OULUN AMMATTIKORKEAKOULU

Jyrki Ahtonen

MAGENTO 2:N TESTAUSAUTOMAATIO

MAGENTO 2:N TESTAUSAUTOMAATIO

Jyrki Ahtonen
Opinnäytetyö
Syksy 2018
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Jyrki Ahtonen
Opinnäytetyön nimi: Magento 2:n testausautomaatio
Työn ohjaaja: Timo Vainio
Työn valmistumislukukausi ja -vuosi: Syksy 2018
Sivumäärä: 49 + 8 liitettä

Opinnäytetyön aiheena oli suunnitella ja toteuttaa yrityksen ohjelmistokehityksen tarpeisiin testaussuunnitelma, luoda moduulitestejä yrityksen Magento 2 -verkkokauppa-alustan moduuleihin ja lisäksi perehtyä Magento 2 -alustan automaattisiin testausjärjestelmiin. Lopputuloksena tavoitteena oli saada yritykselle testaussuunnitelma tulevaisuuden testauksen jatkokehitystä varten ja dokumentaatio ja ohjeet testien tekemiseen.

Suunnittelu ja toteutus tehtiin projektiluontoisesti. Projektin alussa luotiin projektille projektisuunnitelma ja aikataulu. Ensimmäiseksi perehdyttiin testauksen teoriaan, minkä jälkeen opeteltiin käyttämään testauksessa käytettäviä työkaluja. Työkaluihin tutustumisen jälkeen aloitettiin moduulitestausten luonti ja dokumentointi. Lopuksi edettiin opettelemaan Magento 2 -verkkokauppa-alustan integraatio- ja järjestelmätestien suoritusta.

Projektin toteutus onnistui hyvin ja kaikki tavoitteet saavutettiin. Yritykselle saatiin jatkosuunnitelma ja dokumentaatio automaatiotestauksen kehittämiseen ja suorittamiseen. Testausautomaation kehitystä jatketaan projektin päättymisen jälkeen.

Asiasanat: ohjelmistokehitys, testaus, automatisointi, Magento, verkkokauppa

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Software Engineering

---

Author: Jyrki Ahtonen
Title of thesis: Magento 2 testing automation
Supervisor: Timo Vainio
Term and year when the thesis was submitted: Autumn 2018
Pages: 49 + 8 appendices

The subject of this thesis was to design and implement test plan for the company's software development processes, create unit tests for the Magento 2 webstore modules and learn how to use Magento 2 testing automation.

Designing and implementing in this project was done in a project-oriented way. In the beginning of the project, a project plan and a schedule for the project was produced. At first, the theory of testing and testing tools was explored and learned. After that the module testing and documentation was started. The final step was to get to know about Magento 2 integration and system testing.

Project was completed successfully, and all the goals were achieved. Follow-up plan and documentation for the testing automation testing and development was obtained. Testing automation development will be continued after the thesis project.

Keywords: software development, testing, automation, Magento, webstore

SISÄLLYS

SANASTO

Luokka	Luokka tarkoittaa objektin tyyppiä (ilmentymää) ja siinä määritellään objektin toiminnallisuus
Metodi	Ohjelmoinnissa käytettävä aliohjelma. Ohjelmoinnissa itsenäinen osa, joka suorittaa tietyn toiminnon ja jota voidaan kutsua eri puolilta pääohjelmaa tai muista aliohjelmista
Mock-objekti	Testauksessa käytettävä objekti, jonka metodit tarjoavat vastauksia vain testeissä ennalta määrätyillä parametreilla. Lisäksi määritellään, mitä odotuksia metodeilta kutsutaan ja kuinka usein
Objekti	Ohjelmiston perusyksikkö, joka myös tunnetaan nimellä olio. Sisältää joukon loogisesti yhteenkuuluvaa tietoa ja toiminnallisuutta
Refaktorointi	Ohjelmakoodin luotettavuuden, käytettävyyden ja rakenteen parantaminen uudelleenmuokkaamalla
Stub-objekti	Testauksessa käytettävä objekti, jolle testaaja määrittää oletetut kutsut sekä vastaukset niihin
Testitapaus	Yksi ominaisuuden testattavista tapauksista, jossa tietyillä syötteillä odotetaan tietynlaista tulosta

1 JOHDANTO

Ohjelmistotestaus on erittäin tärkeä osa ohjelmistotuotantoa. Testauksen avulla pystytään varmistumaan ohjelmiston laadusta ja virheettömyydestä. Testauksen avulla voidaan osoittaa, että ohjelmisto täyttää kaupalliset ja tekniset vaatimukset, ohjelmisto toimii oletetusti ja suunnitelmien mukaisesti ja ohjelmisto voidaan ottaa käyttöön halutuilla ominaisuuksilla.

Tämän opinnäytetyön ideana oli suunnitella ja toteuttaa automaattinen ohjelmistotestaus ja testaussuunnitelma tilaajayritykselle. Ohjelmistoalustana käytössä oli Magento 2 -verkkokauppa-alusta.

Yrityksellä ei ollut käytettävissä testaussuunnitelmaa, joten suunnitelman suunnittelutyö aloitettiin alusta. Magento 2 -ohjelmisto sisältää testausohjelmistoa, jonka hyödyntämistä haluttiin lähteä kehittämään. Työn tarkoituksena myös oli opetella luomaan uusia testejä.

2 OHJELMISTOTESTAUS

Ohjelmistotestauksen avulla pyritään tuottamaan ohjelmistoja, jotka täyttäisivät niille asetetut vaatimukset ja joiden toiminta olisi virheetöntä. Ajoissa suunniteltu ja toteutettu testaus parantaa ohjelmiston laatua ja vähentää kustannuksia. Ohjelmistotestauksen tulisi aina kuulua erittäin oleellisena osana ohjelmistotuotantoprosessia jo vaatimusmäärittelystä alkaen. Ohjelmistotestaus voi olla yhtä monimutkainen kokonaisuus kuin ohjelmisto itse, minkä takia sen huomioiminen määrittelyvaiheessa on tärkeää. (1, s. 12; 2, s. 4.)

Testauksen tavoitteena on suorittaa ohjelmistolle mahdollisimman kattava testaukset, jotta ohjelmistokoodissa olevat virheet löydettäisiin ja havaittaisiin mahdollisimman aikaisin. Testauksen avulla voidaan varmistaa, että ohjelmiston vaatimusten mukainen toiminto toteuttaa sille annetut vaatimukset. (1, s 12.)

Miksi kaikki eivät testaa ja miksi testaamista ei aloiteta riittävän ajoissa? Tyypillisin väite on, että testaaminen vie liikaa aikaa. Todellisuudessa integrointitestauksen aikana havaittujen virheiden korjaaminen vaatii aikaa moninkertaisesti enemmän kuin ajoissa aloitettu moduulien yksittäinen testaaminen. Parhaatkin ohjelmoijat tekevät virheitä, joten kaikkien ohjelmoijien olisi hyvä opetella hyödyntämään testausta. (3, s. 10.)

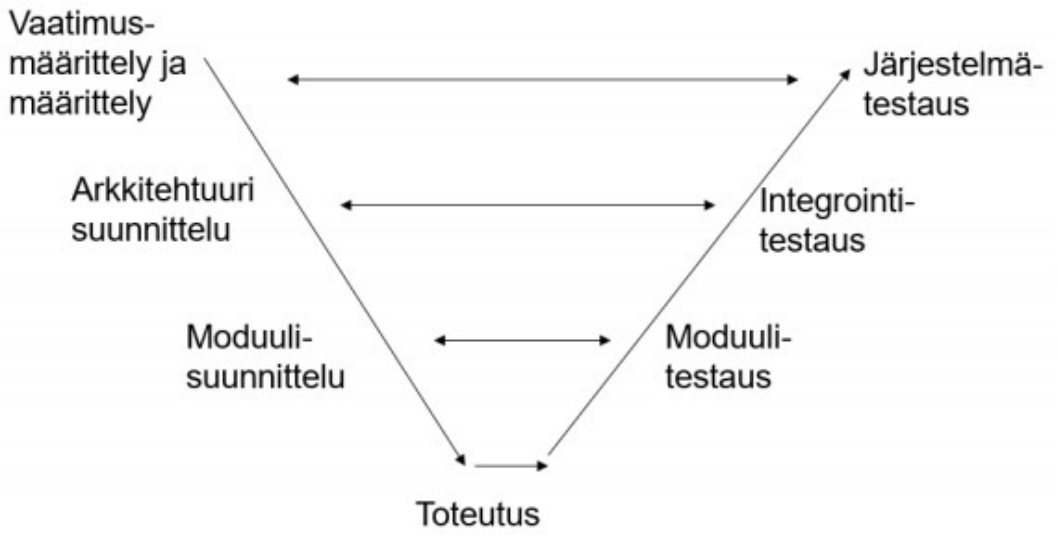
2.1 Testauksen suunnittelu

Testauksen suunnittelu aloitetaan testaussuunnitelman laatimisella. Suunnitelman avulla varmistetaan testauksen kattavuus, tiedetään, minkälaisia lopputuloksia odotetaan ja järjestetään testaus eri vaiheisiin. (2, s. 28.)

Suunnittelu sisältää erilaisten testitapausten suunnittelun, testausmenetelmien valinnan ja testiaineistojen laadinnan. Suunnittelun aikana tulee huomioida sekä ohjelmistolle asetetut vaatimukset että mahdolliset olosuhteet ja tilanteet. Huolellisella suunnittelulla voidaan varmistaa testattavan ohjelmiston toimintovarmuus ja vaatimusten mukainen toiminta. (1, s. 12; 2, s. 28.)

2.2 Testaustasot ja vaiheet

Perinteisessä ohjelmistokehityksessä testitapaukset jaotellaan V-mallin mukaisiin vaiheisiin (kuva 1). Vaiheisiin jakamisella parannetaan testausprosessin hallittavuutta. Aluksi suunnitellaan koko kokonaisuuden testaus ja tämän jälkeen seuraavalla tasolla perusteena ovat edellisen ja kyseisen tason suunnittelun määritykset. Esimerkiksi moduulitestausta suunniteltaessa käytössä ovat moduulisuunnittelun, arkkitehtuurisuunnittelun ja vaatimusmäärittelyn määritelmät. Itse testaus tapahtuu vastaavasti käänteisessä järjestyksessä suunnitteluun nähden. Testausosio V-mallissa jaotellaan moduulitestaukseen, integrointitestaukseen ja järjestelmätestaukseen. (2, s. 13.)



KUVA 1. Testauksen V-malli (1, s. 14)

Jokainen testausvaihe suoritetaan kokonaisuudessaan ennen seuraavaan vaiheeseen siirtymistä niin, että testausvaiheessa olevat testit eivät havainneet ohjelmakoodissa vikaa tai virheellistä toimintaa. Jos virheitä tai vääränlaista toimintaa ilmenee, tuotetaan virheistä testiraportti, jonka avulla ohjelmistosuunnittelija voi paikantaa virheet ja tehdä tarvittavat muutokset virheiden korjaamiseksi. Virheiden korjaamisen jälkeen suoritetaan kaikki virheitä edeltäneet testitapaukset uudelleen, jotta voidaan varmistua siitä, että tehdyt muutokset eivät aiheuttaneet uusia virheitä alemman tason testeissä. (1, s. 13–14.)

2.2.1 Moduulitestaus

Moduulitestaus eli yksikkötestaus on ensimmäinen vaihe testauksessa. Moduulitestauksessa testataan yksittäisen moduulin toimintoja ja metodeita. Vastuussa testauksesta on moduulitestauksen määrittelystä ja suunnittelusta vastaava ohjelmistosuunnittelija, koska suunnittelija tuntee parhaiten ohjelmakoodin, sen rakenteen ja tavoitellut toiminnot ja tulokset. (1, s. 14.)

Moduulitestausta toteutetaan tynkäobjektien avulla (stub ja mock). Tynkäobjektit simuloivat ohjelman ympäristöä ja esittävät aliohjelmaa, jota testattava moduuli kutsuu. Moduulitestauksessa koodi jaotellaan paloittain ja testataan jokainen osa erikseen. Seuraavaan vaiheeseen voidaan siirtyä vasta, kun edellinen vaihe toimii virheettä. (2, s. 17.; 3, s. 15.)

2.2.2 Integrointitestaus

Integrointitestauksen tarkoituksena on testata moduulien tai moduuliryhmien välisiä rajapintoja, jotta voidaan todeta niiden yhteistoimivuus. Tarkoitus on löytää mahdolliset ongelmakohdat, joissa moduulit eivät yhdessä toimi oikein. Testauksen avulla voidaan estää, että moduulien rajapinnoilla ei häviä tarvittavaa tärkeää tietoa. (2, s. 16.)

Virheitä voi syntyä, jos moduulin suunnitteluvaiheessa ei ole osattu ottaa huomioon tilannetta, jossa toinen moduuli kutsuu moduulin metodia ja metodista puuttuukin oikeanlaiset tarkistukset sinne tulevasta tiedosta. Integrointitestauksen avulla löydetään tämän kaltaiset virheet. (2, s. 16.)

Integrointitestauksen suunnittelijan täytyy tuntea testattavien moduulien ja kokonaisuuksien väliset rajapinnat, odotettu toiminta ja tulokset. Testauksen jälkeen havaitut virheet raportoidaan moduulin kehittäneelle ohjelmistosuunnittelijalle, jotta virheet voidaan paikantaa ja korjata. Muutostöiden jälkeen aloitetaan testaus alusta, jotta voidaan varmistaa, että muutokset eivät aiheuta uusia virheitä. (1, s. 15.)

