jamk.fi

Integraatiotestaus .NET-palvelinohjelmistolle

Joose Seppälä

Opinnäytetyö
Joulukuu 2020
Tietojenkäsittely ja tietoliikenne
Insinööri (AMK), tieto- ja viestintätekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Seppälä, Joose	Opinnäytetyö, AMK	Joulukuu 2020
	Sivumäärä 26	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Integraatiotestaus .NET palvelinohjelmistolle

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn ohjaaja(t)
Esa Salmikangas, Jani Immonen

Toimeksiantaja(t)
Document Ocean Oy

Tiivistelmä

Ohjelmistotestauksella on suuri merkitys ohjelmistoon. Testaamalla saadaan pois suljettua inhimilliset ohjelmointivirheet ohjelmistokoodista, mikä tekee ohjelmasta toimivamman sekä turvallisemman käyttää. Testauksella varmistetaan, että ohjelma toimii niin kuin sen kuuluu toimia.

Tutkimuksen tavoitteena oli toteuttaa toimeksiantajan käytössä olevalle palvelinohjelmistolle integraatiotestit, joilla helpotetaan ohjelmiston jatkokehitystä, toimivuutta sekä varmistetaan tietoturvakartoituksessa löydettyjen haavoittuvuuksien korjaukset. Samalla haluttiin kehittää ohjelmiston testeille hyvä testipohja, jota voidaan käyttää ohjelmiston jatkokehitys vaiheessa.

Tutkimus toteutettiin kehitystyönä, missä käytettiin ohjelmiston testaamiseen Microsoftin Visual Studio 2019 -ohjelmistoa RestSharp-kirjaston sekä NUnit-ohjelmistokehyksen kanssa. Näitä työkaluja käyttäen luotiin ja ajettiin integraatiotestejä toimeksiantajan .NET 5 palvelinohjelmistolle.

Kehitystyön tuloksena ovat hyvät testausluokat, joiden pohjalta on helppo jatkokehittää palvelinohjelmistoa sekä luoda uusia testejä uusille ominaisuuksille ja testitapauksille. Tietoturvakartoituksessa löydettyjen haavoittuvuuksien korjauksille sekä ohjelmiston toiminnan kannalta kriittisimmille testitapauksille luotiin ja ajettiin testit onnistuneesti, mutta koko palvelinohjelmiston kattavia testejä ei ehditty toteuttamaan kehitystyön aikana palvelinohjelman laajuuden vuoksi.

Avainsanat (asiasanat)

Testaus, Ohjelmistoala, Ohjelmistotuotanto

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Seppälä, Joose	Bachelor's thesis	December 2020
		Language of publication: Finnish
	Number of pages: 26	Permission for web publication: x

Title of publication

Integration testing for .NET server software

Degree programme

Information and communication technology

Supervisor(s)

Esa Salmikangas, Jani Immonen

Assigned by

Document Ocean Oy

Abstract

Software testing has a great impact on software. Software testing eliminates human errors from software code making the program more stable and safer to use. Testing ensures that the program works as it should.

The aim of the study was to implement integration tests for the assignees server software in use which will facilitate the further development and functionality of the software. Another aim was to ensure the fixes of the vulnerabilities found in the software security mapping with integration tests. At the same time there was a desire to develop a fundamental testing base for software tests that can be used in the further development phase of the software.

The research was carried out as a development project where Microsoft's Visual Studio 2019 software, with the RestSharp library and the NUnit software framework was used to test the software. Using these software tools integration tests were created and run on the assignees .NET 5 server software.

The result of the development project was worthwhile testing classes which made further development of the server software easier including creating new tests for upcoming features. Tests where successfully created and run for the test cases that were most critical to the operation of the server software and the vulnerabilities that were found in software security mapping but the comprehensive server software-wide tests were not implemented during the development project because of the large scope of server software.

Testing, Software engineering, Software business

Miscellaneous (Confidential information)

Sisältö

Kuviot

Taulukot

1 Johdanto

1.1 Sovelluksien täyttämässä maailmassa laatu ratkaisee

Nykyään tuotetaan sovelluksia suuria määriä, ja jokainen meistä käyttää niitä lähes poikkeuksetta päivittäin. Varsinkin älypuhelimet ja internet ovat totuttaneet meidät siihen, että melkein jokaiseen asiaan on olemassa jonkinlainen sovellus, joka helpottaa asioiden hoidossa. Älylaitteiden omistajat käyttävät yleensä useampia sovelluksia mobiililaitteillaan. Jatkuvasti lisääntyvä mobiilisovellusten käyttö haastaa sovellusten tuottajia pysymää muuttuvissa trendeissä mukana luomalla toimialalla jatkuvaa kilpailua. Sovellusten tuottajien on huomioitava sovelluksissaan käyttäjälähtöisyys ja toimivuus, jotta uusia käyttäjiä saadaan sovelluksen piiriin, ja jotta nykyiset käyttäjät saadaan pysymään sovelluksen käyttäjinä.

Käyttäjät vaativat käyttämältään sovellukselta hyvää laatua sekä toimivuutta heti ensimmäisestä käyttökerrasta alkaen. Ensimmäinen käyttökerta vakuuttaa käyttäjän sovelluksen toimivuudesta ja hyödyllisyydestä. Vain noin neljäsosa käyttäjistä jatkaa mobiilisovelluksen käyttöä ensimmäisen käyttökerran jälkeen. (Why Quality Assurance Is a Crucial Part of Software Development 2020.) Jotta käyttäjät saataisiin jatkamaan mobiilisovelluksen käyttöä, tulee sovelluksen olla laadukas ja vastata käyttäjän tarpeisiin. Tämä puolestaan vaatii sovelluksen testaamista ja laadunvalvontaa sovelluksen kehittäjiltä.

Nykypäivänä myös yhä useampi sovellusten käyttäjä on tietoisempi sovelluskehityksen tarjoamista mahdollisuuksista, mikä haastaa entisestään sovelluskehityksen ja nostaa testaamisen aiempaa tärkeämpään rooliin sovelluksen kehittämisessä. Jotta käyttäjät saataisiin palaamaan mobiilisovelluksen pariin ensimmäisen käyttökerran jälkeen, tulee sovelluskehittäjien kiinnittää aiempaa enemmän huomiota testaamiseen ja asiakasystävälliseen käyttökokemukseen. Testausmenetelmäksi soveltuu hyvin esimerkiksi integraatiotestaaminen, jolla voidaan testata mobiilisovelluksen ja palvelinohjelman kommunikaation toimivuutta.

1.2 Toimeksiantaja ja tehtävä

Toimeksiantajaksi tähän opinnäytetyöhön on valittu Document Ocean Oy, joka on perustettu vuonna 2019 ja siihen kuuluu neljä perustajajäsentä. Yritys tarjoaa asiakkailleen kokonaisvaltaista palvelua, jolla voidaan luoda, hallinnoida sekä allekirjoittaa digitaalisesti erilaisia dokumentteja niin verkossa kuin mobiililaitteillakin. Document Ocean Oy:n asiakaskunta koostuu pääosin rakennusalan yrityksistä, joissa tehdään paljon niiden yritystoimintaan liittyviä erilaisia dokumentteja tai asiakirjoja.

Tämän opinnäytetyön tavoitteena on suunnitella ja toteuttaa Document Ocean Oy:n hallinnoimaan Docean-palvelun palvelinohjelmistoon integraatiotestaus, joka kattaa kriittisimmät palvelinohjelmiston toiminnot sekä tietoturvakartoituksessa löydettyjen haavoituksien korjaukset. Lisäksi tavoitteena on luoda yhtenäinen pohja tehdä integraatiotestejä tulevaisuuden uusille ominaisuuksille. Integraatiotestien avulla varmistetaan web-rajapinnan toimivuus sekä tietoturvallisuus, mikä parantaa Docean-palvelun tietoturvaa, toimivuutta ja lopullista käyttökokemusta.

