Mikael Tikka

Mobiilisovelluksen toteutus Unity-pelimoottorilla ja tietokantayhteydellä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

6.5.2024

Tiivistelmä

Tekijä:	Mikael Tikka
Otsikko:	Mobiilisovelluksen toteutus Unity-pelimoottorilla ja tietokantayhteydellä
Sivumäärä:	24 sivua
Aika:	6.5.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Osaamisaluepäällikkö Janne Salonen Perustaja Sami Vanni

---

Insinöörityön tavoitteena oli kehittää asiakasyritykselle sovelluskokonaisuus, johon sisältyi Unity-pelimoottorilla luotu mobiilisovellus Androidille ja iOS:lle, joka kommunikoi palvelimen kanssa, jossa oli API-rajapinta käytettäväksi sekä sovellukselle että verkkosivuille. Sovellus on suunnattu golfin harrastajille, sillä se pystyy sijoittamaan käyttäjän oikealle sijainnille LiDAR-mitattuihin karttoihin, minkä lisäksi sovelluksen avulla käyttäjä pystyy mittaamaan etäisyyksiä käyttäjän ja maamerkkien väliltä.

Insinöörityössä käytiin vastuuhenkilön kanssa läpi sovellukselta edellytettyjä vähittäisvaatimuksia, jotta sovelluskokonaisuus voitaisiin määritellä julkaisuvalmiiksi kokonaisuudeksi. Insinöörityössä tutkittiin myös, mikä olisi Unity-pelimoottorilla luodulle sovellukselle parhaiten sopivin API-rajapinta insinöörityön tekijän osaamisen ja yrityksen tarpeiden puitteissa. API-palvelimen toteutuksessa päädyttiin NERM-, Node.js-, Express-, React-, MongoDB-stackiin.

Mobiilisovelluksen kehityksessä keskeisiä haasteita olivat käyttäjän sijoittaminen mahdollisimman tarkasti karttaan, sovelluksen muuttaminen luonnostelevasta prototyypistä tuotantovalmiiksi ja julkaistavaksi sekä järjestelmän kehittäminen, joka mahdollistaa karttojen lataamisen internetin välityksellä. API-palvelimeen liittyvät olennaiset tehtävät olivat ratkaistava: miten saadaan monta golf-kenttää saman omistajan alle ja kuinka varmistaa, että uhkatoimija ei voisi muutetulla sovelluksella tehdä tietokantaan luvattomia muutoksia.

Sovelluskokonaisuutta testattiin paikan päällä, jossa sovellus todettiin toimivaksi, ja vastuuhenkilön kanssa saatiin ideoita, joilla voidaan parantaa mobiilisovellusta ja sen käyttökokemusta. Lopputuloksena on määritysten mukainen sovelluskokonaisuus, jonka mobiilisovellus saatiin julkaisuvalmiiseen vaiheeseen ja API-palvelin saatiin julkisesti saataville IP-osoitteen takaa.

Avainsanat: Unity, tietokanta, golf, LiDAR

Abstract

Author:	Mikael Tikka
Title:	Implementation of Unity-Based Mobile Application with Database Connection
Number of Pages:	24 pages
Date:	6 March 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Janne Salonen, Head of School Sami Vanni, Founder

---

The goal of the study was to develop a software package for a client company, including a mobile application for Android and iOS created with the Unity game engine. The application was to be able to communicate with a server hosting an API interface for use by both the application and a website. The application is aimed at golf enthusiasts, as it can position the user accurately on LiDAR-measured maps and allows the user to measure distances between themselves and landmarks.

The minimum requirements deemed necessary so that the software package could be considered complete and ready for publication were discussed with the commissioner. The study also investigated the most suitable API interface for the Unity-based application within the expertise of the author and the needs of the company. For the API server implementation, the choice was the NERM stack, which consists of Node.js, Express, React, and MongoDB.

The key challenges in the development of the mobile application included accurately positioning the user on the map, transitioning the application from a conceptual prototype to a production-ready and publishable state, and developing a system for loading maps over the internet. Essential tasks related to the API server included solving how to group multiple golf courses under the same owner and ensuring that malicious actors could not make unauthorized changes to the database using a modified application.

The software package was tested on-site, and was found to be functional, and ideas were discussed on how to further improve the mobile application and its user experience. The result is a software package compliant with the specifications, with the mobile application left in a ready for release state, and the API server is currently publicly accessible behind an IP address.

Keywords: Unity, database, golf, LiDAR

Sisällys

Lyhenteet

API: *Application Programming Interface*. Ohjelmointirajapinta, joka on komponenttien ja moduulien välinen raja ohjelmoitavassa järjestelmässä.

EC2: *Elastic Compute Cloud*. Amazon Web Servicesin pilvilaskenta-alusta, joka vuokraa skaalautuvia ja muokattavia virtuaalisia palvelimia.

Full stack: Pino, tai stack, joka on kokoelma järjestelmiä tai komponentteja, jotka tarvitaan kokonaisen alustan luomiseen niin, että muuta lisäohjelmistoa ei tarvita sovelluksen tukemiseen.

GPS: *Global Positioning System*. Satelliiteilla toimiva maailmanlaajuinen paikallistamisjärjestelmä.

I/O: *Input/Output*. Operaatiot tietojen siirtämiseen tietokonejärjestelmän tai sen ulkopuolisten laitteiden tai ohjelmien välillä.

LiDAR: *Light Detection And Ranging*. Menetelmä, jolla voidaan laservalopulsilla mitata etäisyyksiä esimerkiksi kaukokartoituslaitteesta.

NPM: *Node Package Manager*. Node.js:n paketinhallintajärjestelmä, jolla hallinnoidaan projektin koodiriippuvaisuuksia.

SDK: *Software Development Kit*. Ohjelmistokehityspaketti, joka sisältää työkalut, kirjastot, dokumentaation ja esimerkkikoodin, jolla kehittäjä voi luoda sovelluksia jollekin alustalle, käyttöjärjestelmälle tai laitteistolle.

SSH: *Secure Shell.* Verkkoprotokolla, jota käytetään turvalliseen etäyhteyteen ja tiedonsiirtoon tietokoneiden välillä internetin kautta.

1 Johdanto

Mobiilisovellusten suosio on kosketusnäytöllisten mobiililaitteiden suosion yhteydessä jatkunut kasvamistaan tasaisesti. 2023 mobiilisovellusmarkkinat tuottivat lähestulkoon biljoona dollaria, ja mobiililaitteiden jatkuvan yleistymisen ja helppokäyttöisyyden takia tämä kasvu ei toistaiseksi näyttäisi pysähtyvän [1]. Tämän takia monet yritykset ovat pyrkineetkin pääsemään osaksi tätä tuottoisaa ja kasvavaa markkinaosuutta. Mobiililaitteet ja niiden alati lisääntyvät ominaisuudet tarjoavat yrityksille myös mahdollisuuksia uusien innovaatioiden kehittämiseen – etenkin kun kuluttajilla alkaa olemaan taskussaan laskelmointivoimaltaan vuonna 2013 julkaistuja keskivertopöytätietokoneprosessoreja voimakkaampia älypuhelimia [2].

Insinöörityön tavoitteena on kehittää yritykselle toimiva sovelluskokonaisuus, johon kuuluu 3D-mobiilisovellus ja tietokanta, jonka kanssa mobiilisovellus pystyy kommunikoimaan. Yrityksellä on tarkoituksena myydä golfkentille tätä mobiilisovellusta, jolla heidän asiakkaansa voivat sijoittaa itsensä golfkentälle ja käyttää mobiilisovellusta helpottaakseen heidän pelaamistaan esimerkiksi sijoittamalla etäisyysmittoja mobiilisovelluksen virtuaalikentälle.