2.2.3 Järjestelmätestaus

Järjestelmätestauksen tarkastelun kohteena on koko järjestelmä, joka voi olla esimerkiksi ohjelmisto, laitteisto tai tietokanta. Testauksen tuloksia verrataan määrittelydokumentaatioon. Järjestelmätestauksen suunnittelijan tulee tuntea koko ohjelmiston vaatimukset, jotta voidaan suunnitella ja toteuttaa riittävän kattava testaus. (1, s. 15; 2. s. 14.)

Järjestelmätestaukseen kuuluvat myös ei-toiminnallisten vaatimusten mukaisia testauksia. Ei-toiminnallisia testejä ovat esimerkiksi kuormitustestit, volyymitestit, turvallisuustestit ja luotettavuustestit. Kuormitustesteissä mitataan, kuinka suurta kuormitusta järjestelmä kestää, ja tunnistetaan kuormitushuiput, joissa järjestelmä voi epäonnistua käsittelemään tietoa. Volyymistesteissä testataan, pystyykö järjestelmä käsittelemään riittävää määrää tietoja ja pyyntöjä. Turvallisuustesteillä voidaan osoittaa, että järjestelmä on esimerkiksi tietoturvallinen käyttää. Luotettavuustestauksen avulla saadaan tieto, täyttääkö järjestelmä sille asetetut luotettavuusvaatimukset, kuten esimerkiksi kuinka usein järjestelmä on käytettävissä. (2, s. 15.)

2.3 Testauksen laatu ja seuranta

Testauksen laatua voidaan mitata sillä, kuinka kattavasti ohjelmiston eri toiminnot on testattu. Täydellistä testausta on mahdotonta toteuttaa, koska tämä tarkoittaa, että kaikki ohjelmiston mahdolliset reitit on tutkittu kaikilla mahdollisilla syötteillä jokaisessa mahdollisessa ympäristössä. Testauksen laatua voidaan parantaa automatisoinnilla. Suoritettavat testit voidaan ajaa automatisoidusti aina ohjelmakoodin muutosten yhteydessä tai ajastaa ajamaan automatisoidusti esimerkiksi öisin. Testauksen automatisoinnista kerrotaan enemmän luvussa 3. (1, s. 16; 2, s. 6.)

Ohjelmistotestauksen seurannan tarkoituksena on tarkkailla testien havaitsemia virheitä, virheiden määrien muutoksia ja sitä, missä vaiheessa virheet ilmenevät. Näitä tuloksia analysoimalla voidaan seurata testausprosessin kehittymistä sekä ohjelmistosuunnittelijoiden tuottamien virheiden määrä. Seurannan avulla osataan ongelmien ilmetessä tällöin nostaa tiettyjen testivaiheiden määrä. (1, s. 16.)

2.4 Testivetoinen kehitys

Testivetoinen kehitys (test-driven development, TDD) on tekniikka, joka tukee ohjelmointia. Tekniikassa luodaan ensin uusi testitapaus ja sen jälkeen vasta muokataan kehitettävää ohjelmaa niin, että se läpäisee uuden testin. Moduulitestit kirjoitetaan pienissä osissa ennen varsinaista tuotantokoodia. Testivetoisen kehitystavan avulla pyritään parempaan rajapintasuunnitteluun ja myös varmistumaan ohjelmiston oikeasta toiminnasta. Mikäli moduulitestit aiottaisiin kirjoittaa tuotantokoodin jälkeen, ne hyvin todennäköisesti jäisivät usein tekemättä. (4, s.10-11.)

Kun testikoodi kirjoitetaan etukäteen, tuloksena saadaan jatkuvasti ohjelmointikehityksen aikana kehittyvä testiverkosto, jonka varassa uusien toimintojen kehittäminen ja virheiden korjaaminen on huomattavasti turvallisempaa. Olemassa olevia testejä suorittamalla huomataan, mikäli virhettä korjattaessa tulee tehneeksi mahdollisesti uusia virheitä. Testivetoinen kehitys ei ole testausmenetelmä, vaan suunnittelumenetelmä, vaikka sivutuotteena syntyykin joukko käyttökelpoisia testejä. (4, s.188-189.)

3 TESTAUKSEN AUTOMATISOINTI

Automatisointi on manuaalisten testien suorittamista koneellisesti testausohjelman avulla. Testausautomaatio vaatii yksityiskohtaiset testitapaukset sekä vaatimusmäärittelyihin ja suunnitteludokumentteihin perustuvat odotetut tulokset. Automaatiota varten tulee luoda erillinen testausympäristö, jossa sillä on oma tietokanta. (3, s. 23.)

Automaattisella ohjelmistotestauksella on tarkoitus parantaa ohjelmiston luotettavuutta ja laatua. Testausautomaatiolla pystytään vähentämään riskejä, koska automatisoidut testit ajetaan aina samalla tavalla. Manuaalisessa testauksessa virheitä syntyy ihmisten tekemien virheiden takia. Automatisoinnilla voidaan myös tuoda säästöjä testauksen toteuttamiseen ja vähentää testien suorittamiseen kuluvaa aikaa. Olemassa on myös testejä, joita voidaan pitää käytännössä mahdottomina suorittaa manuaalisesti. (1, s. 23; 5, s. 7.)

3.1 Automatisoinnin suunnittelu

Automatisoinnin suunnittelussa tavoitteena ja tarkoituksena on tarkastella jokainen testitapaus yksittäin ja selvittää onko testitapaus automatisoitavissa ja onko se kannattavaa. Testejä ei kannata automatisoida, jos ne ajetaan todella harvoin tai tuloksien tarkistaminen manuaalisesti ihmisiltä onnistuu helposti, mutta ohjelmallisesti se on todella hankalaa. Suunnittelussa tulee huomioida myös manuaalisen testauksen automatisointiin kuluva aika, joka on 3–10 kertaa manuaalista toteutusta suurempi. (1, s. 23; 5, s. 7.)

3.2 Automatisoinnin suoritus

Kun suunnittelu on saatu valmiiksi, testit voidaan suorittaa toimivan testausympäristön avulla siinä aikataulussa, kuin testaussuunnitelmassa on määritelty. Kaikki eri testausvaiheet, moduuli-, integraatio- ja järjestelmätestaus, kuuluvat suoritus vaiheeseen. Yhdessä ne muodostava koko systeemin testauksen. Löydettyjen virheiden määrä suhteessa suoritettujen testien määrään, on yksi tapa mitata testauksen tehokkuutta. Testaaminen voidaan lopettaa, kun todetaan, että määritellyt vaatimukset ja hyväksymiskriteerit on saavutettu. (3, s.36.)

4 MAGENTO-VERKKOKAUPPA

Magento on avoimen lähdekoodiin perustuva verkkokauppa-alusta, joka on kehitetty PHP-ohjelmointikielellä. Ensimmäinen Magento julkaistiin vuonna 2008 ja sen on luonut Varien, jonka toimitusjohtaja Roy Rubin myi Varienin myöhemmin eBaylle. Magentosta julkaistiin vuonna 2015 Magento 2 -versio. Vuonna 2018 Adobe osti Magenton 1,68 miljardilla dollarilla. Magento mukautuu helposti kaiken kokoisiin verkkokauppoihin ja siinä on kaikki tarvittava verkkokaupan perustamiseen. (6; 7.)

Magentosta on saatavilla open source- ja commerce-versiot. Commerce-versio on lisenssimaksullinen versio ja se sisältää lisäominaisuuksia sekä Magenton tarjoaman virallisen tuen. (6.)

Magenton modulaarinen rakenne mahdollistaa sen monipuolisen jatkokehittämisen ja kolmannen osapuolen palveluiden käyttämisen. Magentossa voidaan hallita montaa kauppaa yhtä aikaa samasta paikasta eli Magenton hallinnasta. Yhdelle kaupalle pystytään luomaan useampi kauppanäkymä, jonka avulla on toteutettavissa eri kieliversiot tai kokonaan erilainen ulkoasu ja valikoimalla toimivat alikaupat. Magenton avulla on mahdollista luoda yhdistetty verkkosivu ja verkkokauppa, johon voidaan tuottaa kaikki yrityksen markkinointimateriaali ja tuotteet. (6; 7.)

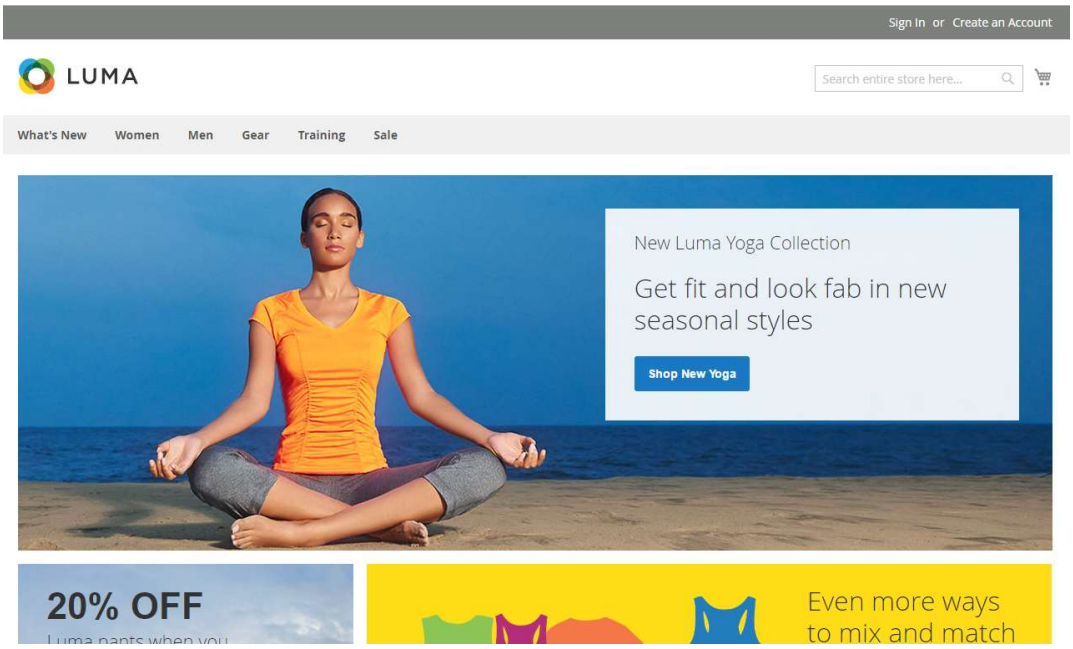
4.1 Järjestelmän näkymät

Magento perustuu kahteen eri näkymään, kuten monet muutkin verkkokauppajärjestelmät. Nämä näkymät ovat Frontend-näkymä eli asiakkaan näkymä ja Backend-näkymä eli hallintapuolinäkymä. (7.)

4.1.1 Asiakkaan näkymä

Asiakkaan näkymässä asiakas voi selata tuotekategorioita, ostaa tuotteita ja lukea kauppiaan tekemiä uutisia tai blogeja (kuva 2). Kauppias voi helposti erilai-

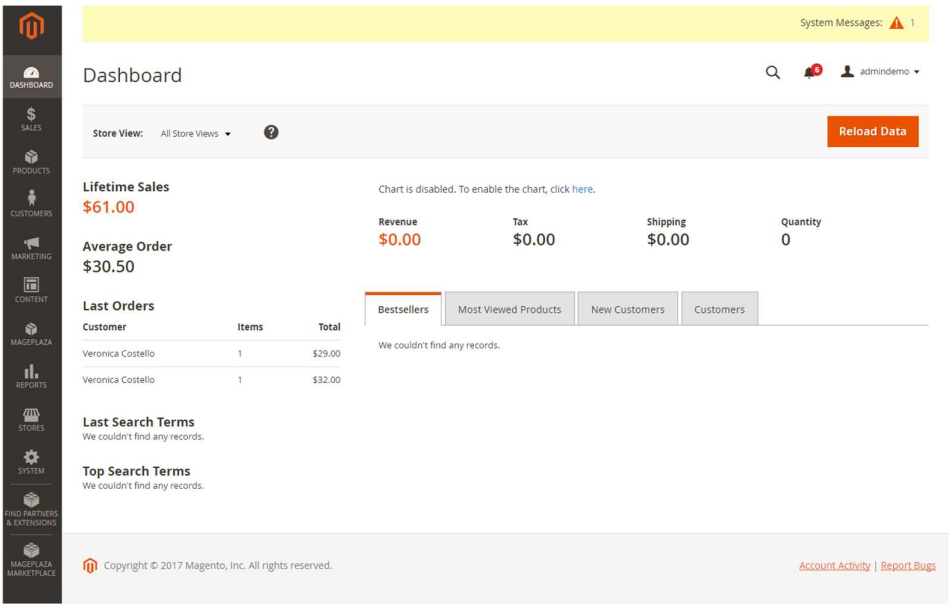
silla lisäosilla tuoda esimerkiksi sosiaalisen median julkaisujaan näkyville asiakkaan näkymään. Mahdollista on myös tuoda esille esimerkiksi myydyimpiä tuotteita tai tuotekortille tuotteeseen liittyviä lisämyyntituotteita.



KUVA 2. Magento 2 demokaupan etusivu

4.1.2 Hallintapaneeli

Hallintapaneelissa kauppiaalla on kätevästi nähtävissä myynnit, tilaukset, suosituimmat hakusanat, myydyimmät tuotteet ja paljon muuta (kuva 3). Hallintapaneelin kautta pystytään luoda uusia ja muokata jo olemassa olevia tuotteita ja kategorioita. Paneelin kautta hallitaan kaupan asetuksia. Sieltä laaditaan eri kieliversioita, määritetään valuutta, asetetaan verosääntöjä, luodaan ja muokataan asiakastietoja ja hallita sisältöä, joka näkyy asiakkaalle. Hallintapaneeli on Magenton ydin, josta kaikkea voidaan säätää ja muokata. (7.).