1.3 Tutkimusmenetelmä

Tutkimusmenetelmänä käytetään tutkimuksellista kehitystyötä, jossa yhdistyy kehittämistoiminta tutkivalla otteella sekä tutkimuksessa saadun aineiston ja tulosten analysointi (Tutkimuksellinen kehittämistyö, 2020). ”Lähtökohtana ovat työelämästä nouseva käytännön ongelma ja kysymykset, jotka ohjaavat tiedon tuottamista käytännön toimintaympäristössä.” (mt). Tätä tutkimusmenetelmää käytetään, koska opinnäytetyön tutkimus on ominaisuuksiltaan luova ja systemaattinen (Tutkimus- ja kehittämistoiminta, 2020).

Opinnäytetyössä tutkitaan, miten voidaan parhaiten soveltaa integraatiotestien tekemistä toimeksiantajan palvelinohjelmistoon. Kehitystyön aikana toteutetaan integraatiotestaus Visual Studio 2019 -ohjelmistolla toimeksiantajan palvelinohjelmiston kriittisimmille toiminnoille sekä luodaan toimeksiantajalle yhtenäinen pohja integraatiotestien luomiseen jatkossa.

2 Ohjelmistotestaus

2.1 Yleistä

Ohjelmistotestaamisella tarkoitetaan sovelluksen ja sen komponenttien toiminnan testaamista. Sillä pyritään varmistamaan, että ohjelmisto toimii niin kuin se on suunniteltu toimimaan sekä pyritään sulkemaan pois mahdolliset ohjelmoinnissa tapahtuneet inhimilliset virheet. Ohjelmistotestauksen perusteiden (2020) mukaan ohjelmistotestaamista toteutetaan eri menetelmillä, joita ovat yksikkötestaus, integraatiotestaus, järjestelmätestaus ja hyväksyntätestaus.

Yksikkötestauksessa testataan ohjelmiston osia pienimmässä mahdollisessa koossa. Niiden tarkoituksena on varmistaa, että jokin pieni ohjelman funktio tai aliohjelma toimii oikein. Yksikkötestien pitäisi toimia ympäristöstä riippumatta sekä voida ajaa milloin vain.

Integraatiotestauksessa testataan isompaa osaa ohjelmasta. Sen tarkoituksena on varmistaa erilaisten ohjelman moduulien yhdessä toimiminen. Integraatiotestit ovat jo jonkin verran ympäristöstä riippuvaisia ja saattavat vaatia tietokantayhteyksiä.

Järjestelmätestauksessa testataan jo koko ohjelman toimintaa sille tarkoitetussa ympäristössä. Näillä testeillä pitäisi ikään kuin simuloida ohjelmiston oikeaa toimintaa oikeanlaisilla laitteilla ja kuormilla.

Hyväksyntätestauksissa varmistetaan, että ohjelma toimii niin kuin sen on tarkoitettu toimivan. Hyväksyntätestien määrittelyssä on vahvasti mukana käyttötapaukset ja asiakas. Hyväksyntätestien perusteella ohjelma luovutetaan asiakkaalle tai laitetaan tuotantoon, sillä nämä testit määrittelevät, milloin ohjelmisto on luovutuskunnossa.

2.2 Integraatiotestaus

Integraatiotestauksella testataan isompia kokonaisuuksia ohjelmistosta sekä erillisten ohjelmiston moduulien yhteistoimivuutta. Tällaisia tapauksia ovat esimerkiksi mobiilisovelluksen ja palvelinohjelmiston rajapintojen sekä palvelinohjelmiston ja muiden mahdollisten ohjelmistojen rajapintojen yhteistoimivuuden testaaminen. Kyseisillä tapauksilla voidaan löytää mahdolliset virheet rajapintojen integroinnissa. Ohjelmointivirheiden sijaintien löytyminen ei ole yhtä helppoa ja nopeaa integrointitestaamisessa kuin yksikkötestaamisessa, koska integraatiotestit käyttävät yleensä useita ohjelmiston funktiota. Integraatiotestejä ajaessa testituloksen oltaessa negatiivinen, voi ohjelmointivirhe sijaita joko rajapinnassa tai rajapintaa käyttävässä ohjelmistossa.

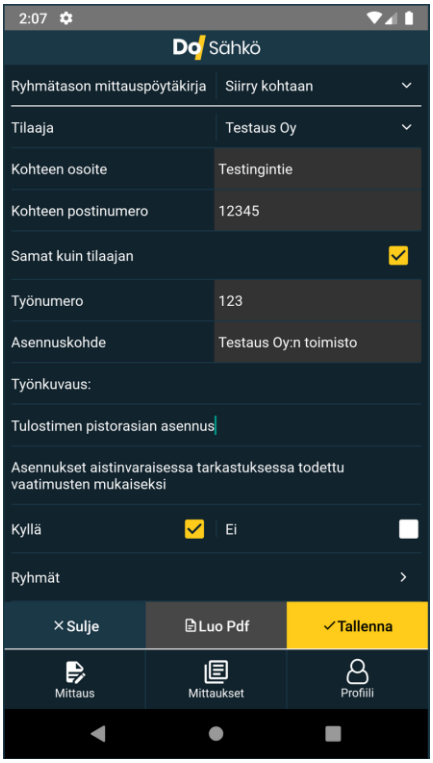
2.3 Testauksen merkitys laatuun

Ohjelmiston laatua voidaan varmistaa testaamalla ohjelmistoa monipuolisesti eritasoisilla ohjelmistotesteillä. Ohjelmistotestauksella on suoranainen vaikutus ohjelmiston laatuun, jota vaikuttaa käyttökokemukseen ja asiakastyytyväisyyteen. Ohjelmistotestaaminen on tietynlaista ohjelmiston laadunvarmistusta, mitä ei voida jättää huomioimatta, jos halutaan pitää käyttäjät tyytyväisinä sekä lisätä käyttäjämäärää ohjelmistolle. Huonolla tai puutteellisella ohjelmiston testaamisella ei saada riittävää laadunvarmistusta, mikä näkyy käyttäjillä negatiivisina käyttökokemuksina. Tällaiset epämiellyttävät käyttökokemukset voivat karkottaa potentiaalisia käyttäjiä, kun ohjelmistoa kokeilleet käyttäjät jakavat negatiivisen käyttökokemuksensa uusille mahdollisille käyttäjille kohderyhmässä. Tällainen tapahtumaketju voi näkyä haitallisena jopa uusien käyttäjien ja asiakkaiden hankinnassa saakka, kun käyttäjä ei halua edes kokeilla ohjelmistoa kuulemansa perusteella. (Laadunvarmistus ja ohjelmistotestaus, 2020.)

