OULUN AMMATTIKORKEAKOULU

Petri Virkkunen

C++-KIRJASTOJEN KÄYTTÖ JA PILVIPOHJAISEN LISENSSIJÄRJESTELMÄN TOTEUTUS

C++-KIRJASTOJEN KÄYTTÖ JA PILVIPOHJAISEN LISENSSIJÄRJESTELMÄN TOTEUTUS

Petri Virkkunen
Opinnäytetyö
Syksy 2017
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, Ohjelmistotuotanto

Tekijä(t): Petri Virkkunen
Opinnäytetyön nimi: C++-kirjastojen käyttö ja Pilvipohjaisen lisenssijärjestelmän toteutus
Työn ohjaaja: Veijo Väisänen ja Pekka Alaluukas
Työn valmistumislukukausi ja -vuosi: Syksy 2017 Sivumäärä: 8 + 2 liitettä

Opinnäytetyö kirjoitettiin kahdessa osassa, joista ensimmäinen oli viiden opintopisteen arvoinen ja toinen kymmenen.

Ensimmäisessä opinnäytetyön osassa käytiin läpi C++-kirjastojen käyttöä Windows-pohjaisella kehitysalustalla käyttäen Visual Studio -editoria. Työn aikana opin käyttämään SDL-nimistä C++-kirjastoa, jota käytetään pääosin pelien luomiseen. Työssä käsiteltiin SDL-kirjastolla ruudun piirtäminen, siihen eri väristen laatikoiden ja tekstin piirtäminen sekä laatikoiden siirtäminen käyttämällä näppäimistöä.

Toisessa osassa dokumentoitiin pilvipohjaisen lisenssijärjestelmän toteutus Biim Ultrasound Oy:lle. Lisenssijärjestelmää käytetään ultraäänilaitteiden käytön rajoittamiseen. Sen avulla voidaan estää luvaton laitteen käyttö sekä maksattaa käyttäjiä tietyin väliajoin. Työssä dokumentoitiin pilvipalvelun ja mobiilisovelluksen vaatimukset, suunnittelu, toteutus sekä testaus.

Asiasanat: pilvipalvelut, C++, Visual Studio, lisenssit, peliohjelmointi, opinnäytetyökooste

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Software development

Author(s): Petri Virkkunen
Title of thesis: Usage of C++ libraries and development of a cloud based licensing system
Supervisor(s): Veijo Väisänen and Pekka Alaluukas
Term and year when the thesis was submitted: Autumn 2017 Number of pages: 8 + 2 attachments

---

This thesis was written in two parts, the first one documenting learning how to use C++ libraries in Visual Studio and the second one documenting the implementation of a cloud-based licensing system for Biim Ultrasound Oy.

The first part of the thesis is meant to document learning how to build with and use a C++ library called SDL, which is mainly used for videogame programming. This library was then used to draw a window, some rectangles and text on that window, and then to move those rectangles using keyboard events.

The purpose of the second part of the thesis was to document the requirements, planning and implementation of a cloud-based licensing system for Biim Ultrasound Oy, which would be used to limit usage of their ultrasound device. The system could also be used to bill customers with a specific interval or even track sales of the device.

Keywords: C++, cloud services, Visual Studio, licenses, game programming

SISÄLLYS

1 JOHDANTO

Tämä opinnäytetyö jaettiin kahteen osaan, eli se tehtiin koosteopinnäytetyönä. Opinnäytetyön ensimmäinen osa toteutettiin keväällä 2016 ja toinen osa syksyllä 2017. Ensimmäinen osa on laajuudeltaan viisi ja toinen osa kymmenen opintopistettä.

Ensimmäisessä osassa käsiteltiin erästä C++-ohjelmointikielen kirjastoa ja sen käyttöä Visual Studio -ympäristössä. Työssä keskityttiin kirjaston yksinkertaisimpiin ominaisuuksiin, kuten tekstin kirjoitukseen ikkunaan.

Toisessa osassa dokumentoitiin töissä tehty järjestelmä, johon käytettiin pilvipalvelua sekä mobiilisovellusta. Työssä dokumentoitiin pilvipalvelun ja mobiilisovelluksen vaatimukset, suunnitelma, toteutus ja testaus.

2 OPINNÄYTETYÖN OSAN 1 AIHE, TAVOITE JA TULOKSET

Ensimmäinen osa opinnäytetyöstä oli luonteeltaan projekti, jossa käytiin läpi C++-kirjaston käyttö Visual Studio -ohjelmassa Windows-ympäristössä. Projektiin valittiin kirjastoksi SDL, joka on tyypiltään graafinen kirjasto, jota käytetään suurimmalta osalta pelien tekemiseen. Projektissa tutustuttiin C++-kirjastojen Visual Studio -projekteihin liittämiseen sekä SDL-kirjaston perusominaisuuksiin, kuten ikkunan luontiin, siihen tekstin ja muotojen piirtämiseen sekä muotojen liikuttamiseen näppäimistön avulla.

SDL-kirjasto on aloittelijalle haastava, mutta C++-kielen osaava pääsee siihen nopeasti kiinni. Kirjastoon on myös saatavilla useita eri lisäosia, esimerkiksi äänien lisäämiseen ja PNG-kuvien piirtämiseen. Projekti näytti vain pienen osan SDL-kirjaston ominaisuuksista, mutta projektissa pääsee käsiksi SDL-kirjaston yleisestä käyttötavasta, josta on hyvä jatkaa haastavampiin projekteihin.

3 OPINNÄYTETYÖN OSAN 2 + 3 AIHE, TAVOITE JA TULOKSET

Toisen ja kolmannen osan yhdistetyn opinnäytetyön aihe oli Biim Ultrasound Oy:n lisenssijärjestelmän dokumentointi. Yritys käyttää järjestelmää hallitsemaan ja seuraamaan ultraäänilaitteiden käyttöä.

Lisenssijärjestelmän tavoitteena oli saada luotettava ja nopea järjestelmä, jolla pystytään estämään luvaton ultraäänilaitteen käyttö sekä maksattamaan asiakkaita tietyin väliajoin. Opinnäytetyössä dokumentoitiin järjestelmän vaatimukset, suunnitelma, toteutus ja testaus.

Järjestelmästä tuli lopussa valmis. Sitä käytetään lähettämällä ultraäänilaitteiden valmistuslinjalla dataa pilvipalveluun, jossa data muutetaan lisenssiksi ja tallennetaan tietokantaan. Tämä lisenssi sitten tallennetaan ultraäänilaitteeseen, josta se tarkistetaan mobiililaitteen puolesta ennen ultraäänikuvan näyttämistä. Järjestelmän pilvipalvelu toteutettiin käyttämällä Amazonin AWS-palvelua, joka oli helppokäyttöisin ja eniten dokumentoitu vaihtoehto. Järjestelmän käyttämä mobiilisovellus on ultraäänisovellus, joka tukee puhelimia ja tabletteja.

Pilvipalvelun testaukseen käytettiin Mocha-testauskirjastoa, joka ajaa ohjelmoidut testit kehittäjän tietokoneella ja palauttaa tuloksen. Mobiilisovellus testattiin normaalilla sovelluksien testaustavalla eli käyttämällä testausproseduureja, joissa on askeleet jotka kokeilevat lisenssin tarkistuksen toimivuutta. Järjestelmä läpäisi kaikki kirjoitetut testit ja asetettiin testauksen jälkeen asiakkaiden käytettäväksi.

4 YHTEENVETO

Opinnäytetyö suoritettiin kahtena osana, joista ensimmäinen osa oli kooltaan viisi opintopistettä ja toinen kymmenen opintopistettä. Opinnäytetyö oli miellyttävä tehdä osissa, ohjaus oli perusteellista ja koulun intrasta löytyi resursseja, joista oli helppoa tarkistaa asioita kuten tekstin muotoilut tai opinnäytetyön seuraava askel.

Ensimmäisessä osassa opettelin C++-kirjastojen käyttöä Visual Studio -ohjelmassa sekä SDL-kirjaston perusteet. SDL-kirjastosta opettelin vain perusominaisuuksia, eli ikkunan luontia sekä muotojen ja tekstin piirtoa ikkunaan ja muotojen siirtelyä näppäimistön avulla. Projektissa käytin Visual Studio -ohjelmaa Windows-ympäristössä. Tätä projektia aloittaessa kykyni C++-kielen kanssa olivat melkein olemattomat, mutta projektin lopussa osasin perustoiminnot jo paljon paremmin.

Toisessa osassa dokumentoin työpaikalla tehtyä lisenssijärjestelmää, johon sisältyi pilvipalvelu ja mobiilisovellus. Pilvipalvelu oli Amazon AWS-pohjainen ja mobiilisovellus Qt-pohjainen. Aikaisempaa kokemusta AWS-palvelusta ei ollut ollenkaan, joten opin siitä paljon kirjoittaessa. Qt-kirjastoa ja työkaluja olin käyttänyt ennenkin, joten siitä oli helpompaa kirjoittaa.

Opinnäytetyöt edistivät ammattiosaamistani huomattavasti. Ensimmäisessä osassa pääsin ensimmäistä kertaa aidosti kokeilemaan C++-kieltä, joka motivoi oppimaan kielestä lisää lukemalla kielelle tarkoitettuja keskustelupalstoja ja artikkeleita. SDL-kirjasto vaikuttaa olevan paras vaihtoehto peliohjelmointiin, jos haluaa tehdä pelin alusta asti käyttämättä ilmaisia pelimoottoreita, jotka usein hidastavat pelien suorituskykyä. Tästä syystä käyttäisin pelikehityksessä aluksi demonstraatioon ilmaista pelimoottoria kuten Unity tai Unreal Engine, mutta siirtyisin SDL-kirjastoon oikean pelikehityksen alkaessa. Toisessa osassa käytin myös C++-kieltä, mutta tällä kertaa Qt-kirjaston kera. Qt-kirjasto antaa mahdollisuuden tehdä ohjelmistoja usealle eri alustalle samalla koodilla, eli ohjelma toimii samalla tavalla Windows-, Mac-, Android- ja iOS-pohjaisella laitteella. Kirjaston järjestelmäriippumaton luonne, helppokäyttöiset komennot ja kätevä käyttöliittymän luonti ajavat minut käyttämään sitä myös kotiprojekteissa. Toisessa osassa opin myös käyttämään tiettyjä AWS-palveluita kuten API Gatewaytä. Tästä opin kuinka helppoa on pystyttää yksinkertainen pilvipalvelu, jota on mahdollista käyttää mistä tahansa maailmalta vaikkapa Qt-pohjaisella ohjelmalla.