KUVA 3. Magento 2 hallintapaneeli

4.2 Moduulit

Lisäosien eli moduulien avulla Magento-verkkokaupasta saadaan muokattua omien tarpeiden mukainen. Moduulien avulla on mahdollista tuoda uutta liiketoiminnallisuutta verkkokauppaan. Moduuleilla voidaan luoda uusia toiminnallisuuksia tai muokata jo olemassa olevia. Tästä hyvänä esimerkkinä on uusien maksu- ja toimitustapojen luonti. Laajan yhteisötuen vuoksi moduuleita on saatavilla paljon myös ilmaiseksi, mikä on hieno asia juurikin avoimen lähdekoodin ohjelmistoissa. Useilla sivustoilla kehittäjät antavat neuvoja toisilleen, minkä ansiosta aloittelevan Magento-ohjelmoijan on helppo löytää apua omiin ongelmiin. (8.)

5 TYÖKALUT

Projektin aikana oli tarkoitus hyödyntää jo opittuja ja oppia käyttämään uusia ohjelmointia ja testaamista auttavia työkaluja. Työkalujen valinnassa projektia varten ehdoton kriteeri oli niiden tehokkuus ja maksamattomuus. Kun työkalut ovat ilmaiseksi saatavilla, kaikilla tätä dokumentaatiota käyttävillä on mahdollista päästä samanlaiseen lopputulokseen.

5.1 Ubuntu

Ubuntu on käyttöjärjestelmä, joka perustuu Debian Linux-jakeluun. Ubuntu on saatavilla avoimen lähdekoodin vapaana ohjelmistona. Käyttöjärjestelmän kehitystä rahoittaa Canontical Ltd. (8.)

Linux-pohjainen käyttöjärjestelmä on paras valinta, kun käyttöjärjestelmällä täytyy luoda omia palvelimia. Sen ominaisuudet ovat optimoitu parhaiten juuri tällaiseen käyttötarkoitukseen ja sen komentorivi on yksi tehokkaimmista työkaluista, mitä Linux pystyy tarjoamaan.

5.2 PhpStorm

PhpStorm on JetBrainsin kehittämä ohjelmointiympäristö, joka tarjoaa kattavan editorin kieliin kuten PHP, HTML, JavaScript ja CSS. Ohjelma tukee myös tietokantojen hallintaa ja versionhallintaa. (9.)

PhpStorm-koodieditori valittiin, koska se on käytettävyydeltään ja ominaisuuksiltaan paras, mitä PHP-ohjelmointiin on löydettävissä. Se on monipuolinen ja auttaa ohjelmoijaa työssään. Lisäksi editoriin on mahdollista ladata erilaisia ohjelmointia tukevia lisäosia.

5.3 PHPUnit

PHPUnit on yksikkötestaukseen käytettävä ohjelmistokehys PHP-kielelle. Idea on, että kehittäjät löytävät nopeasti ja helposti virheitä koodista. Työkalun avulla on helppo testata vaikuttaako uusi koodi jo olemassa oleviin osiin. Sen on luonut Sebastian Bergman. (10.)

PHPUnit oli erittäin hyvä valinta, koska sitä oli mahdollista käyttää niin komentoriviltä kuin myös PhpStorm-editorista. Internetistä löytyvät lukuisat oppaat ja foorumit auttoivat työkalun käytössä.

5.4 Selenium

Selenium on avoimen lähdekoodin ohjelmisto, joka tarjoaa ohjelmistotestauksen työkalut verkkosovelluksille. Selenium on mahdollista saada Windows-, Linux- ja macOS-käyttöjärjestelmille ja se tukee ohjelmointikieliä C#, Groovy, Java, Perl, PHP, Python, Ruby ja Scala. Seleniumin kehitti Jason Huggins vuonna 2004. (11.)

Selenium tuli valituksi projektin aikana, koska ohjeita löytyi Magenton omilta sivuilta ja se on käytetyin järjestelmätestauksessa käytettävä ohjelmisto. Selenium-ohjelmistoa on myös mahdollista käyttää yhtä aikaa usean selaimen kanssa.

6 MODUULITESTAUKSEN SUUNNITTELU JA TOTEUTUS

Opinnäytetyötä lähdettiin tekemään projektiluontoisesti. Aluksi tehtiin kartoitus, minkälaisia testausjärjestelmiä Magentosta löytyy. Tämän jälkeen käytiin läpi yrityksessä, mitä osa-aluetta lähdetään tutkimaan ja kehittämään. Kun aihe ja kehittävä kohde olivat selvillä, pidettiin projektin aloituspalaveri minun, ohjaavan opettajan ja yrityksessä opinnäytetyöstä vastaavan henkilön kanssa. Aloituspalaverissa luotiin lähtötietomuistio (liite 1), jonka perusteella opinnäytetyön toteutus voitiin aloittaa.

Projektin suunnittelu aloitettiin tekemällä projektisuunnitelma, jossa käytiin läpi projektin aloitusasetelmat ja taustat. Suunnitelmassa avattiin Magentossa käytettäviä testausmenetelmiä ja esiteltiin projektin organisaatio. Projektisuunnitelman yhteydessä luotiin projektille aikataulu sekä raportointisuunnitelma.

6.1 Moduulitestauksen suunnittelu

Aluksi suunniteltiin moduulin käyttötarkoitus ja sen toiminnallisuus. Selvitettiin, tarvitaanko ulkoisia rajapintoja tai tarvitseeko moduuli toimiakseen jotakin toista jo olemassa olevaa moduulia. Moduuliin haluttiin vain luoda testattavaa toiminnallisuutta, jotta moduulitestauksesta tulisi mahdollisimman kattava.

6.2 Moduulitestauksen toteutus

Moduulitestauksessa pyritään siihen, että jokaista testattavan luokan metodia kohti on vähintään yksi testitapaus. Moduuli olisi hyvä ohjelmoida testivetoista kehitysmenetelmää käyttäen.

Testivetoisessa kehitysmenetelmässä aluksi kirjoitetaan testi, joka ei mene läpi, koska vaadittu toiminnallisuus puuttuu. Tämä testitapaus testaa ainoastaan yhden pienen asian. Tämän jälkeen luodaan toiminnallisuus, joka läpäisee testitapauksen. Lopuksi koodia refaktoroidaan eli parannetaan koodin laatua ja rakennetta. Kun koodin rakenne on kunnossa, aloitetaan sykli alusta.

Koodi kirjoitetaan täyttämään testin sisältämät vaatimukset. Testien avulla voidaan määrittää, kuinka kyseisen luokan tulisi toimia. Testivetoisessa kehitysmenetelmässä testaus ohjaa kehitystyötä eikä ole erillinen toteutuksen jälkeinen laadunvarmistusvaihe. Moduulin toiminnallisuus on valmis, kun kaikki testit menevät läpi.

Kun moduulin toiminnallisuutta myöhemmin muokataan tai toiminnallisuutta lisätään, testien avulla voidaan heti varmistua, toteuttavatko luokan metodit yhä testeissä vaaditut testitapaukset. Moduulin jatkokehitys tulee tehdä testivetoista kehitysmenetelmää hyväksikäyttäen, jotta testit pysyvät ajan tasalla.

7 MAGENTO 2:N MODUULITESTAUS

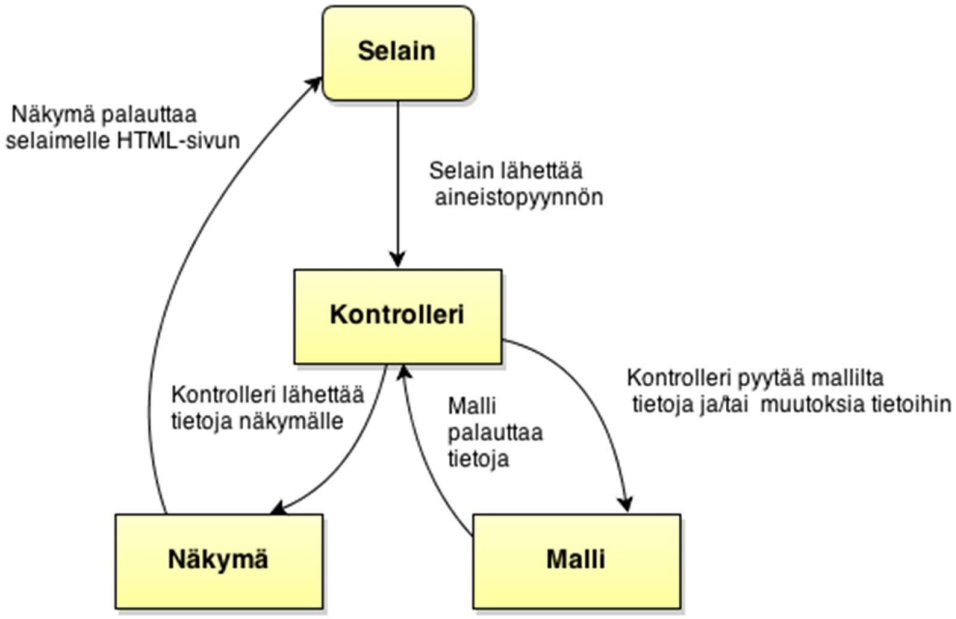
Tässä opinnäytetyössä ei käydä läpi, kuinka Magento 2 asennetaan, vaan perehdytään suoraan moduulien ohjelmoimiseen olettaen, että verkkokauppa on jo asennettu ja että tarvittavat työkalut kuten PphStorm ja PHPUnit ovat asennettuina.

Aluksi luotiin tarvittavat tiedostot moduulia varten, minkä jälkeen oli mahdollista aloittaa moduulin kehitys testivetoista kehitysmenetelmää käyttäen.

Tässä opinnäytetyössä moduulin kehityksessä käytettiin erityisesti tätä projektia varten luotua moduulia, jonka avulla on tarkoitus neuvoa ja ohjeistaa, kuinka moduulitestausta tulee ohjelmoida.

7.1 Magenton tiedostorakenne

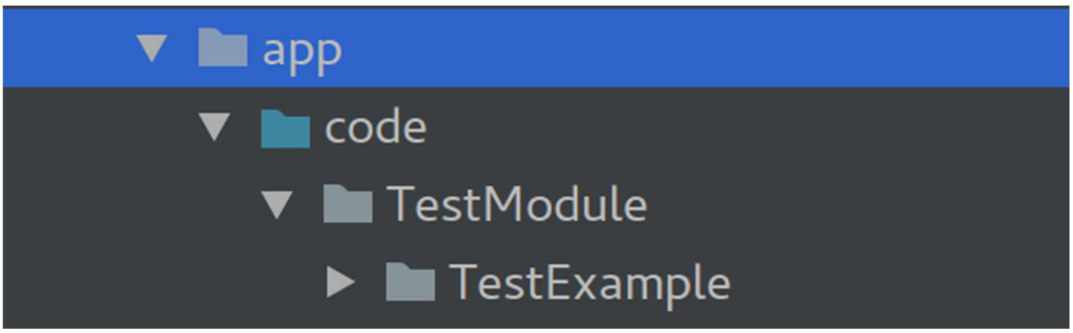
Magentossa käytetään MVC (Model, View, Controller) -ohjelmistoarkkitehtuurityyliä (kuva 4), jonka tarkoituksena on käyttöliittymän erottaminen sovellusaluetiedostoista. Mallia (model) käytetään järjestelmän tiedon tallentamiseen, ylläpitoon ja tietojen käsittelyyn. Näkymän (view) avulla määritetään käyttöliittymän ulkoasu ja tietojen näyttö käyttöliittymässä. Käsittelijä (controller) eli kontrolleri vastaanottaa käskyjä jotka tulevat käyttäjältä, muuttaa mallia ja näkymää vastauksena otettuihin käskyihin, kuten esimerkiksi kun käyttäjä vaihtaa sivua Magento-verkkokaupassa.



KUVA 4. MCV-mallin vastuunjako

7.1.1 App-kansio

Magento-asennuksen juuresta löytyy kansio nimeltä app. App-kansion alla voidaan kehittää helposti omia moduuleita ja tässä opinnäytetyössä perehdyttiin ainoastaan App-kansion alla kehitykseen. Ensimmäiseksi luotiin Magenton app/code-kansiorakenteen alle oma nimiavaruus (namespace), jossa yleensä käytetään yrityksen nimeä. Tässä projektissa esimerkeissä nimiavaruutena käytetiin TestCompany-nimeä. Tämän nimiavaruuden alle sijoitetaan omat moduulit. Moduulin nimeksi projektissa tuli TestModule (kuva 5).