3 Testauskohde ja siinä käytettävät teknologiat

3.1 Yleistä testauskohteesta

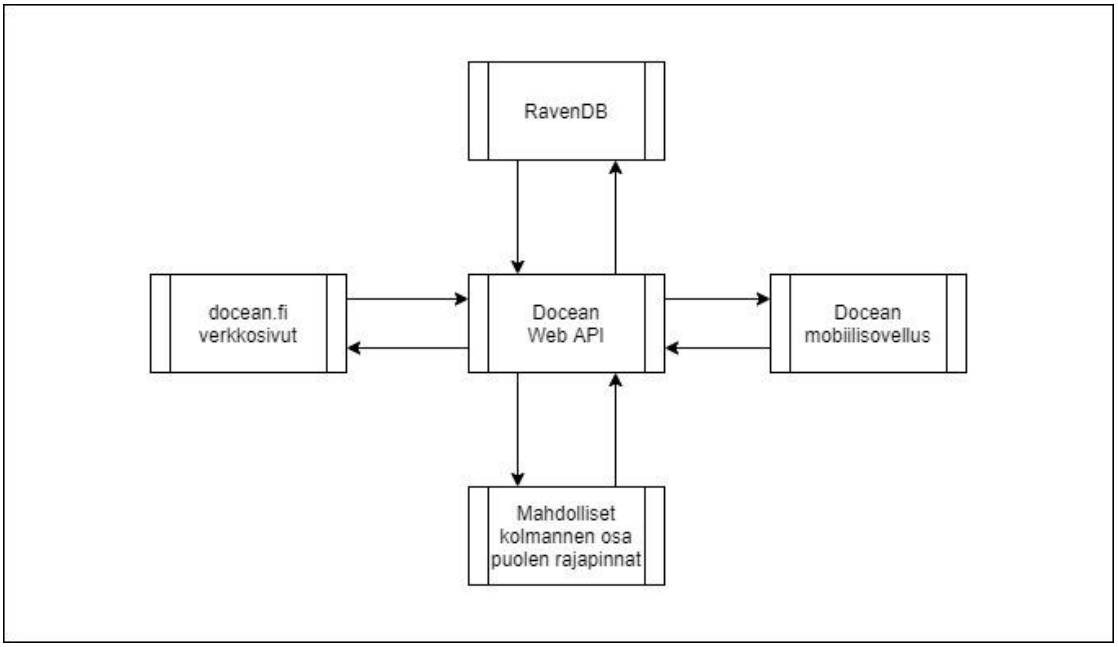
Docean-palvelulla helpotetaan ja nopeutetaan dokumenttien luomista ja allekirjoittamista asiakasyrityksessä siirtymällä paperisesta dokumentoinnista täysin digitaaliseen dokumentointiin. Palvelu tarjoaa asiakkaalleen käyttönsä sekä web- että mobiilisovellukset, joissa molemmissa voidaan hallinnoida dokumentteja sekä muita dokumentteihin liitettäviä tietoja. Kuviossa 1 on esitetty dokumentin muokkausta mobiilisovelluksessa.



Kuvio 1. Dokumentin muokkaus Docean-mobiilisovelluksessa.

Testauksen kohteena on toimeksiantajan Docean-palvelun ASP.NET Core -ohjelmistokehyksellä toteutettu palvelinohjelmisto, joka palvelee web- ja mobiilisovellusta web-ohjelmointirajapinnan kautta. Palvelinohjelmisto käyttää tietokantana RavenDB NoSQL -tietokantaa. Kohteeseen on myös integroitu Osuuspankin tunnistautumispalvelu, jossa voidaan tunnistaa henkilöitä pankkitunnistautumisen avulla. Kohdetta voidaan testata joko paikallisesti ajamalla tietokoneella palvelinohjelmistoa tai testin voi

suoraan ajaa myös palvelimelle, missä palvelinohjelmistoa ajetaan. Kuviossa 2 on esitetty web-rajapinnan kanssa keskustelevat ohjelmat ja tietokanta.



Kuvio 2. Web-rajapinnan kuvaus.

Ohjelmiston web-rajapintaan on luotu käyttäjätilien valvonta eli UAC (User Account Control). Sillä määritetään käyttäjän mukaan käyttäjän pääsy eri web-rajapinnan toimintoihin. Kuviossa 3 on esitetty kuinka UAC on toteutettu toiminnoissa, jolla voidaan hallinnoida yrityksen tietoja. Web-rajapinnan monet toiminnot on myös suojattu käyttäen JWT:tä (JSON Web Tokens).

Rooli	Trial	User	Manager	Boss	Salesman	Admin
Voi lisää yrtityksen	EI	EI	EI	EI	KYLLÄ	KYLLÄ
Voi muokkaa yrityksen tietoja	KYLLÄ	EI	EI	KYLLÄ	KYLLÄ	KYLLÄ
Voi poistaa yrityksen	EI	EI	EI	EI	EI	KYLLÄ
Voi lisätä samantasoisen käyttäjän	EI	EI	EI	KYLLÄ	EI	EI
voi lisätä alemman tason käyttäjän	EI	EI	KYLLÄ	KYLLÄ	KYLLÄ	KYLLÄ
voi lisätä esitäytettyjä dokumennti pohjia	EI	EI	KYLLÄ	KYLLÄ	KYLLÄ	KYLLÄ
Voi poistaa toisen käyttäjän yrityksestään	EI	EI	EI	KYLLÄ	EI	KYLLÄ
Voi poistaa minkätahansa käyttäjän	EI	EI	EI	EI	EI	KYLLÄ

Kuvio 3. UAC yrityksen tietojen hallintaan.

Testauskohteeseen on myös teetetty ulkopuolisella IT-yrityksellä tietoturvakartoitus, jossa ilmeni lieviä ohjelmiston sisäisiä tietoturvahaavoittuvuuksia. Haavoittuvuuksia pystyi hyödyntämään vain siinä tapauksessa, jos hyödyntävä taho olisi pääsy ohjelmistoon käyttäjätunnuksien kanssa. Näin suoranaista ulkoista uhkaa haavoittuvuuksista ei koitunut testauskohteelle.

3.2 Testattavat rajapinnan kontrollerit

Testauskohteessa keskitytään testaamaan web-rajapinnan toimintoja. Toiminnot on jaoteltu kolmeentoista eri kontrolleriin, joista jokainen käsittelee palvelinohjelmiston tiettyä aluetta:

- **Admin -**kontrolleri käsittelee pääkäyttäjätoimintoja, joita on esimerkiksi käyttäjän tai asiakasyrityksen lisääminen järjestelmään.
- **Attachments -**kontrolleri käsittelee kaikki liitetiedostoihin liittyvät toiminnot.
- **Customers -**kontrolleri käsittelee asiakkaiden asiakasrekisteriin liittyvät toiminnot.
- **Docs -**kontrolleri käsittelee dokumenttien lisäämiset, haut ja allekirjoittamisen.
- **Eidentify -**kontrolleri käsittelee tunnistautumispalveluun liittyvän toiminnallisuuden.
- **Email -**kontrolleri käsittelee sähköpostin lähettämisen ohjelmistossa.
- **Errors -**kontrolleri käsittelee mobiiliohjelmiston virheiden lähetyksen ja ilmoittaa asiakkaalle, jos ohjelmistosta on saatavilla uusi päivitys.
- **Layouts -**kontrolleri käsittelee dokumenttipohjien hallinnan.
- **MainContractors -**kontrolleri käsittelee pääurakoitsijoiden hallinnan.
- **Options -**kontrolleri käsittelee asetusten ja yleisen datan hallinnan.
- **Sites -**kontrolleri käsittelee työmaatietojen hallinnan.
- **Users -**kontrolleri käsittelee käyttäjän tietojen hallinnan sekä sisään- ja uloskirjautumisen.
- **Web -**kontrolleri käsittelee erillistoiminnot, jotka ovat vain käytössä web-ohjelmistossa.

Näistä kontrollereista tärkeimpiä testaamisen kannalta ovat Admin, Docs, Eidentify, Options ja Users -kontrollerit, jotka oikein toimiessaan mahdollistavat uusien asiakkaiden ja käyttäjien lisäämisen, asiakkaiden käytettävän datan hallinnan sekä dokumenttien luonnin ja allekirjoittamisen. Admin-kontrollerin testaaminen UAC:in kanssa on todella tärkeää, jotta voidaan varmistaa, ettei alemman tason käyttäjällä voi päästä käsiksi pääkäyttäjien tekemiin ominaisuuksiin.