Mobiilisovellus tulee olemaan yrityksen päätuote sen valmistuessa, joten työn arvo on heille kaupallisesti merkittävä. Yritys on juuri perustettu alle kymmenen ihmisen startup, ja he pyrkivät luomaan 3D golf-sovelluksen, jolla on muita kilpailijoita tarkemmat LiDAR-kartat (Light Detection and Ranging) keskivertokuluttajan mobiililaitteisiin käytettäväksi.

Tässä insinöörityössä käydään ensiksi läpi aiheeseen liittyviä termejä, jota seuraa mobiilisovellus ja sen arkkitehtuuri kokonaisuutena. Tämän jälkeen käsitellään syvemmin itse mobiilisovellus, sen toteutus Unityllä ja tämän toteutuksen vaiheita. Seuraavana käydään läpi API-palvelin (Application Programming Interface) ja sen osat, jotka yhdessä muodostavat palvelimen backendin, joka syöttää sisältöä sekä mobiilisovellukselle että verkkosivuille. Viimeisenä käydään

läpi sovelluksen käytännön testausta golfkentällä ja sitä, mitä ideoita tästä testauksesta saatiin jatkokehitystä varten. Vaikka sovelluksen kehitykseen kuuluu myös frontendin kehitystä, tämä insinöörityö keskittyy itse mobiilisovellukseen ja backendiin, jonka kanssa mobiilisovellus kommunikoi.

2 Golf ja termistö

Tämän työn yhteydessä käsitellään golfiin liittyviä termejä, minkä vuoksi ne esitellään tässä luvussa. Golfkenttä on golfin harrastamiseen ja oheistoimintaan tarkoitettu alue, josta löytyy golfrata tai ratoja sekä yleensä driving range ja harjoitteluviheriö. Golfkenttä voi olla yksityisessä, kaupallisessa tai kunnallisessa omistuksessa, joita tässä työssä kutsutaan yleisesti (kentän) omistajiksi.

Golfrata on alue, jossa golfia pelataan. Golfradalla on yhdeksän tai 18 reikää. Tässä työssä käsiteltäessä mobiilisovelluksen ratojen latausjärjestelmää rata on yksikkö, joka ladataan aina kerralla ja on yksi kokonaisuus.

Reikä on kenttä, jossa on tiiauspaikka, väylä, puttausviheriö ja reikä. Reikään viitataan usein arkikielessä myös väylällä, mutta todellisuudessa väylä on reiän alue, jossa pelaajat pyrkivät pitämään lyöntinsä.

Driving range on lyöntiharjoitteluun tarkoitettu väylä, ja harjoitteluviheriö on puttausviheriö, joissa on monia reikiä harjoittelua varten. Näillä ei ole kuitenkaan työn kannalta merkitystä.

3 Sovelluskokonaisuus

Tässä luvussa käsitellään sovelluskokonaisuutta ja sen kehitystä yleisesti ja sitä, miten se toimii kokonaisuutena. Luvussa käydään ensin läpi sovelluskokonaisuuden vaatimusmäärittely ja mitkä kriteerit piti täyttää, jotta sovelluskokonaisuuden voitaisiin todeta olevan julkaisukuntoinen ensimmäinen versio. Tämän jälkeen käsitellään mobiilisovellusta ja API-palvelinta yleisesti, ennen kuin näitä käydään lävitse tarkemmin omissa luvuissaan.

3.1 Vaatimusmäärittely

Projektin vastuuhenkilön kanssa kahdenkeskisessä kokouksessa määriteltiin vaatimukset siitä, mitä haluttiin mobiiliohjelmalta ja mitä resursseja tämä vaatisi toimiakseen. Pitkän kaavan tähtäimessä oli kattava Unityllä luotu mobiilisovellus, jossa olisi monia erinäisiä ominaisuuksia, joilla parannetaan käyttäjän golf-pelikokemusta. Ensimmäisen version tavoitteena oli kuitenkin tuottaa sovellus, joka täyttäisi MVP:n vaatimukset.

MVP:ksi määriteltiin sovellus, jossa on LiDAR-mitattu kartta, pelaajanappula ja lippu. Pelaajanappulan alkusijainnin piti olla jokaisen reiän tiiauspaikalla ja sen piti sijoittaa itsensä kartalle kännykän GPS-paikannuksen (Global Positioning System) perusteella käyttäjän halutessa. Sovelluksesta piti saada esille tiedot reiästä sekä lippuetäisyydet tiiauspaikoilta ja mahdollisuus nähdä reiän ylilento-video. Mobiilisovelluksen haluttiin myös kommunikoivan palvelimen kanssa, josta muun muassa mobiilisovellus voisi sijoittaa reiän lipun palvelimelta saatavien koordinaattien avulla ja johon voisi jälkeenpäin kehittää verkkosivu-frontendin, jota kautta yrityksen asiakkaat voisivat muuttaa tietojansa, jotka on tallennettu palvelimelle. Tällöin mobiilisovelluksen luomisen lisäksi tavoitteisiin kuului myös selvittää, minkälainen full stack -palvelimelle kuuluisi luoda, jotta se pystyisi kommunikoimaan mobiilisovelluksen kanssa ja jolle voidaan jälkeenpäin luoda mahdollisimman helposti frontend-verkkoselaimelle.

3.2 Mobiilisovellus

Mobiilisovellus toteutettiin Unityllä yrityksen määräyksestä. Syitä tähän olivat muun muassa Unityn helppokäyttöisyys, kattava dokumentaatio ja se, kuinka helposti Unityllä pystyi kehittämään ja viemään sovelluksen sekä Androidille että iOS-laitteille.

Unity on järjestelmäriippumaton 3D-pelimoottori, jota kehittää Unity Technologies. Unity on eniten tunnettu sen helppokäyttöisyydestään, joka on tehnyt Unitystä erittäin suositun aloittavien yksittäisten indie-pelikehittäjien keskuudessa

[3], mutta Unityä käytetään myös yleisesti peliteollisuuden ulkopuolella [4, s. 17–18]. Unityn toiminta pohjautuu kohtaukseen (engl. Scene) eli säiliöön, jonka sisällä pelioliot (engl. Game Object) ja niiden hierarkiat ovat. Kohtaus onkin yleisesti tietty osuus koko pelistä tai sovelluksesta, kuten päävalikko, pelitaso tai välianimaatio. Peliolio on olennainen rakennuspalikka, joka muodostaa kohtausten sisällön, kuten kentän, valot, hahmot ja painikkeet. Peliolio muodostuu komponenteista (engl. Component). Tyhjällä pelioliolla on vain Transform-komponentti, joka antaa pelioliolle sijainti-, koko- ja suuntamuuttujat. Kehittäjä pystyy lisäämään kohtaukseen valmiita peliolioita kuten niitä, joita mainittiin esimerkeissä tai lisäämään tyhjän peliolion, johon hän voi lisätä komponentteja luodakseen tälle oman toiminnallisuuden. Komponentteja voi myös lisätä valmiisiin peliolioihin. Kehittäjällä on myös mahdollisuus lisätä skriptikomponentti, jolla hän voi C#-ohjelmointikielellä luoda täysin uutta toiminnallisuutta haluamallaan tavalla.

Yksi tärkeimmistä huomioitavista asioista mobiilisovelluksen kannalta muuhun kokonaisuuteen suhteutettuna oli LiDAR-kartat ja niiden koot. Täysikokoisina nämä kartat olivat liian suuria mobiilikäyttäjälle ladattavaksi internetin välityksellä, ja huolena oli myös, miten keskivertoinen mobiililaite suoriutuisi kentän avaamisessa sovelluksen sisällä. Unityssä kartat muutetaan Terrain-peliolioiksi, jotka puolestaan muodostuvat tiileistä. Korkeuskartoissa oli oletuksena ilman pienennystä 4097x4097 datapisteen tarkkuuden tiilet, joita tavanomaisesti oli 12–15 kappaletta kentässä. Tämä antoi pahimmillaan kentälle kooksi noin 700 MB. Tämän vuoksi vastuuhenkilölle suositeltiin korkeuskarttojen tarkkuuden pienentämistä tarpeeksi pieneksi, jotta kenttien koko olisi noin 100–200 MB. Tällöin koko olisi tarpeeksi pieni, jotta kenttä olisi ladattavissa mobiiliyhteyksillä inhimillisissä aikamääreissä eikä veisi liikaa tilaa mobiililaitteilta, vaikka yksittäinen käyttäjä lataisikin monta kenttää, mutta tarkkuus ei kärsisi liikaa.