Nämä kaksi oppimiskokemusta yhdistettäessä opin tekemään kokonaisen järjestelmän mihin tahansa tarkoitukseen ja kasvatin ansioluetteloani nähtävästi. Qt ja AWS ovat kummatkin alan jättiläisiä, joten näiden teknologioiden perusosaaminen auttaa työnhakua havaittavasti.

Petri Virkkunen

C++-KIRJASTOJEN KÄYTTÖ

C++-KIRJASTON KÄYTTÖ

Petri Virkkunen
Opinnäytetyö, osa 1
Kevät 2016
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

SISÄLLYS

SANASTO

2D	Kaksi ulottuvuutta, leveys ja korkeus.
2D-Grafiikkakirjasto	Kahta ulottuvuutta käyttävä grafiikkakirjasto
C++	C++ ohjelmointikieli
If-lause	Ohjelmoinnissa käytettävä komento, jolla voidaan tarkistaa arvoja.
Metodi	Ohjelmoinnissa käytettävä toiminto
RGBA	Värien asettaminen numeroiden avulla järjestyksessä Red, Green, Blue ja Alpha.
SDL	Simple Directmedia Layer-Grafiikkakirjasto C++:lle
Surface	Tapa tallentaa kokoelma pikseleitä ohjelman piirtämisen avuksi.
Rectangle	Neliön muotoinen alue
Tekstuuri	Pinnan rakenne

1 JOHDANTO

Työn tarkoituksena on edistää omaa osaamistani C++:n käytössä ja aiheena on kirjoittaa C++-pohjaisesta pelien tekoon käytetyn 2D-grafiikkakirjaston käytöstä. Testaan sitä tekemällä kirjaston pohjalta pienen ohjelman.

Kirjaston valitsin hakemalla Googlella C++-kirjastoja, joita aloittelijan on helppo käyttää. Löysin Simple Directmedia Layer (SDL) -kirjaston, josta on kattavat dokumentaatiot ja ohjeistukset sen käyttöön.

SDL on ilmainen kirjasto, jota saa käyttää missä vain ohjelmassa, ja sitä on käyttänyt mm. Valve. Se käyttää OpenGL- ja Direct3D-rajapintoja ja sitä on käytetty jopa emulointiin.

Ohjelmaan piirrän neliön ja teen jonkin tavan ohjata sitä, jotta ohjelmalla voi mahdollisesti testata myös elementtien yhteentörmäyksen havaitsemista. Kokeilen myös, miten kirjastolla saadaan tekstiä ilmestymään.

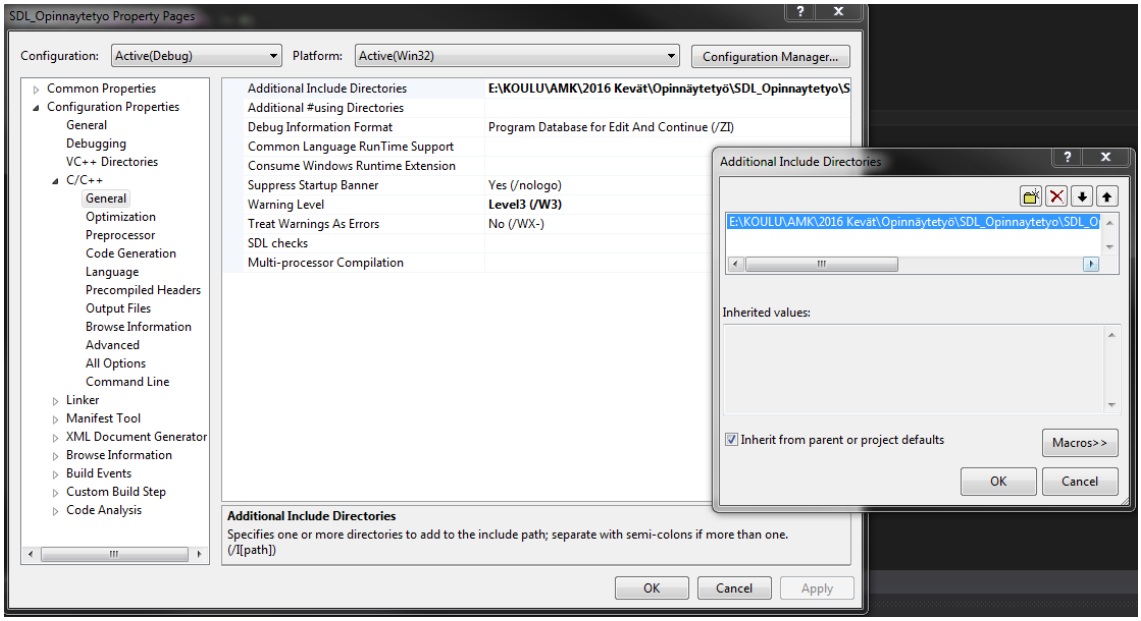
2 SDL-GRAFIIKKAKIRJASTON KÄYTTÖ

SDL on grafiikkakirjasto, jota voidaan käyttää melkein millä tahansa alustalla. Virallisesti se tukee Windowsia, Mac OS X:ää, Linuxia, iOSia ja Androidia. C++:n lisäksi sitä voidaan käyttää myös C#:lla ja Pythonilla (1).

SDL:llä vaikuttaa olevan aika tarkka dokumentaatio asennuksesta ja kirjaston käyttämisestä, joten saan luultavasti tehtyä koko testiohjelman pelkän virallisen dokumentaation avulla. Seuraavassa luvussa kerron SDL:n asentamisesta sekä uuden projektin luomisesta käyttämällä sitä. Tämä projekti on ohjelma, jossa testaan kirjaston käyttöä.

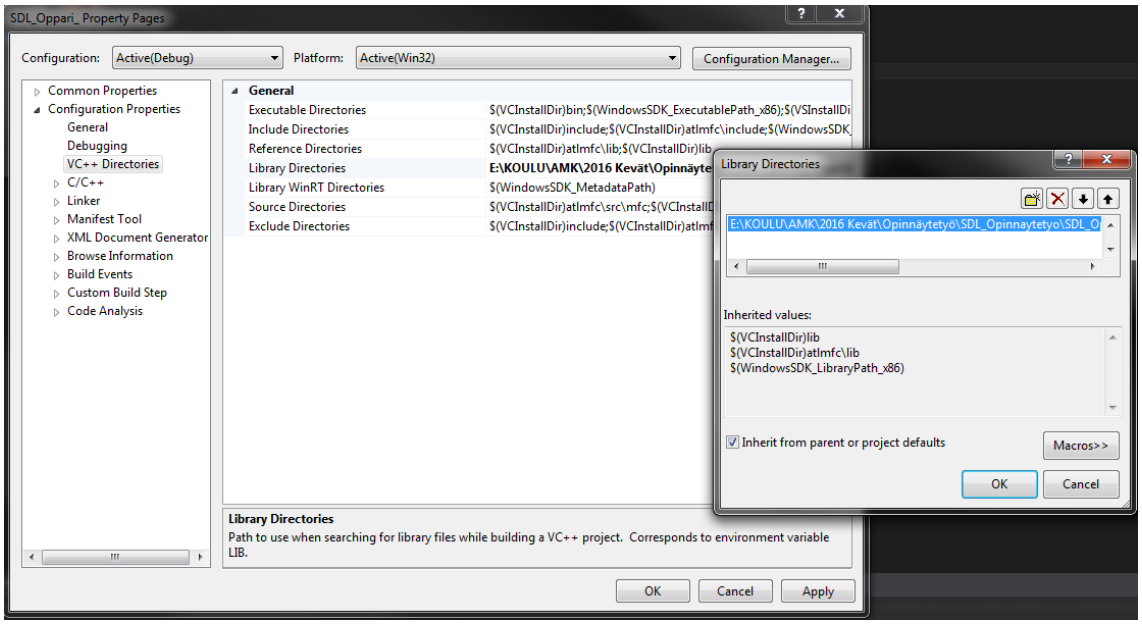
2.1 Asennus

Projekti aloitetaan lataamalla SDL-verkkosivuilta zip-tiedosto, jossa on include- ja libs-kansiot. Luodaan uusi Win32 Console-tyyppinen projekti Visual Studio 2015:lla, jotta voidaan käyttää konsoli-ikkunaa kehityksen aikana. Include-kansio kopioidaan projektin kansioon ja lisätään se projektin asetuksissa Additional Include Directories alle, jotta ohjelma tunnistaa kansiot. (Kuva 1.)



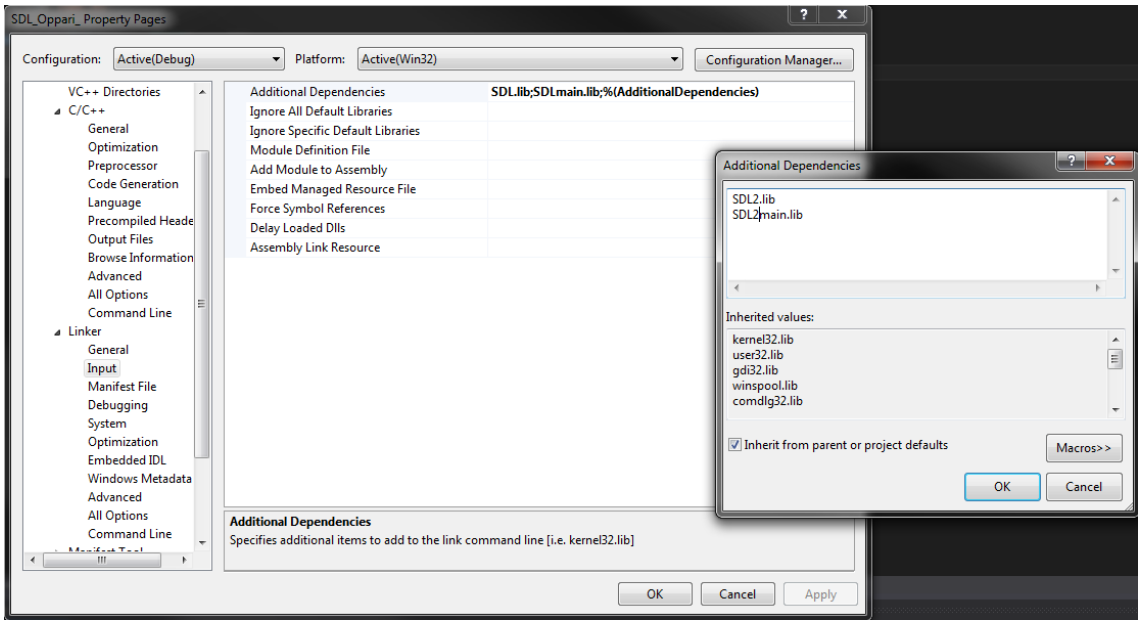
KUVA 1. Include-kansion lisääminen projektiin

Tämän jälkeen projektiin täytyy lisätä ”SDL library”-kansio, joka tehdään samassa valikossa kuin ”include”-kansio (Kuva 2).