3.3 JSON Web Tokens

JWT on yleisesti web-rajapinnoissa käytetty turvallinen tapa siirtää tietoja eri tahojen välillä JSON-objekti muodossa. Sen yleisimmät käyttötarkoitukset ovat tiedon vaihtaminen eri osapuolien välillä tai käyttäjän varmentaminen sisäänkirjautumisen jälkeen. JWT:n sisältö voidaan salata joko algoritmia tai RSA/ECDSA-avainparia käyttäen. JWT muodostuu kolmesta eri osasta, joita ovat Header, Payload ja Signature. Header sisältää tiedot käytetystä algoritmista ja tokenin tyypistä. Payloadiin tallennetaan tiedot, joita halutaan siirtää taholta toiselle. Signature-osalla varmistetaan, ettei JWT ole muuttunut siirron aikana käyttämällä avainpareja, jolloin vastaanottaja voi varmentaa, että lähettäjä todella on luotettava taho. (Introduction to JSON Web Tokens 2020.)

3.4 ASP.NET Core -ohjelmistokehys

ASP.NET Core on Microsoftin julkaisema alustariippumaton, suorituskykyinen ja avoimen lähdekoodin ohjelmistokehys, jolla voidaan luoda nykyaikaisia pilvipohjaisia internettiin kytkettyjä sovelluksia. Se on suosittu ja sitä voi ohjelmoida Microsoftin kehittämillä F#- ja C#-kielillä. (Introduction to ASP.NET Core 2020.)

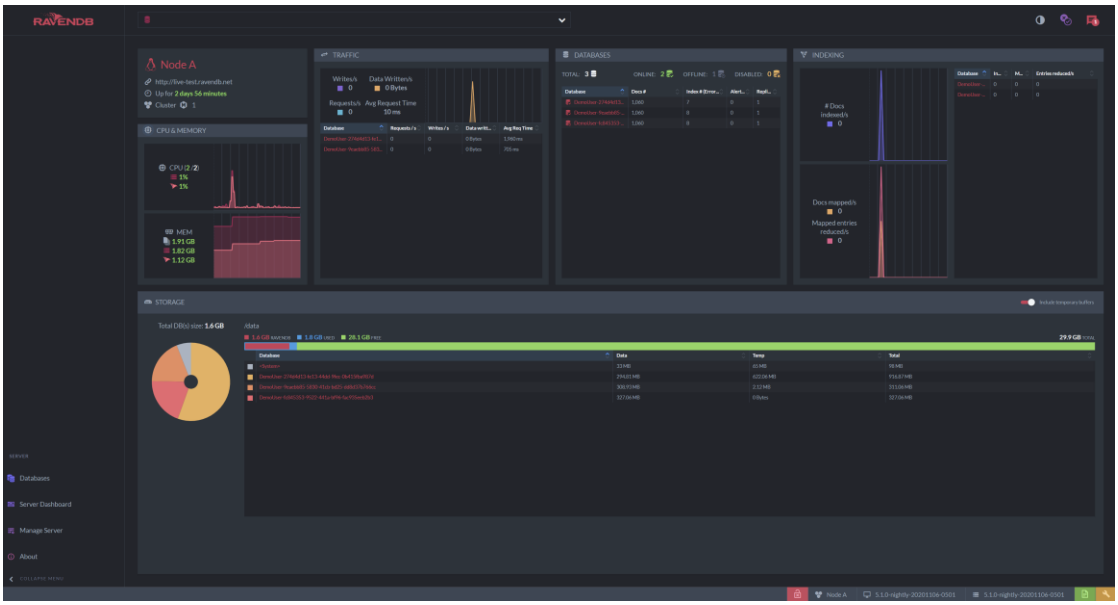
3.5 Web-ohjelmointirajapinta

Web-ohjelmointirajapinta mahdollistaa tietoja tai niiden käsittelyä toisille sovelluksille tai muille tietojärjestelmille verkon ylitse. Sama rajapinta pystyy palvelemaan useita eri sovelluksia tai tietojärjestelmiä. (What is Web API 2020.)

Ohjelmointirajapinta voi olla toiminnallinen tai datarajapinta. Datarajapinta tarjoaa ainoastaan tietoja, kun taas toiminnallinen rajapinta tarjoaa mahdollisuuden muuttaa tietoja järjestelmässä rajapinnan kautta. (Rajapinta 2020.)

3.6 RavenDB

RavenDB on Hibernating Rhinosin kehittämä täysin transaktionaalinen NoSQL-tietokanta, joka sijoittuu kymmenen käytetyimmän NoSQL-tietokannan joukkoon. Ensimmäinen 1.0 versio siitä julkaistiin vuonna 2008. Nykyinen versio tietokannasta on 5.0, joka on julkaistu vuonna 2020. (About RavenDB 2020.)



Kuvio 4. RavenDB:n käyttöliittymän etusivu

Tietokanta tallentaa datan JSON-muotoisina dokumentteina sekä sisältää web-käyttöliittymän tietokannan hallinnointiin ja selaamiseen. Kuviossa 4 on esitetty RavenDB:n web-käyttöliittymä, jolla voidaan hallinnoida tietokantaa ja sen asetuksia.

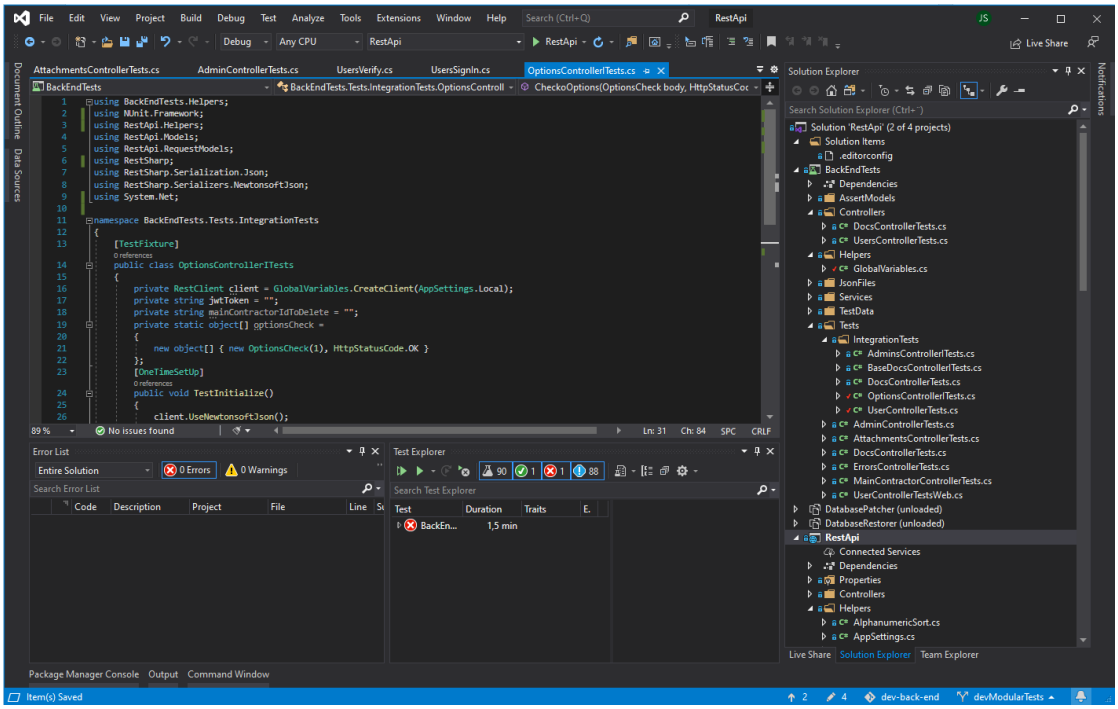
4 Testauksessa käytetyt työkalut

4.1 Testauskokoonpano

Testaamiseen käytettiin Visual Studio 2019 -ohjelmistoa, jonka kanssa käytettiin testausta helpottavia kirjastoja: Nunit ja RestSharp. Nämä kirjastot yksinkertaistivat koodia ja helpottivat rajapintaan tehtävien pyyntöjen tekemistä, mikä helpottaa testien luontia ja ajamista.