KUVA 5. Kansiorakenne moduulia varten

Kun moduulin kansiot oli luotu, luotiin rekisteritiedosto registration.php, joka tulee moduulin kansion alle (liite 2). Rekisteritiedostossa tuli määrittää moduulin nimiavaruus kokonaisuutena eli tässä tapauksessa TestCompany_TestModule.

```
<?php
    \Magento\Framework\Component\ComponentRegistrar::register(
        \Magento\Framework\Component\ComponentRegistrar::MODULE,
        'TestCompany_TestModule',
        __DIR__
    );
```

7.1.2 Etc-kansio

Seuraavaksi luotiin moduulin sisään kansio nimeltä etc. Etc-kansio alle luodaan moduulin kokoonpanotiedostot. Etc-kansio luotiin module.xml-tiedosto (liite 2).

```
<?xmlversion="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/mod-
ule.xsd">
    <module name="TestCompany_TestModule" setup_version="0.0.1"/>
</config>
```

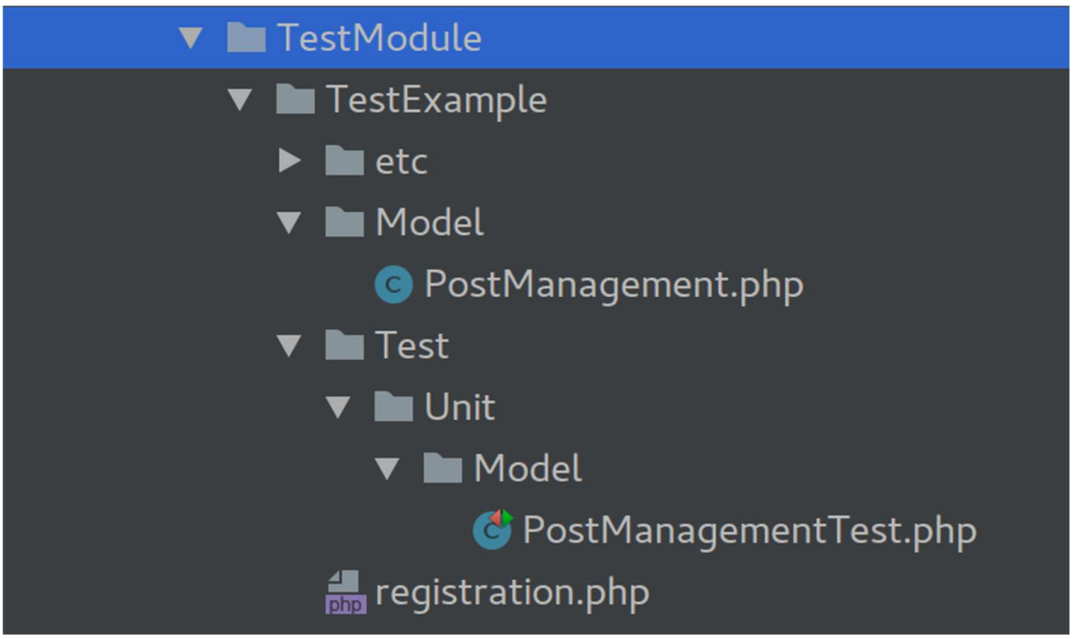
Moduulin xml-tiedosto on hyvin yksinkertainen ja siinä määritetään moduulin nimi ja versio.

7.1.3 Model-, Test- ja ResourceModel -kansiot

Model-kansio sisältää moduulin mallitiedostot. Kansioon luotiin mallitiedosto nimeltä PostManagement.php (liite 3). Tähän tiedostoon alettiin luomaan koodia TDD-kehitysmenetelmää käyttäen.

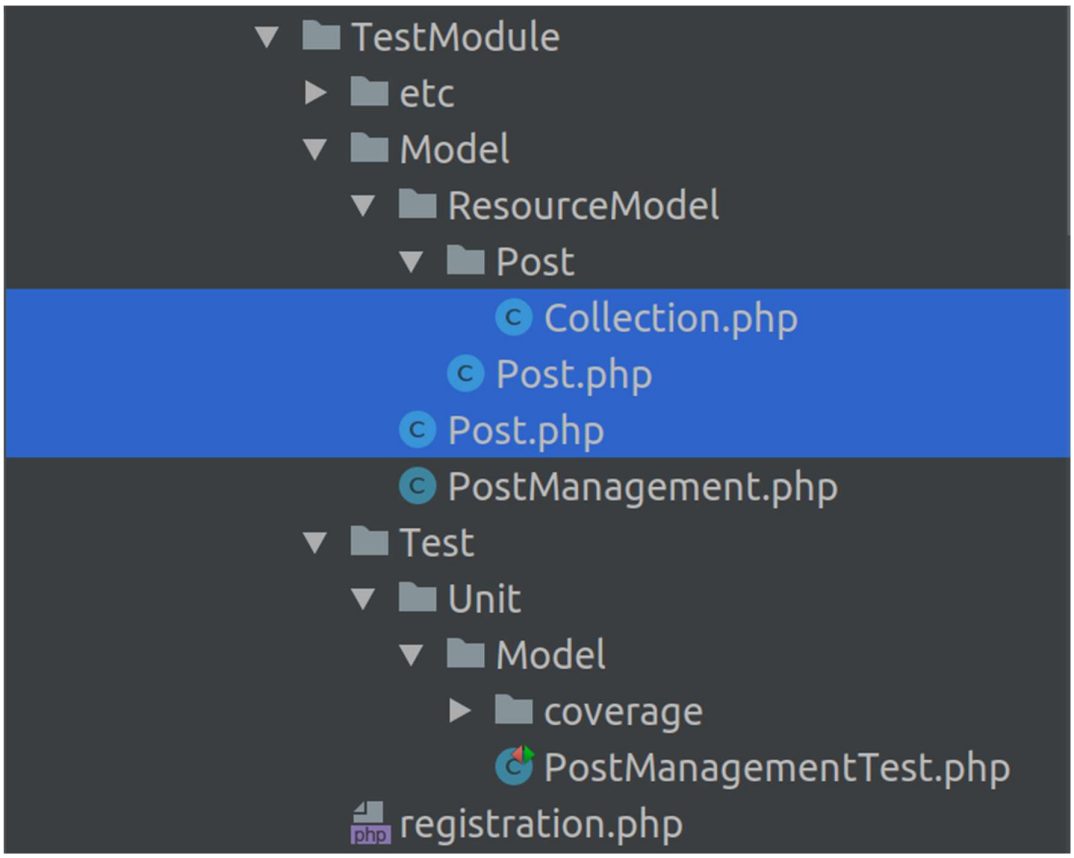
Ennen koodin luomista täytyi luoda Test-kansion alle testitiedosto, joka testaa luotua mallitiedostoa. Test-kansio alle luodaan kaikki testauksessa käytettävät tiedostot. Test-kansion alle luotiin alikansio nimeltä Unit, jonne moduulitestauksessa käytettävät tiedostot laitettiin. Testitiedosto täytyi nimetä PostManagementTest.php nimellä, jotta tiedettiin, mitä luokkaa kyseinen tiedosto testasi (liite

4). Selkeyden vuoksi tiedosto laitettiin TestCompany\TestModule\Test\Unit\Model-kansiorakenteen alle (kuva 6).



KUVA 6. Kansiorakenne malli- ja testitiedoston luonnin jälkeen

Magentossa käytetään kokoelmia hakemaan tietoa esimerkiksi tuotteista ja kategorioista. Kokoelmia on mahdollista luoda myös omista tietokantatauluista. Testejä varten luotiin omat kokoelmatiedostot, jotta testien aikana oli mahdollista harjoitella kokoelmien testausta. Kokoelman käyttöä varten tuli luoda TestCompany\TestModule\Test\Unit\Model-kansioon Post.php-tiedosto (liite 5), TestCompany\TestModule\Test\Unit\Model\ResourceModel-kansioon Post.php-tiedosto (liite 5) ja TestCompany\TestModule\Test\Unit\Model\ResourceModel\Collection-kansioon Collection.php-tiedosto (liite 5). Moduulin lopullinen kansiorakenne on esitetty kuvassa 7. Sinisellä merkityt tiedostot ovat uusia kokoelmaa varten luotuja tiedostoja.

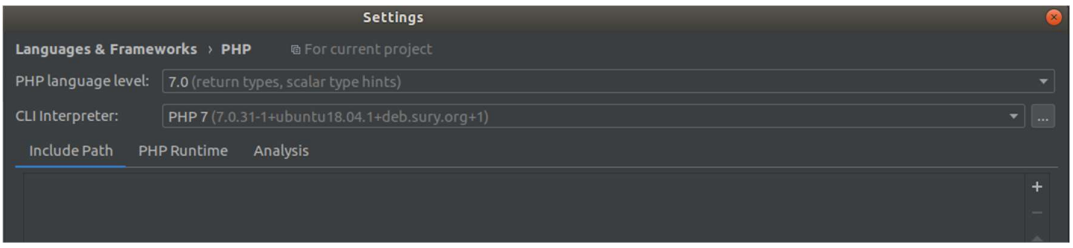


KUVA 7. Moduulin lopullinen kansiorakenne

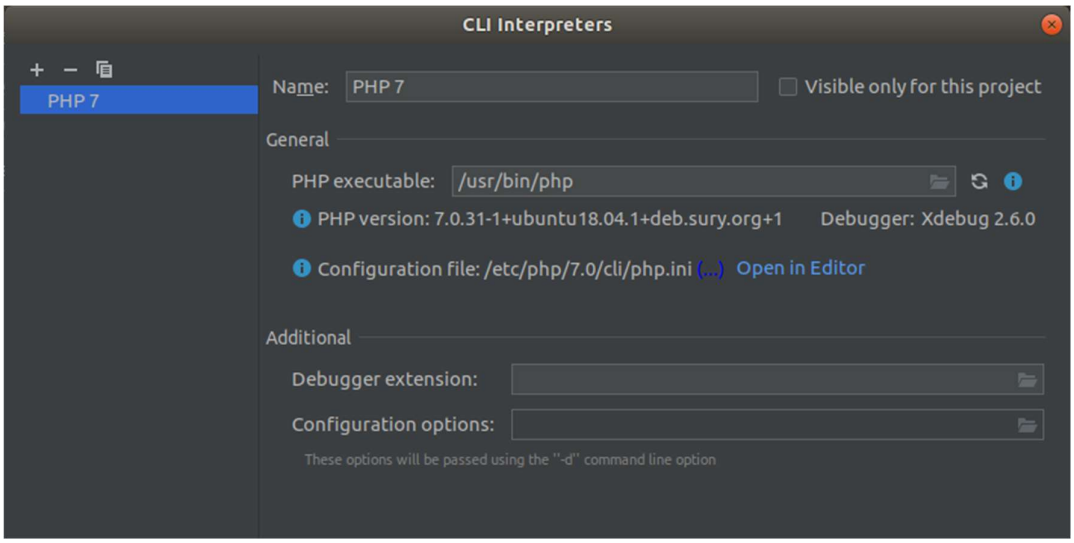
7.2 Testin asetukset

Testejä oli mahdollista suorittaa komentorivillä tai PhpStorm-editoria hyväksikäyttäen. Työtä tehdessä käytettiin testien suorittamiseen koko ajan PhpStorm-editoria, koska se oli huomattavasti nopeampaa ja helpompaa.

Aluksi tuli määrittää PHP-tulkki (PHP interpreter), jolla testit oli tarkoitus ajaa. Ohjelmointikielivalikon alta löytyi PHP-kielelle oma osio, jonka alta asetukset tuli määrittää (kuva 8). Valittiin sopiva taso kielelle ja käytettävä tulkki. Jos tulkin valikko oli tyhjä, oli mahdollista lisätä uusi tulkki painamalla kolmea pistettä pudotusvalikon vierestä. Painiketta painamalla avautui uusi ikkuna, josta pystyi määrittämään halutun tulkin käyttöön (kuva 9).

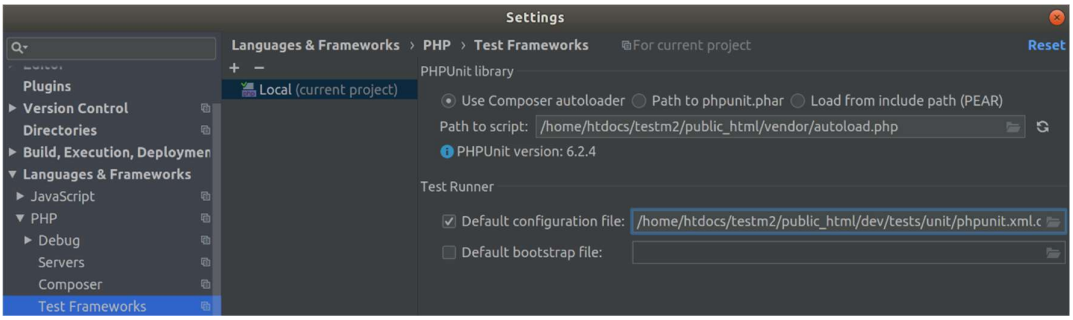


KUVA 8. PHP-ohjelmointikielen version ja tulkin valinta



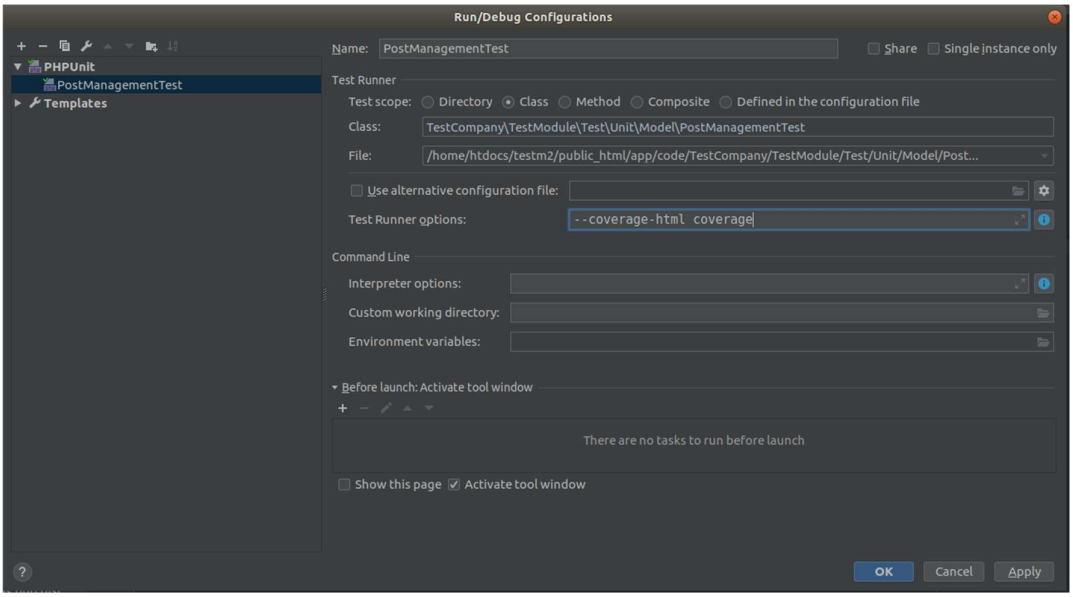
KUVA 9. Tulkin määritysvalikko.