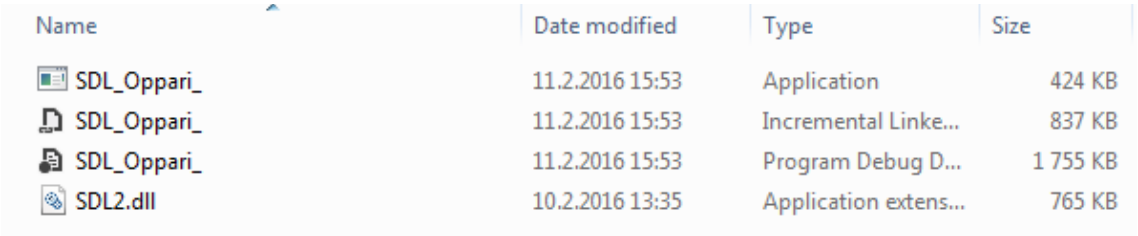
KUVA 2. Libs-kansion lisääminen projektiin

Lib-kansion asettamisen jälkeen täytyy lisätä projektin Linkerin ”Input”-osioon mitä tiedostoja kansiossa on, eli SDL2.lib ja SDL2main.lib. Tämä tehdään menemällä samaan valikkoon kuin aiemmin ja lisäämällä riville ”Additional Dependencies” teksti ”SDL2.lib; SDL2main.lib;”. (Kuva 3.) Luin aluksi vanhasta ohjeesta, että tekstin pitäisi olla ”SDL.lib; SDLmain.lib;”, mutta nämä olivat SDL:n vanhan version tiedostot, ja projektissa käytän SDL:n 2. versiota.



KUVA 3. Lib-tiedostojen lisääminen projektin Linkeriin

Lopuksi täytyy kopioida SDL2.ddl-tiedosto libs-kansiosta projektin debug-kansioon, jonka jälkeen ohjelman pitäisi käynnistyä ihan oikein (kuva 4).



Name	Date modified	Type	Size
SDL_Oppari_	11.2.2016 15:53	Application	424 KB
SDL_Oppari_	11.2.2016 15:53	Incremental Linke...	837 KB
SDL_Oppari_	11.2.2016 15:53	Program Debug D...	1 755 KB
SDL2.dll	10.2.2016 13:35	Application extens...	765 KB

KUVA 4. SDL2.dll debug-kansiossa

Nyt projekti on valmis ajamaan SDL-kirjaston ominaisuuksia ja aletaan tekemään projektiin perustoimintoja, kuten muotojen piirtämistä.

2.2 Kirjaston käyttö

Kirjaston asennus uuteen projektiin oli aika helppoa. Löysin ohjeet jotka kertoivat mitä tehdä askel askeleelta. Kirjaston toimivuuden testaukseksi ajetaan pieni ohjelma, joka alustaa kirjaston, sulkee kirjaston ja sulkee itsensä (Kuva 5). Jos kirjasto ei aukea, ohjelma näyttää virheviestin konsolissa siihen asti, että laitan ohjelman itse kiinni.

```
#include "stdafx.h"
#include <iostream>
#include <SDL.h>

int _tmain(int argc, _TCHAR* argv[])
{
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) //Kirjasto initialisoidaan, jos ei palauta 0 eli ei initialisoi oikein, tulostaa virheviestin.
    {
        std::cout << "SDL Init Error: " << SDL_GetError() << std::endl; //Virheen tulostus
        for(;;); //Jättää ohjelman päälle kunnes itse suljen sen.
        return 1;
    }
    SDL_Quit(); //Sulkee SDL:n
    return 0; //Sulkee ohjelman
}
```

KUVA 5. Testiohjelman koodi

Seuraavaksi kirjoitin koodin, jolla saan ikkunan ilmestymään. Tähän ikkunaan piirrän kuvioita myöhemmin, ja konsoli-ikkuna jää näkyville, jotta saan tulostettua virheviestejä tehdessäni ohjelmaa.

Koodi luo ensin ”SDL_Windowin”, johon täytyy laittaa ominaisuudeksi mm. ikkunan nimi ja sen koko. Ikkunan jälkeen luodaan ”SDL_Renderer”, eli olio, joka piirtää kaiken ikkunalle. (Kuva 6.) Sille laitettiin ominaisuuksiin mm. ikkuna, joka juuri luotiin.

```
void CreateWindow()
{
    SDL_Window *window = SDL_CreateWindow("SDL Oppari", 100, 100, SCREEN_WIDTH, SCREEN_HEIGHT, SDL_WINDOW_SHOWN);
    if(window == nullptr)
    {
        std::cout << "CreateWindowError: " << SDL_GetError() << std::endl;
        for(;;);
    }
    SDL_Renderer *renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC);
    if(renderer == nullptr)
    {
        std::cout << "CreateRendererError: " << SDL_GetError() << std::endl;

        for(;;);
    }


}
```

KUVA 6. SDL_Window- ja SDL_Renderer-olioiden luonti

Ohjelma toimi ihan oikein ensimmäisellä yrityksellä. Se loi ensin konsoli-ikkunan koska projekti on konsolityyppinen, jonka jälkeen oma koodini loi kirjaston avulla ikkunan ja sille ”renderer”-olion. (Kuva 7.)

KUVA 7. Ikkuna, joka juuri luotiin. Konsoli taustalla.

2.2.1 Neliön piirtäminen

Neliön piirtäminen aloitetaan SDL-kirjastossa luomalla ”SDL_Rect”-olio, koodissani nimeltä ”rect”. Tälle oliolle asetetaan leveys ja korkeus sekä x- ja y-koordinaatit. (Kuva 8.) (3.)

```
SDL_Rect rect;
rect.x = 50;
rect.y = 50;
rect.w = 50;
rect.h = 50;
```

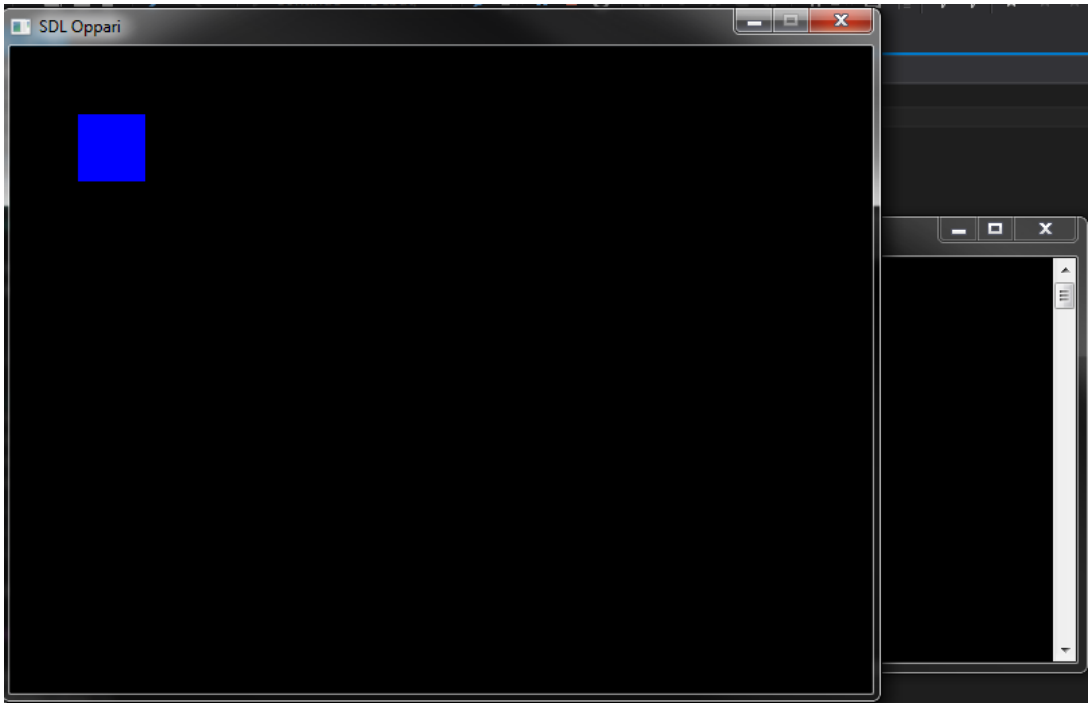
KUVA 8. Neliön leveyden ja korkeuden asettaminen.

Koon ja koordinaattien asettamisen jälkeen neliölle voidaan asettaa väri RGBA-muodossa ja piirtää se ikkunaan (Kuva 9).

```
SDL_SetRenderDrawColor(renderer, 0, 0, 255, 255);
SDL_RenderFillRect (renderer, &rect);
SDL_RenderPresent(renderer);
```

KUVA 9. Neliön värin asettaminen ja sen piirtäminen ohjelmaan.

Neliön piirtäminen onnistui myös ihan mallikkaasti. Ikkunassa tausta on musta ja neliö on sininen, niin kuin oli tarkoituskin (Kuva 10).



KUVA 10. Piirretty sininen neliö näkyy oikein.

2.2.2 Tekstin kirjoitus

Jotta kirjastolla voidaan kirjoittaa tekstiä ruudulle, siihen täytyy ladata ja asentaa lisäosa nimeltä ”SDL2_ttf”. Tämä lisätään projektin asetuksiin samalla tavalla kuin itse kirjasto aluksi tehtiin.
Itse tekstin kirjoituksen saa aikaan seuraavilla askelilla. Ensin initialisoidaan ”SDL_ttf”-kirjasto, jotta ohjelma osaa käyttää kirjaston elementtejä. Initialisoin kirjaston if-lauseen sisällä, jotta voin helposti laittaa konsoliin näkymään virheviestin, jos jokin menee vikaan. (Kuva 11.)

```
if(TTF_Init() == -1)
{
    std::cout << "TTF Error:" << TTF_GetError() << std::endl;
}
```

KUVA 11. TTF-kirjaston initialisointi.