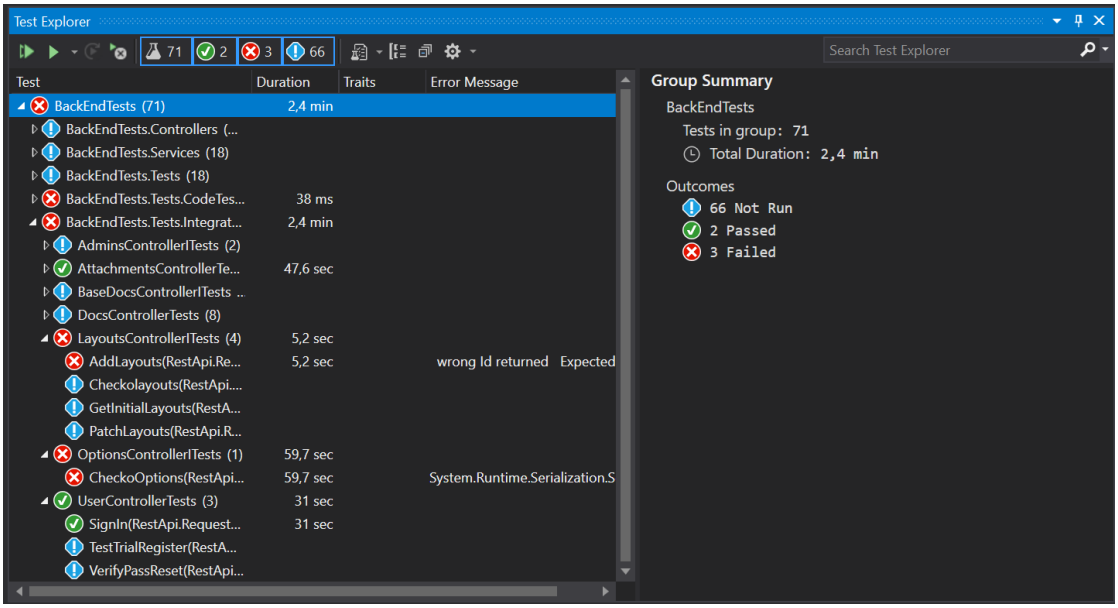
4.2 Visual Studio 2019

Visual Studio 2019 on Microsoftin kehittämä ohjelmoijille suunnattu sovellus, jolla voidaan kehittää, analysoida, debugata, testata, yhteistyöohjelmoida sekä julkaista ohjelmistoja. Se sopii niin pieniin kuin suuriinkin ohjelmistokehitysprojekteihin. Kuvissa 5 on esitelty Visual Studion käyttöliittymää ohjelmoitaessa.



Kuvio 5. Visual Studio 2019 -käyttöliittymä

Testaamisessa käytettiin Visual Studion sisäänrakennettua Test Explorer -ohjelmaa. Kuviossa 6 on esitelty Test Explorerin -käyttöliittymää, mikä etsii automaattisesti tehdyt testit ja sillä voi hallinnoida luotuja testejä sekä saada muuta hyödyllistä tietoa testeistä. Testien hallintaan kuuluu esimerkiksi testien ajoa, virheiden etsimistä testeistä, tulosten tarkastelua sekä ryhmittelyä. (Run unit tests with Test Explorer 2020.) Microsoftin Azure DevOps -palvelua käyttäessä on myös mahdollista liittää kyseiset testitapaukset suoraan Azure DevOps:ssa luotuihin testaus dokumentaation testitapauksiin. Azure DevOps on Microsoftin kehittämä ohjelmistoprojekteille tarkoitettu hallinta ympäristö.



Kuvio 6. Test Explorer -näkymä

4.3 NUnit-ohjelmistokehys

NUnit on avoimen lähdekoodin testaukseen tarkoitettu Visual Studiolle luotu ohjelmistokehys, joka toimii kaikkien .NET-ohjelmistokielien kanssa. Se on lisensoitu MIT-lisenssillä ja on ilmainen käyttää kaupallisissa sovelluksissa ilman rajoituksia. (Nunit 2020.) Kuviossa 7 on esitelty NUnit:a käyttäen tehty testausluokka.

```
namespace BackEndTests.Tests.IntegrationTests
{
    [TestFixture]
    0 references
    public class OptionsControllerITests
    {
        private RestClient client = GlobalVariables.CreateClient(AppSettings.Local);
        private string jwtToken = "";
        private static object[] optionsCheck =
        {
            new object[] { new OptionsCheck(1), HttpStatusCode.OK }
        };
        [OneTimeSetUp]
        0 references
        public void TestInitialize()
        {
            client.UseNewtonsoftJson();
            jwtToken = GlobalVariables.SignIn(new UsersSignIn(1), client);
        }

        [Test, TestCaseSource("optionsCheck")]
        0 references
        public void CheckoOptions(OptionsCheck body, HttpStatusCode exceptedResult)
        {
            var request = GlobalVariables.CreateRequestWithToken("options/check", Method.POST, jwtToken, body);
            var response = client.Execute(request);
            var deserialize = new JsonDeserializer();
            var output = deserialize.Deserialize<SendCustomersAndOptions>(response);
            Assert.That(response.StatusCode, Is.EqualTo(exceptedResult), "request failed");

        }
    }
}
```

Kuvio 7. NUnit:a käyttäen tehty testausluokka

4.4 RestSharp-kirjasto

RestSharp on .NET-ohjelmistokehykselle suunnattu HTTP-pyyntöjen luonti- ja käyttökirjasto. Se helpottaa HTTP-pyyntöjen tekemistä rajapinnoille sekä vastauksien käsittelyä ja sarjoittamista. (Introduction 2020.) Kuviossa 8 on luotu testausta varten RestSharp-kirjastoa käyttäen HTTP-pyyntö, joka lähetetään web-rajapinnalle.

```
        /// <summary>
        /// Creates request without authentication header
        /// </summary>
        /// <param name="endpoint">Endpoint where direct the request</param>
        /// <param name="method">Request method</param>
        /// <param name="body">Request body</param>
        11 references | 1/9 passing
        public static RestRequest CreateRequestWithoutToken(string endpoint, Method method, object body = null)
        {
            var request = new RestRequest(endpoint, method);
            request.UseNewtonsoftJson(jsonSettings);
            request.RequestFormat = DataFormat.Json;
            if(body != null)
            {
                request.AddJsonBody(body);
            }
            return request;
        }
```

Kuvio 8. HTTP-pyynnön luonti ja lähettäminen web-rajapinnalle RestSharp-kirjastolla

5 Testauksen toteutus

5.1 Testiprojektin luominen

Testejä varten palvelinohjelman ohjelmistokokonaisuuteen lisättiin uusi projekti, jossa käytettiin NUnit Test Project (.NET core) -sapluunaa. Jokaiselle testattavalle kontrollerille luotiin omat testiluokat, joissa testattiin kyseisen kontrollerin toimintoja. Lisäksi luotiin erillinen apufunktioluokka, jonne sijoitettiin kaikki yleishyödylliset funktiot, joita kontrollerien eri testitluokat voivat yhdessä käyttää.