3.3 API-palvelin

Ottaen huomioon yrityksen pienen koon palvelinratkaisua etsiessä oli tärkeää huomioida, että päivittäiskustannukset olisivat mahdollisimman pienet ja ratkaisun voisi tarpeen tullen skaalata suuremmaksi mahdollisimman nopeasti.

Valintaan vaikuttivat sekä ennestään löytyvä kokemus eri stackeista sekä kielen käyttöönoton sopivuus yritykselle. Pääharkinnan alaisena olivat Wordpress ja sen valmis LAMP (Linux, Apache, MySQL, PHP) stackiin pohjautuva ratkaisu, LAMP stackiin pohjautuvan ratkaisun kirjoittaminen alusta saakka sekä MERN (MongoDB, Express.js, React, Node.js). Harkinnassa oli myös .NET stack (ASP.NET, SQL, C#/F#), koska vaikka siitä ei ollut aiempaa kokemusta, C# oli puolestaan vahvasti hallinnassa. Näistä vaihtoehdoista Wordpress olisi tuonut valmiin mallipohjan, mutta tätä olisi joutunut muokkaamaan jonkin verran yrityksen käyttötarkoituksia varten – eritoten tietokantaa, johon olisi mitä varmemmin joutunut turvautumaan kolmansien osapuolien ylläpitämiin ratkaisuihin. Wordpressin mukana olisi myös tullut ominaisuuksia, joille ei ollut käyttöä, eli ne olisivat toimineet rasitteena. Vaihtoehtona Wordpressille oli LAMP stackiin pohjautuvan toteutuksen kirjoittaminen itse, mutta vaikka LAMP on tuttu ja perinteinen stack, siinä näkyy tietyssä määrin ikäkin, kun otetaan huomioon sen resurssien käyttö, jota käsitellään tässä insinöörityössä myöhemmin.

Lopulta päädyttiin MERNiin, koska se vastasi parhaiten yrityksen tarpeisiin. Tämä johtui Node.js:n epäestävästä I/O-arkkitehtuurista (Input/Output), joka sallii monen samanaikaisen yhteyden palvelimelle pienilläkin resursseilla.

Perinteisesti palvelimet luovat uusia säikeitä uusille kutsuille ja jäävät odottamaan esimerkiksi tietokannasta vastauksia varaten muistia palvelimelta. Node.js-arkkitehtuurin epäestävät, tapahtumavetoiset, asynkroniset I/O-kutsut toimivat puolestaan yhdessä säikeessä, jossa käytetään tapahtumasilmukka-suunnittelumallia (engl. event loop software pattern). Käyttäjien käyttämät toiminnat ohjelmassa luovat ohjelmissa rekisteröityjä tapahtumia, jotka palvelimen ainoa säie käsittelee asynkronisesti, eli kun säikeessä tulee vastaan joku I/O-

tehtävä, joka normaalisti pysäyttäisi säikeen. Node.js siirtyy suorittamaan muita tapahtumia samalla, kun se odottaa esimerkiksi tietokannalta vastausta tälle tapahtumalle. [5, s. 7–8.]

Tapahtumasilmukkasuunnittelumallin hyödyntäminen näkyy palvelimen suorituksessa, kun sitä vertaillaan Apache-PHP-ympäristöön, jossa luetaan isoa tekstitiedostoa samanaikaisesti monen simuloidun käyttäjän toimesta. Node.js suoriutuu käytännössä kaksinkertaisesti nopeammin kuin Apache-PHP [5, s. 53] ja kuusi kertaa nopeammin kuin Python-Web. Node.js myös pystyi suorittamaan selkeästi enemmän pyyntöjä samanaikaisesti kuin PHP ja Python-Web [6, s. 664–665].

4 Mobiilisovellus

Tässä luvussa käsitellään mobiilisovelluksen kehitystä tarkemmin, ja siinä vastaan tulleita ongelmia. Ensimmäiseksi käsitellään mobiilisovelluksen vaatimusmäärittely. Tämän jälkeen käydään läpi käyttäjän sijoittamista karttaan mobiililaitteen GPS:n avustuksella, jota seuraa sovelluksen muuttamista prototyypistä julkaisuvalmiiksi sovellukseksi, jonka voi ladata Google Playhin ja Apple Storeen ladattavaksi. Viimeisenä käsitellään mobiilisovelluksen latausjärjestelmää ja sen toimintaperiaatetta.

4.1 Vaatimusmäärittely

Työnantajan kanssa pidettiin useita kokouksia, joissa käytiin läpi tavoitteita, jotka olisi määrä saavuttaa, jotta päästäisiin mobiilisovelluksen kanssa haluttuun lopputulokseen. Kokouksissa myös suunniteltiin malli siitä, miltä sovellus näyttäisi käyttäjän näkökulmasta.

Käyttäjän avatessa sovelluksen hänelle pitäisi tulla ensin eteen päävalikko. Päävalikon päätarkoitus on valita golfrata, jota hän aikoo pelata, mutta päävalikosta olisi myös mahdollista hallinnoida käyttäjän mobiililaitteeseen ladattuja golfratoja, eli ladata lisää tai poistaa ladattuja ratoja. Avatessa golfradan

sovellus etenee suoraan pelinäkymään. Pelinäkymässä pelaajanappulan, joka näyttää käyttäjän sijainnin, on tarkoitus olla ensimmäisellä reiällä tiiauspaikalla, josta hän näkee loppuväylän. Käyttäjällä on mahdollisuus käyttää GPS-paikannin sijoittamaan pelaajanappula hänen nykyiselle oikealle sijainnilleen, vaihtaa näkymää yleisnäkymästä pelaajanappulan ensimmäisen persoonan näkökulmaan, luoda lisämerkkejä sekä vaihtaa reikää. Pelaajan näkymässä pitäisi olla myös maalinappula, jolla käyttäjä voi tähdätä tulevaa lyöntiänsä. Pelinäkymä näkyy havainnollistettuna kuvassa 1. Sovellukseen kuuluisi myös radanhallinnointinäkymä, jossa kenttätyöntekijät voivat vaihtaa sovelluksen kautta reikien lippujen sijaintia, joka yleensä vaihdetaan päivittäin golfkentillä.

Kuva 1. Projektin ensimmäinen kuvakaappaus mobiilisovelluksesta pelinäkymässä.

Kuten kuvassa 1 nähdään, mobiilisovelluksen pelinäkymä antaa käyttäjälle paljon tietoa senhetkisestä reiästä. Punainen nappula esittää pelaajan senhetkistä sijaintia, kun taas keltainen nappula on sovelluksen antama maalinappula. Pelinäkymän ylärivillä näkyvät reiän tiedot: mones reikä on kyseessä, reiän ihannelyöntimäärä ja sen tasoiteluku. Ylävasemmalla on puolestaan maali- ja pelaajanappulan välinen korkeusero, kun taas yläoikealla on etäisyysmittari, jonka kohdetta voi vaihtaa näiden kahden käyttöliittymäelementin välistä löytyvästä vihreästä näppäimestä.

4.2 Käyttäjän sijoittaminen karttaan

Käyttäjän sijoittaminen karttaan oli yksi keskeisimmistä ongelmista. Työnantaja antoi ohjeistukseksi selvittää, olisiko MapBox vai Google parempi vaihtoehto.