Itse tekstin kirjoitukselle ja piirtämiselle tein oman metodin, ”DrawText()”, johon laitoin fontin ja tekstin värien luontiin sekä tekstin tulostukseen käytettävän koodin.
Fontin luontiin koodissa tarvitaan ensin fonttitiedosto, jonka hain Googlen fonts-sivustolta ja laitoin projektikansioon. Koodissa täytyy myös valita tekstille fonttikoko. Koodasin myös if-lauseen, jolla ohjelma saa virheet kiinni ja kirjoittaa mahdollisista virheistä konsoliin (Kuva 12).

```
void DrawText()
{
    TTF_Font *Oswald = TTF_OpenFont("Oswald-Regular.ttf", 35);
    if(Oswald == nullptr){std::cout << "Font error: " << SDL_Error << std::endl;}
```

KUVA 12. Fontin luonti DrawText()-funktiossa

Fontin luonnin jälkeen täytyy tekstille päättää värit, joten tein kaksi värioliota. Nämä liittyvät itse SDL-kirjastoon (Kuva 13).

```
    SDL_Color White = {255, 255, 255};
    SDL_Color bg = {0, 0, 0};
```

KUVA 13. Väriolioiden luonti

Värien jälkeen voidaan teksti piirtää erilliselle Surface-oliolle, joka on myös osa SDL-kirjastoa. Surfacen luonnissa siihen täytyy asettaa teksti, tekstin väri ja taustan väri. Surface täytyy sitten

muuntaa tekstuurimuotoon, jotta se voidaan piirtää ohjelmaan. Surfacen tekstuuriin muuttaessa täytyy antaa renderer-olio sekä surface, joka aiemmin tehtiin (Kuva 14).

```
SDL_Surface *surfaceMessage = TTF_RenderText_Shaded(Oswald, "Text here", White, bg);
SDL_Texture *Message = SDL_CreateTextureFromSurface(renderer, surfaceMessage);
```

KUVA 14. Surface-olion luonti ja sen muutos tekstuuriksi.

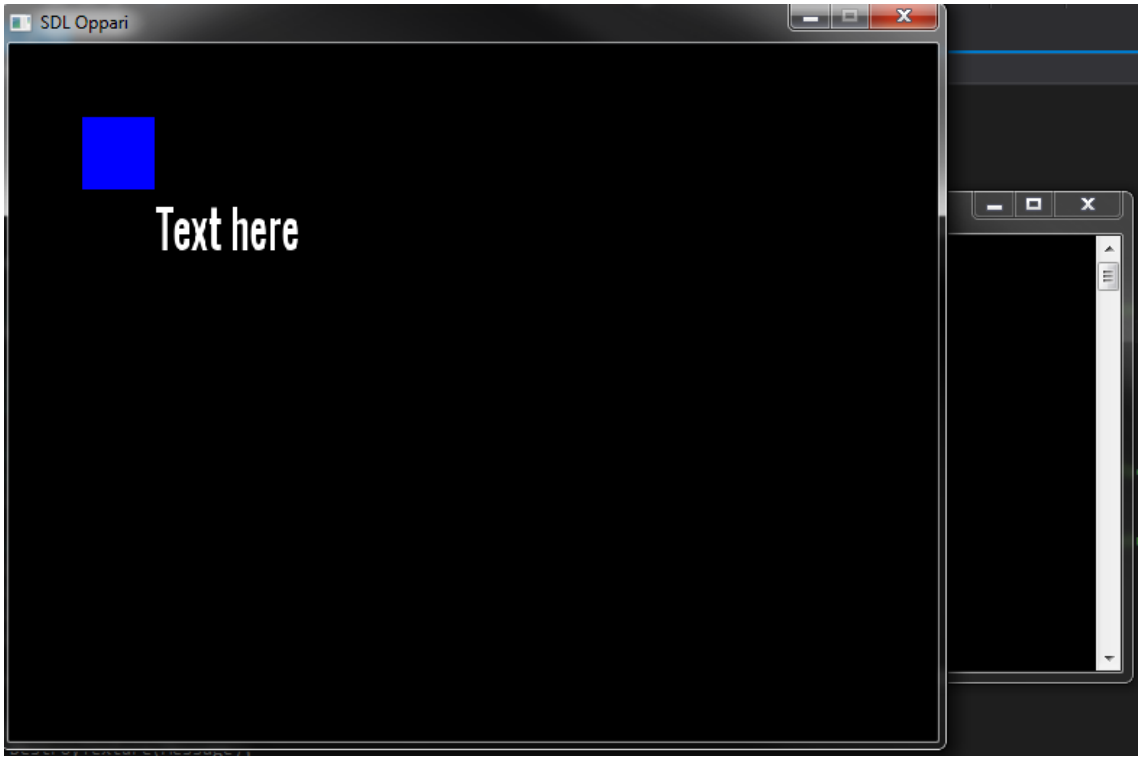
Tekstuurin lisäksi tekstin piirtämiseen vaaditaan Rectangle eli neliö, joka tehtiin kerran jo aikaisemmassa luvussa, joten en käy sitä sen erikoisemmin läpi. Tähän neliöön asetetaan alue, jolle teksti halutaan piirtää. Sitä käytetään periaatteessa vain tekstin alueen koon ja koordinaattien asettamiseen.

Viimeinen askel on piirtää teksti näytölle käyttämälle aikaisemmin tehtyjä olioita SDL_RenderCopy-metodin sisällä (Kuva 15).

```
SDL_RenderCopy(renderer, Message, NULL, &Msg_Rect);
```

KUVA 15. Tekstin piirtäminen näytölle

Loppujen lopuksi sain tekstin piirrettyä ikkunaan ilman suurempia ongelmia, teksti näkyy oikeassa paikassa ja värit ovat oikein (Kuva 16).



KUVA 16. Teksti piirrettynä ikkunaan.

2.2.3 Objektien liikuttaminen näppäimistöllä

Jotta voin testata kirjaston objektien törmäämisen huomioinnin, minun täytyy pystyä liikuttamaan olioita ohjelman ollessa päällä.

Objektien liikuttamisen saa onnistumaan käskemällä ohjelmaa seuraamaan koko ajan näppäimistöä ja tarkistamaan, onko tiettyjä näppäimiä painettu. Kirjastossa tämä tehdään tarkistamalla SDL_PollEvent-tapahtumaa. Jos tämä tapahtuma palauttaa arvon, katsotaan arvosta, onko se tyyppiä ”SDL_KEYDOWN”. Jos tämäkin on totta, voidaan tarkistaa, mitä näppäimiä käyttäjä painoi, ja asettaa toimintoja jokaiselle mahdollisuudelle (Kuva 17).

```
while(!quit)
{
    while(SDL_PollEvent(&event))
    {
        if (event.type == SDL_KEYDOWN)
        {
            switch (event.key.keysym.sym)
            {
                case SDLK_UP:
                case SDLK_DOWN:
                case SDLK_LEFT:
                case SDLK_RIGHT:
                    moveRect(event);
                break;
                case SDLK_ESCAPE:
                    quit = 1;
                break;
            }
        }
        if (event.type == SDL_QUIT)
        {
            quit = 1;
        }
    } //While SDL_PollEvent
```

KUVA 17. Tapahtuman havaitseminen ja näppäimen tunnistaminen.

Asetin neliön liikkumisen näppäimistön nuolinäppäimille, joten tein uuden ”moveRect”-funktion ohjelmaan, jolla käsittelen neliön liikkumisen. Tapahtuman tarkistus ajaa tämän funktion, jos se havaitsee jonkin nuolinäppäimen painamisen. Funktion ajamiseen vaaditaan ”Event”-olio, joka saadaan samalla kun tapahtuma havaitaan. Funktio tarkistaa tapahtuman arvoa ”switch”-silmukalla, joka siirtää neliötä nuolinäppäimien osoittamaan suuntaan (Kuva 18).

```
void moveRect(SDL_Event event)
{
    switch(event.key.keysym.sym)
    {
    case SDLK_UP:
        rect.y -= movespeed; //Y-Akselilla ylöspäin
    break;
    case SDLK_DOWN:
        rect.y += movespeed; //Y-Akselilla alaspäin
    break;
    case SDLK_RIGHT:
        rect.x += movespeed; //X-Akselilla oikealle
    break;
    case SDLK_LEFT:
        rect.x -= movespeed; //X-Akselilla vasemmalle
    break;
    }
}
```

KUVA 18. Suunnan havaitseminen ja neliön liikuttaminen.

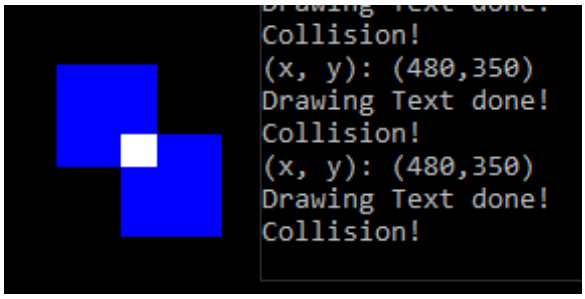
2.2.4 Törmäyksen havaitseminen

Törmäyksen havaitseminen on tärkeä osa mitä tahansa pelien tekoon käytettävää kirjastoa - tämän vuoksi ohjelmassa testataan myös sitä. Kirjastossa on sisäänrakennettu neliöiden törmäyksen havainnointi, joka toimii antamalla sille kaksi neliötä sekä yhden uuden neliön. Tyhjään neliöön funktio tallentaa alueen, jolla kaksi aiempaa neliötä osuvat toisiinsa. Tätä kolmatta neliötä voidaan käyttää esimerkiksi törmäävän alueen korottamisessa alueen väriä muuttamalla. (2)
Ohjelmaan laitettiin komento, jolla se piirtää uuden valkoisen neliön törmäysalueelle sen korostamiseksi (Kuva 19).

```
if (SDL_IntersectRect(&rect, &collision_rect, &result_rect))
{
    std::cout << "Collision!" << std::endl;
    DrawRectangle(result_rect, 1);
}
```

KUVA 19. Törmäyksen havainto sekä uuden neliön piirtäminen.

Törmäyksen havaitseminen oli loppujen lopuksi aika helppoa. Neliöiden osuessa toisiinsa konsolissa lukee ”Collision!” ja ikkunassa näkyy osuma-alueen kohdalla valkoinen laatikko (Kuva 20).



KUVA 20. Neliöiden törmäysalue ja konsoliin tulostettu viesti.

3 YHTEENVETO

Työn tavoitteena oli SDL-kirjaston oppiminen kirjoittamalla ohjelma, jossa käytetään kirjaston perusteita. Ohjelmointi sujui hyvin. Siinä ei suurempia ongelmia tullut vastaan, osin oman aikaisemman ohjelmointikokemuksen ja osin kirjaston dokumentoinnin ja helppokäyttöisyyden johdosta.

Kirjaston oppiminen oli ihan sulavaa. Metodien ja olioiden nimet ovat loogisia, mutta en ehdottaisi C++:aa kielenä aloittelijalle, vaan esim. Javaa tai C#:a.

Hankalin osa työn tekemisessä oli itse kirjaston asentaminen. Ohjelmointi ei ollut niinkään haastavaa, koska olin sitä tehnyt jo aiemmin, mutta kirjastoja en ollut ikinä ennen käyttänyt projekteissa.

Työ muistutti minua, miten pelien ohjelmointi eroaa normaalien ohjelmien ohjelmoinnista, ja tämä auttaa ainakin omien projektien suunnittelussa. Opin myös käyttämään kirjastoja Visual Studiolla, mitä en ollut aiemmin tehnyt.

LÄHTEET

1. What is it? SDL Wiki. Saatavissa: https://wiki.libsdl.org/FrontPage Hakupäivä 29.1.2016

2. SDL_IntersectRect. SDL Wiki 2010. Saatavissa: https://wiki.libsdl.org/SDL_IntersectRect Hakupäivä 29.3.2016

3. SDL_Rect. SDL Wiki 2015. Saatavissa: https://wiki.libsdl.org/SDL_Rect Hakupäivä 29.3.2016

Petri Virkkunen

PILVIPOHJAISEN LISENSSIJÄRJESTELMÄN TOTEUTUS

PILVIPOHJAISEN LISENSSIJÄRJESTELMÄN TOTEUTUS

Petri Virkkunen
Opinnäytetyö, osat 2 + 3
Syksy 2017
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, Ohjelmistotuotanto

Tekijä(t): Petri Virkkunen
Opinnäytetyön nimi: Pilvipohjaisen lisenssijärjestelmän toteutus
Työn ohjaaja: Pekka Alaluukas
Työn valmistumislukukausi ja -vuosi: Syksy 2017 Sivumäärä: 25

Tämän opinnäytetyön tarkoituksena on dokumentoida Biim Ultrasound Oy:n pilvipohjaisen lisenssijärjestelmän toteutusta. Biim Ultrasound Oy valmistaa kannettavia langattomia ultraäänilaitteita, joista tuleva kuva lähetetään langattomalla yhteydellä mobiililaitteeseen, eli puhelimeen tai tablettiin. Tabletissa olemassa ohjelmistossa pystyy myös ottamaan ultraäänilaitteesta tulevasta videosta kuvia sekä lyhyempiä videoita.

Lisenssijärjestelmän tavoitteena on estää maksamattomien ultraäänilaitteiden käyttö. Tällä järjestelmällä yritys voi kerätä käyttäjiltään maksua tietyin väliajoin, rajoittaa ohjelmiston ominaisuuksia lisenssin tason perusteella tai estää asiakkaiden ohjelmiston väärinkäytön. Lisenssijärjestelmä koostuu JSON-formaatissa tallennettavasta lisenssistä, pilvipalvelusta sekä mobiili- ja ultraäänilaitteen ohjelmistosta. Mobiililaitteen ohjelmiston pohjalla on Qt-kirjasto, jossa käytetään C++, Javascript ja QML-kieliä. Pilvijärjestelmä on Amazon Web Services-pohjainen.

Lisenssijärjestelmä onnistui hyvin ja tekee tehtävänsä turvallisesti, nopeasti ja järkevästi. Järjestelmää on helppo lähteä tästä jatkamaan muihin suuntiin ja lisäämään siihen uusia ominaisuuksia. Järjestelmälle tehtiin toteutuksen jälkeen myös varmuuskopiointiohjelma, joka hakee päivittäin ultraäänilaitteet ja lisenssit tietokannasta ja tallentaa ne paikalliselle kovalevylle.

Avainsanat: pilvipalvelut, ultraääni, lisenssi, C++

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Software development

Author(s): Petri Virkkunen
Title of thesis: Development of a cloud based licensing system
Supervisor(s): Pekka Alaluukas
Term and year when the thesis was submitted: Autumn 2017 Number of pages: 25

The purpose of this thesis is to describe the process behind developing Biim Ultrasound Oy's cloud based licensing system. Biim Ultrasound Oy develops small wireless ultrasound devices which send their ultrasound data over a wireless connection to a mobile device. The Biim Ultrasound software then takes that data and allows the user to capture images or small video clips of it. The user can then edit the captured images by adding accurate measurements and text annotations as well as freely drawing on the image.

The goal of this licensing system is to prevent unpaid users from using the device and to allow Biim Ultrasound Oy to bill them at a specific interval, as well as allowing Biim Ultrasound Oy to remove access from customers misusing the system.

The tablet software uses the Qt software library which involves using C++, Javascript and QML. The cloud service uses Amazon Web Services for its' databases, endpoints and functions.

At the end of this thesis, the system is working well. It is secure, fast and easy to use and expand with new features. Once the system was done, a backup system was added that takes daily license and ultrasound device data from the cloud database and saves it on a local hard drive to make sure no data is lost in the event of a catastrophic error.

Keywords: Cloud, Qt, License, Ultrasound, Amazon

SISÄLLYS

SANASTO

API	Application Programming Interface
AWS	Amazon Web Services
JSON	JavaScript Object Notation
Node	JavaScript-ohjelmointikirjasto
Qt	Järjestelmäriippumaton ohjelmointikirjasto

1 JOHDANTO

Opinnäytetyön tarkoituksena on dokumentoida lisenssihallintajärjestelmän suunnittelu ja toteutus Biim Ultrasound Oy:lle. Järjestelmän tarkoitus on antaa yritykselle mahdollisuus estää ultraäänilaitteen käyttö joko maksusyistä tai väärinkäytön seurauksena. Tulevaisuudessa on myös mahdollista rajoittaa ohjelmiston tai ultraäänilaitteen ominaisuuksia tietynlaisille lisensseille.

Lisenssit toimivat siten, että käyttäjä ostaa tietylle ajalle laitteelle käyttöluvan. Tämä aika voi olla mikä tahansa, mutta aluksi yksi maksu vuodessa. Tästä maksusta luodaan vuoden mittainen lisenssi laitteen käyttöön, ja tästä lisenssistä luodaan pienemmän ajan lisenssejä sopimuksen mukaan. Lisenssi ladataan laitteen käyttöönoton aikana pilvipalvelusta mobiilisovellukseen, joka varmentaa lisenssin oikeuden ja tallentaa lisenssin myös ultraäänilaitteeseen. Kun pienen lisenssin aika loppuu, mobiilisovellus kehottaa käyttäjää lataamaan uusi lisenssi pilvipalvelusta. Jos käyttäjä kieltäytyy lataamasta lisenssiä ennen lisenssin viimeistä käyttöpäivää, mobiilisovellus lopettaa toiminnan.

Tässä opinnäytetyössä kuvataan Lisenssijärjestelmän kaikki osat paitsi laiteohjelmisto. Kuvataan siis pilvipalvelu, mobiilisovelluksen lisenssiin liittyvät osat sekä lisenssin luonti, sisältö ja sisällön suojaus, että pilvipalvelun ja mobiilisovelluksen testaus.

2 JÄRJESTELMÄN VAATIMUKSET

Järjestelmä tulee käyttöön lääketieteen alalle, joten se ei saa olla kiinni tai pois päältä pitkiä aikoja. Koska järjestelmä toimii lääketieteen alalla, se ei saa paljastaa tietoja potilaista eikä asiakkaista, joten sen täytyy olla joka puolin turvallinen käyttää.

2.1 Pilvipalvelun vaatimukset

Pilvipalvelun täytyy olla aina käytettävissä. Jos käyttäjä haluaa hakea tai luoda lisenssin, hänen täytyy pystyä siihen mihin aikaan tahansa. Tästä syystä palvelimen täytyy olla luotettava, maailmanlaajuisesti käytettävä sekä tarpeeksi nopea. Reaaliaikaisuutta ei kuitenkaan tarvita, joten pilvipalvelua voidaan odottaa yhdestä viiteen sekuntiin asti.

Pilvipalvelun täytyy olla suojeltu palvelunesto-hyökkäyksiltä, joissa lähetetään pilvipalvelun osoitteeseen suuri määrä dataa, jolloin palvelin hidastuu tai saattaa jumittua kokonaan eikä tällöin pysty palvelemaan asiakkaita.

Pilvipalvelua täytyy myös pystyä testaamaan, joten tarvitaan toinen pilvipalvelu, joka ei käytä asiakkaiden tietoja eikä pääse niihin ollenkaan koskemaan, mutta ajaa kuitenkin täysin samaa koodia samalla tavalla kuin asiakasversio.

2.1.1 Lisenssin salaaminen pilvipalvelussa

Lisenssissä täytyy pystyä tallentamaan salattuna tietoa, esimerkiksi lisenssille linkitetyn laitteen sarjanumero ja mallinumero. Lisenssiin täytyy myös pystyä tallentamaan lisenssin alku- ja loppupäivämäärät sekä lisenssin päivityspäivämäärä jolloin mobiilisovellus ilmoittaisi käyttäjälle lisenssin tarvitsevan päivitystä. Lisenssin täytyy olla suojattu muokkaamiselta, jottei käyttäjä pysty tätä siirtämään toiselle laitteelle tai muuttamaan päivämääriä saadakseen lisenssin kestämään kauemman aikaa.

2.1.2 Lisenssin luonti pilvipalvelussa

Lisenssien luonnissa täytyy käyttää hyvin suojattua avainta, sertifikaattia sekä satunnaistekstiä. Lisenssien luontitapa ei saa olla selvä käyttäjälle, jos hän lukee lisenssitiedostoa.

2.1.3 Lisenssin haku pilvipalvelussa

Lisenssien haku tehdään ainoastaan mobiililaitteelta, joten lisenssien haun täytyy olla tarpeeksi nopeaa, jotta se ei hidasta ohjelmiston käyttökokemusta. Lisenssien hausta täytyy palauttaa oikeita virheilmoituksia, jotta mobiiliohjelmisto osaa näyttää oikeita viestejä käyttäjälle.

2.2 Mobiilisovelluksen vaatimukset

Mobiilisovelluksen täytyy osata tarkistaa lisenssin voimassaolo ja kelpoisuus sekä hakea lisenssiä ultraäänilaitteesta, mobiililaitteen muistista tai pilvipalvelusta. Mobiilisovelluksen täytyy myös näyttää järkeviä virheilmoituksia käyttäjälle, jos lisenssiä ei ole tai siinä on jotain vikaa. Mobiilisovellukseen täytyy lisätä valikko, jossa näkyy, kuinka kauan aikaa nykyinen käytössä oleva lisenssi on vielä voimassa.

3 JÄRJESTELMÄN SUUNNITELMA

3.1 Pilvipalvelu

Pilvipalvelun implementaatiossa oli mahdollisuutena käyttää Google Cloud -palvelua, Amazonin AWS-palvelua tai itserakennettua palvelinta. Lopuksi päädyttiin Amazonin AWS-palveluun, koska sillä oli eniten dokumentaatiota verkossa ja moni suuri nettipalvelu käyttää AWS-palveluita. AWS-palvelulla on myös suuri määrä ominaisuuksia jokaiseen tarpeeseen.

AWS-palveluista aiottiin käyttää seuraavia: AWS API Gateway, AWS Lambda ja AWS DynamoDB. Näistä palveluista API Gateway hallitsee HTTP-kutsujen – jotka tulevat mobiililaitteesta sekä laitteiden tuotannosta – vastaanottoa, edelleen lähetystä ja niiden määrän rajoitusta. Lambda ottaa vastaan API Gatewayn lähettämät viestit ja ajaa niiden perusteella ohjelman, joka tallentaa tai ottaa lisenssejä DynamoDB-tietokannasta.

3.2 Mobiilisovellus

Mobiilisovelluksessa täytyy tarkistaa lisenssin olemassaolo ja sen loppu- ja alkupäivämäärä. Lisenssin olemassaolo tarkistetaan ensin mobiililaitteen muistista, sitten yhdistetystä ultraäänilaitteesta. Jos kummassakaan ei ole lisenssiä, sovellus pyytää käyttäjän yhdistämään mobiililaitteen verkkoon ja pyytää sitten automaattisesti lisenssiä pilvipalvelulta. Jos pilvipalvelussakaan ei ole lisenssiä, tätä ultraäänilaitetta ei voi käyttää mobiilisovelluksen kanssa.

Lisenssistä tarkistetaan sen alku- ja loppupäivämäärä. Jos tarkistuksen päivämäärä on ennen lisenssin alkupäivämäärää, käyttäjälle ilmoitetaan, että lisenssiä ei ole vielä olemassa. Tällöin käyttäjä pystyy alkamaan käyttämään laitetta vasta lisenssin aloituspäivämääränä. Jos lisenssin loppupäivämäärä on ennen tarkistusta, laitteen käyttö estetään ja käyttäjälle annetaan virheviesti.

Lisenssit suojataan salaisilla allekirjoituksilla. Näihin allekirjoituksiin sisältyy mm. lisenssin alku- ja loppupäivämäärät. Jos allekirjoitukset ovat oikein tarkistushetkellä, sovellus tallentaa lisenssin ultraäänilaitteeseen ja tallentaa lisenssin hajautusarvon suojattuun tietokantaan, jota vasten tarkistetaan lisenssi aina, kun käyttäjä yrittää päästä käyttämään ultraäänilaitetta.

4 JÄRJESTELMÄN TOTEUTUS

Toteutusvaiheessa lisenssijärjestelmän suunnitelmaa muutettiin jonkin verran. Päätettiin, että ensimmäiset laitteet toimitetaan päättymättömillä lisensseillä, jotka kestävät vuoteen 2995 asti. Nämä lisenssit asennetaan laitteisiin sisään jo tehtaalla. Tästä syystä pilvipalvelun ja mobiiliohjelmiston täytyy tunnistaa päättymättömät lisenssit. Nämä lisenssit merkataan asettamalla lisenssin loppumispäivämäärä tietyksi, jolloin mobiiliohjelmisto huomaa, että lisenssistä ei tarvitse ilmoittaa käyttäjälle muulloin kuin jos se korruptoituu tai sitä ei ole olemassa.

4.1 Lisenssi

Lisenssi tallennetaan JSON-muodossa pilveen, mobiililaitteeseen ja ultraäänilaitteeseen. Lisenssissä on seuraavat kentät:

- creationDate – Lisenssin luontipäivämäärä
- startDate – Lisenssin voimassaolon alkamispäivämäärä
- endDate – Lisenssin voimassaolon loppumispäivämäärä
- partNumber – Lisenssille tarkoitetun ultraäänilaitteen osanumero
- serialNumber – Lisenssille tarkoitetun ultraäänilaitteen sarjanumero
- pollDate – Päivä, jolloin mobiilisovellus muistuttaa käyttäjää päivittämään lisenssin
- pollReminderDate – Päivä, jonka jälkeen mobiilisovellus alkaa muistuttaa useammin lisenssin päivityksestä
- probeId – Ultraäänilaitteen salainen identifikaatioluku

Lisenssissä on myös muita salaisia kenttiä, joita tässä opinnäytetyössä ei mainita turvallisuuden vuoksi.

4.2 Pilvipalvelu

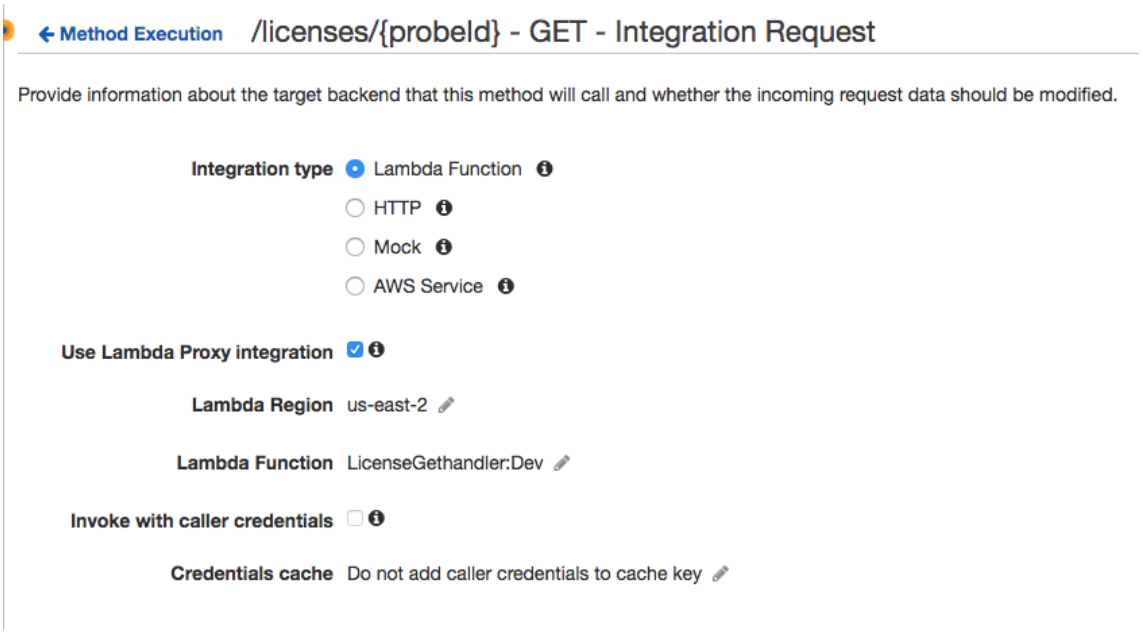
Pilvipalvelu on AWS-pohjainen. Pilvipalvelun rakenne koostuu seuraavista AWS-palveluista: API Gateway, Lambda, CloudWatch ja DynamoDB. Pilvipalvelu siis toimii käyttämällä API Gatewaytä ottamaan kaikki pyynnöt vastaan ja Lambdaa käsittelemään pyynnöissä mukana tulevaa dataa, lukemaan DynamoDB-tietokannasta ja kirjoittamaan sinne sekä palauttamaan dataa API Gatewaylle, joka palauttaa sen pyynnön lähettäjälle.

4.2.1 API Gateway

Pilvipalvelu käyttää API Gateway-palvelua pilvipalvelun rajapinnan hallintaan. Rajapinta hoitaa mm. HTTP-pyyntöjen edelleen lähetykseen Lambda-funktioille.

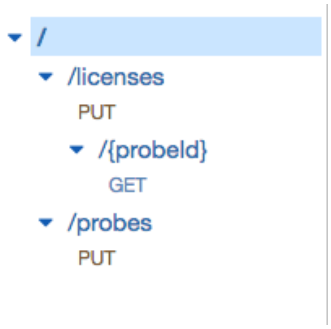
API Gateway -palvelussa on mahdollista tallentaa erilaisia rajapintakonfiguraatioita, joita voidaan linkittää eri internet-osoitteisiin. Rajapinnan asiakasversio on linkitetty *https://example.com/v1/* -osoitteeseen, ja testiversio on linkitetty osoitteeseen *https://example.com/dev/*.

Testiversiolla on käyttölupa ainoastaan *Test*-version, ja asiakasversiolla ainoastaan *Production*-version funktion kutsumiseen. Tämä saatiin aikaiseksi käyttämällä Lambdan Alias-ominaisuutta, jota voidaan käyttää ulkopuolisten AWS-palvelujen osoittamiseen eri versioihin samasta Lambda-funktiosta. API Gatewayssä tämä on mahdollista saavuttaa asettamalla rajapinnan päätepisteen kutsuman Lambda-funktion nimen perään haluttu Lambda-funktion Alias, joka on edelletty kaksoispisteellä (kuva 1).



KUVA 3. Lisenssien haun päätepisteen testiversion asetukset.

Lisenssijärjestelmän pilvipalvelun rajapinnalla on kolme päätepistettä (kuva 2). Näistä päätepisteistä muodostuu http-kutsuille osoitteita. Esimerkiksi lisenssin haun asiakasversion päätepiste on osoitteessa https://example.com/v1/licenses/{probeId}, jossa {probeId} korvataan ultraäänilaitteesta saatavalla uniikilla identifikaationumerolla.



KUVA 4. Rajapinnan rakenne

4.2.2 Lambda

AWS Lambda on palvelu, joka antaa ajaa koodia ilman sen kummempaa palvelimien konfigurointia. Lambda skaalautuu automaattisesti, eli se käynnistää niin monta instanssia itsestään kuin sillä hetkellä tarvitaan. Jos mobiilisovelluksesta tulee esimerkiksi 1000 pyyntöä sekunnin aikana rajapinnan läpi, Lambda käynnistää jokaiselle pyynnölle oman instanssin. (1.)

Lisenssijärjestelmän pilvipalvelun Lambda-funktiot ovat kirjoitettu JavaScript-kielellä, käyttäen Node-kirjastoa. Node valittiin, koska muut vaihtoehdot eli Java, C# ja Python olivat vähemmän dokumentoituja. Java- ja C#-pohjaiset funktiot olivat myös hitaampia käynnistymään.

Lambda-funktioita on kolme, joista yksi palauttaa dataa tietokannasta ja kaksi kirjoittaa dataa tietokantaan. Lisenssien hakuun käytettävä Lambda-funktio on nimeltään GetLicenseHandler, lisenssien tallennukseen käytettävä Lambda-funktio on nimeltään PutLicenseHandler, ja ultraäänilaitteiden tietojen tallennukseen käytetään funktiota PutProbeHandler.

PutLicenseHandler

PutLicenseHandler-funktion tarkoitus on tarkistaa pyynnössä lähetetty data (kuva 3), salata se ja tallentaa salattu data tietokantaan.

Funktio tarkistaa ensin datan olemassaolon viestissä, ja sitten tarkistaa datan sisällä olevat tiedot perusteellisesti. Funktio käyttää seuraavia sääntöjä datan vahvistamiseen:

- Datassa täytyy olla seuraavat kentät:
    - startDate.
    - endDate.
    - partNumber.
    - serialNumer.
    - pollDate.
    - pollReminderDate.
- Kenttien startDate, endDate, pollDate ja pollReminderDate täytyy olla kelpaavia päivämääriä
- Kentän startDate täytyy olla ennen kentän endDate päivää
- Kentän pollDate täytyy olla ennen päivää endDate ja päivän startDate jälkeen
- Kentän pollReminderDate täytyy olla ennen päivää endDate ja päivän pollDate jälkeen

- Jokaisen päivämääräkentän täytyy loppua kirjaimeen Z, joka kertoo päivämäärän olevan UTC-aikavyöhykkeellä
- Kentän partNumber täytyy alkaa kirjaimella P, olla 10 merkkiä pitkä ja sen kolmanneksi viimeinen merkki täytyy olla väliviiva.
- Kentän serialNumber täytyy olla kelpaava numero, 9 merkkiä pitkä, arvoltaan alle 999999999 ja suurempi kuin 0.

Ainoastaan kaikki nämä säännöt läpäisevä lisenssi päästetään koodissa läpi salaukseen. Tarkistuksen jälkeen haetaan laitetietokannasta vielä lisenssiin liittyvän laitteen tiedot. Laitteen tiedot haetaan generoimalla lisenssin kentistä salainen, uniikki merkkijono, joka on laitteen valmistuksen aikana generoitu samalla tavalla ja tallennettu pilven tietokantaan. Lisenssi- ja laitedata yhdistetään yhteen JSON-objektiin, jolle tehdään salausprosessi.

```
[
  {
    "startDate": "2017-07-20T12:51:02Z",
    "endDate": "2017-10-20T12:51:02Z",
    "pollDate": "2017-09-20T01:00:00Z",
    "pollReminderDate":"2017-10-10T01:02:00Z",
    "partNumber": "P111111-11",
    "serialNumber": "111111111"
  }
]
```

KUVA 5. Kelpaava funktioon lähetetty lisenssi

Lisenssi salataan käyttämällä Noden Crypto-kirjaston metodeja, mutta salauksen tarkempi prosessi pidetään yksityisenä turvallisuussyistä. Salauksen jälkeen lisenssi kirjoitetaan tietokantaan.

GetLicenseHandler

GetLicenseHandler-funktion tarkoitus on palauttaa pyynnölle vastauksena merkkijonoksi muutettu lisenssi, joka haetaan tietokannasta.

Funktio ottaa vastaan probeId-nimisen polkuparametrin rajapinnalta. Parametri probeId on uniikki merkkisarja, joka generoidaan ensin laitteelle tuotantolinjalla ja myöhemmin myös lisenssin luontifunktiossa, jolloin se tallennettiin pilvessä tietokantaan.

Parametrin saatuaan funktio hakee tietokannasta tällä parametrilla lisenssin. Jos lisenssi tälle parametrille löytyy, funktio tarkistaa lisenssin voimassaolon tarkistamalla lisenssin endDate-kentän. Jos kentässä oleva päiväys on jo mennyt, funktio palauttaa virheviestin ja -koodin. Funktio palauttaa myös virheviestin ja –koodin jos lisenssiä ei löydy tai jos funktio ei saa parametria.

4.2.3 CloudWatch

CloudWatch on seurantapalvelu, jolla pystyy seuraamaan AWS-palveluiden käyttäytymistä luomalla lokitiedostoja ja asettamalla hälytyksiä. CloudWatch-lokit voidaan myös tallentaa vaikkapa S3- tai Glacier-palveluun, jossa niitä voidaan säilyttää niin kauan kuin halutaan. (2.)

Pilvipalvelussa CloudWatchia käytetään Lambda-funktioiden koodin sisäisien lokien kirjoittamiseen ja lukuun. Lambdassa voidaan tallentaa lokiviestejä suoraan CloudWatchiin, ja näitä viestejä voidaan virhetilanteissa lukea CloudWatch-palvelusta selaimella tai AWS-komentorivityökalulla.

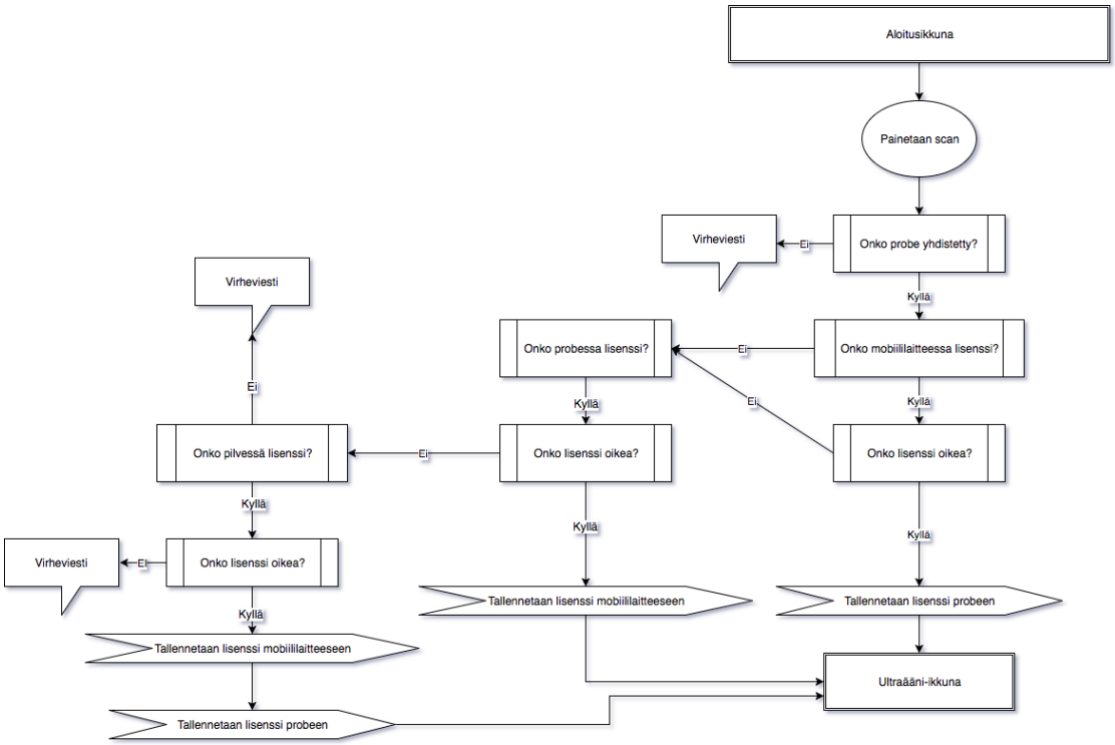
4.2.4 DynamoDB

DynamoDB on NoSQL-tyyppinen tietokanta, jonka vasteajat ovat pienet ja skaalautuvuus hyvä. DynamoDB tarjoaa muuttumatonta vasteaikaa, eli tietokanta saa olla minkä tahansa kokoinen ja se silti palauttaa vastauksia pyynnöille tasaisella vasteajalla. Koska AWS tarjoaa automaattisia varoituspalveluita jotka ilmoittavat, jos tietokanta paisuu liian suureksi verrattuna maksettuun tietokannan kokoon sekä mahdollisuuden antaa tietokannan skaalautua automaattisesti huolimatta hinnasta, tietokannan kokoa ei tarvitse seurata manuaalisesti. (3.)

Näistä syistä päätettiin käyttää DynamoDB-palvelua lisenssien tallennukseen. Koska DynamoDB on NoSQL-tyyppinen tietokanta, lisenssien muotoa voidaan muuttaa huolimatta tietokannassa olemassa olevien muiden lisenssien muodosta. Lisenssijärjestelmässä on kaksi DynamoDB-taulua, joista toinen on testiversio ja toinen tuotantoversio, jossa asiakkaiden lisenssit ovat.

4.3 Mobiilisovellus

Mobiilisovellus tukee iOS- ja Android-käyttöjärjestelmiä. Sovelluksen pohjana on Qt-kirjasto, jossa käytetään C++- ja QML-ohjelmointikieliä. Se hallinnoi lisenssin tallennuksen ultraäänilaitteeseen ja mobiililaitteeseen, virheviestien näytön käyttäjälle, lisenssien oikeuden tarkistuksen sekä lisenssin luvun ultraäänilaitteesta, mobiililaitteesta tai pilvestä. Mobiilisovelluksen lisenssiin liittyvässä käyttöliittymässä on kaksi osaa: aloitusikkuna joka tulee näkymään, kun sovellus käynnistetään sekä ultraääni-ikkuna, jossa näkyy ultraäänilaitteesta tuleva ultraäänikuva. Ultraääni-ikkunaan pääsee ainoastaan, jos sovellus on löytänyt jostain voimassaolevan lisenssin, ja ultraäänilaite on yhdistettynä mobiililaitteeseen. (Kuva 4.)



KUVA 6. Mobiilisovelluksen askeleet lisenssin haussa

5 JÄRJESTELMÄN TESTAUS

Järjestelmän testauksen kuvaus rajataan tässä opinnäytetyössä pilvipalvelun ja mobiilisovelluksen testauksen kuvaukseen.

5.1 Pilvipalvelu

Pilvipalvelu testattiin aluksi laittamalla koodimuutokset suoraan AWS-palveluun ja testaamalla perustoimivuutta. Tämä prosessi on kuitenkin liian hidas, eikä sitä pystytä turvallisuussyistä jatkamaan enää, kun palvelu julkaistaan asiakkaiden käyttöä varten. Tästä syystä pilvipalvelun testauksessa siirryttiin käyttämään automatisoitua paikallista testausprosessia. Paikallisessa testausprosessissa käytettiin Noden Mocha- ja DynamoDB-Local-moduulia.

Testit ajetaan terminaalista käyttämällä komentoa *npm test*, joka ajaa package.json-tiedostossa määritellyn test-komennon. Tässä tilanteessa määritelty test-komento ajaa vain Mochan.

5.1.1 Mocha

Käynnistyttyään Mocha ajaa oletuksena test-kansion sisällä olevat Javascript-tiedostot peräkkäin aakkosjärjestyksessä.