5.2 Testien apufunktioluokka GlobalVariables

5.2.1 HTTP-clientin luonti

Kuviossa 9 on esitelty funktio CreateClient, joks luo RestSharp-kirjastoa käyttäen HTTP-clientin. Funktiolle annetaan parametrille url verkko-osoite, jonne pyyntö tullaan luomaan. Funktiota käytetään jokaisen testausluokan alussa, jolla alustetaan testausluokkien HTTP-client.

```
/// <summary>
/// Creates new rest client
/// </summary>
/// <param name="url">Server url where requests are sent</param>
13 references
public static RestClient CreateClient(string url)
{
    var client = new RestClient(url);
    client.UseNewtonsoftJson();
    return client;
}
```

Kuvio 9. HTTP-clientin luonti

5.2.2 Käyttäjän sisäänkirjaus

Kuviossa 10 on esitetty SignIn-funktio, jolla voidaan kirjata testikäyttäjä sisään. Funktio ottaa vastaan käyttäjäolion, jossa on sisäänkirjautumistiedot sekä HTTP-clientin.

Funktiosta palautuu takaisin käyttäjän pohjalta luotu JWT, mikä mahdollistaa JWT:llä suojattujen toimintojen testaamisen.

```
/// <summary>
/// Signin user and return logintoken
/// </summary>
7 references
public static string SignIn(UsersSignIn user, RestClient client)
{
    var request = CreateRequestWithoutToken("web/signin", Method.POST, user);
    request.UseNewtonsoftJson(jsonSettings);
    var response = client.Execute(request);
    var deserialize = new JsonDeserializer();
    var output = deserialize.Deserialize<ResponseUsersSignIn>(response);
    return output.Auth.Token;
}
```

Kuvio 10. Käyttäjän sisäänkirjaaminen

5.2.3 HTTP-pyynnön luonti JWT:n kanssa ja ilman JWT:tä

Kuviossa 11 on esitetty, kuinka luodaan funktio, jolla lähetetään HTTP-pyyntöjä web-rajapinnan avoimiin toimintoihin ja joita ei ole suojattu JWT:llä. Funktio on monipuolinen ja sillä voidaan luoda erityyppisiä pyyntöjä. Parametrien avulla voidaan määrittää mille web-rajapinnan toiminnolle pyyntö lähetetään, mikä on HTTP-pyynnön metodi sekä mahdollinen pyynnön sisään tuleva data.

```
/// <summary>
/// Creates request without authentication header
/// </summary>
/// <param name="endpoint">Endpoint where direct the request</param>
/// <param name="method">Request method</param>
/// <param name="body">Request body</param>
14 references | 0/12 passing
public static RestRequest CreateRequestWithoutToken(string endpoint, Method method, object body = null)
{
    var request = new RestRequest(endpoint, method);
    request.UseNewtonsoftJson(jsonSettings);
    request.RequestFormat = DataFormat.Json;
    if(body != null)
    {
        request.AddJsonBody(body);
    }
    return request;
}
```

Kuvio 11. HTTP-pyynnön luonti ilman JWT:tä

Kuviossa 12 on esitetty HTTP-pyynnön luonti JWT kanssa, jota käytetään, kun on tarve luoda pyyntö JWT:llä suojattuihin web-rajapinnan toimintoihin. Funktio on lähes samanlainen kuin edellinen esitetty funktio. Erona on vain, että funktiolle lähetetään jwt-niminen parametri, joka lisää pyyntöön JWT:n.

```
/// <summary>
/// Creates request with token and application/json body.
/// </summary>
/// <param name="endpoint">Endpoint where direct the request</param>
/// <param name="method">Request method</param>
/// <param name="jwt">JWT token for authentication header</param>
/// <param name="body">Request body</param>
32 references | 0/27 passing
public static RestRequest CreateRequestWithToken(string endpoint, Method method, string jwt, object body)
{
    var request = new RestRequest(endpoint, method);

    request.UseNewtonsoftJson(jsonSettings);
    request.RequestFormat = DataFormat.Json;
    request.AddHeader("Authorization", $"Bearer {jwt}");
    request.AddHeader("Content-Type", "application/json; charset=utf-8");
    if (body != null)
        request.AddJsonBody(body);
    return request;
}
```

Kuvio 12. HTTP-pyynnön luonti JWT:n kanssa

5.3 Testausluokat

Kuviossa 13 on esitetty kontrollerien testausluokkien koodijäsentely. Alussa luodaan testausluokka, joka määritetään käyttämään NUnit-ohjelmistokehystä käyttämällä testiluokalle attribuuttia TestFixture. Sen jälkeen alustetaan testausluokan client-kenttä apufunktioluokan CreateClient-funktiolla sekä jwtToken-kenttä, johon voidaan tallentaa mahdollinen pyynnöstä palautunut JWT.

```
[TestFixture]
0 references
public class UserControllerTests
{
    private RestClient client = GlobalVariables.CreateClient(AppSettings.Local);
    private string jwtToken = "";
```

Kuvio 13. Testausluokan kenttien alustaminen

NUnit:n OneTimeSetUp-attribuutilla voidaan määrittää alustusfunktio, joka ajetaan ennen jokaista testiä. Useimmissa kontrollerien testiluokissa on kuvion 14 mukaisesti luotu testien alustusfunktio, jolla kirjataan käyttäjä sisään käyttämällä apufunktioluokan SignIn-funktiota.

```
[OneTimeSetUp]
0 references
public void TestInitialize()
{
    client.UseNewtonsoftJson();
    jwtToken = GlobalVariables.SignIn(new UsersSignIn(3), client);
}
```

Kuvio 14. Testiluokan testien alustusfunktio

Kuviossa 15 on esitetty GetPdf-testifunktio. NUnit:ssa määritellään testifunktiot attribuutilla Test sekä testin käyttämät testitapaukset joko attribuuteilla TestCase tai TestCaseSource. Testifunktion yläpuolella on getPdf-kentän sisällä ajettavat testitapaukset. Testillä varmistettiin, ettei web-rajapinnassa voida käyttää haitallista path travelsal -haavoittuvuutta, jolla voitaisiin päästä käsiksi mihinkä tahansa tiedostopolkuun palvelimella.

```
private static object[] getPdf =
{
    new object[] { new DocsGetPdf() { DocRef = "%2e%2e%2flol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "%2e%2e/lol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "..%2flol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "%2e%2e%5clol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "..%255clol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "..%u2216lol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "%cc%b7lol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "%cc%b8lol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "%e2%81%84lol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "%e2%88%95lol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "%ef%bc%8flol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "%c1%1clol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "%c0%aflol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "../lol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "../../lol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "../../../lol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "../../../../lol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "/kissa/lol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "/kissa/kala/lol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "/kissa/../lol" }, HttpStatusCode.NotFound },
    new object[] { new DocsGetPdf() { DocRef = "lol" }, HttpStatusCode.OK },
};

[Test, TestCaseSource("getPdf")]
0 references
public void GetPdf(DocsGetPdf body, HttpStatusCode exceptedResult)
{
    var request = GlobalVariables.CreateRequestWithToken("docs/getPdf", Method.POST, jwtToken, body);
    var response = client.Execute(request);
    Assert.That(response.StatusCode, Is.EqualTo(exceptedResult), "request failed");
}
```

Kuvio 15. Testaus funktio ja sen testitapaukset

5.4 Testien tulokset

Kehitystyön aikana ehdittiin tekemään sekä ajamaan testejä neljälle eri kontrollerille. Kontrollerit sekä tehdyt testit valikoituivat sovelluksen käytettävyyden kannalta olennaisimpiin ominaisuuksiin. Ajetut testit on esitelty alla olevissa taulukoissa 1.–4. kontrollereittain.

Taulukko 1. Admin-kontrollerille ajetut testit.