PhpStorm-editorin asetuksista, ohjelmointikielivalikon alta, löytyi myös Test Frameworks-niminen kohta, jossa pystyi asettamaan haluamansa PHPUnit version ja konfigurointitiedoston testauksia varten (kuva 10). Magento 2 -alustassa pystyttiin käyttämään siinä tulevaa konfiguraatiotiedostoa (phpunit.xml.dist), joka löytyy <magento2_root_dir>/dev/tests/unit-kansiosta.



KUVA 10. PHP-kielen testauskehyksen asetukset

Kun PHP-kielelle oli määritetty asetukset, asetettiin seuraavaksi halutulle testiluokalle tarvittavat asetukset, jotta se pystyttiin suorittamaan (kuva 11). Asetuksiin tuli määrittää testattava luokka ja sille oli mahdollisuus antaa lisämääritteitä ajon ajaksi. Kattavuustulostuksen (lisää luvussa 7.4) sai luokasta kirjoittamalla Test Runner options-kenttään ”--coverage-html coverage”-määritteen. Tämän jälkeen testi oli mahdollista ajaa painamalla PhpStorm-editorista Play-nappia.



KUVA 11. Testiluokan asetukset

7.3 Testin luominen

PostManagementTest.php-tiedoston (liite 4) täytyi periä (extend) luokka TestCase toimiakseen. Jos halutaan testiluokassa suorittaa useampi testi, jotka tarvitsevat samalla tavalla alustettuja objekteja, voidaan luoda alustusmetodi setUp. Luokkaan oli suunniteltu useampi metodi, joten testin alustukset luotiin setUp-metodiin, jota kutsutaan joka kerta, ennen kuin kutsutaan testimetodia. PostManagement-luokasta luotiin objekti ObjectManager-luokkaa käyttäen, jota moduulitestauksella testattiin. Ensimmäinen yksinkertainen testi luotiin metodia getPostString varten, joka palauttaa yksinkertaisen merkkijonon. Testiluokan metodi nimettiin testGetPostStringReturnEqual, jonka avulla tiedettiin, mitä metodia kyseinen metodi testaa ja mitä se palauttaa. Metodista luotiin alla olevan koodin näköinen.

```
public function testGetPostStringReturnEqual()
{
      $expectedString = post;
      $this->assertEquals($expectedString,
      $this->postManagementModel->getPostString());
}
```

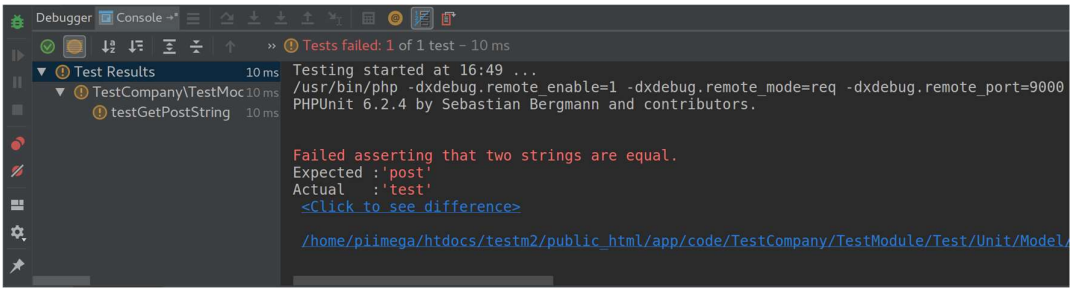
7.3.1 Väitteet

Väitteiden eli asserts avulla testataan odotettu testin paluuarvo. AssertEquals-metodin avulla testattiin, palauttaako luokan PostManagement getPostString-metodi odotetun merkkijonon. Kun merkkijonot ovat yhtäläisiä toisiinsa nähden, testi menee hyväksytysti läpi.

PostManagement.php (liite 3) mallitiedostoon luotiin metodi, jota aiemmin luotu testi testaa. Metodin tuli ainoastaan palauttaa yksinkertainen merkkijono.

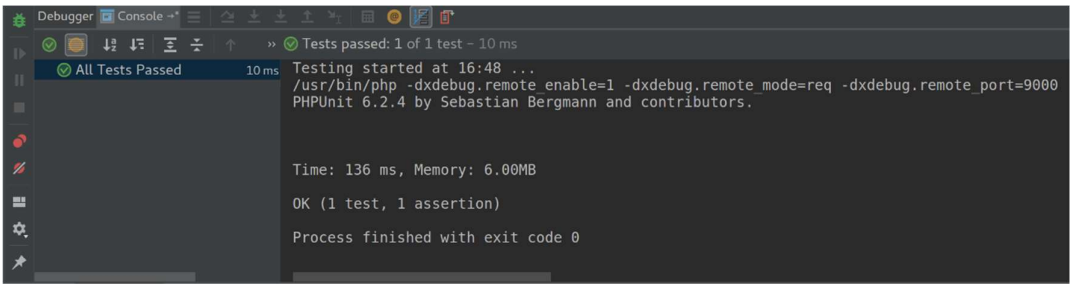
```
public function getPostString
{
      return “test”;
}
```

Metodi palautti merkkijonon ”test” ja PHPUnit huomautti virheestä (kuva 12).



KUVA 12. Hylätty testi. Merkkijonot eivät ole yhtäläisiä

Kun merkkijono vaihdettiin samaksi merkkijonoksi mikä testiin oli laitettu, eli merkkijonoksi ”post”, palautti testi hyväksytyn lopputuloksen (kuva 13).



KUVA 13. Hyväksytty testi

7.3.2 Sijaisobjektit testauksessa

Tynkäobjektien avulla voidaan luoda sijaisobjekteja, jotka simuloivat alkuperäisen luokan toimintaa. PostManagementTest.php-tiedostossa (liite 4/1) metodissa setUp luotiin mock-objekti CollectionFactory-luokasta. Objektille määriteltiin metodi, jonka objekti suorittaa testauksen aikana tai muuten testi aiheuttaa virheen. Seuraavaksi alla esimerkki, kuinka luoda ohjelmakoodi mock-objektin luomiseksi.

```
$this->getMockBuilder(CollectionFactory::class) ->disableOriginalConstructor()->set-
Methods(['create'])->getMock();
```

Mock-objekti olettaa, että se suorittaa testin aikana tietyt metodit tietyillä parametreilla. Stub-objekteilla ei ole oletuksia, mutta niiden avulla voidaan korvata ja toteuttaa haluttuja toiminnallisuuksia. Yhteenvetona on, että mock-objekti on periaatteessa stub-objekti, jolla on toteutus ja oletus.

Testausmetodissa testGetPostCollection käytettiin postCollectionMock-nimistä stub-objektia, joka luotiin Collection-luokasta (liite 4/2). Tätä sijaisobjektia verrattiin assertInstanceOf-väitteellä, minkä avulla tiedettiin, onko getPostCollection-metodin paluuarvo yhtäläinen Collection-luokan kanssa. Stub-objektin luonti alla olevassa koodissa.

```
$postCollectionMock = $this->getMockBuilder(Collection::class)
->disableOriginalConstructor()
->getMock();
```

7.3.3 Merkinnät

Merkinnät eli annotaatiot sijaitsevat metodin yläpuolella niin kutsutussa ”DocBlock” -tilassa, kuten alla olevassa esimerkissä näkyy.

```
/**
    * @dataProvider providerTestGetPostType
    * @param string $postType
    * @param string $expectedResult
    */
 public function testGetUrlByPostTypeReturnEqualString(string $postType, string
$expectedResult){…}
```

Metodissa getUrlByPostType on ideana, että kun post-parametrista löytyy merkkijono, esimerkiksi facebook, palauttaa metodi tällöin tietyntyylisen URL-osoitteen eteenpäin (liite 3/3). Testimetodissa testGetUrlByPostTypeReturnEqualString käytettiin dataProvider-annotaatiota, minkä avulla oli mahdollista suorittaa yksi testi useilla eri informaatiojoukoilla (liite 4/4). Tämä mahdollisti sen, että yhtä testiä ei tarvinnut monistaa moneen kertaan. Alla esimerkki metodista, jota käytettiin annotaation kautta tuomaan testitietoa testGetUrlByPostTypeReturnEqualString-funktion testausta varten.

```
public function providerTestGetPostType()
   {
      return
         [
            ['facebook', 'https:\\facebook.com'],
            ['youtube', 'https:\\youtube.com']
         ];
   }
```

Metodi providerTestGetPostType palauttaa taulukossa arvot, mitkä yhdistetään dataProvider-annotaation kautta testGetUrlByPostTypeReturnEqualString-metodiin. Taulukon ensimmäinen arvo on testGetUrlByPostTypeReturnEqualString-metodiin menevä postType-parametri ja jälkimmäinen expectedResult-parametri.

Testissä oletetaan ensimmäisellä kerralla, että postModelMock-mock-objekti palauttaa getData-metodissa arvon ”facebook”, jolloin getUrlByPostType-metodi tulisi palauttaa ”'https:\\facebook.com'”-merkkijono, jota verrataan assertEquals-väitteellä palautuvaan arvoon. Toisella testikerralla käydään automaattisesti providerTestGetPostType-metodin palautuvan taulukon toiset arvot.

7.3.4 Suojattujen metodien testaus

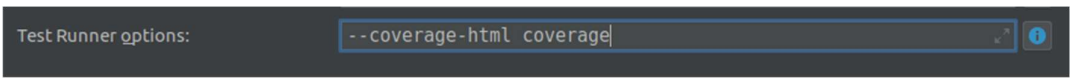
Testit tehdään julkisille (public) metodeille. Testien tulee olla niin kattavia, että kaikki julkisen metodin sisällä käytetyt suojatut (protected, private) metodit ja niiden logiikka testataan. Mock- ja stub-objekteilla ei ole mahdollista matkia metodeja joiden näkyvyysmääre on final, private, protected tai static.

Joissakin tapauksissa halutaan varmistaa, että jokin suojattu metodi suorittaa varmasti tietynlaisen logiikan. Metodi convertTagsToArray on suojattu metodi ja se poistaa merkkijonossa olevien sanojen välistä turhat välilyönnit pois ja lisää sanat taulukkoon (liite 3/3).

Suojatun metodin testaamisessa tuli käyttää ReflectionMethod-luokkaa, minkä avulla oli mahdollista suorittaa PostManagement-luokan valittu metodi halutuilla arvoilla ja antaa testille oikeudet suojattuun convertTagsToArray-metodiin.

7.4 Moduulitestauksen kattavuustulostus

Moduulitestauksesta oli mahdollista tulostaa visuaalinen kattavuustulostus. PhpStorm-editorissa luokan testin määrityksiin laitetiin kuvan mukainen komento (kuva 14).



KUVA 14. Lisämääritykset testiluokalle

Tulostuksen avulla oli mahdollista nähdä, minkä osan tehdyt testit kattavat. Tulostuksia pystyi analysoimaan kansio-, tiedosto ja metoditasolla (kuvat 15, 16, 17). Prosenttien avulla oli nopea todeta tehtyjen testien kattavuus.

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		71.11%	32 / 45		55.56%	5 / 9		0.00%	0 / 4
TestModule		71.11%	32 / 45		55.56%	5 / 9		0.00%	0 / 4

KUVA 15. Kattavuus kansiotaso

Total		78.05%	32 / 41		55.56%	5 / 9		0.00%	0 / 4
ResourceModel		0.00%	0 / 4		0.00%	0 / 2		0.00%	0 / 2
Post.php		0.00%	0 / 2		0.00%	0 / 1		0.00%	0 / 1
PostManagement.php		91.43%	32 / 35		83.33%	5 / 6		0.00%	0 / 1

KUVA 16. Kattavuus tiedostotaso

	Code Coverage								
	Classes and Traits			Functions and Methods			Lines		
Total		0.00%	0 / 1		83.33%	5 / 6	CRAP	91.43%	32 / 35
TestCompany\TestModule\Model\PostManagement		0.00%	0 / 1		83.33%	5 / 6	18.20	91.43%	32 / 35
__construct					100.00%	1 / 1	1	100.00%	2 / 2
getPostString					100.00%	1 / 1	1	100.00%	1 / 1
getPostCollection					0.00%	0 / 1	11.56	75.00%	9 / 12
filterPostsByTags					100.00%	1 / 1	2	100.00%	11 / 11
convertTagsToArray					100.00%	1 / 1	1	100.00%	4 / 4
getUrlByPostType					100.00%	1 / 1	3	100.00%	5 / 5

KUVA 17. Kattavuus metoditaso

Kattavuuden avulla pystyi katsomaan metodikohtaisesti, minkä osan testi kattoi metodista (kuva 18). Visuaalisen näkymän ansiosta oli helppoa lähteä kehittämän testiä niin, että saatiin täysi 100 prosentin kattavuus (kuva 19).

```
/**
 * @param $post
 * @return string
 */
public function getUrlByPostType($post)
{
    if ($post->getData('post_type') === 'facebook') {
        return 'https:\\facebook.com';
    } else if ($post->getData('post_type') === 'youtube') {
        return 'https:\\youtube.com';
    } else {
        return '';
    }
}
```

KUVA 18. Vihreällä olevan osan testi kattaa, mutta myös punainen osa tulisi testata.

```
/**
 * @param $post
 * @return string
 */
public function getUrlByPostType($post)
{
    if ($post->getData('post_type') === 'facebook') {
        return 'https:\\facebook.com';
    } else if ($post->getData('post_type') === 'youtube') {
        return 'https:\\youtube.com';
    } else {
        return '';
    }
}
```

Kuva 19. Metodi on väriltään täysin vihreä, kun sen kattavuus on 100 prosenttia

8 MAGENTO 2:N INTEGRAATIOTESTAUS

Integraatiotestauksella on mahdollista testata kaikki Magento 2-alustassa olevat moduulit ja niiden välinen toiminta. Ennen integraatiotestaamista täytyi tehdä pieniä valmisteluja.