MapBox tarjoaisi 25000 kuukausittaiselle käyttäjälle ilmaisen pääsyn mobiilille tarkoitettuun SDK:hon (Software Development Kit), johon sisältyisi käyttäjän paikallistaminen [7]. Sen mukana tulisi kuitenkin myös kartta, joka olisi ylimääräinen kustannus, koska sille ei ollut tarvetta omien LiDAR-mitattujen karttojen takia.

Selvitystyön ollessa käynnissä Googlella oli tarjolla Google Maps Platform gaming services -palvelu. Heillä oli käytössä Daily Active Users (DAU) -tyyppinen hinnoittelu, jossa jokaisesta yksittäisestä käyttäjästä laskutettiin riippumatta siitä, kuinka paljon kyseinen käyttäjä käytti palvelua yhden päivän aikana. Ensimmäinen hintaporras oli 10 dollaria tuhatta DAU:ta kohti, mutta he tarjosivat ensimmäiset 200 dollaria ilmaiseksi. Palvelu kuitenkin lakkautettiin lokakuussa 2021. [8.]

Tiedossa oli kuitenkin myös, että puhelimen omaa GPS:ää oli mahdollista käyttää suoraan, ja Unityn oma mobiilikohtainen kirjasto tuki tätä. Tämän myötä suunniteltiin järjestelmä, jossa valmiiksi saatavilla oleviin karttoihin sijoitettiin manuaalisesti ankkurit kartan luode- ja kaakkoispäätyihin. Nämä ankkurit ovat pisteitä kartalla, joiden vastaava sijainti oikeassa maailmassa voidaan

mahdollisimman tarkasti löytää, ja antaa niille vastaavat koordinaatit datapisteiksi. Näitä koordinaatteja käyttäen pystyttiin laskemaan, missä kohtaa käyttäjä on suhteellisesti näiden kahden ankkurin välillä, ja sijoittamaan pelaajanappula vastaavaan kohtaan virtuaalikartalla, kuten näkyy esimerkkikoodissa 1.

```
float x = anchor1.transform.position.x + ((Input.loca-
tion.lastData.longitude - location1.coordinates.longitude) /
longScale) * xScale;
float z = anchor1.transform.position.z + ((Input.location.lastData.la-
titude - location1.coordinates.latitude) / latScale) * zScale;
float y;
Ray ray = new Ray(new Vector3(x, 100, z), new Vector3(0, -1000, 0));
if (Physics.Raycast(ray, out RaycastHit hit, Mathf.Infinity, layer-
Mask))
{
          y = hit.point.y + 0.5f;
}else{
          y = 0;
}
transform.position = new Vector3(x, y, z);
```

Esimerkkikoodi 1. Osa koodia, joka päivittää pelaajanappulan sijaintia.

Esimerkkikoodin 1 Input.location.lastData on osa Unityn sisäistä kirjastoa, jolla päästään mobiililaitteen viimeisimpään GPS-sijaintiin käsiksi. Jos katsotaan esimerkiksi, miten saadaan pelinappulan tuleva X-koordinaatti, käyttäjän puhelimen viimeisen sijainnin longitudin desimaalista vähennetään nollapisteenä toimiva, luoteisen ankkurin longitudidesimaali, ja jaetaan koko kentän pituudella longitudissa (longScale). Näin saadaan prosenttiluku sijainnista, jota käytetään kertomaan koko kentän koko pelikoordinaateissa (xScale). Tämän tulos lisätään luoteisen ankkurin pelikoordinaatteihin, jolloin pelaaja sijoittuu pituussuunnassa. Sama tehdään Z-koordinaatille latitudilla leveyssuuntaisen sijainnin saamiseksi. Lopuksi mitataan, kuinka korkealla maa on kyseisessä kohdassa, jotta pelaajanappula voidaan sijoittaa oikealla korkeudelle. Ratkaisu esitettiin työnantajalle ja se todettiin toimivaksi verrattuna kolmannen osapuolen palvelun käyttämiseen.

Myöhemmin sijoittamisessa ilmaantui virhe, kun GPS:n toimintaa testattiin emulaattorilla, jolle syötettiin käsin GPS-tiedot sen varmistamiseksi, että pelaajanappula sijoitetaan oikealle sijainnille. Kuten kuvassa 2 on havainnoitu, jos ankkurien välille kuvitellaan akseli, joka halkaisee radan, mitä pitemmälle käyttäjä kulkee akselista, sitä enemmän hänen paikkansa vääristyy. Vääristyksessä oli kuitenkin havaittavissa kuvio. Pelaajan oikean sijainnin ja sovelluksen ilmoittaman väärän sijainnin välinen akseli oli aina rinnakkainen ankkurien välisen akselin kanssa. Jos pelaaja oli ankkurien välisen akselin pohjoispuolella, pelaajan sijainti vääristyi luoteeseen. Päinvastaisesti jos pelaaja oli akselin eteläpuolella, pelaajan sijainti vääristyi kaakkoa kohti.

Kuva 2. Havainnollistava kuva, jonka esittelin vastuuhenkilölle. Vääristymät ovat liioiteltuja havainnoinnin vuoksi. Vihreät pisteet esittävät, missä käyttäjän oikea sijainti on, ja punaisten nuolen kärki on, missä sovellus luuli käyttäjän olevan.

Ongelmaa tutkittiin, mutta siihen ei löytynyt käytetyillä termeillä mitään vastaavaa. Ongelma päätyttiin ratkaisemaan muuttamalla pelaajanappulan sijaintia sen oikeaa sijaintiansa kohti riippuen sen etäisyydestä ankkurien välisestä akselista, kuten näkyy esimerkkikoodissa 2. Tämä toimii vain väliaikaisena ratkaisuna, koska sen alla piileviä syitä ei voitu ratkaista, mutta mikä jouduttiin valitsemaan, jotta projekti voitaisiin viimeistellä suunnitelluissa aikamääreissä.

```
nearestPointInLine.transform.position = NearestPointOnFiniteLine(an-
chor1.transform.position, anchor2.transform.position, transform.posi-
tion);
DistanceToLine = Vector3.Distance(transform.position, nearestPointIn-
Line.transform.position);
longFromAnchor = Input.location.lastData.longitude - location1.coordi-
nates.longitude;
latFromAnchor = Input.location.lastData.latitude - location1.coordi-
nates.latitude;
percentLong = longFromAnchor / longScale;
percentLat = latFromAnchor / latScale;
correctionX = DistanceToLine * errorCorrectionPercentX;
correctionZ = DistanceToLine * errorCorrectionPercentZ;
if (nearestPointInLine.transform.position.z < transform.position.z)
{
           correctionX *= -1;
           correctionZ *= -1;
}
float x = anchor1.transform.position.x + percentLong * xScale - cor-
rectionX;
float z = anchor1.transform.position.z + percentLat * zScale + correc-
tionZ;
```