Admin-kontrollerin ajetut testit				
Toiminto	Ajettu roolilla	Oletettu HTTP-statuskoodi	Token	Testin tulos
Käyttäjän lisäys (>= Boss)	SuperAdmin	469	kevyt	OK
	Admin	469	kevyt	OK
	Boss	469	kevyt	OK
	Manager	469	kevyt	OK
	User	403	kevyt	OK
Käyttäjän lisäys (Manager)	SuperAdmin	200	kevyt	OK
	Admin	200	kevyt	OK
	Boss	200	kevyt	OK
	Manager	469	kevyt	OK
	User	403	kevyt	OK
Käyttäjän lisäys (User)	SuperAdmin	200	kevyt	OK
	Admin	200	kevyt	OK
	Boss	200	kevyt	OK
	Manager	200	kevyt	OK
	User	403	kevyt	OK
Data credittien lisäys	SuperAdmin	200	kevyt	OK
	Admin	200	kevyt	OK
	Boss	403	kevyt	OK
	Manager	403	kevyt	OK
	User	403	kevyt	OK
Lisenssi credittien lisäys	SuperAdmin	200	kevyt	OK
	Admin	200	kevyt	OK
	Boss	403	kevyt	OK
	Manager	403	kevyt	OK
	User	403	kevyt	OK
Yrityksen tietojen haku	SuperAdmin	200	kevyt	OK
	Admin	200	kevyt	OK
	Boss	200	kevyt	OK
	Manager	200	kevyt	OK
	User	403	kevyt	OK

Taulukko 2. Attachment-kontrollerille ajetut testit.

Attachment-kontrollerin ajetut testit				
Toiminto	Ajettu roolilla	Oletettu HTTP-statuskoodi	Token	Testin tulos
Yrityslogon päivitys vanhalla aikaleimalla	SuperAdmin	200	kevyt&vahva	OK
	Admin	200	kevyt&vahva	OK
	Boss	200	kevyt&vahva	OK
	Manager	200	kevyt&vahva	OK
	User	200	kevyt&vahva	OK
Yrityslogon päivitys ajantasaisella aikaleimalla	SuperAdmin	204	kevyt&vahva	OK
	Admin	204	kevyt&vahva	OK
	Boss	204	kevyt&vahva	OK
	Manager	204	kevyt&vahva	OK
	User	204	kevyt&vahva	OK
Uuden logon uploadaus	SuperAdmin	200	kevyt	OK
	Admin	200	kevyt	OK
	Boss	200	kevyt	OK
	Manager	403	kevyt	OK
	User	403	kevyt	OK
Yrityslogon haku	SuperAdmin	200	kevyt&vahva	OK
	Admin	200	kevyt&vahva	OK
	Boss	200	kevyt&vahva	OK
	Manager	200	kevyt&vahva	OK
	User	200	kevyt&vahva	OK
Liitetiedoston lisäys	SuperAdmin	200	kevyt&vahva	OK
	Admin	200	kevyt&vahva	OK
	Boss	200	kevyt&vahva	OK
	Manager	200	kevyt&vahva	OK
	User	200	kevyt&vahva	OK
Webissä lisätyn liitetiedoston haku	SuperAdmin	200	kevyt&vahva	OK
	Admin	200	kevyt&vahva	OK
	Boss	200	kevyt&vahva	OK
	Manager	200	kevyt&vahva	OK
	User	200	kevyt&vahva	OK

Taulukko 3. Docs-kontrollerille ajetut testit.

Docs-kontrollerin ajetut testit				
Toiminto	Ajettu roolilla	Oletettu HTTP-statuskoodi	Token	Testin tulos
Uuden dokumentin lisäts webissä, lisättiin 5 erilaista dokumenttia	SuperAdmin	200	kevyt	OK
	Admin	200	kevyt	OK
	Boss	200	kevyt	OK
	Manager	200	kevyt	OK
	User	200	kevyt	OK
Uuden dokumentin lisäys mobiililla, lisättiin 5 erilaista dokumenttia	SuperAdmin	200	vahva	OK
	Admin	200	vahva	OK
	Boss	200	vahva	OK
	Manager	200	vahva	OK
	User	200	vahva	OK
5 erilaisen dokumentin poisto	SuperAdmin	200	kevyt&vahva	OK
	Admin	200	kevyt&vahva	OK
	Boss	200	kevyt&vahva	OK
	Manager	200	kevyt&vahva	OK
	User	200	kevyt&vahva	OK
8 erilaisen dokumentin allekirjoittaminen joissa dokumentin allekirjoitus parametrit on oikein	SuperAdmin	200	vahva	OK
	Admin	200	vahva	OK
	Boss	200	vahva	OK
	Manager	200	vahva	OK
	User	200	vahva	OK
2 erilaisen dokumentin allekirjoittaminen joissa dokumentin allekirjoitus parametrit on väärinä	SuperAdmin	400	vahva	OK
	Admin	400	vahva	OK
	Boss	400	vahva	OK
	Manager	400	vahva	OK
	User	400	vahva	OK
Valmiiden dokumenttien haku pdf muodossa, 8 erilaisen dokumentin haku	SuperAdmin	200	kevyt&vahva	OK
	Admin	200	kevyt&vahva	OK
	Boss	200	kevyt&vahva	OK
	Manager	200	kevyt&vahva	OK
	User	200	kevyt&vahva	OK
Valmiiden pdf dokumenttien haku, väärillä parametreilla, haettiin kahdilla eri väärillä parametreilla	SuperAdmin	404	kevyt&vahva	OK
	Admin	404	kevyt&vahva	OK
	Boss	404	kevyt&vahva	OK
	Manager	404	kevyt&vahva	OK
	User	404	kevyt&vahva	OK
Path travelsal - haavoittuvuus korjauksen testaaminen, ajettiin 20 eri parametrillä jolla saadaan parth travelsal aikaan	SuperAdmin	404	kevyt&vahva	OK
	Admin	404	kevyt&vahva	OK
	Boss	404	kevyt&vahva	OK
	Manager	404	kevyt&vahva	OK
	User	404	kevyt&vahva	OK

Taulukko 4. Users-kontrollerille ajetut testit.

Users-kontrollerin ajetut testit				
Toiminto	Ajettu roolilla	Oletettu HTTP-statuskoodi	Token	Testin tulos
Sisäänkirjautuminen mobiililla	SuperAdmin	200	kevyt&vahva	OK
	Admin	200	kevyt&vahva	OK
	Boss	200	kevyt&vahva	OK
	Manager	200	kevyt&vahva	OK
	User	200	kevyt&vahva	OK
Uloskirjautuminen	SuperAdmin	200	kevyt&vahva	OK
	Admin	200	kevyt&vahva	OK
	Boss	200	kevyt&vahva	OK
	Manager	200	kevyt&vahva	OK
	User	200	kevyt&vahva	OK
Tunnistautumisen access tokenin haku	SuperAdmin	200	kevyt	OK
	Admin	200	kevyt	OK
	Boss	200	kevyt	OK
	Manager	200	kevyt	OK
	User	200	kevyt	OK
Salasanan vaihto pyyntö	SuperAdmin	200	-	OK
	Admin	200	-	OK
	Boss	200	-	OK
	Manager	200	-	OK
	User	200	-	OK
Uuden salasanan asettaminen	SuperAdmin	200	email token	OK
	Admin	200	email token	OK
	Boss	200	email token	OK
	Manager	200	email token	OK
	User	200	email token	OK
Uuden salasanan asettaminen väärällä tokenilla	SuperAdmin	401	väärä token	OK
	Admin	401	väärä token	OK
	Boss	401	väärä token	OK
	Manager	401	väärä token	OK
	User	401	väärä token	OK
Email tokenin validointi	SuperAdmin	200	email token	OK
	Admin	200	email token	OK
	Boss	200	email token	OK
	Manager	200	email token	OK
	User	200	email token	OK

6 Pohdinta

6.1 Tavoitteet ja tulokset

Tavoitteena oli suunnitella ja luoda integraatiotestit Document Ocean Oy:n palvelinohjelmistolle, yhtenäisen pohjan luominen jatkossa käytettävien integraatiotestien tekoon sekä varmistaa tietoturvakartoituksessa löydettyjen haavoittuvuuksien paikkauksien toimivuus. Tavoitteiden pohjana oli, että toimeksiantaja halusi parantaa palvelinohjelmiston toimivuutta sekä varmistaa löydettyjen tietoturvahaavoittuvuuksien paikkaukset. Varmin tapa ohjelmiston sekä haavoittuvuuksien paikkauksien toiminnalle on luoda ohjelmistotestit. Tavoitteisiin päästiin käyttämällä oikeainlaista testausohjelmaa sekä oikeanlaisia kirjastoja, että ohjelmistokehyksiä testausohjelmiston kanssa. Työn tuloksia olivat monipuolinen ja hyvä pohja integraatiotestien tekoon, integraatiotestit ohjelmiston käytön kannalta kriittisimmille ominaisuuksille ja tietoturvahaavoittuvuus korjauksille.

