekuntia.