SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

OPINNÄYTETYÖ

InWorks notifikaatiopalkki

TEKIJÄ/T: Tuomas Karhunen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Tuomas Karhunen	
Työn nimi InWorks notifikaatiopalkki	
Päiväys 20.12.2018	Sivumäärä/Liitteet 33
Ohjaaja(t) Jussi Koistinen, lehtori ja Keijo Kuosmanen, lehtori	
Toimeksiantaja/Yhteistyökumppani(t) Tauno Hyvärinen, jäjestelmäarkkitehti / työn johtaja, Solteq Oyj	

Tiivistelmä

Työssä toteutettiin inWorks tuotteeseen tilattu notifikaatiopalkki. Sen tehtävänä oli toimia kommunikaatiovälineenä inWorksin asiakkaiden ja ylläpidon välillä. Sillä oli tarkoitus pystyä tehdä yleisiä ilmoituksia ympäristön toimintaan liittyen, sekä huomauttaa asiakkaita esimerkiksi huoltokatkosta.

Hallinta työkalu toteutettiin ASP.NET MVC projektina, kun taas inWorksin notifikaatiopalkki tehtiin lisäosaksi olemassa olevaan tuotteeseen. Eniten käytettyjä ohjelmointikieliä olivat C#, TypeScript ja Angular. Muita käytettyjä tekniikoita oli SQLServer, SignalR, EntityFramework ja Razor.

Opinnäytetyön lopputuloksena valmistui prototyyppi inWorksin ilmoituspalkista. Ilmoituksia hallitaan Solteqin ylläpitoliittymällä, josta voidaan lähettää viesti haluttuihin inWorkseihin. Kun ilmoitus on luotu, päättelee palvelimen logiikka, ketkä ovat oikeutettuja lukemaan sen. Ilmoituksien hallintaan on erilaisia toimintoja. Niille voidaan määrittää mm. prioriteetti ja voimassaoloaika, joiden perusteella järjestelmä tietää kuinka tärkeä ilmoitus on. Tärkeät ilmoitukset ovat aina näkyvillä notifikaatiopalkissa. Vähemmän tärkeät ilmestyvät siihen vain lukemattomina, jos palvelimen logiikka on päätellyt, että kyseinen käyttäjä saa vastaanottaa ilmoituksen. Kaikki asiakkaalle osoitetut poistamattomat ilmoitukset ovat kuitenkin myöhemmin luettavissa ilmoitushistoriasta. Muita toimintoja hallintatyökalussa on mm. ilmoitusten poisto, palautus, kopiointi sekä muokkaus.

Avainsanat
inWorks

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Tuomas Karhunen			
Title of Thesis InWorks Notification Bar			
Date	20 December 2018	Pages/Appendices	33
Supervisor(s) Mr. Jussi Koistinen, Senior Lecturer and Mr. Keijo Kuosmanen, Senior Lecturer			
Client Organisation /Partners Mr. Tauno Hyvärinen, System Architect / Team Leader, Solteq Oyj			

Abstract

The purpose of this thesis was to create a notification system for an inWorks product. It was needed, because inWorks did not have a build-in communication tool for customers and admins. There were two parts to this project. The first part was the creation of a management tool to create and control notifications and the other was an add-on for the inWorks product that could receive notifications and display them for the user.

The management tool was created as an ASP.NET MVC project and the inWorks notification bar was added to the existing product. The management tool had to be created first with the database structure before the notification bar add-on to inWorks could be implemented. The main programming languages were C#, TypeScript and Angular. Some other technologies that were used were SQLServer, SignalR, EntityFramework and Razor.

As a result of this thesis, a prototype of the inWorks notification bar was completed. It can be used to send notifications to different virtual customer environments. Admins had rights to create, delete and control notifications and customers can receive and read them. Created notifications were saved into the SQLServer database and the back end logic of the server decides if the customer is legitimate to receive these notifications.

Keywords
inWorks

Sisältö

1 JOHDANTO

Tämän opinnäytetyön tarkoitus on toimia inWorksin asiakkaiden, tuen ja tuotekehityksen kommunikointi välineenä. Esimerkiksi jos asikasympäristöön ollaan tekemässä versiopäivitystä, voidaan siitä ilmoittaa kaikille ympäristön käyttäjille samanaikaisesti. Tämän ansiosta asiakkaat osaavat keskeyttää laskutusprosessit, ennen kuin päivityksen vaatima käyttökatkos katkaisee yhteyden.

Opinnäytetyö on kaksi osainen. Se sisältää ilmoitusten hallinta työkalun ja inWorks tuotteeseen upotetun notifikaatiopalkin ja sen toiminnot.

Kiitokset Solteq Oyj:lle, Tauno Hyväriselle ja Jussi Pöntyselle tästä työuran avaavasta mahdollisuudesta.

2 LYHENTEET, TEKNINEN SANASTO

Frontend
Frontend on asiakaspäädyn ohjelmointiin käytetty termi. Tämä tarkoittaa sitä, että koodin sisältö prosessoidaan käyttäjän selaimessa katseltavaan muotoon. Yleisimmät frontend ohjelmointi kielet ovat HTML ja CSS. Näillä kielillä hallitaan sivuston elementtejä ja niiden visuaalisia tyylejä. Lisäksi frontendiin kuuluu myös ohjelmointikieli JavaScript, joka toimii tapahtumankäsittelijänä käyttäjän selaimessa, kun jotain sivun elementtiä klikataan. Frontendiä hallitaan verkkosivun selaukseen käytetetyllä selaimella.

Frontend kielillä kirjoitetut sivustot sisältävät vähän muuttuvaa dataa ja ovat yleensä informaatiosivustoja, joita tarvitsee muuttaa harvoin, sillä frontend sivuston sisällön muuttaminen on työlästä. Muuttuvien sivujen rakentamiseen käytetäänkin backend logiikkaa ja tietokantoja.
(Frontend vs. backend)

Backend
Backend tarkoittaa palvelimenpuolen ohjelmointia. Backend määrittää miten sivusto toimii, päivittyy ja muuttuu. Tämä tarkoittaa palvelimen ja tietokantojen käyttöä.

Backendin tavoitteena on tehdä sivustoja, jotka sisältävät paljon muuttuvaa dataa. Backend ohjelmointikielillä kyetään keskustelemaan tietokantojen kanssa, joilla voidaan esimerkiksi määrittää mitä sisältöä näytetään käyttäjälle. Backend ohjelmoinnin suurin haaste onkin turvallisuus. Kenelläkään ylimääräisellä ei saa olla pääsyä muiden ihmisten dataan, sillä monet tietokantaohjelmistot sisältävät henkilökohtaisia tietoja.
(Frontend vs. backend)

HTML (Hyper Text Markup Language)
HTML on frontend ohjelmointikieli, joka on kaikkien web-sivustojen pohja. HTML koodi pohjautuu tageihin, joilla määritellään web-sivuston elementit, sekä niiden järjestys.
(What is HTML 2000-2018)

CSS (Cascading Style Sheets)
CSS on frontend ohjelmointikieli, jolla muokataan HTML sivujen ulkoasua. CSS:llä hallitaan sivuston ulkonäköä ja teemaa. Sillä voidaan hallita elementtien ja tekstin väriä, kokoa, välitystä, fonttia yms.
(What is CSS)

HTTP (HyperText Transfer Protocol)
HTTP on asiakkaan ja palvelimen väliseen tiedonsiirtoon käytetty yhteiskäytäntö. Palvelin kuuntelee asiakkaanpyyntöja tietyssä portissa, käsittelee sen ja palauttaa selaimelle haluttua dataa mikä esitetään käyttäjälle sovelluksen toiminnon vaatimalla tavalla. REST (Representational State Transfer) kutsut toimivat hyvin samaan tapaan palvelimen ja käyttöliittymän kommunikointi välineenä.
(What is HTTP)

3 WEB-OHJELMOINTI YLEISESTI

3.1 Web-toimintaympäristö

Webin toiminta malliin kuuluu yleisesti kolme osaa: (lainaus 1. s2)

- asiakasohjelma (client) lähettää palvelimelle pyyntöjä käyttäjän ohjaamana
- palvelinohjelma (server) vastaa asiakasohjelman pyyntöihin
- yhteyskäytäntö (protokolla), joka määrittelee, miten asiakasohjelma ja palvelinohjelma viestivät keskenään

Käytännössä asiakasohjelmana toimii web-selain (esim. Firefox, Chrome), joka lähettää palvelimelle pyyntöjä käyttäjän toimintojen perusteella. Käyttäjä voi esimerkiksi painaa selaimestaan linkkiä, jolloin palvelin lähettää vastauksena selaimelle web-sivun käyttäjän katseltavaksi. Pyynnöt lähetetään HTTP-siirtoyhteyskäytännöllä.

Palvelimen vastauksena tulevat web-dokumentit kuvataan HTML-kielellä ja voivat olla staattisia tai dynaamisia. Staattisia dokumentteja voi käyttää, jos käyttäjän ei tarvitse muuttaa sivuston sisältöä mitenkään. Staattista sivua voi muuttaa vain, jos sen tekijä tekee muutokset web-palvelimen koodiin. Tälläisiä sivuja voivat olla esimerkiksi lyhyet informaatiosivut, jotka tarvitsevat tiedonmuutoksia erittäin harvoin.