Tavoitteisiin päästiin kohtalaisesti. Työn aikana onnistuttiin luomaan hyvä yhtenäinen pohja integraatiotestien luomiseen. Ohjelmiston käytön kannalta kriittisille toiminnoille sekä tietoturvakartoituksessa löydetyille tietoturvariskeille saatiin myös luotua sekä ajettua testit. Koko palvelinohjelmiston kattavia testejä ei ehditty tekemään. Ohjelmiston rajapinnat ovat laajat ja ohjelmistoon on parhaillaan kehitteillä uusi päivitys, joka muokkaa palvelinohjelmiston logiikkaa melkein jokaisessa kontrollerissa. Liikaa testejä ei täten kannattanut ryhtyä tekemään ohjelmistolle tässä kehitysvaiheessa, sillä päivityksen myötä tulee muutoksia HTTP-pyyntöjen ja vastausten sisältöön. Lisäksi monia vanhempia toimintoja ollaan poistamassa ja korvaamassa aivan uusilla toiminnoilla. Tämä aiheuttaisi sen, että kaikki testit, jotka tässä vaiheessa luodaan, joudutaan muokkaamaan päivityksen kanssa yhteentoimivaksi. Kehitystyön tärkeimmäksi tulokseksi muodostuikin, että löydettiin hyvä tapa tehdä testausluokkia, joista saatiin tehtyä mainiot testipohjat. Tämä nopeuttaa testien tekoa huomattavasti jatkossa, mikä puolestaan nopeuttaa sovelluksen kehittämistä ja päivittämistä tulevaisuudessa. Työn tulokset ovat

hyödynnettävissä erilaisissa .NET-palvelinohjelmistoissa, jotka käyttävät web-rajapintaa JSON Web Tokenien kanssa.

6.2 Jatkokehitysideat ja testien ajoitus

Testauspohjaa voisi kehittää monipuolisemmaksi luomalla ja kehittämällä testauspohjan yksikkötestejä varten. Näillä saataisiin vielä varmemmin poissuljettua virheiden mahdollisuutta testattavasta ohjelmistosta. Työn alussa yritettiin tehdä yksikkötestejä NUit:a käyttäen, mutta se ei tuntunut järkevältä vielä tässä ohjelmistokehitysvaiheessa, koska ohjelma muuttui jatkuvasti ja muokattavat tiedot olivat isoja, jolloin pienenkin päivityksen jälkeen testien päivittämiseen kului paljon aikaa. Nopeammaksi tavaksi testata palvelinohjelmisto todettiin tällöin integraatiotestit.

Integraatiotestien ja testi-funktioiden pohjalta voisi myös luoda pidempien käyttötapauksia simuloivia testejä yhdistämällä testi-funktioita yhdeksi isommaksi funktioksi, jolla voidaan ajaa useita käyttötapauksia putkeen. Teoriassa olisi mahdollista tehdä testipohjalla yksi testi, jossa sisäänkirjataan käyttäjä, luodaan, poistetaan, katsellaan ja allekirjoitetaan dokumentteja sekä lopuksi uloskirjataan käyttäjä.

Palvelinohjelmistoon on järkevintä ryhtyä tekemään testejä, kun palvelinohjelmistossa on osia valmiina eikä ohjelmiston toimintoihin tule suuria muutoksia. Tällöin testien teko on nopeinta, koska jatkuva pyyntöjen ja vastausten sisällön sekä luokkien muuttuminen ohjelmistossa aiheuttaa sen, että testien päivittämiseen ajan tasalle kuluu paljon aikaa. Ohjelmistokehityksessä on suosittua kehittää TDD (Test Driven Development) menetelmällä, missä ennen varsinaisen ohjelmistokoodin luomista luodaan ensin testit. Testin pohjalta luodaan sitten toiminto. Yksinkertaisille ja pienille ohjelmille tämä on parempi tapa kuin isommille ohjelmille. TDD-menetelmä myös vaatii sen, että suunnitteluvaiheessa on luotu erittäin tarkka vaatimusmäärittely, jonka pohjalta ohjelmisto luodaan. Ketterän kehityksen tai agile-tyyppisissä ohjelmistokehitysprojekteissa on järkevämpää luoda testit ohjelmistosta luotujen toimintojen pohjalta.

Lähteet

About RavenDB. 2020. RavenDB yleinen esittely. Viitattu 08.11.2020. https://ravendb.net/about.

Introduction. 2020. RestSharpin esittely. Muokattu 20.6.2020. Viitattu 08.11.2020. https://restsharp.dev/getting-started/.

Introduction to ASP.NET Core. 2020. ASP.NET Core 3.1 dokumentaatio. Muokattu 17.4.2020. Viitattu 8.11.2020. https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.1.

Introduction to JSON Web Tokens. 2020. JSON Web tokenien esittely. Viitattu 08.11.2020. https://jwt.io/introduction/.

Laadunvarmistus ja ohjelmistotestaus. 2020. Laadunvarmistus ja ohjelmistotestaus artikkeli Ite wikin digitalisoinnin oppaasta. Viitattu 18.11.2020. https://www.ite-wiki.fi/opas/laadunvarmistus-ja-ohjelmistotestaus/.

Nunit. 2020. Nunitin kotisivut. Viitattu 08.11.2020. https://nunit.org/.

Ohjelmistotestauksen perusteet. 2020. Artikkeli ohjelmistotestauksen perusteista Vertics ohjelmistotalon kotisivustolla. Viitattu 15.11.2020. https://vertics.co/ohjelmistotestauksen-perusteet/.

Rajapinta. 2020. Avoin rajapinta kotisivut. Viitattu 08.11.2020. http://avoinrajapinta.fi/.

Run unit tests with Test Explorer. 2020. Visual Studio 2019 dokumentaatio. Muokattu 14.7.2020. Viitattu 8.11.2020. https://docs.microsoft.com/en-us/visualstudio/test/run-unit-tests-with-test-explorer?view=vs-2019.

Tutkimus- ja kehittämistoiminta. 2020. Tutkimus- ja kehittämistoiminnan määritelmä tilastokeskuksen mukaan. Viitattu 15.11.2020. https://www.stat.fi/meta/kas/t_ktoiminta.html#tab2.

Tutkimuksellinen Kehittämistyö. 2020. Jyväskylän ammattikorkeakoulun perustiedot ja ohjeet opinnäytetyöprojektin läpiviemiselle. Viitattu 15.11.2020. https://oppimateriaalit.jamk.fi/opinnaytetyo/toteutustavat-ja-rakenne/tutkimuksellinen-kehittamistyo/.

What is Web Api. 2020. Tutorials Teacher Web API -tutoriaali. Viitattu 08.11.2020. https://www.tutorialsteacher.com/webapi/what-is-web-api.

Why Quality Assurance Is a Crucial Part of Software Development. 2020. Artikkeli Better Software Groupin sivuilla. Julkaistu 12.3.2020. Viitattu 08.11.2020. https://www.bsgroup.eu/quality-assurance-crucial-part-of-software-development/