8.1 Integraatiotestauksen valmistelut

Aluksi tuli luoda integraatiotestausta varten sille tarkoitettu tietokanta. Oikeaa Magenton tietokantaa ei saanut käyttää, koska kaikki siellä oleva tieto tulee häviämään. Tietokannan luonnin yhteydessä kantaan luotiin myös käyttäjä, jolla oli oikeudet ainoastaan luotuun testitietokantaan turvallisuussyistä. Tietokannan ja käyttäjän luonti onnistui seuraavanlaisella SQL-komennolla.

```
CREATE DATABASE magento_integration_tests;

CREATE USER 'magento2_test_user'@'localhost' IDENTIFIED BY 'qwerty1234';

GRANT ALL ON magento_integration_tests.* TO 'magento2_test_user'@'localhost'
IDENTIFIED BY 'qwerty1234';
```

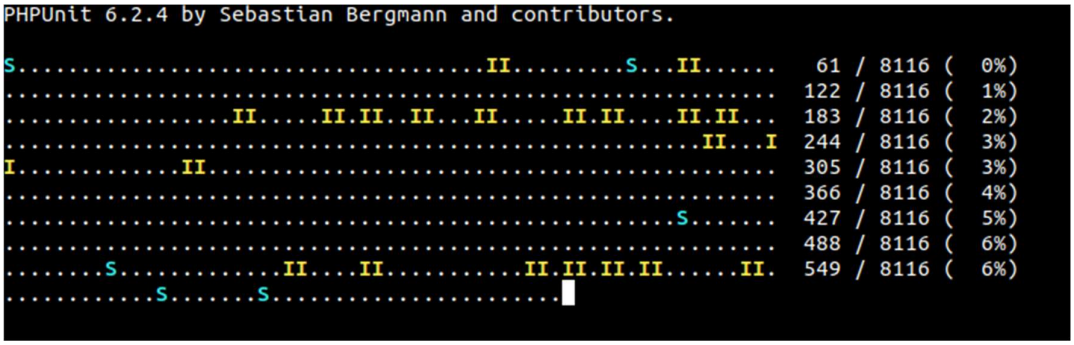
Magenton integraatiotestauskehys sisälsi konfiguraatiotiedoston (install-config-mysql.php.dist), joka sijaitsi <magento2_root_dir>/dev/tests/integration/etc/-kansiossa. Tämä tiedosto tuli kopioida ja tiedoston perästä tuli ottaa pois ”.dist”-jälkiliite. Tiedoston sisältöön muutettiin oman testauskannan nimi, käyttäjä ja salasana (liite 6).

8.2 Integraatiotestin suorittaminen

Integraatiotestin suorittaminen onnistui sekä PhpStorm-editorin että komentorivin kautta. Komentorivillä ollessa Magenton juurikansiossa tuli mennä <magento2_root_dir>/dev/tests/integration-kansioon. Tämän jälkeen, jos haluttiin suorittaa kaikki integraatiotesti, annettiin seuraavanlainen komento.

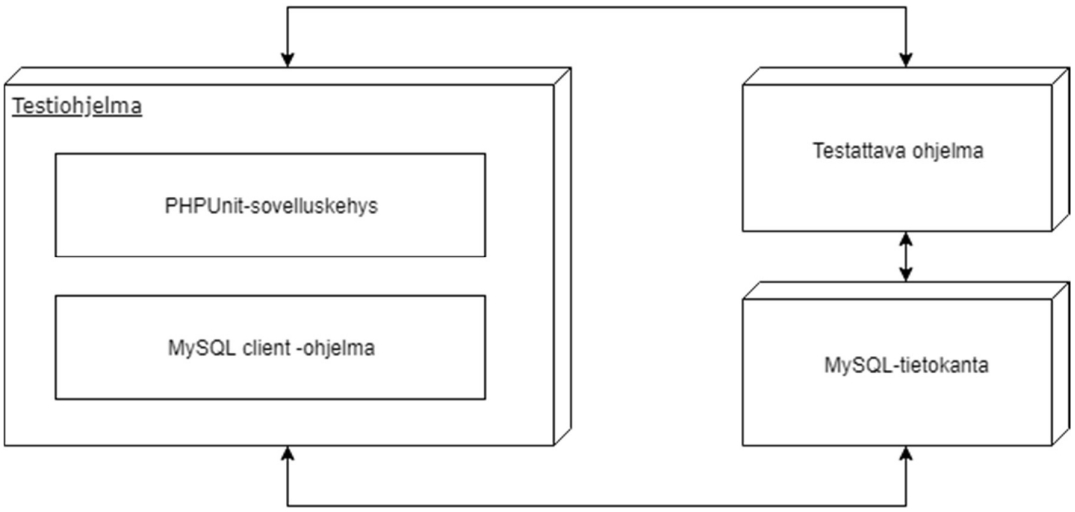
```
$ ../../../vendor/bin/phpunit
```

Komentoriville alkoi tulostumaan tietoa siitä, monesko testi oli ajossa ja montako prosenttia ajo oli edennyt (kuva 20). Lopuksi integraatiotesteistä tuli yhteenveto suoritettujen testien ja virheiden määrästä. Aikaa integraatiotestien ajamiseen kului noin puoli tuntia. Ongelmallista integraatiotestien suorittamisessa oli välillä se, että tyhjällä verkkokaupalla Magento 2.2.5-versiolla integraatiotestit eivät suoriutuneet loppuun asti vaan pysähtyivät Magenton koodissa olevan virheen vuoksi.



KUVA 20. Integraatiotestin ajamista komentorivillä

Integraatioita oli mahdollista suorittaa myös yhdelle luokalle kerralla, kun komentoon annettiin halutun luokan polku. Magenton omilla sivuilta löytyi myös ohjeet, kuinka integraatiotestejä voisi itse aloittaa tekemään. Kuvassa 21 on kuvattu integraatiotestin toimintaa.



KUVA 21. Integraatiotestien toiminta

9 MAGENTO 2:N JÄRJESTELMÄTESTAUS

Magento 2 sisältää järjestelmätestauksen tarvittavat testit. Järjestelmätestien suoritukseen tarvitaan PHPUnit, Selenium-palvelin ja internetselain. Järjestelmätestit käyvät läpi automaattisesti Magenton ominaisuudet, kuten tuotteiden ja kategorioiden luonnin, kassan testauksen, hallinnan asetuksien testaamisen eri variaatioilla ja Magentoon integroitujen rajapintojen testauksen. Magento tarjoaa käyttäjille kattavat ohjeet omilla sivuillaan (https://devdocs.magento.com/guides/v2.0/mtf/mtf_introduction.html), kuinka järjestelmätestausta voidaan suorittaa ja kuinka testejä voi alkaa itse luomaan.

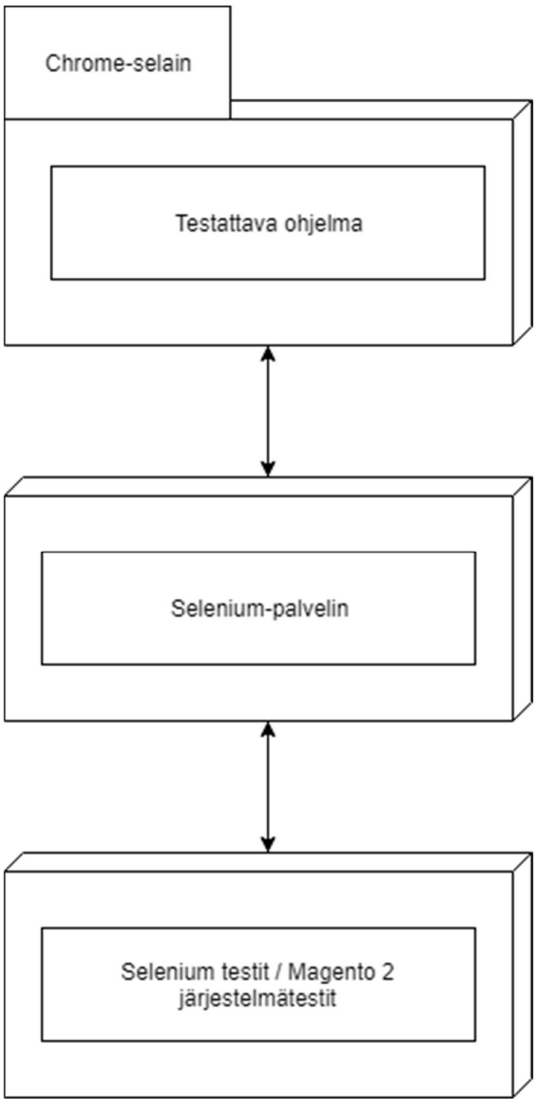
9.1 Järjestelmätestauksen valmistelut

Järjestelmätestauksen valmistelut olivat huomattavasti työläämmät kuin integraatiotestauksen valmistelut. Ensimmäiseksi tuli asentaa Selenium-palvelin (kuva 22), mikä oli mahdollista ladata ilmaiseksi Seleniumin omilta sivuilta. Testauksessa käytettävän selaimen pystyi itse päättämään, ja tässä projektissa valittiin selaimeksi Chrome. Chromelle tuli asentaa Chromedriver, joka sisälsi ajurin verkkosovellusten automaatiotestausta varten.

Järjestelmätestauskehys (Functional Testing Framework) vaati, että myös Composer-ohjelmisto oli asennettu. Kun Composer oli asennettu, mentiin järjestelmätestauksen kansioon. Tämän jälkeen syötettiin komentoriville komento Composer-ohjelmistoa varten, joka latasi tarvittavat paketit järjestelmätestausta varten. Kansioon <magento2_root_dir>/dev/tests/functional/ asentui vendor-kansio, joka sisälsi tarvittavat tiedostot testausta varten.

```
$ cd <magento2_root_dir>/dev/tests/functional/

$ composer install
```



KUVA 22. Selenium-palvelinohjelman toiminta

Kun työkalut oli asennettu, täytyi konfiguroida phpunit.xml-tiedosto <magento2_root_dir>/dev/tests/functional/-kansioon (liite 7). Tiedoston pystyi luomaan kansiossa olevasta phpunit.xml.dist-tiedostosta. Tiedostoon täytyi määrittää oman kaupan frontend- ja backend-osoitteet.

Seuraavaksi kopioitiin config.xml.dist-tiedosto ja luotiin siitä config.xml-tiedosto <magento2_root_dir>/dev/tests/functional/etc/-kansioon (liite 8). Tiedostoon määritettiin hallintapaneelin salasana ja osoite, kaupan tietokannan tiedot ja selain, jota testauksessa haluttiin käyttää.

Magenton järjestelmätestauksen ohjeiden mukaan täytyi hallinnan asetuksia käydä vaihtamassa, jotta järjestelmätestillä oli tarvittavat oikeudet kaupan hallinnan puolelle. Lopuksi asennettiin uusin Java-versio Selenium-palvelinta varten. Palvelin käynnistettiin seuraavalla komennolla kansiossa, jossa Selenium-palvelimen jar-paketti sijaitsi.

```
$ java -Dwebdriver.chrome.driver=/usr/bin/chromedriver -jar selenium-server-standalone-3.14.0.jar
```

Ennen testien suorittamista täytyi generoida vielä tarvittavat tiedostot.

```
$ cd <magento2_root_dir>/dev/tests/functional/utils

$ php generate.php
```

9.2 Järjestelmätestien suorittaminen

Järjestelmätestausta suoritettiin komentorivin kautta. Seuraavalla komennolla oli mahdollista suorittaa kaikki Magento 2 -järjestelmätestauksen testit, jotka komento ajaa automaattisesti <magento2_root_dir>/dev/tests/functional/tests/app/Magento/-kansiosta.

```
$ cd <magento2_root_dir>/dev/tests/functional

$ vendor/bin/phpunit
```

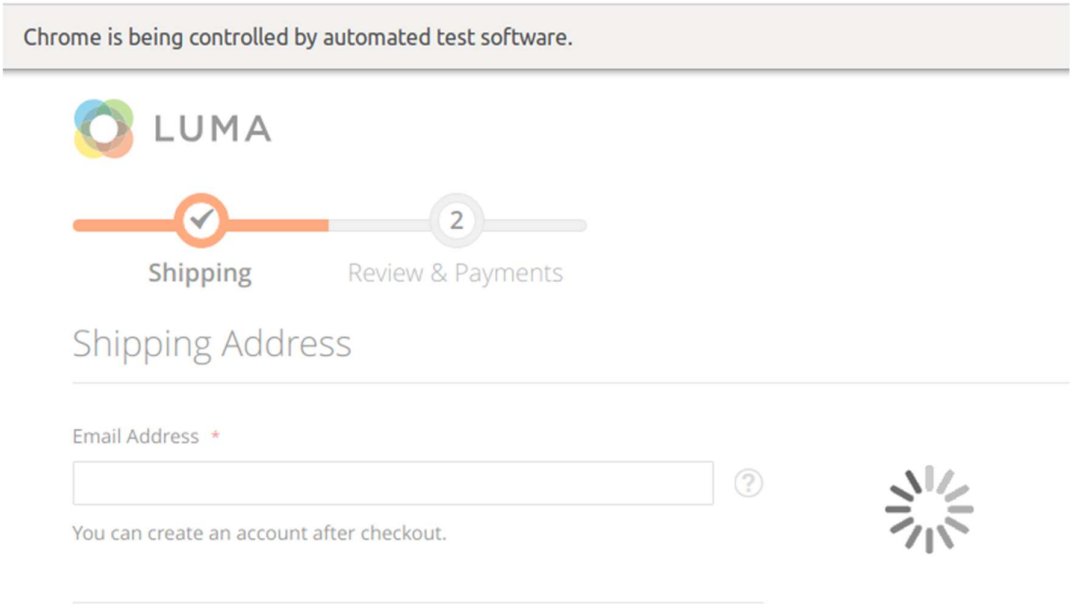
Terminaalissa näkyi, että testit lähtivät suoriutumaan ja ohjelmisto merkkasi punaisella E-kirjaimella, jos jotain virheitä oli ilmentynyt (kuva 23). Kun suoritus alkoi, käynnistyi selain automaattisesti ja ilmoitti olevansa automaattisen testin kontrolloitavana (kuva 24).

```
PHPUnit 5.5.7 by Sebastian Bergmann and contributors.

Runtime:       PHP 7.0.31-1+ubuntu18.04.1+deb.sury.org+1 with Xdebug 2.6.0
Configuration: /home/htdocs/testm2/public_html/dev/tests/functional/phpunit.xml

.EEE......E.^C
```