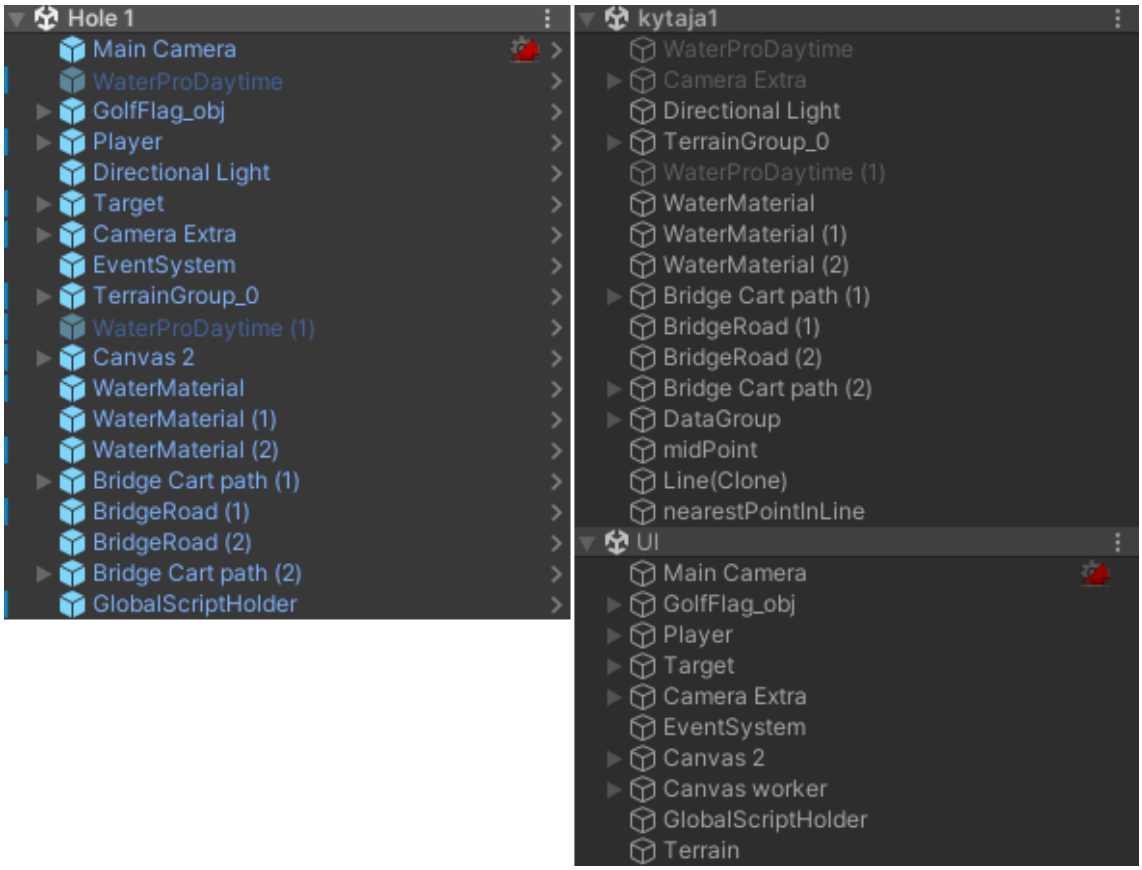
Esimerkkikoodi 2. Vastaava koodi kuin esimerkkikoodi 1:ssä, mutta korjauksien kanssa.

Esimerkkikoodi 2 laskee, kuinka pitkällä käyttäjän lähin piste ankkurien väliseltä akselilta on, josta saadaan distanceToLine-muuttuja. Tätä muuttujaa käytetään vaikuttamaan correctionX- ja correctionZ-muuttujiin riippuen pelaajan etäisyydestä akselista. Nämä muutokset muutetaan käänteisiksi, jos käyttäjä on akselin eteläpuolella.

4.3 Prototyypistä tuotantovalmiiksi

Kun Unity-projekti oli vaiheessa, jossa suurimmaksi osaksi päästiin asetettuihin tavoitteisiin, eli mobiilisovellukselle oli olemassa valmis pelinäkymä, siirryttiin seuraavaan vaiheeseen. Tämä vaihe oli mobiilisovelluksen muuttaminen niin, että siinä pystyi pelaamaan muitakin golfratoja kuin kehitysvaiheessa käytettyä rataa. Tätä varten piti luoda alkuvalikko, mutta piti myös erottaa kentän ja sovelluksen toimintaperiaatteet toisistaan. Sovellus oli prototyyppivaiheessa luotu tämä mielessä pitäen, joten suurimmaksi osaksi tämä osa oli peliolioiden siirtämistä erikseen eri kohtauksiin ja virheviestien selvittelyyn niiden tullessa vastaan. Virheviestit olivat tavanomaisia, kun skriptit eivät löytäneet viittauksiaan (Reference not found, null exception ja niin edelleen).

Kuvasta 3 nähdään, kuinka pelioliot siirrettiin uuteen kohtaukseen nimeltä UI, joka nimestään huolimatta sisälsi käyttöliittymäelementtien lisäksi muitakin peliolioita. Kaikki pelioliot, joita voitiin käyttää uudestaan tai tarvittiin muissakin golfradoissa, siirrettiin tähän kohtaukseen. Näistä peliolioista ylivoimaisesti suurin osa oli toiminnallisia peliolioita, ja reiän kohtaukseen jäi ainoastaan reiän 3D-pelioliot ja DataGroup-peliolio, jossa on kaikki kyseiseen reikään liittyvä data.



Kuva 3. Kuvakaappaukset hierarkioista ennen ja jälkeen. Hole 1 nimettiin uudelleen kytaja1:ksi.

Erittelyn jälkeen sovelluksen toimintaa muutettiin niin, että valitessa ladattavan kentän päävalikosta sovellus lataa kyseisen radan, ja tämän radan ensimmäisen reiän. Tämän jälkeen ensimmäinen asia, jonka reikä tekee oman kohtauksen alustuksen jälkeen, on UI-kohtauksen lataaminen päälle. GlobalScriptHolder on julkinen peliolio, jota kautta muut pelioliot ja niiden skriptit pääsevät käsiksi muihin peliolioihin. Suurimmaksi osaksi UI-kohtauksen skriptit kytkeytyivät toisiin UI-kohtauksen skripteihin, mutta niihin muutamaan skriptiin, jotka tarvitsivat pääsyn kenttä kohtauksen Terrain-olioon (nimeltä TerrainGroup_0) piti muokata GlobalScriptHolderia niin, että UI:n alustuksen jälkeen GlobalScriptHolder etsi TerrainGroup_0:n nimeltään ja asetti tämän itsellensä julkiseksi referenssiksi, jota kautta UI-kohtauksen pelioliot pääsivät Terrain-olioon käsiksi.

4.4 Kenttien latausjärjestelmä

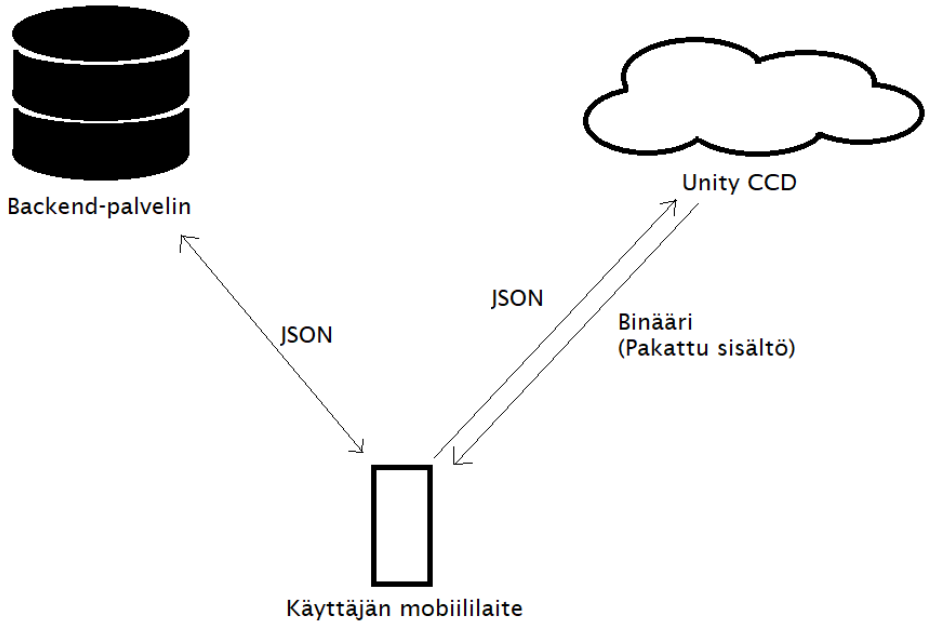
Seuraavaksi selvitettiin, millä tavalla radat olisivat parhaiten saavutettavissa käyttäjille. Tavoitteena olisi, että radat olisivat saavutettavissa loppukäyttäjien mobiililaitteille internetin välityksellä jostakin palvelimesta tai sisällönjakeluverkosta ladattavaksi. Tämä toteutui Unityn Addressables- ja Cloud Content Delivery (CCD) -järjestelmillä, jotka on luotu toisiaan varten, vaikkakin järjestelmät toimivat myös yhdistettyinä muihin kolmansien osapuolien järjestelmiin.