Muuttuvaa tietoa sisältäviä web-dokumentteja kutsutaan dynaamisiksi. Tällöin dokumenttiin voidaan hakea tietokannasta sisältöä sen perusteella mitä käyttäjä haluaa nähdä. Esimerkiksi haku on dynaaminen toiminto johon käyttäjä kirjoittaa hakusanan, jonka jälkeen palvelin voi hakea tietokannasta hakua vastaavia artikkeleita tai muuta sellaista. Dynaamisiin kutsuihin voidaan myös lisätä muutakin palvelinpuolen logiikkaa, kuten vaikka käyttäjänhallinta, jolloin käyttäjälle näytetään vain asioita, joihin tällä on lukuoikeus. On kuitenkin täysin sovelluskohtaista, minkälaisia toimintoja dynaamiselta sivulta vaaditaan.
(Docendo, Web-ohjelmointi 2005, Ari Rantala)

3.2 Käyttöliittymä

Web-sovellustenkäyttöliittymä rakentuu minimissään HTML-elementeistä. Käyttöliittymä tarkoittaa selaimessa näkyvää sivustoa, jonka käyttäjä näkee ja jota hänen olisi osattava käyttää. Siitä syystä sivustoilla käytetään erilaisia elävöittämistekniikoita (CSS, JavaScript). Elävöittämistekniikoilla muokataan sivun tyyliä ja toiminnallisuutta, jotta sivusto olisi helppolukuinen ja -käyttöinen. Asiakastekniikoja käyttämällä voidaan myös luoda tietynlaista dynaamisuutta web-sovellukseen, mutta se rajoittuu pääasiassa käyttöliittymän hallintaan.
(Docendo, Web-ohjelmointi 2005, Ari Rantala)

3.3 Sovelluslogiikka

Sovelluslogiikan suoritukseen sekä organisaatioiden tietojärjestelmien hyödyntämiseen tarvitaan palvelinpuolen tekniikkaa. Monet sovellukset hyödyntävät tietokantoja, joissa olevan datan perusteella käyttäjäkokemusta muokataan. Perinteisesti käytetään relaatiotietokantoja, joita käsitellään SQL-kielellä (Structured Query Language). Tyypillisesti web-sovellus integroidaan näihin tietokantoihin. Tavallisesti palvelinpuolen infrastruktuuri koostuu käyttöjärjestelmä-, web-palvelin ja tietokantapalvelinratkaisuista. Ohjelmoijan onkin hyvä olla tietoinen kaikista näistä ratkaisuista.
(Docendo, Web-ohjelmointi 2005, Ari Rantala)

3.4 Käyttöliittymän ja sovelluslogiikan välinen viestintä

Käyttöliittymä ja palvelin voivat viestiä HTTP-protokollan avulla. Eräs tapa viestiä on esimerkiksi REST kutsut. Käyttöliittymässä voi olla esimerkiksi nappi, johon liitetty JavaScript toiminto lähettää REST kutsun palvelimelle. Se mitä palvelin palauttaa asiakasympäristöön riippuu käytettävästä metodista (GET, POST, PUT, DELETE) ja sen parametreista. Kun palvelin määrittää mitä metodia käytetään, voidaan toteuttaa palvelinpuolen logiikka. Tämän jälkeen lähetetään käyttöliittymälle vastaus. Vastauksen tyyppi määritellään REST kutsun headerissa ja yleensä vastaus palautetaan JSON muodossa. Kun käyttöliittymä vastaanottaa JSON tiedoston, voidaan sen perusteella muokata käyttöliittymän näkymää palautetun datan perusteella, oli sitten kyse tuotteiden listauksesta tai mistä tahansa, mitä sovelluksella on tarkoitus näyttää käyttäjälle.
(Docendo, Web-ohjelmointi 2005, Ari Rantala)

4 TYÖSSÄ KÄYTETYT TEKNIIKAT

4.1 ASP.NET MVC

ASP.NET on Microsoft web-palvelimien käyttämä upotustekniikka. ASP.NET on ASP:n uudempi versio perustuen Microsoftin .NET-arkkitehtuuriin. Sivut rakentuvat tapahtuma- ja oliopohjaisuuden varaan. ASP.NET sovelluksia ohjelmoidaan yleensä C# ohjelmointikielellä ja näin ollen muistuttavatkin enemmän Windows sovellusten ohjelmointia, kuin esimerkiksi PHP-ohjelman tekemistä.

MVC lyhenne tulee sanoista model-view-controller. Käytännössä se tarkoittaa, että ASP.NET:ssä backend logiikkaa hallinnoidaan kontrollereilla. Nämä kontrollerit voivat palauttaa näkymän (view). Näkymässä voi olla esimerkiksi taulukoita, joihin pystyy linkittämään malleja (model). Kun taulukko täytetään ja tiedot lähetetään takaisin palvelimelle, sidotaan taulukon tiedon malliin, joka sitten tuo datan controllerille parametrinä. Tätä dataa voidaan muokata ja tallentaa tietokantaan myöhempää käyttöä varten.
(Learn about ASP.NET MVC)

4.2 EntityFramework (EF)

EF mahdollistaa työskentelyn relationaalisella datalla käyttäen toimialuekohtaisia olioita. EF eliminoi tarpeen suurimmalle osalle SQL koodia, jota ohjelmoijat joutuisivat muuten normaalisti kirjoittamaan. Näin ollen datan tietokantaliikenteen muodostaminen helpottuu.
(Entity Framework 6)

4.3 C#

C# on ECMA-organisaation standardoima olio-ohjelmointikieli, joka muistuttaa syntaksiltaan C++ ja Java-ohjelmointikieltä. C#:n tavoitteena on olla simppeli, moderni ja monikäyttöinen. Sen on tarkoitus tukea hyvin ohjelmistokehityksen tarpeita esimerkiksi tyypin ja taulun rajojen tarkastamisella, havaitsemalla alustamattomien muuttujien käytön ja automaattisella roskan keräyksellä. Roskan keräyksellä tarkoitetaan muistin hallintaa, jonka avulla ohjelmoijan ei tarvitse vapauttaa käyttämättömien olioiden muistipaikkoja. Tällä hetkellä tuorein versio on C# 7.3, joka julkistettiin 2018 Visualstudio 2017 version 15.7.2 kanssa.
(C#-perusteet 2016, C# Language Specification, Reference counting garbage collection)

4.4 Razor

Razor on ASP .NET ympäristössä käytettävä tekniikka, jolla voidaan upottaa backend logiikkaa cshtml tiedostoihin C# kielellä. Sillä saadaan tehtyä kahdensuuntainen datan sidonta näkymän, mallin ja kontrollerin välillä.
(Introducing Razor Syntax)

4.5 JavaScript/jQuery

JavaScript on frontend ohjelmointikielistä ensimmäinen, joka tuo logiikkaa ja dynaamisuutta web-sivustoon. Sillä hallitaan erilaisia tapahtumia ja voidaan muokata sivuston sisältöä HTML:n ja CSS:n osalta, animoida, hakea dataa palvelimelta sekä tehdä monia muitakin toiminnallisuuksia.

JavaScript kirjastoista yksi käytetyimmistä on jQuery. Se lyhentää ja yksinkertaistaa JavaScript koodin kirjoittamista.
(What is JavaScript 2005-2018)

4.6 TypeScript

TypeScript on skaalautuva JavaSript, joka mahdollistaa mm. olio-ohjelmoinnin JavaScriptin yhteydessä ja lisää vaihtoehtoisia tyyppejä, luokkia ja moduleita JavaScript ohjelmointiin. Sitä käytetään Microsoft Visual Studio:lla ja on yleensä käytetty .NET ohjelmoinnissa. TypeScript itsessään kirjoitetaan TypeScript kielellä, joka myöhemmin kääntyy JavaScriptiksi. Sillä voidaan kehittää toimintoja sekä palvelin-, että asiakaspuolelle.
(TypeScript language 2012-2018, TypeScript README)

4.7 Angular

Angular on TypeScript pohjainen vapaan lähdekoodin frontend ohjelmointialusta. Angularin perus rakennuspalikkana toimii ngModulit. Toiminnoilla on aina vähintään juurimoduli, joka mahdollistaa Angular ohjelman käynnistymisen. Näillä moduleilla voidaan myös sitoa dataa ja kahdensuuntainen datan sidonta on perus ominaisuus Angularille. Tämä tarkoittaa, että käyttöliittymällä tehdyt muutokset heijastuvat samantein ohjelma dataan.
(Angular fundamentals)

4.8 SQLServer

Microsoftin kehittämä relationaalinen tietokantatekniikka, jota ohjelmoidaan T-SQL (Transact-SQL) kielellä. T-SQL on kehitetty versio SQL ohjelmointikielestä. Dataa hallitaan tietokannoilla. Tietokannat sisätävät tietotauluja, joihin data tallennetaan. Taulut voivat olla itsenäisiä tai linkitettyjä muihin tauluihin.
(SQL Server Tutorial / Transact-SQL)

4.9 SignalR

ASP.NET:ssä SignalR on kirjasto, joka helpottaa reaaliaikaisen web-toiminnallisuuden lisäämistä ohjelmiin. Tällä tarkoitetaan sitä, että palvelimelta saadaan puskettua koodi sisältöä asiakkaan käyttöliittymään saman tien, kun se on saatavilla ilman että käyttäjän tarvitsee päivittää sivua.

Normaalissa tapauksessa selain ottaa yhteyden palvelimelle ja palvelin lähettää vastauksena tarvittavan datan. SignalR:n ansiosta palvelimen koodista voidaan lähettää dataa suoraan selaimen JavaScriptiin, joka päivittää asiakasnäkymää ilman erillistä palvelin kutsua. Tästä syystä SignalR mahdollistaa uudentyyppisiä web-ohjelmia, jotka vaativat paljon reaaliaikaisia päivityksiä palvelimelta. (Introduction to SignalR)

4.10 Git

Git on yksi yleisin moderni versionhallintaohjelma. Se on aktiivisesti ylläpidetty avoimen lähdekoodin projekti, joka on tehty 2005 Linus Torvaldsin toimesta.