KUVA 23. Järjestelmätestien suoritusta



KUVA 24. Selain ilmoitti, että se oli automaattisen testin suorituksen kontrolloitavana

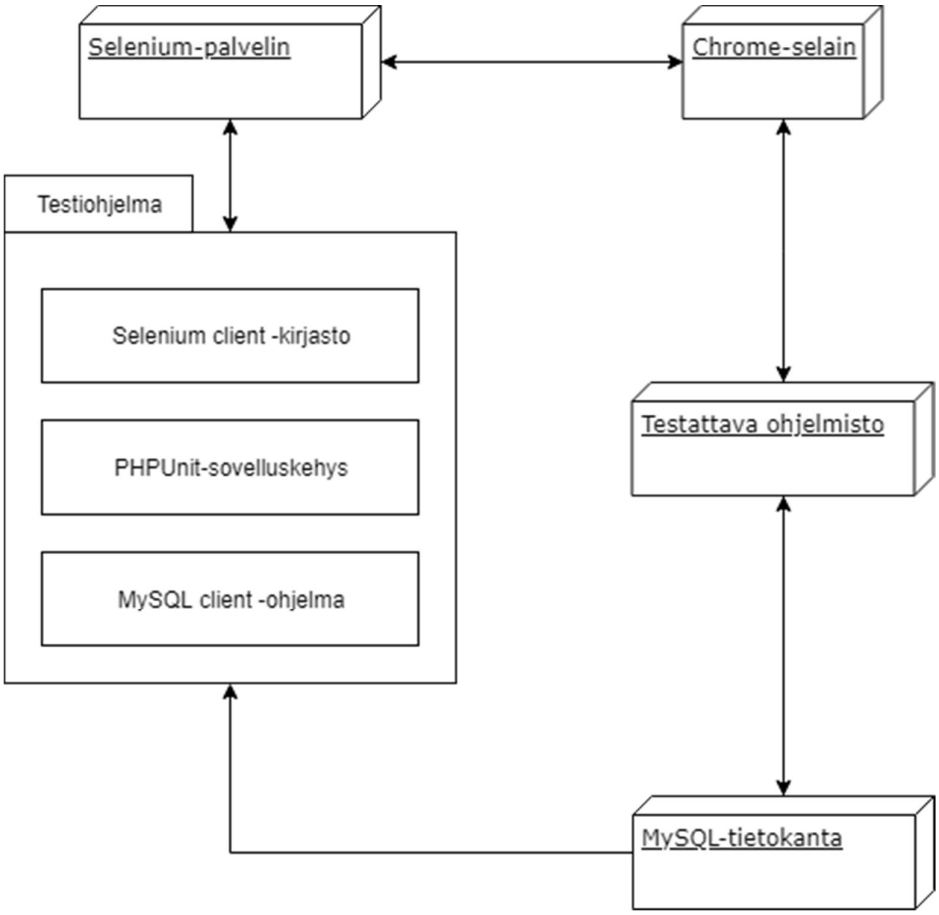
Aikaa kului kaikkien järjestelmätestien suorittamiseen noin kaksi tuntia. Lopuksi komentoriville tulostui pitkä lista havaituista virheistä sekä lopputulos testeistä (kuva 25). Tarkempia tietoja suoritetuista testeistä ja virheistä löytyy Magenton järjestelmätestien <magento2_root_dir>/dev/tests/functional/var/log-kansiosta, josta löytyvät logit ja kuvakaappaukset löydetyistä virheistä. Testauksen lopputulos oli 585 suoritettua testiä. Suurin osa ilmenneistä virheistä liittyi vajavaisista oikeuksista esimerkiksi kolmannen osapuolen testiympäristöön, jonka takia lopputuloksessa näkyi 13 vaillinaista testiä. Kolme testiä epäonnistui kokonaan. Kuvassa 26 on kuvattu järjestelmätestien toimintaa.

```
15) Magento\Search\Test\TestCase\CreateMultipleSynonymGroupsTest::test with data set "Cre
MAGETWO-63120: Search icon is disabled on collapsed toolbar

/home/htdocs/testm2/public_html/dev/tests/functional/vendor/magento/mtf/Magento/Mtf/TestC
/home/htdocs/testm2/public_html/dev/tests/functional/vendor/magento/mtf/Magento/Mtf/TestC
/home/htdocs/testm2/public_html/dev/tests/functional/vendor/magento/mtf/Magento/Mtf/TestC
/home/htdocs/testm2/public_html/dev/tests/functional/vendor/magento/mtf/Magento/Mtf/TestC
/home/htdocs/testm2/public_html/dev/tests/functional/vendor/magento/mtf/Magento/Mtf/TestC

ERRORS!
Tests: 585, Assertions: 879, Errors: 607, Failures: 3, Incomplete: 15.
```

KUVA 25. Järjestelmätestien lopputulos



KUVA 26. Järjestelmätestien toiminta

Järjestelmätestauksessa oli myös mahdollista suorittaa yksittäisiä testejä seuraavalla komennolla komentorivillä, jossa kohta ”<name of test>” korvattiin PHP-tiedoston nimellä.

```
$ cd <magento2_root_dir>/dev/tests/functional

$ vendor/bin/phpunit --filter <name of test>
```

Esimerkiksi kategorioiden luonti -testi voidaan suorittaa seuraavanlaisesti.

```
$ cd <magento2_root_dir>/dev/tests/functional

$ vendor/bin/phpunit --filter CreateCategoryEntityTest
```

10 TESTAUKSEN KEHITTÄMINEN

Yrityksen ohjelmiston laadun parantamiseksi ja kehittämiseksi on erittäin tärkeää ottaa käyttöön mahdollisimman pian Magento-verkkokauppa-alustan tarjoamat työkalut ohjelmistotestausta varten. Testauksen automatisoinnilla voidaan vähentää huomattavasti manuaalisen testauksen työmäärä ja pystytään havaitsemaan virheet jo kehitysvaiheessa. Hyvissä ajoin havaitut virheet ohjelmakoodissa alentavat huomattavasti kehityskustannuksia ja vähentävät ohjelmakoodin myöhempää korjausta.

Aluksi moduulitestauksen lisääminen ohjelmistoon kannattaa aloittaa pienellä määrällä työntekijöitä. Opinnäytetyöstä saatu tieto on hyvä jakaa ja syventää verkkokauppatiimin kesken. Ohjelmistosuunnittelijoiden kanssa tulee käydä läpi testauksen perusteet, idea ja hyödyllisyys. Testien luonti, suunnitteleminen ja oppiminen moduulin kehittämisen aikana vie aikaa, joten on erittäin hyvä lähteä kehittämään asiaa pienissä palasissa eteenpäin. Vaikka aluksi yhden moduulin kehittämiseen voi kulua huomattavasti paljon enemmän aikaa, vähentävät testit tulevaisuudessa moduulien korjaamisen tarvetta. Kun ohjelmistosuunnittelijat oppivat tekemään testejä rutiinilla, kehitysaika pienenee huomattavasti ja ohjelmiston laatu paranee merkittävästi.

Projektin aikana tuli havaittua, että erityisesti liiketoimintakriittisiin moduuleihin, kuten toimitus- ja maksutapoihin, on erittäin tärkeää luoda moduulitestit. Uusien ominaisuuksien lisääminen on helpompaa ja turvallisempaa, kun moduulitestit ovat ajan tasalla niiden kehittyessä moduulin mukana. Tällöin on heti mahdollista huomata jo kehitysvaiheessa mahdolliset virheet ja ristiriidat koodissa. Moduuli voidaan päivittää tuotantoon ilman pelkoa sen toiminnallisuuden hajoamisesta.

Työtä tehdessä ilmeni, että testien tekeminen vanhoihin moduuleihin on erittäin työlästä ja osin jopa mahdotonta. Koodia joutui refaktoroimaan todella paljon, jotta se oli helpommin ja järkevämmin testattavissa. Refaktorointiprosessi vie erityisen paljon aikaa, joten havaintojen perusteella päädyttiin siihen, että ainoas-

taan tärkeät vanhemmat liiketoiminallisesti kriittiset moduulit ovat ensisijalla testauksen kehittämisen osalta. Uusien moduulien kehittäminen tulee aloittaa testaus edellä joka tapauksessa.

Kun moduulitestaus on jo vakiintunut ohjelmiston kehitykseen, on hyvä ottaa myöhemmässä vaiheessa käyttöön myös integraatiotestaus. Integraatiotestauksen avulla on mahdollista testata isompaa kokonaisuutta ja saadaan selville jo Magentossa olevien moduulien ja yrityksen omien moduulien välinen toiminta ja yhteensopivuus. Integraatiotestien tekeminen vaatii oman osaamisensa, mutta niihin perehtyminen on taloudellisesti kannattaa tulevaisuutta ajatellen, kun moduulit toimivat luotettavasti kokonaisuutena.

Järjestelmätestaus, järjestelmätestien tekeminen ja niiden ylläpitäminen on iso ja haastava kokonaisuus. Järjestelmätestien tekemiseen vaaditaan laajempaa kokemusta testauksesta. Uusien järjestelmätestien kehittämiseen ja ylläpitämiseen yrityksen tarpeiden ja vaatimusten mukaisesti olisi hyvä hankkia testaukseen erikoistunut osaaja tai osaajia. Taloudellisesti järjestelmätestauksen käyttöönotto on kallista, mutta pitkällä aikavälillä katsottuna se tuo taloudellisia säästöjä. Erityisen tärkeää olisi hyödyntää järjestelmätestauksen kuormitustestausta, jonka avulla olisi mahdollista kartoittaa, minkä kokoista palvelinkapasiteettia yrityksen asiakkaat yksilöllisesti tarvitsevat. Tämän avulla osattaisiin varautua suurien asiakasmäärien nostattamaan kuormitukseen, etenkin asiakkaille tärkeinä alennuspäivinä, kuten Black Friday -päivänä. Aluksi olisi hyvä käyttää yksittäisiä osia järjestelmätestauksesta, kuten testejä, jotka testaavat Magenton kassaa. Tämä vähentäisi manuaalisen testaamisen määrää huomattavasti ja näin saataisiin testattua liiketoimintaosaa verkkokaupasta.

11 YHTEENVETO

Opinnäytetyön tarkoituksena oli luoda jatkosuunnitelma Magento 2 -verkkokaupan testaamisesta, perehtyä yleisesti testaamiseen ja oppia luomaan yrityksen moduuleihin moduulitestejä ja laatia ohjeet testausten tekemiseen.

Opinnäytetyön aihetta mietittäessä tuli erityisesti ilmi, että testaukseen perehtymiselle olisi selkeää tarvetta. Ohjelmointitestaus nostaisi ohjelmiston laatua ja automaattisesta testauksesta tulisi olemaan konkreettista hyötyä yrityksen ohjelmistokehityksessä. Aihe oli erittäin laaja ja minulle täysin uusi, mutta mielestäni sopivan haastava. Koko projektin ajan toimin täysin itsenäisesti.

Moduulitestaus oli korkeimmalla prioriteetilla projektissa ja testien luonti ja toteutus onnistui hyvin. Moduulitestauksen tekemisestä jäi kattava osaaminen ja dokumentaatio tämän opinnäyteyön avulla. Testejä on luotu uusiin moduuleihin projektin alkamisesta lähtien. Projektin aikana ehti myös perehtymään integraatio- ja järjestelmätestauksen toiminnallisuuteen ja suorittamiseen.

Projektia tehdessä opin erittäin paljon uusia asioita ohjelmistotestaamisesta ja testaustyökalujen käytöstä. PHPUnit-työkalu tuli minulle täysin uutena työkaluna tutuksi projektin aikana.

Tämän opinnäytetyön avulla on mahdollista saada käsitys testauksesta ja oppia luomaan alusta omia Magento 2-moduulitestejä.

LÄHTEET

1. Hannila, Esa 2018. Ohjelmistojen automaattinen testausjärjestelmä. Opinnäytetyö. Oulu: Oulun ammattikorkeakoulu, tieto- ja viestintätekniikan tutkinto-ohjelma. Saatavissa: http://www.theseus.fi/bitstream/handle/10024/141431/hannila_esa.pdf?sequence=1&isAllowed=y. Hakupäivä 21.4.2018.

2. Rauhala, Eija 2010. Ohjelmistotestauksen suunnittelu - Case: A-lehdet Oy:n laskujen tulostusohjelma. Opinnäytetyö. Haaga-Helia, tietojenkäsittelyn koulutusohjelma. Saatavissa: https://www.theseus.fi/bitstream/handle/10024/23687/Rauhala_Eija.pdf?sequence=1&isAllowed=y. Hakupäivä 21.4.2018.

3. Pohjolainen, Pentti 2003. Ohjelmiston testauksen automatisointi. Pro gradu -tutkielma. Kuopio: Kuopion yliopisto, tietojenkäsittelytieteen laitos. Saatavissa: http://www.cs.uku.fi/tutkimus/Teho/PenttiPohjolainen_Gradu.pdf. Hakupäivä 21.4.2018.