Addressables on Unityn sisäinen sisällönhallintajärjestelmä, joka auttaa dynaamisen sisällön lataamisen kanssa. Järjestelmä pystyy lataamaan sisältöä mistä tahansa sijainnista pakkaamalla sisällön paketteihin sovelluksen rakentamisen yhteydessä. Kun sisältö – kuten skripti tai 3D-malli – merkitään Addressables-järjestelmässä, siitä tulee Addressable. Addressables-järjestelmä antaa tälle sisällölle osoitteen, jota pystyy käyttämään mistä vain sovelluksen sisällä. Järjestelmä paikantaa paikallisesti tai ulkoisesta sijainnista pakatun sisällön ja sen tarvitsemat riippuvuudet ja palauttaa sen.

Unity CCD on pilvijärjestelmä, jonne voi ladata Unityn sisältöä, joka on tätä kautta Unityllä luotujen sovellusten käytettävissä internetin välityksellä. Lataamalla Addressables-järjestelmällä luodut paketit CCD:hen, on kaikki tarvittava sisältö saatavana etänä.

Latausjärjestelmää varten luotiin seuraavanlainen arkkitehtuuri: käyttäjä lisää golfradan mobiililaitteeseensa syöttämällä julkisesti saatavilla olevan koodin sovellukseen. Kuten kuvassa 4 havainnoidaan, sovellus ottaa yhteyttä API-palvelimelle, ja jos se löytää käyttäjän antamalle koodille vastaavuuden, palvelin palauttaa vastauksena kentän nimen ja Addressable-osoitteen. Sovellus pyytää Addressables-järjestelmältä golfradan esiladattavaksi, jolloin järjestelmä ottaa yhteyttä CCD:hen ja alkaa lataamaan rataa sovellukseen. Samalla sovellus tallentaa mobiililaitteen muistiin kentän nimen ja kyseiseen kenttään liittyvän Addressable-osoitteen. Kun sovellus käynnistetään, aloittaa käyttäjä päävalikosta, joka luo näppäimen jokaisesta tallennetusta nimestä. Painaessa

näppäintä sovellus pyytää Addressables-järjestelmältä näppäimeen yhdistettyä osoitetta esiladatun golfradan ensimmäisen reiän avattavaksi.



Kuva 4. Kaavakuva mobiililaitteen yhteydenotoista latauksen yhteydessä.

Myöhemmin huomattiin perustavanlaatuinen ongelma Addressables-järjestelemässä. Jos rakentamisen yhteydessä Addressableksi merkityssä sisällössä on tapahtunut muutos, Addressables-järjestelmän luoma paketti saa uuden nimen. Näin Addressables-järjestelmän kuuluukin toimia, mutta Addressable-paketti saattaa saada näennäisesti täysin satunnaisesti uuden nimen, vaikka sisällössä itsessään ei ole tapahtunut muutosta. Tämä rikkoo sovelluksen determinismiä ja saattaa aiheuttaa sovelluksen vanhoissa versioissa ongelman, jossa Addressables-järjestelmä alkaa lataamaan sisältöä uusiksi, vaikka se löytyisikin jo käyttäjän mobiililaitteesta, koska se löytyy eri nimellä. [9.] Tämä on varsin haitallista etenkin sovelluksissa, jossa on huomattavia latauskokoja, mutta valitettavasti tämä vaikuttaisi olevan tarkoitettu seuraus, joka tapahtuu aina, jos sovellus rakennetaan eri tietokoneella.

5 API-palvelin

Mobiilisovelluksen toimintaa varten tarvitaan palvelin, josta sovellus saa ajantasaiset tiedot, kuten tämänpäiväisten lippujen sijainnin reiällä. Tehtävänantoon kuului selvittää sopivin backend-tyyppi palvelimelle, joka täyttäisi sekä sovelluksen vaatimukset että voisi toimia myös nettisivujen backendinä. Tätä tarkoitusta varten päätettiin tehdä REST API -palvelin, jonka kanssa sekä nettisivut että sovellus pystyisivät kommunikoimaan. Tämän vuoksi ainoa palvelinta koskeva vaatimus oli, että se suoriutuisi mahdollisimman hyvin mahdollisimman pienillä resursseilla.

5.1 Full stack

API-rajapinta toteutettiin Node.js:llä, Expressillä ja MongoDB:llä. Vaikka Node.js:ää ja sen luomaa käyttöympäristöä on käsitelty jo aikaisemmassa luvussa, keskitytään tässä luvussa tarkemmin siihen, miten Node.js toimii muiden stackin osien kanssa. Lisäksi käsitellään muita API-rajapinnan osia.

Node.js

Node.js on JavaScript-ajonaikainen ympäristö, joka perustuu Chromen V8 JavaScript -moottoriin. Se mahdollistaa JavaScriptin suorittamisen verkkoselaimen ulkopuolella, yleensä palvelimessa. Node.js käyttää tapahtumapohjaista, epäblokkaavaa I/O-mallia, mikä tekee siitä kevyen ja tehokkaan skaalautuville verkkosovelluksille [5, s.1; 6, s. 661]. Full stack -web-sovelluksessa Node.js toimii palvelimen puolen suoritusympäristönä, mikä hoitaa myös HTTP-pyyntöjen käsittelyä ja dynaamisen sisällön tarjoamista käyttäjille.

Express

Express on Node.js-web-sovelluskehys, joka laajentaa Node.js:n ominaisuuksia verkkosovellusten ja mobiilisovellusten rakentamiseen. Se yksinkertaistaa palvelinpuolen logiikan rakentamista ja HTTP-pyyntöjen käsittelyä tarjoamalla yksinkertaisen ohjelmointirajapinnan kehittäjille. Express tarjoaa ominaisuuksia

kuten istuntojen hallinnan, middleware-tuen, mallimoottorien integroinnin ja virheidenkäsittelyn. Tämän palvelimen toteutuksessa Express lähinnä käsittelee reititystä, tapahtumakirjausta, middlewarea ja vuorovaikutusta tietokantojen kanssa.

MongoDB

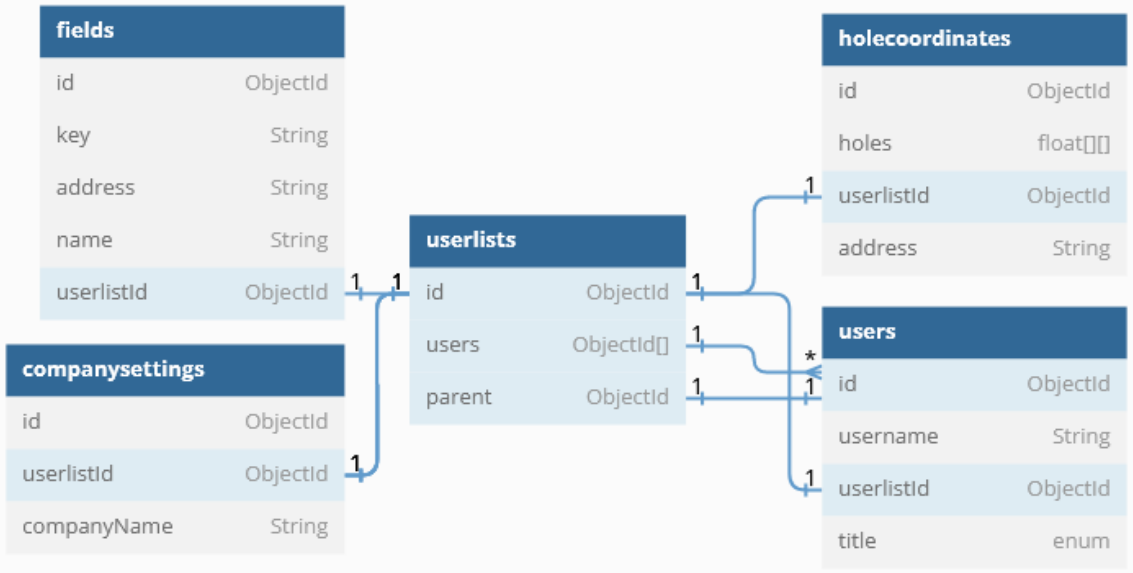
MongoDB on NoSQL-tietokanta, joka tallentaa tiedot joustavina JSON-dokumentteina. Se tunnetaan skaalautuvuudestaan, joustavuudestaan ja suorituskyvystään erityisesti sovelluksissa, joissa on suuria määriä rakenteettomia tai puolirakenteellisia tietoja [10, s. 91]. MongoDB tallentaa tiedot kokoelmiin sarakkeiden sijaan ja käyttää joustavaa skeemaa, joka mahdollistaa erilaisten datatyyppien tallentamisen samaan kokoelmaan. MongoDB:lle sopivia käyttökohteita ovat muun muassa sisällönhallintajärjestelmät, reaaliaikainen analytiikka, välimuistina toimiminen ja käyttäjätietojen hallinta [11, s. viii-ix]. Full stack -web-sovelluksessa MongoDB toimii tietokantakerroksena tallentaen ja halliten sovelluksen tietoja.

5.2 Toteutus

Palvelimen lähdekoodi ladattiin GitHubiin, jota kautta se ladattiin Amazon Web Servicesin (AWS) EC2-mikroinstanssille (Elastic Compute Cloud) ja otettiin käyttöön tälle asennetun NPM:n avulla (Node Package Manager). Instanssille asennettiin myös MongoDB:n paikallinen asennus. AWS:n käyttöoikeusryhmien (engl. security group) ja elastisen IP:n (engl. elastic IP) konfiguroinnin jälkeen palvelin oli saatavilla julkisesti IP-osoitteen takana. Käyttöoikeusryhmät konfiguroitiin ottamaan vastaan kaikki portti 3001:een tulevat pyynnöt, jotta palvelin pystyy vastaanottamaan pyyntöjä sen ohjelmointirajapintaan. Muuten kaikki pyynnöt evättiin pois lukien ne, joita tarvittiin palvelimen hallinnointiin kehittäjien toimesta, kuten SSH (Secure Shell). Vastuuhenkilölle suositeltiin verkkotunnuksen liittämistä vastaisuudessa palvelimeen, jotta sovelluksen julkaisuversio voisi käyttää staattista osoitetta ottaakseen yhteyttä palvelimeen.

5.3 Arkkitehtuuri

Vastuuhenkilö tiedotti, että usein yksi golfkenttä jakautui moniin golfratoihin, ja monella kentällä saattoi olla myös sama omistaja, joten tietokannan arkkitehtuuri piti suunnitella niin, että samalla tunnuksella pystyi hallinnoimaan monta kenttää tai rataa. Tämän vuoksi päädyttiin ratkaisuun, jossa kaiken keskellä oli käyttäjälistat-taulu (engl. userlists), kuten näkyy kuvassa 5. Käyttäjälistojen sisällä on id, joka oletuksena löytyy jokaisesta MongoDB:n sisällä olevasta kokoelmasta: parent, joka viittaa yhteen käyttäjään, joka on kentän omistaja ja jolla on kaikki oikeudet, ja users, jonka sisällä on lista kaikista käyttäjistä ja jotka koostuvat kenttien työntekijöistä. Kaikilla kenttiin liittyviltä tauluilta löytyy userlistId, joka viittaa tämän kyseisen kentän käyttäjälistaan, jossa on kaikki ihmiset, joilla pitäisi olla tämän kentän hallinnointioikeudet riippuen heidän roolistaan (title). Tällä tavalla jokainen kenttä saman omistajan alla voi viitata samaan ID:hen, ja hakemalla haluamastaan taulustaan listan kaikista kentistä, joissa on oikea userlistId, saa listan kaikista kentistä, joilla on sama omistaja.



Kuva 5. Kuva taulujen välisistä relaatioista.

Fields-taulun sisällä on luvussa 3.4 mainitut tiedot, jotka mobiilisovellus hakee tietokannasta, eli address ja name, jos käyttäjän syöttämä koodi vastaa key-

kenttää. Holecoordinates sisältää reikien lipun tämänhetkisen sijainnin holes-2D-taulukon sisältä liukulukuina. Companysettings sisältää kaikki golfkenttää tai kenttiä hallinnoivan yrityksen liittyvät tiedot. Taulukko sisältää companyNamen lisäksi osoitetiedot, jotka jätettiin kuvasta 5 pois selkeyden vuoksi.

5.4 Tietoturvallisuus

Koska mobiilisovellus on luotu Unityllä, joka on erittäin suosittu pelimoottori, sen purkaminen sen alkuelementteihin on verrattain yksinkertaista, kun otetaan huomioon tätä tarkoitusta varten luotujen ohjelmien määrän. Tämä tarkoittaisi sitä, että uhkatoimija voisi purkaa ohjelmiston ja koota sen uudestaan niin, että hänellä on suora pääsy aikaisemmin mainittuun kenttätyöntekijälle tarkoitettuun näkymään. Näin hänellä olisi mahdollista tehdä ilkivaltaa vaihtelemalla lippujen paikkoja mielivaltaisesti. Vaikka vahingon määrä tästä teoreettisesta ilkivallasta olisi verrattain pientä, oli silti hyvä selvittää, miten tämä voitaisiin estää.

Palvelimeen tehtiin muutoksia niin, että jokaisen muuttavan pyynnön mukana tuli olla mukana tunniste (engl. token). Kenttätyöntekijä saa palvelimelta mobiilisovellukseen oikeilla tunnisteilla kirjautuessaan tunnisteen, jonka sovellus tallentaa muistiinsa. Kun kenttätyöntekijä lähettää muutoksensa palvelimelle, sovellus automaattisesti sisällyttää tunnisteen pyynnön mukana. Jos tälle tunnisteelle löytyy vastaavuus palvelimelta, muutokset hyväksytään ja toteutetaan tietokannassa. Tällä tavalla, vaikka uhkatoimija antaisi itsellensä pääsyn kenttätyöntekijänäkymään muutetulla sovelluksella, hän ei pystyisi vaikuttamaan palvelimeen, koska palvelin ei ole koskaan luonut ja antanut tälle muutetulle sovellukselle tunnistetta.

6 Kenttätestaus

Sovelluskokonaisuutta testattiin vastuuhenkilön kanssa lataamalla mobiilisovellus molempien puhelimeen ja sopimalla tapaaminen Kytäjän golfkentällä. Käytössä oli Android- ja iOS-käyttöjärjestelmän omaavat puhelimet, jotta sovellusta pystyttiin kokeilemaan molemmilla alustoilla. Tarkoituksena oli simuloida

golfpelaajia, jotka kiertävät kaikki 18 väylää läpi, jotta voitaisiin saada mahdollisimman hyvä kuva siitä, miten sovelluksessa valmistuneet ominaisuudet toimivat, ja ideoida mahdollisia uusia ominaisuuksia, joita sovellus voisi kaivata käyttäjien näkökulmasta.

Kentän omistajan kanssa sovittiin testistä, jota varten lainattiin golfautoa. Mobiilisovelluksilla ladattiin Kytäjän rata paikan päällä. Lataus onnistui molemmilla alustoilla ja rataa lähdettiin kiertämään läpi reikä kerrallaan ikään kuin golferää pelaten.