Git on yleisesti käytettynä ohjelmoinnin versionhallintaan, sillä se on joustava. Se sopii niin pieniin kuin isoihin projekteihin. Haarautumisen ansiosta voidaan kehittää eri versioita samasta koodista, joista esimerkiksi master haara sisältää tuotannon koodin ja muissa haaroissa testataan ja kehitetään mahdollisia muutoksia. Myöhemmin nämä muutokset voidaan yhdistää master haaraan. Jos projektissa työskentelee useita henkilöitä, on mahdollista tulla konflikteja muutosten kanssa. Tämä aiheutuu siitä, että kaksi henkilöä on tehnyt muutoksia samaan kohtaan. Git sisältää työkalut näiden tapauksien vertailemiseksi ja korjaamiseksi, joilla kummastakin koodista voidaan valita oleelliset osat lopulliseen fuusioon.
(What is Git)

5 INWORKS

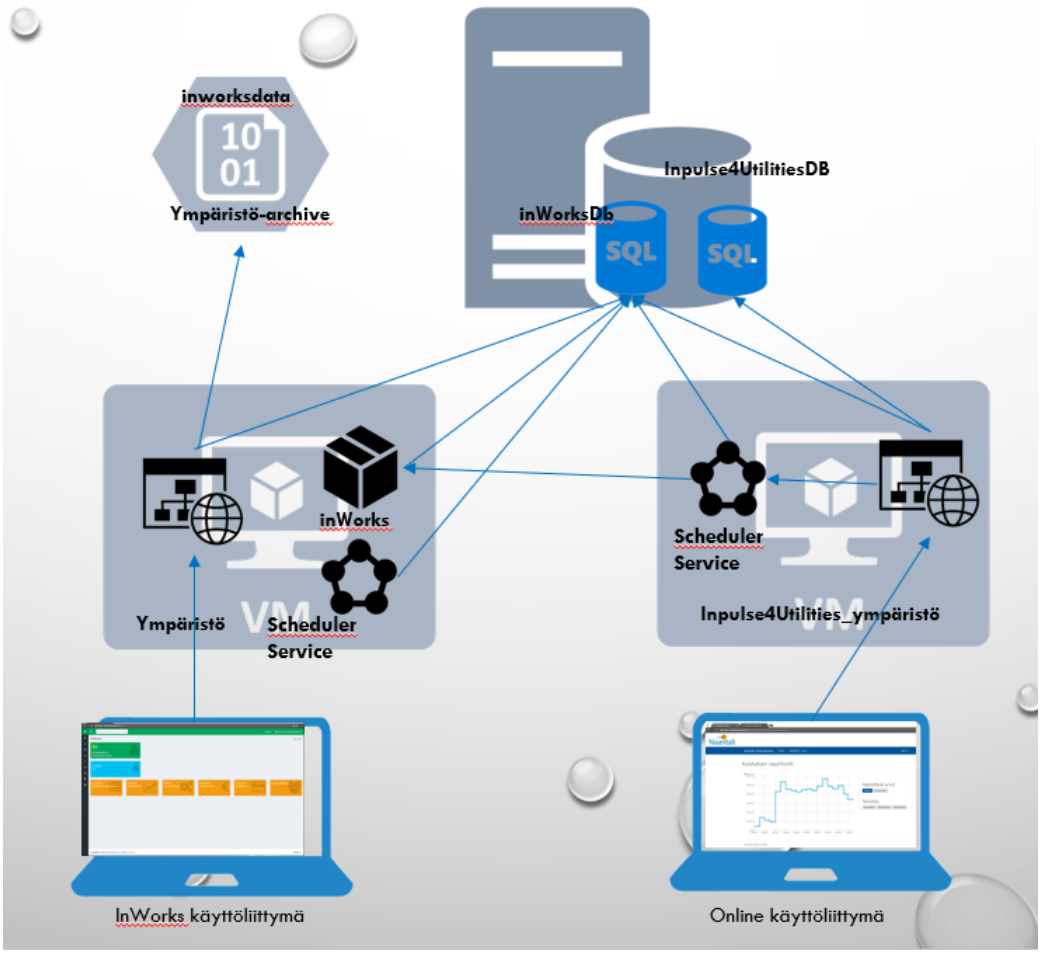
5.1 InWorks yleisesti

InWorks on Solteq Oyj:n kehittämä ja ylläpitämä tuote, jota käytetään erilaisten hyödykkeiden kuten esimerkiksi veden, sähkön ja lämmön kulutuksen laskuttamiseen. Näin ollen inWorksin asiakkaita ovat esimerkiksi sähköyhtiöt, jotka mittaavat asiakkaiden kulutusta, tuovat tiedot inWorksiin vaikkapa tuntilukemina ja muodostavat laskut asiakkaille heidän tilaamien tuotteiden ja kulutuksen perusteella. Laskuille tulleita maksuja voi myöhemmin seurata ja hallita järjestelmään rakennetusta reskontrasta.

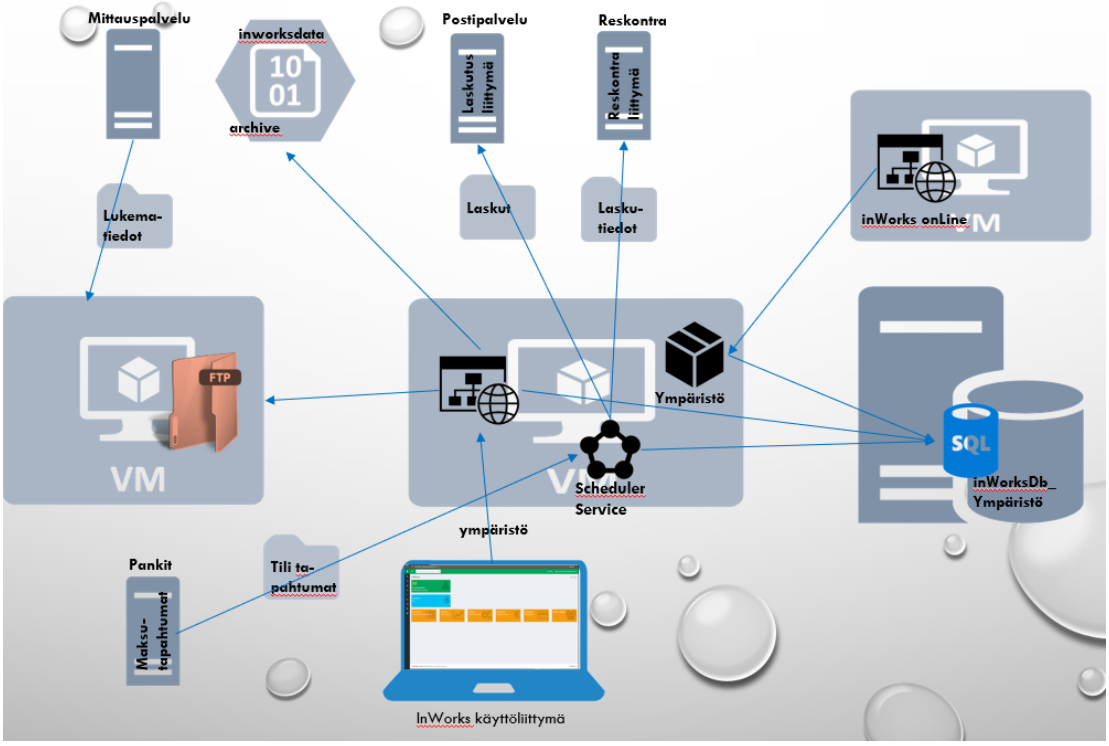
5.2 Arkkitehtuuri

InWorksin tuoteperheeseen kuuluu kaksi osaa. Itse inWorks ja inPulse4Utilities, joka on online järjestelmä loppuasiakkaalle. Esimerkiksi sähkösopimuksen tilaaja pystyy tarkastelemaan omaa kulutustaan näiltä sivustoilta. InWorks ja inPulse4Utilities ovat kuitenkin kaksi täysin eri tuotetta, jotka vain toimivat yhteistyössä toistensa kanssa.

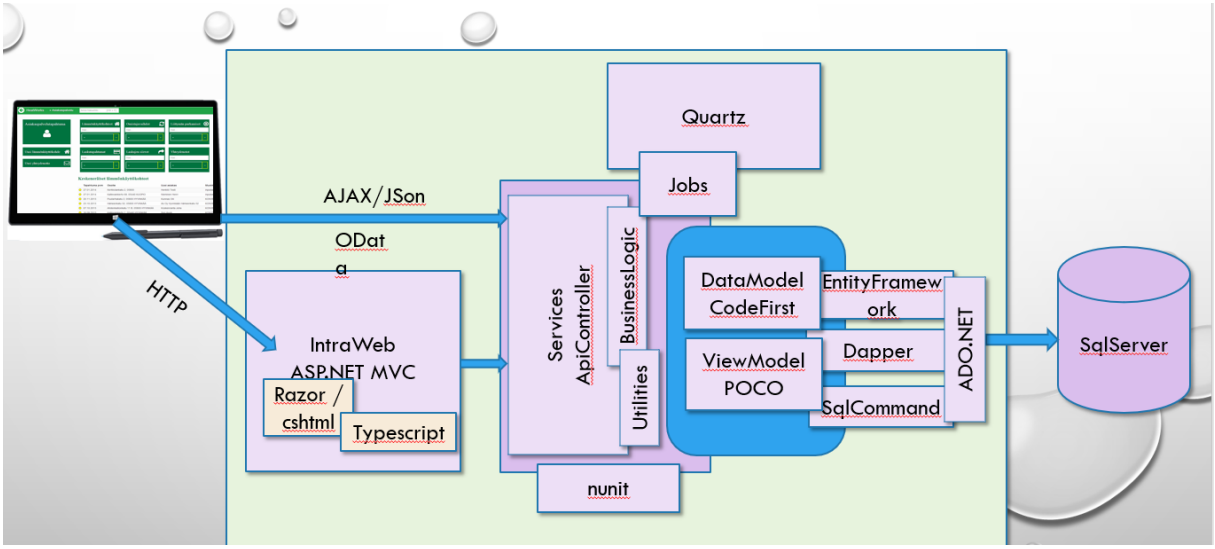
InWorksillä on kantapalvelin, johon dataa saadaan joko käyttäjän syöttämänä tai esimerkiksi lukema tiedostoina. Kun inWorks on vastaanottanut tiedoston, puretaan se tietokantaan ajastettujen tehtävien avulla virtuaalipalvelimella. Monet näistä tehtävistä on ajastettu tapahtumaan työpäivän ulkopuolella, jottei ne aiheuttaisi hitautta ja kantalukkoja palvelimella, kun asiakas itse käyttää järjestelmää. Kun inWorks on saanut siirrettyä lukemat omaan kantaansa, voi inPulse4Utilities kerätä tarvitsemansa tiedot tästä kannasta, muokata datan omaan käyttöönsä ja tallentaa sen omaan tietokantaansa. Näin tieto saadaan näkyvään muotoon online puolella loppukäyttäjälle.
Eli lyhykäisyydessään inWorks prosessoi datan, hallitsee asiakkuuksia ja muodostaa laskut. Näistä tiedoista inPulse4Utilities ottaa omaan käyttöönsä tarvittavan datan asiakkaalle näytettäväksi.



Kuva 1. InWorks arkkitehtuuri ja datan kulku inWorksin ja I4U:n välillä. Tietokantoja päivitetään käyttäjän toimintojen ja ajastettujen tehtävien perusteella.