4. Beck, Kent 2002. Test-Driven Development By Example. Yhdysvallat: Addison Wesley.

5. Jäsberg, Jarkko 2011. Web-sovellusten testauksen automatisointi. Opinnäytetyö. Jyväskylän ammattikorkeakoulu, ohjelmistotekniikka. Saatavissa: http://www.theseus.fi/bitstream/handle/10024/27475/jasberg_jarkko.pdf?sequence=1&isAllowed=y. Hakupäivä 13.5.2018.

6. Itewiki. Magento. Saatavissa: https://www.itewiki.fi/opas/magento/. Hakupäivä 13.5.2018.

7. Magento DevDocs. Saatavissa: https://devdocs.magento.com. Hakupäivä 13.5.2018.

8. Ubuntu. Saatavissa: https://www.ubuntu.com/. Hakupäivä 6.10.2018

9. PhpStorm. Saatavissa: https://www.jetbrains.com/phpstorm/. Hakupäivä 10.6.2018.

10. PHPUNit. Saatavissa: https://phpunit.de/. Hakupäivä 10.6.2018.

11. Selenium. Saatavissa: https://www.seleniumhq.org/. Hakupäivä 6.10.2018

LIITTEET

Liite 1 Lähtötietomuistio

Liite 2 Moduulin tiedostot registration.php ja module.xml

Liite 3 Moduulin tiedosto PostManagement.php

Liite 4 Moduulin tiedosto PostManagementTest.php

Liite 5 Moduulin tiedostot Post.php, Post.php ja Collection.php

Liite 6 Integraatiotestin asetukset install-config-mysql.php

Liite 7 Järjestelmätestauksen asetukset phpunit.xml

Liite 8 Järjestelmätestaukset asetukset config.xml

TestCompany\TestModule\registration.php

```
<?php
    \Magento\Framework\Component\ComponentRegistrar::register(
        \Magento\Framework\Component\ComponentRegistrar::MODULE,
        'TestCompany_TestModule',
        __DIR__
    );
```

TestCompany\TestModule\etc\module.xml

```
<?xmlversion="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:ma-
gento:framework:Module/etc/module.xsd">
    <module name="TestCompany_TestModule" setup_version="0.0.1"/>
</config>
```

```
<?php

namespace TestCompany\TestModule\Model;

use TestCompany\TestModule\Model\ResourceModel\Post\CollectionFactory;

class PostManagement
{
    /**
     * @var CollectionFactory
     */
    protected $_postCollectionFactory;

    /**
     * @var $tagsString
     */
    protected $tagsString;
    /**
     * @var array $postParams
     */
    protected $postParams = [];

    public function __construct(
        CollectionFactory $postCollectionFactory
    ) {
        $this->_postCollectionFactory = $postCollectionFactory;
    }

    public function getPostString() {
        return 'post';
    }
```

```
public function getPostCollection($siteId, $sectionId) {
    if ((!isset ($siteId) || empty($siteId)) && (!isset($sectionId) || empty($sectionId))){
        return null;
    }
    $postCollection = $this->_postCollectionFactory->create();
    if (isset($siteId) && !empty($siteId)) {
        $postCollection->addFieldToFilter('site_id', ['eq' => $siteId]);
    } else if (isset($sectionId) && !empty($sectionId)) {
        $postCollection->addFieldToFilter('section_id', ['eq' => $sectionId]);
    }
    if (!$this->postParams['tags'] == ' ') {
        $tagsFilter = $this->filterPostsByTags($this->postParams['tags']);
        $postCollection->addFieldToFilter('tags', $tagsFilter);
    }
    $postCollection->setOrder('updated_at','DESC');
    return $postCollection;
}

protected function filterPostsByTags(string $tagsString) {
    $tagsArray = $this->convertTagsToArray($tagsString);
    $likeCondition = [];
    foreach ($tagsArray as $tag) {
        $likeCondition = array_merge($likeCondition, array(
            array('like' => $tag),                  // First and only tag
            array('like' => $tag . ',%'),           // First tag
            array('like' => '%, ' . $tag . ',%'), // Tag in the middle
            array('like' => '%,' . $tag . ',%'),  // Tag in the middle without whitespace
            array('like' => '%, ' . $tag),         // Last tag
            array('like' => '%,' . $tag),          // Last tag without whitespace
        ));
    }
    return $likeCondition;
}
```

```
protected function convertTagsToArray(string $tagsString) {
        $tagsString = str_replace(', ', ',', $tagsString);
        $tagsString = str_replace(' ,', ',', $tagsString);
        $tagsArray = explode(",", $tagsString);
        return $tagsArray;
    }

    /**
     * @param $post
     * @return string
     */
    public function getUrlByPostType($post)
    {
        if ($post->getData('post_type') === 'facebook') {
            return 'https:\\facebook.com';
        } else if ($post->getData('post_type') === 'youtube') {
            return 'https:\\youtube.com';
        } else {
            return '';
        }
    }
}
```

```
<?php
namespace TestCompany\TestModule\Test\Unit\Model;
use TestCompany\TestModule\Model\PostManagement;
use TestCompany\TestModule\Model\ResourceModel\Post\CollectionFactory;
use TestCompany\TestModule\Model\ResourceModel\Post\Collection;
class PostManagementTest extends \PHPUnit\Framework\TestCase
{
    /**
     * @var PostManagement
     */
    protected $postManagementModel;
    /**
     * @var CollectionFactory
     */
    protected $postCollectionFactoryMock;
    /**
     * @var $postCollectionMock
     */
    protected $postCollectionMock;
    public function setUp()
    {
        $this->postCollectionFactoryMock = $this->getMockBuilder(CollectionFactory::class)
            ->disableOriginalConstructor()
            ->setMethods(['create'])
            ->getMock();
        $this->postCollectionMock = $this->getMockBuilder(Collection::class)
            ->disableOriginalConstructor()
            ->getMock();
        $objectManager = new \Magento\Framework\TestFramework\Unit\Helper\ObjectManager($this);
        $this->postManagementModel = $objectManager->getObject(PostManagement::class,
            [
                'postCollectionFactory' => $this->postCollectionFactoryMock,
                'postParams' => ['tags' => 'article']
            ]);}
```

```
public function testGetPostString()
    {
        $expectedString = 'post';
        $this->assertEquals($expectedString, $this->postManagementModel->getPostString());
    }
    public function testGetPostCollection()
    {
        $postCollectionMock = $this->createMock(Collection::class);
        $this->postCollectionFactoryMock->expects($this->once())
            ->method('create')
            ->will($this->returnValue($postCollectionMock));
        $this->assertInstanceOf(Collection::class, $this->postManagementModel->getPostCollection(1, 1));
    }
    public function testConvertTagsToArrayReturnEqual()
    {
        $testString = 'article, example';
        $expectedResult = ['article', 'example'];
        $method = new \ReflectionMethod(PostManagement::class, 'convertTagsToArray');
        $method->setAccessible(true);
        $result = $method->invoke($this->postManagementModel, $testString);
        $this->assertEquals($expectedResult, $result);
     }
```

```
public function testFilterPostsByTagsReturnEqual()
    {
        $testTag = 'article';
        $expectedResult =
            [
                ['like' => 'article'],
                ['like' => 'article,%'],
                ['like' => '%, article,%'],
                ['like' => '%,article,%'],
                ['like' => '%, article'],
                ['like' => '%,article'],
            ];
        $method = new \ReflectionMethod(PostManagement::class, 'filterPostsByTags');
        $method->setAccessible(true);
        $result = $method->invoke($this->postManagementModel, $testTag);
        $this->assertEquals($expectedResult, $result);
    }
```

```
    /**
     * @dataProvider providerTestGetPostType
     * @param string $postType
     * @param string $expectedResult
     */
    public function testGetUrlByPostTypeReturnEqualString(string $postType, string $expectedResult) {
        $postModelMock = $this->getMockBuilder(\TestCompany\TestModule\Model\Post::class)
            ->disableOriginalConstructor()
            ->setMethods(['getData'])
            ->getMock();
        $postModelMock->expects($this->any())
            ->method('getData')
            ->willReturn($postType);
        $this->assertEquals($expectedResult, $this->postManagementModel->getUrlByPostType($postModelMock));
    }

    public function providerTestGetPostType()
    {
        return
            [
                ['facebook', 'https:\\facebook.com'],
                ['youtube', 'https:\\youtube.com'],
                [' ',' ']
            ];
    }

}
```

TestCompany\TestModule\Test\Unit\Model\Post.php

```
<?php
namespace TestCompany\TestModule\Model;
use Magento\Framework\Model\AbstractModel;


class Post extends AbstractModel
{
    protected function _construct() {
        $this->_init('TestCompany\TestModule\Model\ResourceModel\Post');
    }
}
```

TestCompany\TestModule\Test\Unit\Model\ResourceModel\Post.php

```
<?php
namespace TestCompany\TestModule\Model\ResourceModel;

use Magento\Framework\Model\ResourceModel\Db\AbstractDb;

class Post extends AbstractDb
{
    protected function _construct() {
        $this->_init('test_posts', 'post_id');
    }
}
```

TestCompany\TestModule\Test\Unit\Model\ResourceModel\Post\Collection.php

```
<?php
namespace TestCompany\TestModule\Model\ResourceModel\Post;
use Magento\Sales\Model\ResourceModel\Collection\AbstractCollection;

class Collection extends AbstractCollection
{
    protected function _construct() {
        $this->_init('TestCompany\TestModule\Model\Post', 'TestCompany\TestModule\Model\ResourceModel\Post');
    }
}
```

Magento2_root\dev\test\integration\etc\install-config-mysql.php

```
<?php

return [
    'db-host' => 'localhost',
    'db-user' => 'magento2_test_user',
    'db-password' => 'qwerty1234',
    'db-name' => 'magento_integration_tests',
    'db-prefix' => '',
    'backend-frontname' => 'backend',
    'admin-user' => \Magento\TestFramework\Bootstrap::ADMIN_NAME,
    'admin-password' => \Magento\TestFramework\Bootstrap::ADMIN_PASSWORD,
    'admin-email' => \Magento\TestFramework\Bootstrap::ADMIN_EMAIL,
    'admin-firstname' => \Magento\TestFramework\Bootstrap::ADMIN_FIRSTNAME,
    'admin-lastname' => \Magento\TestFramework\Bootstrap::ADMIN_LASTNAME,
];
```

Magento2_root\dev\test\functional\phpunit.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
/**
 * Copyright © Magento, Inc. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<phpunit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="http://schema.phpunit.de/6.2/phpunit.xsd"
         colors="true"
         bootstrap="bootstrap.php"
         backupGlobals="false"
         verbose="true">
    <testsuites>
        <testsuite name="All Tests">
            <directory suffix="Test.php">tests</directory>
        </testsuite>
    </testsuites>
```

```
<listeners>
        <listener class="Magento\Mtf\System\Browser\Listener" />
        <listener class="Magento\Mtf\System\Isolation\Listener">
            <arguments>
                <object class="Magento\Mtf\System\Isolation\Driver\Base" />
            </arguments>
        </listener>
        <listener class="Magento\Mtf\System\Event\StateListener" />
        <listener class="Yandex\Allure\Adapter\AllureAdapter">
            <arguments>
                <string>var/allure-results</string> <!-- XML files output directory -->
                <boolean>false</boolean> <!-- Whether to delete previous results on rerun -->
                <array> <!-- A list of custom annotations to ignore (optional) -->
                    <element key="0">
                        <string>ZephyrId</string>
                    </element>
                    <element key="1">
                        <string>Group</string>
                    </element>
                </array>
            </arguments>
        </listener>
    </listeners>
```

```
<php>
        <env name="app_frontend_url" value="http://testm2.local/" />
        <env name="app_backend_url" value="http://testm2.local/hallinta/" />
        <env name="testsuite_rule" value="basic" />
        <env name="testsuite_rule_path" value="Magento/Mtf/TestSuite/InjectableTests" />
        <env name="log_directory" value="var/log" />
        <env name="events_preset" value="base" />
        <env name="module_whitelist" value="Magento_Install,Magento_Setup" />
        <env name="basedir" value="var/log" />
        <env name="credentials_file_path" value="./credentials.xml.dist" />
        <env name="mage_mode" value="developer" />
        <env name="magento_timezone" value="America/Los_Angeles" />
    </php>

</phpunit>
```

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="config.xsd">
    <application>
        <reopenBrowser>testCase</reopenBrowser>
        <backendLogin>piimega</backendLogin>
        <backendPassword>1bnwkolms</backendPassword>
        <appBackendUrl>http://testm2.local/hallinta/</appBackendUrl>
        <backendLoginUrl>admin/auth/login</backendLoginUrl>
    </application>
    <isolation>
        <resetUrlPath>dev/tests/functional/isolation.php</resetUrlPath>
        <testSuite>none</testSuite>
        <testCase>none</testCase>
        <test>none</test>
    </isolation>
    <install>
        <host>127.0.0.1</host>
        <user>root</user>
        <password>root</password>
        <dbName>testm2</dbName>
        <baseUrl>http://testm2.local/</baseUrl>
        <backendName>backend</backendName>
    </install>
```

```
<server>
      <item name="selenium"
           type="default"
           browser="Google Chrome"
           browserName="chrome"
           host="localhost"
           port="4444"
           seleniumServerRequestsTimeout="90"
           sessionStrategy="shared">
         <item name="resolution" width="1366" height="768"/>
      </item>
   </server>
   <handler>
      <webapi priority="0">
         <token>91r30uixu6y9rqs5yyu2mf67lc1s4gmc</token>
      </webapi>
      <curl priority="1"/>
      <ui priority="2"/>
   </handler>
</config>
```