Javascript-tiedostoissa käytetään Mochan antamia funktioita describe, it, before ja after. Describe-funktiota käytetään testien osittamiseen, jolloin niille voidaan tulostaa oma otsikko ja tehdä omat rutiinit ennen ajoa (kuva 5). Kuvan esimerkissä ajetaan setupPreConditions-funktion ennen testien käynnistämistä käyttämällä Mochan before-funktiota.

```
describe('biim-endpoints', function () {
    before(function(done) {
        setupPreConditions(dynDB, DynamoDBLocal, dynamoPort, "Probes-DEV", function(res) {
            if(res) {
                done();
            } else {
                done(new Error("Failure to set up preconditions, check logs"));
            }
        });
    });
```

KUVA 7. Describe-funktio koodissa

Describe-komennon sisällä, before-komennon jälkeen, Mocha ajaa it-komentoja, jotka kuvaavat yksittäisiä testejä. It-funktiot antavat tulostaa komentoriville kuvauksen testistä sekä testin tuloksen. It-funktio ajetaan asynkronisena, jolloin se antaa funktion, jota kutsutaan, kun testi on ajettu, jolloin Mocha tietää testin tuloksen ja osaa jatkaa testien ajoa seuraavasta describe-alueesta (kuva 6).

Testien tulosten vertailuun käytetään Noden Assert-moduulin equal-komentoa, johon asetetaan kaksi muuttujaa. Ensimmäinen muuttuja on arvo, joka pitäisi tulla testistä, ja toinen on luku, joka testistä tuli. Mocha tarkistaa arvojen samanlaisuuden ja tulostaa sen mukaan komentoriville viestin.

```
        it('should return 200 OK, when the input data is a valid duplicate', function(done) {
            let event = {
                body: JSON.stringify(probe),
                requestContext: {
                    stage: "Dev"
                }
            };
            biim.handleProbesPut(event, context, docClient, function(err, response) {
                if(err) {
                    console.log(err);
                }
                assert.equal(200, response.statusCode);
                done(err);
            });
        });
    });
```

KUVA 8. It-funktio koodissa

5.1.2 DynamoDB-Local

DynamoDB-Local on Javascript-kirjasto, joka simuloi DynamoDB-palvelua. Kirjaston avulla voidaan luoda Javascript-olio, joka käyttäytyy kuten oikean DynamoDB:n rajapinta. Tätä kirjastoa

hyödyntäen voidaan ajaa testejä, jotka käyttävät DynamoDB:tä jopa paikallisessa ympäristössä ilman internet-yhteyttä tai oikeaa DynamoDB-palvelinta.

Käytännössä DynamoDB-tietokannan simulointi onnistuu luomalla uusi Javascript-olio, joka ottaa sisältönsä paikallisesta Javascript-kirjastosta. Tämä saadaan tehtyä käyttämällä Noden require-ominaisuutta, johon annetaan argumentiksi vain halutun moduulin nimi, jolloin Javascript-olio saa tämän moduulin sisällöt omakseen (kuva 7).

```
/* Database setup START */
let DynamoDBLocal = require('dynamodb-local');
let AWS = require("aws-sdk");
AWS.config.update({
    region: "us-east-2",
    endpoint: "http://localhost:8000"
});

let docClient = new AWS.DynamoDB.DocumentClient();
let dynamoPort = 8000;
let dynDB = new AWS.DynamoDB();
/* Database setup END */
```

KUVA 9. DynamoDB-olion luonti

DynamoDB-olion luonnin jälkeen kutsutaan sen Launch-funktio, jolle annetaan osoite, jolle DynamoDB-palvelin halutaan käynnistää. Koska testaamiseen käytetään paikallista ympäristöä, käynnistetään palvelin osoitteeseen http://localhost, porttiin 8000 (kuva 8).

```
function initializeDatabase(localDB, port, tableName, callback) {
    localDB.launch(port, null, ["-sharedDb"]).then(function() {
        callback(true);
    });
}
```

KUVA 10. DynamoDB-Local-palvelimen käynnistys

Testauksessa täytyy myös usein poistaa valmiiksi olemassa oleva taulu tietokannasta, jotta testien tulokset olisivat luotettavat. DynamoDB:n avulla taulut voidaan poistaa kutsumalla suoraan deleteTable-funktiota, jolle annetaan argumenttina vain taulun nimi (kuva 9).

```
function deleteExistingTables(database, tableName, callback) {
    database.deleteTable({
        TableName: tableName
    }, function(err, data) {
        if(err) {
            if(err.code === "ResourceNotFoundException") {
                callback(true);
            } else {
                console.log("deleteExistingTables Failed\n", err);
                callback(false);
            }
        } else {
            callback(true);
        }
    });
}
```

KUVA 11. DynamoDB-taulun poistaminen testiympäristössä

Taulujen poistamisen jälkeen kannattaa varmistaa, että DynamoDBssä on olemassa testeihin tarvittavat taulut. Tarvittavat taulut voidaan luoda käyttämällä DynamoDB:n createTable-funktiolla, jolle annetaan argumenttina Javascript-olio, jossa määritellään taulun nimi, sen perusavain ja mahdollisesti sen kirjoitus- ja lukunopeuden rajoitukset (kuva 10).

```
function initializeProbeTable(database, tableName, callback) {
    let tableStructure = {
        TableName: tableName,
        KeySchema: [
            { AttributeName: "probeId", KeyType: "HASH" }
        ],
        AttributeDefinitions: [
            { AttributeName: "probeId", AttributeType: "S" }
        ],
        ProvisionedThroughput: {
            ReadCapacityUnits: 5,
            WriteCapacityUnits: 5
        }
    };

    database.createTable(tableStructure, function(err, data) {
        if(err) {
            console.log("initializeProbeTable Failed\n", err);
            callback(false);
        } else {
            callback(true);
        }
    });
}
```

KUVA 12. DynamoDB-taulun luonti testiympäristössä

5.2 Mobiilisovellus

Mobiilisovellus testattiin kirjoittamalla kaikille mahdollisille lisenssin tiloille oma proseduuri, jota testaaja seuraa ja varmistaa, että sovellus toimii vaatimusten mukaisesti. Mobiilisovelluksen testausta ei ole vielä automatisoitu, joten testaaja käy jokaisen testin askeleet läpi ja kirjaa ylös testin onnistumisen tai epäonnistumisen.

Testiproseduuri on sarja toimintoja, jotka testaaja suorittaa varmistaakseen tietyn järjestelmän osan toiminnan, esimerkiksi lisenssin ultraäänilaitteessa tai mobiililaitteessa olemassaolon tarkistuksen (kuva 11).

Test Case ID:	GUITC-090		
Test Name:	GUI SW checks license validity and notifies the missing license		
Description:	This test procedure verifies that application checks probe license validity including integrity. This validation will be executed always when probe connection is established. If license is missing scanning is prohibited.		
Related GUISWR:	4.3.13.1, 4.3.13.5		
Prerequisite:	Probe and tablet without a valid license, valid license in cloud.		
Step	**Description**	**Expected Results**	**Results**
1	Establish a Wi-Fi connection to Probe	Verify that connection is done	Pass / Fail
2	Start the application	Verify that Home view is visible with "Probe License Not Found" pop-up.	Pass / Fail
3	Select Cancel	Verify that Home view is visible	Pass / Fail
4	Tap Go to Scan	Verify that "Probe License Not Found" pop-up is visible.	Pass / Fail
5	Tap Cancel	Verify that Home view is visible.	Pass / Fail
6	Go to Configuration, License Info	Verify that License Info is visible	Pass / Fail
7	Connect to Internet	Verify that Internet connection is done	Pass / Fail
8	Put application back to FG	Verify that License information is updated	Pass / Fail
9	Back to Home	Verify that Home view is visible	Pass / Fail
10	Establish a Wi-Fi connection to same Probe	Verify that connection is done	Pass / Fail
11	Select Go to Scan	Scan view is open	Pass / Fail

Test Configuration ID: _____

Overall Test Pass/Fail PASS | FAIL

Test Conductor Signature/Date: _____

KUVA 13. Lisenssin olemassaolon ultraäänilaitteessa tai mobiililaitteessa tarkistuksen testiproseduuri

6 LOPPUSANAT

Työn aiheena oli dokumentoida Biim Ultrasound Oy:n lisenssijärjestelmän suunnittelu, toteutus ja testaus. Järjestelmään kuului pilvipalvelu ja mobiilisovellus.

Lisenssijärjestelmällä oli alun perin tarkoitus olla tapa laskuttaa käyttäjiä tietyn väliajoin lukitsemalla mobiilisovelluksen skannausominaisuus, jos lisenssi vanhenee. Oli tarkoitus, että lisenssit ostettaisiin kerran vuodessa ja jaettaisiin kuukauden tai kahden pituisiin paloihin, joita olisi tallennettu mobiililaitteeseen ja ultraäänilaitteeseen. Näitä paloja pitäisi niiden ajan loppuessa käydä hakemassa pilvipalvelusta.

Kuitenkin toteutuksen aikana tuli päätös, että lisenssijärjestelmä piilotetaan ensimmäisissä ultraäänilaitteissa sillä tavalla, että ensimmäiset versiot lisenssistä tulevat kestämään loputtomasti. Tämä tarkoitti sitä, että mobiilisovelluksella täytyi tunnistaa loputtomat lisenssit ja pilvipalvelulla täytyi pystyä luomaan ja käsittelemään niitä.

Varsinainen järjestelmä tuli käyttöön syksyllä 2017. Ultraäänilaitteen tuotantolinjalla ajetaan komentorivisovellus, joka lähettää pilvipalvelulle ultraäänilaitteen tiedot ja pilvipalvelu luo niistä lisenssin ultraäänilaitteen tietojen mukaan. Tämän jälkeen ultraäänilaite tallentaa lisenssin sisäänsä, joten käyttäjän ei tarvitse koskaan ladata uutta lisenssiä.

Lisenssien käyttöä voidaan tulevaisuudessa kehittää vaikkapa myynnin seurausta avustamaan ja rajaamaan mobiilisovelluksen tai ultraäänilaitteen ominaisuuksia lisenssin perusteella.

Kokonaisuutena järjestelmä on toiminut kiitettävästi syksyn ajan, ja se vaikuttaa olevan tarpeeksi turvallinen ja stabiili asiakkaiden käyttöä varten.

LÄHTEET

1. What is AWS Lambda? Saatavissa: http://aws.amazon.com/lambda. Amazon Web Services, Inc. 2017. Hakupäivä 10.10.2017.
2. Amazon CloudWatch. Saatavissa: http://aws.amazon.com/cloudwatch/. Amazon Web Services, Inc. 2017. Hakupäivä 16.10.2017.
3. Amazon DynamoDB. Saatavissa: http://aws.amazon.com/dynamodb/. Amazon Web Services, Inc. 2017. Hakupäivä 25.10.2017