Kuva 2. InWorks arkkitehtuuri ja kommunikointi ulkoisten tekijöiden, kuten pankin kanssa ja luke-matiedostojen vastaanotto. Kun maksutiedot saapuvat pankilta, voi niitä seurata reskontrasta.



Kuva 3. InWorks arkkitehtuuri. Dataliikenne käyttöliittymältä tietokantaan.

Käyttöliittymän toiminnoilla tehdään REST kutsu kontrollerille. Kun palvelin on vastaanottanut käyttöliittymältä saapuvan datan, voidaan sitä muokata tarvittavalla tavalla, tallentaa tiedot malliin ja lopulta tietokantaan, jossa se pysyy tallessa ja valmiina muuta käyttöä varten. REST kutsulla voidaan tarvittaessa myös palauttaa dataa käyttöliittymälle, jos siihen on tarvetta.

6 KÄYTTÖLIITTYMÄ & TOIMINNOT

Projektissa on kaksi osaa. Manager työkalu, jolla hallitaan ja luodaan ilmoituksia ja inWorks tuotteeseen upotettun notifikaatiopalkki, jonka avulla käyttäjät voivat seurata ilmoituksia.

6.1 Manager

Manager on työkalu, jolla ylläpitäjät pääsevät tekemään ilmoituksia inWorksiin. Ilmoitukset tallennetaan kantaan ja inWorksistä tulevalle api kutsuille palautetaan vastaukseksi asiakaskohtaiset ilmoitukset. Managerin yläpalkissa on myös linkki käyttöliittymän etusivulle.

6.1.1 Etusivu

Etusivulta käyttäjä pääsee valitsemaan kolmesta päätoiminnosta. Pikaviesti kaikille, valitse vastaanottajat sekä ilmoitukset. Näistä kaksi ensimmäistä ovat molemmat ilmoituksen luontia varten. Pikaviesti lähettää yleisen ilmoituksen kaikille ympäristöille, kun taas "valitse vastaanottajat" osiossa pääsee valitsemaan ympäristöt, joille viesti kohdistetaan. Ilmoitukset sivu on datan hallintaa varten.

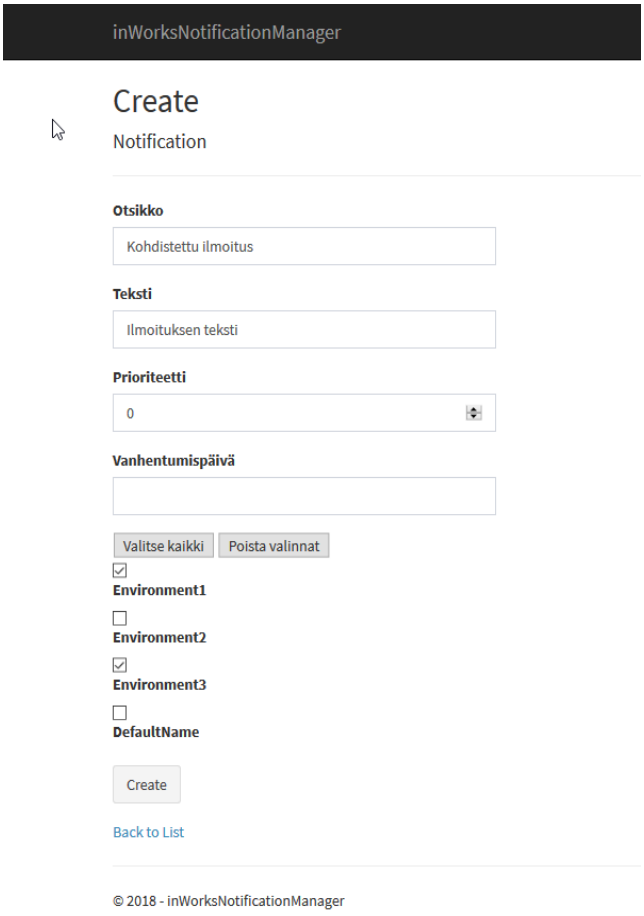
Kuva 4. Etusivun toimintovalikko.

6.1.2 Ilmoitusten luonti

"Valitse vastaanottajat" painikkeesta päästään sivulle, jolla on ilmoitus lomake. Lomakkeelle käyttäjä täyttää ilmoituksen otsikon, tekstin, prioriteetin ja valitsee vastaanottajat. Prioriteetti määrittelee, kuinka tärkeä ilmoitus on. Ykkös prioriteetilla merkityt ilmoitukset rullaavat aina näkyvillä inWorksin puolella olevassa notifikaatio palkissa ja sijaitsevat pysyvästi osiossa, minne kaikki uudet huomiota vaativat ilmoitukset tulevat.

Käyttäjä voi myös valita ilmoitukselle vanhentumispäivän. Kun kyseinen päivämäärä ylittyy, ei ilmoitus enää näy aktiivisena inWorksin käyttäjille. Vanhentuneet ilmoitukset ovat kuitenkin luettavissa historiasta. Vanhenemispäivän voi jättää tyhjäksi, jolloin ilmoitus pysyy aktiivisena niin kauan, kun ylläpitäjä käy jälkeenpäin merkkaamassa sen epäaktiiviseksi.

Ympäristövalikosta valitaan, mille kaikille ympäristöille ilmoitus kohdistetaan. Tässä yhteydessä on myös napit, joilla voi pikaisesti valita kaikki tai tyhjentää valitut. Tämä helpottaa ympäristöjen valintaa, jos esimerkiksi halutaan lähettää ilmoitus lähes joka ympäristölle. Tähän on vielä kehitysmahdollisuuksia esimerkiksi ympäristöjen valinta hyödykkeiden perusteella, jos halutaan vaikkapa lähettää viesti ympäristöille, joilla on vesihyödyke käytössä.



Kuva 5. Ilmoitusten luontisivu.

Etusivun "Pikaviesti kaikille" painike vie samalle sivulle ilman ympäristön valintaa, sillä tätä kautta viesti menee aina jokaiselle järjestelmässä olevalle ympäristölle.

Kun tiedot on täytetty, luontinappia painamalla ilmoituksen data siirtyy lomakkeelta kontrollerille razorin tiedonsidontatekniikalla, jolla lomakkeen data sidotaan datamalliin. Kontrollerista tiedot tallennetaan kantaan ja ne näkyvät inWorksissä valituilla ympäristöillä.

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references | TuomasKarhunen, 172 days ago | 1 author, 1 change | 0 requests | 0 exceptions
public async Task<IActionResult> Create([Bind(include:"Header,Text,Priority,ReleaseDate,ExpirationDate,DeactiveFlag")] Notification notification, string[] selectedEnvironments)//[
{
    notification.ReleaseDate = DateTime.Now;
    notification.DeactiveFlag = null;
    if (ModelState.IsValid)
    {

        if(selectedEnvironments.Count() != 0)
        {
            foreach (var item in selectedEnvironments)
            {
                var x = await _context.Tenants.SingleOrDefaultAsync(i => i.Code == item);
                _context.Add(new NotificationReceiver { EnvironmentCode = item, EnvironmentName = x.EnvironmentName, Notification = notification });
            }
        }
        else
        {
            _context.Add(new NotificationReceiver { EnvironmentCode = "", Notification = notification });
        }

        notification.ObjectInfo = new ObjectInfo { Created = DateTime.Now, Creator = "CreatorName", Modified = DateTime.Now, Modifier = "ModifierName(Creator)", Deleted = false };
        _context.Add(notification);

        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(notification);
}
```

Kuva 6. Koodiesimerkki kun käyttäjä painaa ilmoituksen luontinappia.

Data tuodaan käyttöliittymän lomakkeelta sidottuna "Notification" mallin kenttiin. Post metodin parametreinä ovat siis ilmoitus objekti, jonka malli sisältää kaiken sidonnassa mainitun datan ja listan ympäristöistä, joille tämä ilmoitus olisi tarkoitus näyttää.

Aluksi ilmoitukselle lisätään nykyinen aika ilmoituksen luontiajaksi ja määritellään se aktiiviseksi. Ilmoitus on epäaktiivinen jos "DeactiveFlag" kentällä on päivämäärä arvo. Null tarkoittaa, ettei sille ole määritelty arvoa. Tämän jälkeen tarkastetaan, että sidonnassa tehty malli on validi ja lähdetään käsittelemään valittuja ympäristöjä, jos niitä on valittu. Jos kohde ympäristöjä on valittuna, haetaan x nimiseen muuttujaan kohdeympäristön tiedot koodikentän perusteella ja lisätään kantaan rivi ilmoituksen vastaanottajasta. Jos taas ei ollut valittuna ilmoituksen vastaanottajia, jätetään objektilta "EnvironmentCode" muuttuja tyhjäksi, jolloin viestin pystyy vastaanottamaan mikä tahansa ympäristö.

Kun tieto on lisätty ketkä saavat vastaanottaa ilmoituksen, lisätään itse ilmoitusobjektille tiedot sen luonnista, sekä modifioinnista. Tämän jälkeen ilmoitus lisätään kontekstiin ja tallennetaan tietokantaan, jolloin se on luettavissa ilmoituksen vastaanottajilla. Kun tallennus on tehty, palautetaan käyttäjä etusivulle. Lopussa on vielä toinen palautus ilmoituksen tekosivulle, jos mallin datasidonta ei ole

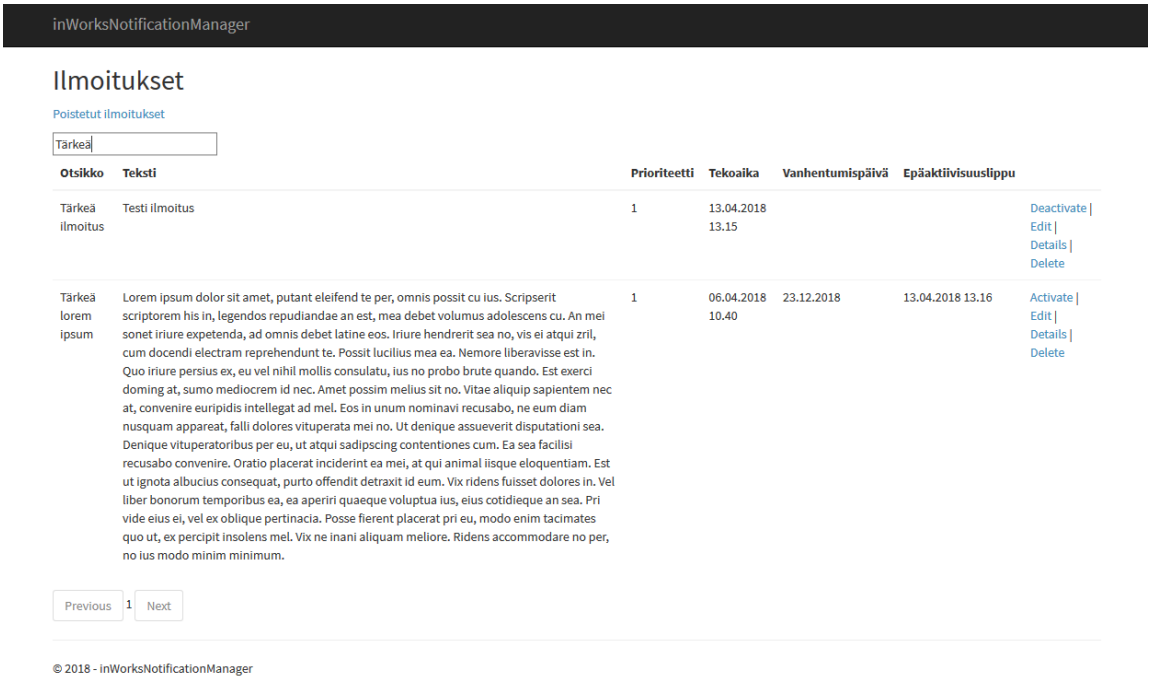
onnistunut ja koska siinä palautetaan parametrinä myös ilmoituksen tiedot, ei sivu pyyhkiydy tyhjäksi, vaan käyttäjän aiemmin syöttämät arvot ovat edelleen sivulla muokattavina.

6.1.3 Ilmoitusten hallinta

Etusivun ilmoitukset painikkeella päästään hallinta sivulle, missä kaikki tehdyt ilmoitukset ovat listattuna. Tämän sivuston sisältä löytyy monia toimintoja, joilla inWorksin ylläpitäjät voivat hallita ilmoituksia.

6.1.4 Haku

Sivuston yläreunasta löytyy haku palkki, jonka avulla voidaan filtteröidä listan ilmoituksia otsikon perusteella. Tämän alta löytyy lista kaikista hakuehdot täyttävistä ilmoituksista. On kuitenkin huomioitava, että poistetuksi merkityt ilmoitukset ovat omassa listassaan. Haku toimii JavaScriptillä. Kun käyttäjä kirjoittaa hakukenttään, pienen viiveen jälkeen tehdään api kutsu palvelimelle, josta palautuu lista hakuehdot täyttävistä ilmoituksista. Haun viive johtuu siitä, että siinä käytetään "on input" metodia. Jotta kuitenkin vältettäisiin kutsujen lähettäminen palvelimelle jokaisen kirjaimen jälkeen, odotetaan että käyttäjä lopettaa kirjoittamisen ja lähetetään kysely sitten.



Kuva 7.1. Ilmoitussivu ja haku toiminto, joka etsii tekstin sisältävät ilmoitukset.

```
$("#searchNotification").on("keyup", function () {

    var input = <HTMLInputElement>this;
    var data = {
        "Search": input.value,
        "Type": "normal",
        "page": 1
    };

    delay(function () {
        $.post("/Notification/SearchView", data, function (result) {
            $(".notificationList").html(result);
        });
    }, 500);
});
```

Kuva 7.2. JavaScript toiminnot, kun käyttäjä kirjoittaa haku palkkiin. Kun käyttäjä on ollut kirjoittamatta puoli sekunttia, lähetetään post kutsu palvelimelle.

```
var delay = (function () {
    var timer = 0;
    return function (callback, ms) {
        clearTimeout(timer);
        timer = setTimeout(callback, ms);
    };
})();
```

Kuva 4.3. Viive ajastimen hallinta.

```
[HttpPost]
0 references | TuomasKarhunen, 172 days ago | 1 author, 1 change | 0 requests | 0 exceptions
public async Task<IActionResult> SearchView(string Search, string Type, int? Page)
{
    var query = Enumerable.Empty<Notification>().AsQueryable();
    Notification notification = new Notification();
    query = GetSearchQuery(Type, Search);

    int pageNumber = (Page ?? 1);
    notification.Plist = await PaginatedList<Notification>.CreateAsync(query, pageNumber, pageSize);

    return View(notification);
}
```

Kuva 7.4. Kontrolleri vastaanottaa hakusanan ja tiedon onko ilmoitus poistettu parametreinä ja palauttaa lopuksi sivutetun listan ilmoituksista.

```
1 reference | TuomasKarhunen, 172 days ago | 1 author, 1 change | 0 requests | 0 exceptions
public IQueryable<Notification> GetSearchQuery(string type, string search)
{
    var query = Enumerable.Empty<Notification>().AsQueryable();

    if (search != "" && search != null)
    {
        switch (type)
        {
            case "deleted":
                query = _context.Notifications.Where(i => i.Header.Contains(search) &&
                i.ObjectInfo.Deleted == true).OrderByDescending(x => x.NotificationId).AsNoTracking();
                ViewBag.listType = "deleted";
                break;
            default:
                query = _context.Notifications.Where(i => i.Header.Contains(search) &&
                i.ObjectInfo.Deleted == false).OrderByDescending(x => x.NotificationId).AsNoTracking();
                break;
        }
        return query;
    }

    switch (type)
    {
        case "deleted":
            query = _context.Notifications.Where(i => i.ObjectInfo.Deleted == true).OrderByDescending(x => x.NotificationId).AsNoTracking();
            ViewBag.listType = "deleted";
            break;
        default:
            query = _context.Notifications.Where(i => i.ObjectInfo.Deleted == false).OrderByDescending(x => x.NotificationId).AsNoTracking();
            break;
    }
    return query;
}
```

Kuva 7.5. Metodi joka muodostaa kantakutsun sen perusteella, onko ilmoitus poistettujen listassa vai ei. Jos hakuehdosta on tullut tyhjä merkkijono, palautetaan ilmoitukset normaalisti.

6.1.5 Sivutus

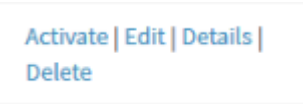
Listan yhdelle sivulle mahtuu konfiguraatioissa määritelty määrä ilmoituksia. Sen alapuolelta löytyy sivutus painikkeet. Edellinen painikkeella mennään yksi sivu taaksepäin ja vastaavasti seuraava painike avaa seuraavan sivun. Painikkeiden välissä on linkki ensimmäiselle ja viimeiselle sivulle, sekä tieto millä sivulla ollaan tällä hetkellä menossa. Sivutus toimii niin että kontrollerille tulee tieto parametreina hakuehdoista ja millä sivulla ollaan menossa. Tämän tiedon avulla kantahaun tuloksista rajataan sivulle tuleva data.

```
$(".notificationList").on("click", ".searchPageBtn", function (e) {
    e.preventDefault();
    var search;
    var input = <HTMLInputElement>this;
    var type = input.id.substr(0, input.id.indexOf("_"));
    var page = parseInt($(".currentPage").html());

    if ($(input).hasClass("plus"))
        page++;
    else if ($(input).hasClass("minus"))
        page--;

    if (type == "deleted")
        search = (<HTMLInputElement>document.getElementById("searchDeletedNotification"));
    else
        search = (<HTMLInputElement>document.getElementById("searchNotification"));


    var data = {
        "Search": search.value,
        "Type": type,
        "Page": page
    };

    $.post("/Notification/SearchView", data, function (result) {
        $(".notificationList").html(result);
    });

});
```

Kuva 7.6. Esimerkki kuinka hakusanalla haetun datan sivutusta hallitaan. Post kutsu viittaa kuvan 7.4 metodiin.

6.1.6 Hallintatyökalut

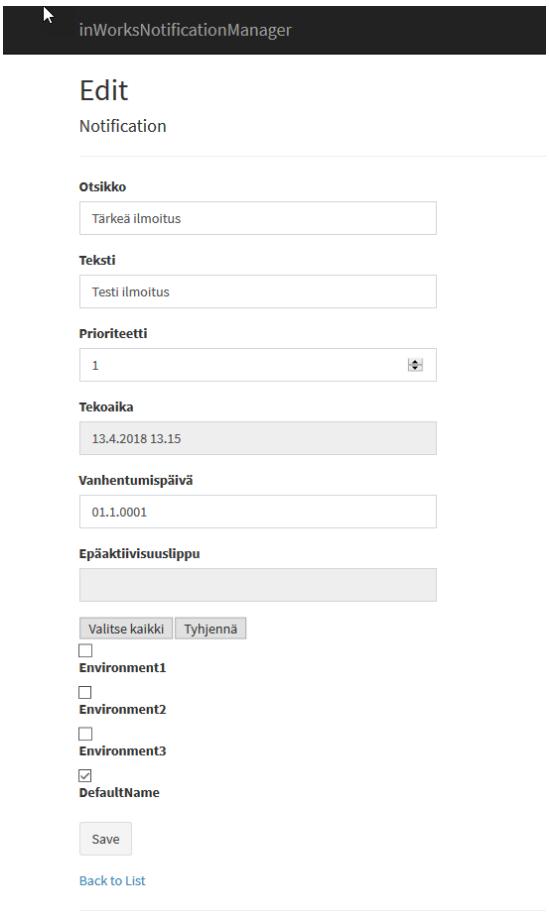
Listan oikeasta laidasta löytyy itse hallintatyökalut. Aktivointi painikkeesta muutetaan ilmoituksen aktiivisuus statusta. Aktiivinen ilmoitus näkyy loppukäyttäjälle normaalisti. Epäaktiiviseksi merkitty ilmoitus ei näy inWorksin ilmoituspalkissa eikä lukemattomissa, vaan ainoastaan ilmoitushistoriassa. Tätä toimintoa käytetään pääasiassa, kun tärkeällä prioriteetilla merkattu ilmoitus ei ole enää ajankohtainen ja se pitää poistaa inWorksin notifikaatiopalkista. Epäaktiiviseksi merkityt ilmoitukset ovat jälkeenpäin vielä luettavissa, jotta asiakas voi tarkistaa, että esimerkiksi versiopäivityksen aiheuttama käyttökatko on tapahtunut.



Kuva 8. Ilmoituksen hallinta työkalut.

6.1.7 Muokkaus

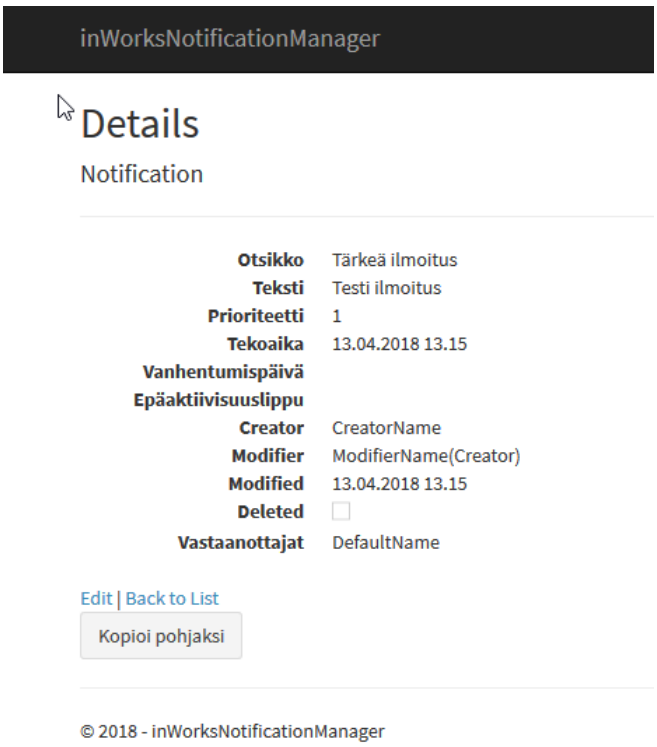
Editointi painikkeesta päästään ilmoituksen luontia vastaavalle sivulle, jolla kentät on täytetty kyseisen ilmoituksen datalla. Täällä ilmoitusta voidaan muokata halutulla tavalla ja vaikka vaihtaa vastaanottavia ympäristöjä, jos luontivaiheessa on tapahtunut virhe.



Kuva 9. Ilmoituksen muokkaussivu.

6.1.8 Ilmoituksen tiedot

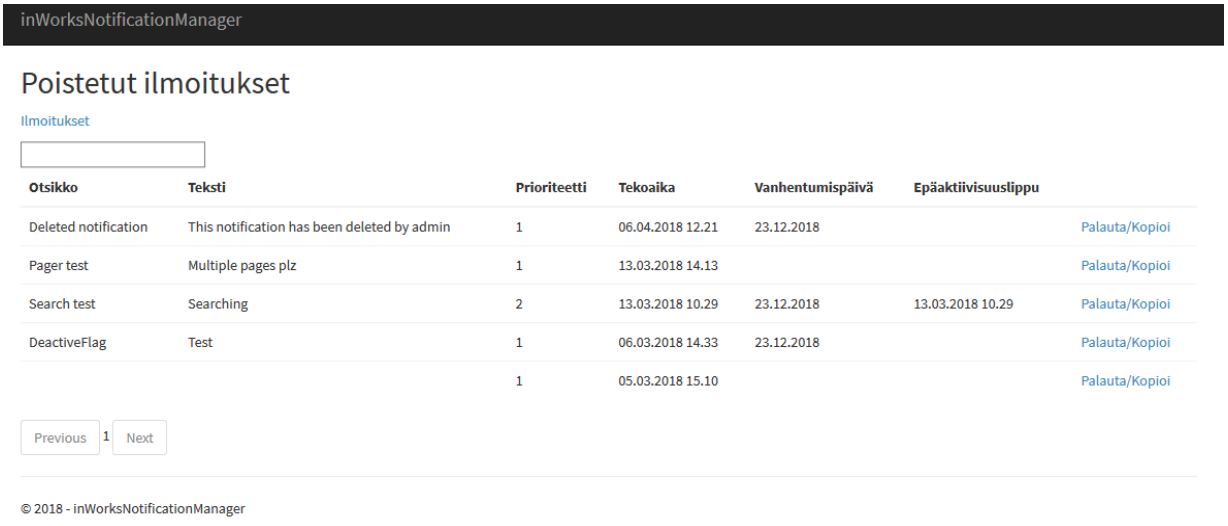
Tiedot sivulla pääsee tarkastelemaan ilmoituksen tietoja. Perustietojen lisäksi näkee, kuka on luonut ja muokannut ilmoitusta. Sivulta voi joko palata takaisin ilmoituslistaan, muokata ilmoitusta tai kopioida ilmoituspohjan. Ilmoituspohjan kopiointi lähtee luomaan uutta ilmoitusta kopioidun arvoilla, joita sitten voi muokata haluamallaan tavalla. Editointi sivu vastaavasti muokkaa aktiivista ilmoitusta.



Kuva 10. Tietosivu, jolta näkee kaiken informaation ilmoituksista.

6.1.9 Ilmoitusten poisto

Poista napilla ilmoitus merkitään poistetuksi. Sitä ei todellisuudessa poisteta oikeasti, vaan muokataan tietokannasta "Deleted" -kentän totuusarvoa. Kun ilmoitus on merkitty poistetuksi, ei loppuasiakas enää näe sitä omissa ilmoituksissaan, mutta ylläpidon puolella se listautuu poistettujen ilmoitusten listaan. Tämä lista vastaa muuten ilmoituslistan sisältöä, mutta tiedot sivulta ilmoituksen voi joko palauttaa tai kopioida pohjaksi. Palautuksen jälkeen loppuasiakas kykenee jälleen lukemaan ilmoituksen, kun se palautuu ilmoituslistaan.



Kuva 11. Poistetut ilmoitukset -sivu. Sivun ylälaidassa otsikon alla on ilmoitukset linkki, jolla päästään takaisin normaaliin ilmoituslistaan.

```
// POST: Notification/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
0 references | TuomasKarhunen, 172 days ago | 1 author, 1 change | 0 requests | 0 exceptions
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var notification = await _context.Notifications.SingleOrDefaultAsync(m => m.NotificationId == id);
    //_context.Notifications.Remove(notification); // poisto kokonaan kannasta, vaatii huomioida myös poistot foreign tauluista
    //(managerista ei ole tarkoitus poistaa pysyvästi)
    ObjectInfo info = _context.Notifications.Where(i => i.NotificationId == notification.NotificationId).Select(x => x.ObjectInfo).First();
    info.Deleted = true;
    notification.ObjectInfo = info;
    _context.Update(notification);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}
```

Kuva 12. Koodiesimerkki ilmoitusten poistosta.

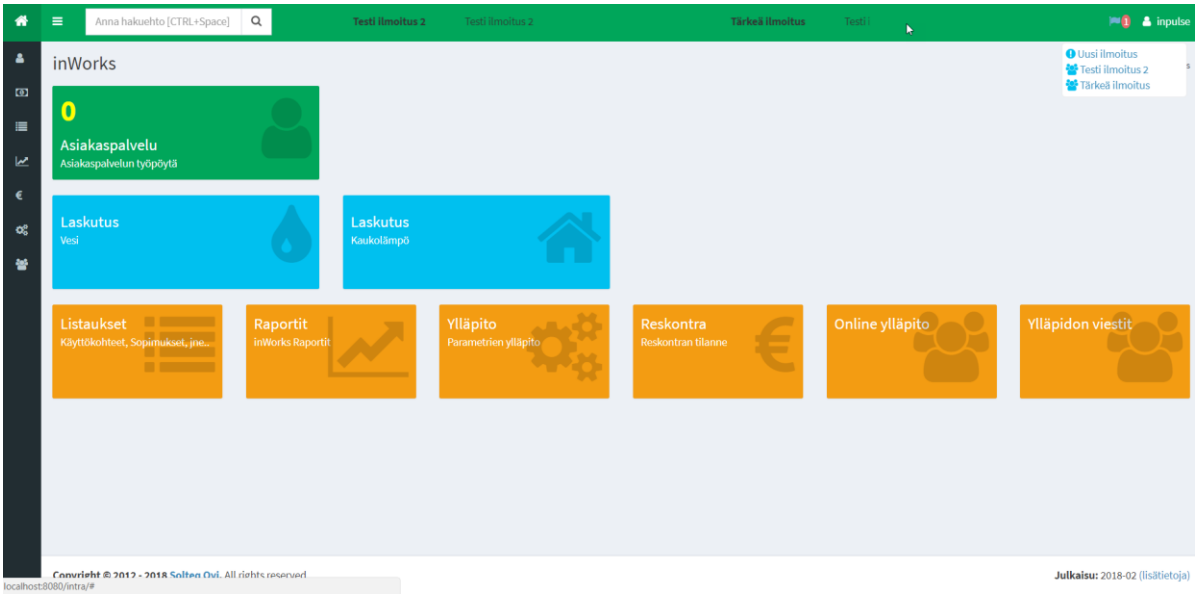
Tietokantaan päivitetään ilmoitusrivin poiston totuusarvoksi tosi, joka merkkaa ilmoituksen poistetuksi. Tämä on turvallinen tapa poistaa dataa. Jos poisto on tehty virheellisesti, voidaan data palauttaa ilman totuusarvon muuttamista kummempia toimenpiteitä.

6.1.10 Luetut ilmoitukset

Luetut ilmoitukset sivulta pystytään tarkastamaan ketkä ovat ladanneet ja lukeneet minkäkin ilmoituksen. Jos tulee ongelmia, että asiakas ei ole kiinnittänyt huomiota johonkin ilmoitukseen, voidaan tarkistaa, onko hänellä ollut mahdollisuutta nähdä sitä. Näin ollen mahdollisissa virhetaupauksissa voidaan todeta, onko esimerkiksi inWorksin huoltokatkosta ilmoitettu asiallisesti.

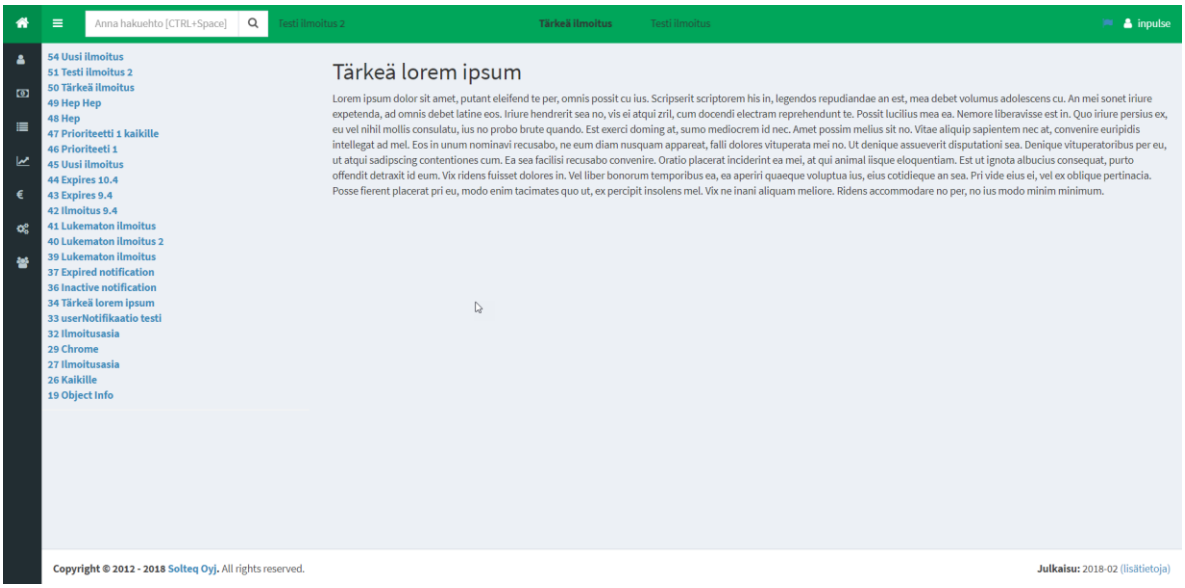
6.2 InWorks notifikaatiot

InWorksin käyttöliittymällä tärkeät ilmoitukset rullaavat yläpalkissa aina näkyvillä. Rullausefekti on tehty jQueryllä. Tekstin nopeutta pystyy muuttamaan koodimuutoksella. Rullaava teksti myös pysähtyy, kun käyttäjä vie hiiren sen päälle, jotta se olisi helposti luettavissa. Lukemattomien ilmoitusten määrä näkyy ilmoitusvalikon päällä, josta voi avata haluamansa ilmoituksen ja lukea sen sisällön.



Kuva 13. InWorksin käyttöliittymän etusivu. Ylälaidan palkissa näkyy teksti pätkiä, jotka rullaavat luettavalla nopeudella. Oikeasta laidasta löytyy lista lukemattomista ja tärkeistä ilmoituksista.

Ilmoituksen avattuaan käyttäjä pääsee ilmoitussivulle, jonka sivupalkista löytyy ilmoitushistoria. Ilmoitushistoriassa on kaikki käyttäjälle kohdistetut poistamattomat ilmoitukset.



Kuva 14. Ilmoitusten tarkastelusivu inWorksin puolella. Käyttäjä voi lukea ilmoituksia ja valita luettavan ilmoituksen sivun laidasta löytyvästä ilmoitushistoriasta.

```
[HttpPost]
[Route("GetActiveUserNotifications")]
0 references | TuomasKarhunen, 144 days ago | 1 author, 1 change | 0 requests | 0 exceptions
public async Task<List<ResNotification>> GetActiveUserNotifications(string environment, string user)
{
    List<Notification> list = new List<Notification>();
    List<ResNotification> resList = new List<ResNotification>();
    //var notifications = _context.Notifications;
    var apiKey = Request.Headers["ApiKey"];
    bool correctApiKey = _context.Tenants.Any(t => t.Apikey == apiKey && t.Code == environment);

    if(!correctApiKey)
    {
        return resList;
    }

    list = await _context.Notifications.Where(n => n.NotificationReceivers.Any(u => u.EnvironmentCode == environment || u.EnvironmentCode == "")
    && n.DeactiveFlag == null && (n.ExpirationDate == null || n.ExpirationDate > DateTime.Now) && n.ObjectInfo.Deleted == false
    && (n.UserNotifications.Any(v => v.MessageRead == null && v.UserID == user)
    || n.Priority == 1)).OrderByDescending(x => x.NotificationId).AsNoTracking().ToListAsync();
```

```
foreach (var item in list)
{
    ResNotification data = new ResNotification();
    data.NotificationId = item.NotificationId;
    data.Header = item.Header;
    data.Text = item.Text;
    data.Priority = item.Priority;
    data.NewNotification = false;

    if (!_context.UserNotificationData.Any(x => x.NotificationID == item.NotificationId && x.UserID == user && x.MessageRead != null))
        data.NewNotification = true;


    //data.ReleaseDate = item.ReleaseDate;


    if (!_context.UserNotificationData.Any(x => x.NotificationID == item.NotificationId && x.UserID == user))
    {//päivitetään userNotification taulua, jos käyttäjä ei ole aiemmin ladannut ilmoitusta
        UserNotification userNotification = new UserNotification();
        userNotification.MessageLoaded = DateTime.Now;
        userNotification.NotificationID = item.NotificationId;
        userNotification.EnvironmentCode = environment;
        userNotification.UserID = user;

        _context.UserNotificationData.Add(userNotification);
    }
    resList.Add(data);
}
await _context.SaveChangesAsync();
return resList;
```

Kuvat 15.1 ja 15.2. Käyttäjän aktiivisten ilmoitusten haku, sekä tiedon päivitys tietokantaan, että käyttäjä on ladannut ilmoituksen.

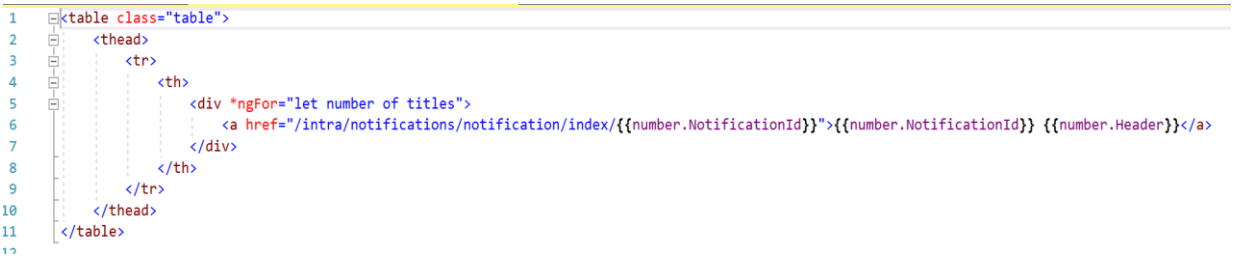
Koodiesimerkki angularkoodista, joka hakee dataa ilmoituksista ja näyttää sen käyttöliittymällä:

```
import { Injectable } from "@angular/core";
import { InpulseServiceBase } from "../../../common/communication"
import { ContractProvider, ContractListType, ReadingsProvider } from "../../../common/interfaces";
import { Http, Response, Headers, RequestOptions } from '@angular/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/catch';
import { Subject } from 'rxjs/Subject';
@Injectable()
export class NotificationService extends InpulseServiceBase {

    //constructor(private http: Http) { }

    constructor(protected http: Http) {
        super(http);
    }

    getData(): Observable<any> {
        var url = ApiUrl + "NotificationComponent/GetNotifications";
        return this.get(url);
    }
    getNotification(id: number): any {
        var url = ApiUrl + "NotificationComponent/GetSingleNotification";
        return this.getWithId(url, id);
    }

    getReadNotifications(): Observable<any>{
        var url = ApiUrl + "NotificationComponent/GetReadNotifications";
        return this.get(url);
    }

    getFilteredNotifications(searchText: string): Observable<any> {
        var url = ApiUrl + "NotificationComponent/GetFilteredNotifications";
        return this.getWithSearchText(url, searchText);
    }
}
```

Kuva 16.1. NotificationService.ts sisältää metodit.

```
import { Component, OnInit } from '@angular/core';
import { NotificationService } from './NotificationService';
 declare var rootUrl;

@Component({
    selector: 'notification',
    //template: `<div *ngFor="let number of titles">{{number.NotificationId}} {{number.Header}} {{number.Text}}</div>`
    //animations: []
    templateUrl: rootUrl + "scripts/components/views/notifications/index/notificationbar.component.html?v=" + Date.now()
    //styleUrls: ['./notificationbar.component.css']
})

export class NotificationComponent implements OnInit {
    title = 'Testi title'; //<div>{{title|async}}</div>
    titles = []; //<div *ngFor="let number of titles">{{number}}</div>

    constructor(private notificationService: NotificationService) {

    }

    ngOnInit() {
        this.getData();
    }

    getData(): void {
        this.notificationService.getData().subscribe(titles => { this.titles = titles; console.log(this.titles); }, err => { console.log(err); });
    }
}
```

Kuva 16.2. NotificationBar.component.ts rakentaa komponentin logiikan ja hakee datan serviceltä.



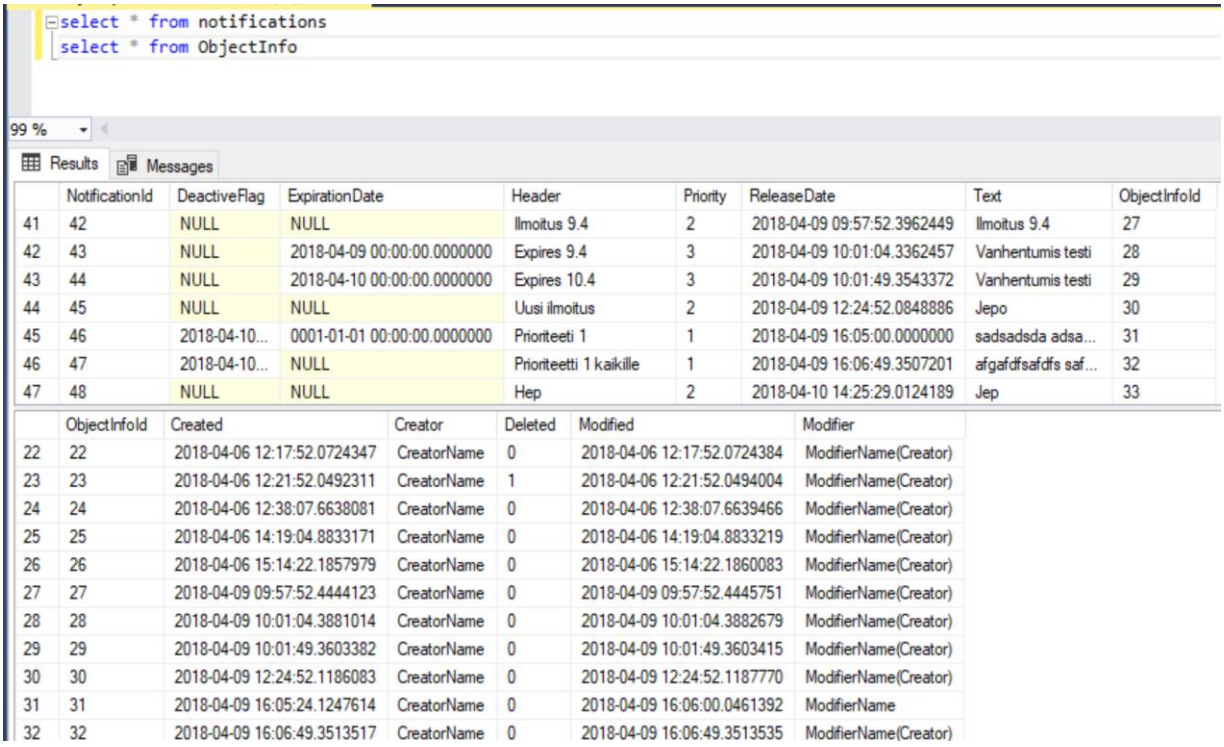
Kuva 16.3. NotificationBar.component.html sisältää HTML koodin, jolla rakennetaan komponentin sisältö.

Angularissa voidaan tehdä "omia" html elementtejä, joihin voidaan sisällyttää paljon enemmän sisältöä. Tätä elementtiä voidaan sitten käyttää tarvittaessa useaan kertaan, ilman että tarvitsee kirjoittaa samaa koodia useasti.

7 POHDINTAA

Työn teon teon aikana ja jälkeen olen paljon uutta inWorksistä. Olisikin monia asioita, joita voisi tehdä nyt paremmin. Opinnäytetyön teko hetkellä ei ollut vielä kovin hyvää visiota, miten inWorks toimii ja miten sitä käytetään.

Ensimmäinen asia mitä parantelisin opinnäytetyössä olisi tietokantarakenne ja mallit. Nykyään työskennellessä inWorksin parisssa, nämä ovat tulleet paremmin tutuiksi ja muuttaisinkin vastaamaan enemmän inWorksin mallia. Tämä olisi hyödyllistä ylläpidon kannalta, sillä tietokannan luku olisi nopeampaa. Alun perin monet asiat tuli turhaan laitettua omaan alitauluunsa vierasavain viitteellä, kuten tiedot ilmoitusten luojista ja luontiajoista.



```
select * from notifications
select * from ObjectInfo
```

99 %

Results | Messages

	NotificationId	DeactiveFlag	ExpirationDate	Header	Priority	ReleaseDate	Text	ObjectInfoId
41	42	NULL	NULL	Ilmoitus 9.4	2	2018-04-09 09:57:52.3962449	Ilmoitus 9.4	27
42	43	NULL	2018-04-09 00:00:00.0000000	Expires 9.4	3	2018-04-09 10:01:04.3362457	Vanhentumis testi	28
43	44	NULL	2018-04-10 00:00:00.0000000	Expires 10.4	3	2018-04-09 10:01:49.3543372	Vanhentumis testi	29
44	45	NULL	NULL	Uusi ilmoitus	2	2018-04-09 12:24:52.0848886	Jepo	30
45	46	2018-04-10...	0001-01-01 00:00:00.0000000	Prioriteeti 1	1	2018-04-09 16:05:00.0000000	sadsadsda adsa...	31
46	47	2018-04-10...	NULL	Prioriteetti 1 kaikille	1	2018-04-09 16:06:49.3507201	afgafdfsafdfs saf...	32
47	48	NULL	NULL	Hep	2	2018-04-10 14:25:29.0124189	Jep	33

	ObjectInfoId	Created	Creator	Deleted	Modified	Modifier
22	22	2018-04-06 12:17:52.0724347	CreatorName	0	2018-04-06 12:17:52.0724384	ModifierName(Creator)
23	23	2018-04-06 12:21:52.0492311	CreatorName	1	2018-04-06 12:21:52.0494004	ModifierName(Creator)
24	24	2018-04-06 12:38:07.6638081	CreatorName	0	2018-04-06 12:38:07.6639466	ModifierName(Creator)
25	25	2018-04-06 14:19:04.8833171	CreatorName	0	2018-04-06 14:19:04.8833219	ModifierName(Creator)
26	26	2018-04-06 15:14:22.1857979	CreatorName	0	2018-04-06 15:14:22.1860083	ModifierName(Creator)
27	27	2018-04-09 09:57:52.4444123	CreatorName	0	2018-04-09 09:57:52.4445751	ModifierName(Creator)
28	28	2018-04-09 10:01:04.3881014	CreatorName	0	2018-04-09 10:01:04.3882679	ModifierName(Creator)
29	29	2018-04-09 10:01:49.3603382	CreatorName	0	2018-04-09 10:01:49.3603415	ModifierName(Creator)
30	30	2018-04-09 12:24:52.1186083	CreatorName	0	2018-04-09 12:24:52.1187770	ModifierName(Creator)
31	31	2018-04-09 16:05:24.1247614	CreatorName	0	2018-04-09 16:06:00.0461392	ModifierName
32	32	2018-04-09 16:06:49.3513517	CreatorName	0	2018-04-09 16:06:49.3513535	ModifierName(Creator)

Kuva 17. Esimerkki kuinka ilmoituksen luontitiedot ovat omassa taulussaan. Tilanteesta riippuen tämä on hyvä tapa jaotella dataa eri taulujen välille, mutta tässä tapauksessa olisi järkevintä, että "ObjectInfo" taulun sisältö olisi samassa taulussa ilmoitusten kanssa.

Toinen asia mikä tuli tehtyä vähän kummallisesti alun perin, koska ei ollut vielä tarpeeksi tietämystä inWorksistä ja miten tätä ilmoitusten hallinttatyökalua halutaan käyttää, oli listattu data ilmoitusten tarkasteluista. Tämä tuli tehtyä inWorksin ylläpidon puolelle, joka on asiakkaiden käytössä, jos heillä on siihen tarvittavat oikeudet. Jälkeen päin ajateltuna tämän toiminnon tekisi enemmän järkeä olla managerin puolella, ellei tätä manageriakin myöhemmin päätettäisi ottaa käyttöön myös inWorksin asiakkaiden käytettäväksi. Esimerkiksi jonkin käyttöoikeuden taakse. Tämä toiminto voisi olla hyödyllinen joillekkin asiakkaille, sillä se vähentäisi tarvetta käyttää ulkoisia viestintävälineitä, kuten sähköpostia. Tältä kantilta ajatellen ilmoituksia voisi eri ympäristöjen sijaan lähettää eri henkilöille, jotka

voitaisiin myös ryhmittää esimerkiksi laskuttajiin ja käyttöpaikkojen ylläpitäjiin. Näin ilmoitusten ylläpitäjä oikeudella varustettu henkilö, oli se sitten asiakasympäristön tai inWorksin henkilökuntaa, pystyisi kohdistamaan ilmoitusluontoisen asian sitä koskeville käyttäjäryhmille.

Yksi olennainen asia missä on kehitettävää varsinkin, jos manageria haluttaisiin lisätä inWorksin puolelle, on käyttöliittymä ja visuaalisuus. Opinnäytetyön tekovaiheessa käyttöliittymän ulkonäköön on panostettu melko minimaalisesti, jotta toiminnot saataisiin kuntoon nopeammin. Etusivulla olevien klikattavien laatikoiden tyylitkin on napattu suoraan inWorksistä. Design ei varsinaisesti olekkaan yleensä ohjelmoijan vahvimpia puolia, vaan ohjelmoijat yleensä toteuttavat designerin tekemää muotoilua ja osaavat itse suunnitella vähän simppelimmän näköisiä käyttöliittymiä. Tämä on tietysti henkilökohtaista, mitenkä hyvä taiteellinen näkemys henkilöllä on.

LÄHTEET JA TUOTETUT AINEISTOT

Docendo, Web-ohjelmointi 2005, Ari Rantala.

What is HTML 2000-2018. Saatavilla 10.11.2018:
http://www.yourhtmlsource.com/starthere/whatishtml.html

What is CSS. Saatavilla 10.11.2018:
https://www.tutorialspoint.com/css/what_is_css.htm

What is JavaScript 2005-2018. Saatavilla 10.11.2018:
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

What is JSON. Saatavilla 10.11.2018:
https://developers.squarespace.com/what-is-json/

Introduction to SignalR. Saatavilla 10.11.2018:
https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr

SQL Server Tutorial / Transact-SQL. Saatavilla 10.11.2018:
https://www.techonthenet.com/sql_server/index.php

C#-perusteet 2016. Saatavilla 10.11.2018:
http://appro.mit.jyu.fi/gko/luennot/luento1/

C# Language Specification. Saatavilla 10.11.2018:
http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf

Reference counting garbage collection. Saatavilla 10.11.2018:
https://blogs.msdn.microsoft.com/abhinaba/2009/01/27/back-to-basics-reference-counting-garbage-collection/

Entity Framework 6. Saatavilla 10.11.2018:
https://github.com/aspnet/EntityFramework6/blob/master/README.md

TypeScript language 2012-2018. Saatavilla 10.11.2018:
https://www.typescriptlang.org/

TypeScript README. Saatavilla 10.11.2018:
https://github.com/Microsoft/TypeScript

What is Git. Saatavilla 10.11.2018:
https://www.atlassian.com/git/tutorials/what-is-git

Angular fundamentals. Saatavilla 10.11.2018:

https://angular.io/guide/architecture

What is HTTP. Saatavilla 10.11.2018

https://www.webopedia.com/TERM/H/HTTP.html

Frontend vs. backend. Saatavilla 10.11.2018

https://blog.teamtreehouse.com/i-dont-speak-your-language-frontend-vs-backend

Learn about ASP.NET MVC. Saatavilla 10.11.2018

https://www.asp.net/mvc

Introducing Razor Syntax. Saatavilla 10.11.2018

https://docs.microsoft.com/en-us/aspnet/web-pages/overview/getting-started/introducing-razor-syntax-c