Sovellus toimi testauksen aikana odotusten mukaisesti, eikä mitään ongelmia tullut vastaan. Sovelluksen GPS-paikoitusominaisuus myös toimi hyvin paikan päällä, ja verrattaessa ympäristöön havaittiin, että sovellus sijoitti pelaajanappulan aika tarkalleen sinne, missä puhelin oli todellisuudessa kentällä, vaikkakin hieman paikannussovelluksille ominaisesti hyppelehtien. Eriäväisyyksiä verrattuna emulaattorilla tehtyyn testiin kehitysvaiheessa, jossa pelaajanappula ei juuri ollenkaan poikennut GPS:llä asetetusta sijainnista, aiheutui mitä luultavammin puhelimien GPS:n epätarkkuuksista.

Kenttätestauksen aikana saatiin kolme kehitysideaa: 1) jos GPS on laitettu päälle, sen pitäisi pysyä päällä reikää vaihdettaessa, 2) GPS:ää tulisi tarkentaa ohjelmoimalla ja 3) jos GPS on päällä ja ohjelma näkee, että käyttäjän sijainti on vaihtunut jo uuteen reikään, sovelluksen pitäisi vaihtaa kohtaus tähän vastaavaan reikään.

GPS:n tarkkuutta parannettiin seuraavanlaisesti. Esimerkkikoodi 1:ssä käydyssä Input.location.lastData:ssa on longitudin ja latitudin lisäksi horizontalAccuracy-niminen muuttuja. Tämä muuttuja kertoo, kuinka tarkka viimeisin sijainti on metrien tarkkuudella. Ohjelmaa voidaan muuttaa niin, että sen sijaan, että pelaajanappula sijoitetaan suoraan viimeisimmän saadun longitudin ja latitudin perusteella, ohjelma luo ympyrän käyttäen horizontalAccuracyä säteenä, ja luo tämän ympyrän sisälle esimerkiksi 20 senttimetrin välin näkymättömiä peliolioita, jotka ovat käyttäjän mahdollisia sijainteja. Näiden peliolioiden

keskimääräinen sijainti lasketaan ja tätä arvoa käytetään sijoittamaan pelinappula. Pelaajanappulan sijainti tarkentuu, kun saadaan seuraava lastData, ja ohjelma luo taas ympyrän, mutta tällä kertaa poistaa kaikki pelioliot, jotka ovat tämän uuden ympyrän ulkopuolella. Tällöin kahden ympyrän ollessa limittäin, peliolioiden, eli käyttäjän mahdollisten sijaintien, määrä vähenee ja sijainti tarkentuu entisestään. Jos yhtään pelioliota ei jää jäljelle, voidaan olettaa, että käyttäjä on vaihtanut paikkaa, skripti aloittaa taas alusta luomalla pelioliot ympyrän sisälle.

7 Yhteenveto

Insinöörityön tavoitteena oli saada aikaiseksi sovelluskokonaisuus, jossa olisi Unityllä luotu mobiilisovellus Androidille ja iOS:lle, jota tukisi API-palvelin, johon voidaan päivittää ajankohtaiset tiedot. Sovelluksen tuli muun muassa sijoittaa käyttäjä mahdollisimman tarkasti kartalle GPS:n perusteella sekä pystyä lataamaan lisää kenttiä sovellukseen internetin välityksellä.

Insinöörityön aikana määriteltiin sovelluksen vähittäisvaatimukset ja kehitettiin sovelluskokonaisuus nämä huomioon ottaen. Tavoitteisiin päästiin, lopputuloksena on mobiilisovelluksen ensimmäinen julkaisukelpoinen versio, jossa on MVP:ksi määritetyt ominaisuudet ja sitä tukeva API-palvelin.

Sovelluksen käytännön testaus onnistui ilman mitään ongelmia, ja vastuuhenkilön kanssa saatiin ideoita, joilla sovellusta voitaisiin kehittää ja käyttäjän käyttökokemusta voitaisiin parantaa sekä pienillä että isoilla muutoksilla. Näihin lukeutuivat GPS:n pitäminen päällä reikien välillä, paikantamisen tarkennus ohjelmoimalla ja reiän vaihtaminen automaattisesti, jos käyttäjän sijainti on jo toisella reiällä.

Insinöörityötä voi käyttää ohjeena, miten Unityä ja MERN-stackia voidaan käyttää luodakseen sovelluskokonaisuuden, jossa Unityssä luotu sovellus on yhteydessä internetin yhteydessä olevaan REST API -tietokantaan ja jota kautta Unity-sovellus saa dynaamisesti luotua sisältöä sovelluksen käytettäväksi.

Työtä pystyy helposti soveltamaan Unityllä luotuihin muiden alustojen (kuten tietokoneiden) sovelluksiin tai käyttämällä muuta pilvitallennustarjoajaa kuin Unity CCD:tä.

Työn perusteella seuraavaksi selvitettäviin asioihin lukeutuu muun muassa selvitys, mikä aiheutti käyttäjän paikannuksessa olevan virheen, kun siirryttiin kauemmas karttojen ankkureista, ja miten pystytään minimoimaan tai kiertämään Addressables-järjestelmässä osoittautunutta virhettä, jos tälle ei näiltä näkymin olla Unityn puolesta tekemässä mitään.

Sovelluksen kehitystä voidaan jatkaa lisäämällä siihen lisäominaisuuksia, jotka parantavat käyttäjän käyttökokemusta. Esimerkki tällaisesta ominaisuudesta on tuloskortti, johon käyttäjät voivat kirjata heidän tuloksiaan ja seurata heidän edistymistään pitkällä aikavälillä. Toinen varteenotettava kehityskohde voisi olla mahdollisuus tilata ruokaa reiälle, jos tämä on mahdollisuutena golfkentällä, kun hyödynnetään käyttäjän puhelimen GPS-ominaisuutta, jotta kenttätyöntekijät voivat mahdollisimman nopeasti toimittaa ruoan.

Lähteet

1 Radić, Drako. 2024. App Usage Statistics 2024: Downloads, Revenue, Popularity. Verkkoaineisto. SerpWatch. <https://serpwatch.io/blog/app-usage-statistics/>. Luettu 1.4.2024.

2 Qualcomm SM8250 vs Intel Core i7-4770R @ 3.20GHz. 2023. Verkkoaineisto. PassMark Software. <https://www.cpubenchmark.net/compare/5404vs2137/Qualcomm-SM8250-vs-Intel-i7-4770R>. Luettu 8.4.2024.

3 Axon, Samuel. 2016. Unity at 10: For better—or worse—game development has never been easier. Verkkoaineisto. Ars Technica. <https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/>. 27.9.2016. Luettu 8.4.2024.

4 Haas, John. 2014. A History of the Unity Game Engine. Verkkoaineisto. CORE. <https://core.ac.uk/works/61695138>. 6.3.2014. Luettu 8.4.2024.

5 Chhetri, Nimesh. 2016. A Comparative Analysis of Node.js (Server-Side JavaScript). Verkkoaineisto. St. Cloud State University. <https://repository.stcloudstate.edu/csit_etds/5/>. 3.2016. Luettu 25.3.2024.

6 Lei, Kai; Ma, Yining; Tan, Zhi. 2015. Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js. Verkkoaineisto. IEEE. <https://ieeexplore.ieee.org/abstract/document/7023652>. 29.1.2015. Luettu 25.3.2024.

7 Mapbox pricing. Verkkoaineisto. Mapbox. <https://www.mapbox.com/pricing#maps>. Luettu 15.3.2024.

8 Google Maps Platform gaming services Usage and Billing. Google. <https://developers.google.com/maps/documentation/gaming/usage_billing>. Luettu 15.3.2024.

9 All Addressable bundles get rebuilt with different names after "Update a Previous Build". 2020. Verkkoaineisto. Unity. <https://forum.unity.com/threads/all-addressable-bundles-get-rebuilt-with-different-names-after-update-a-previous-build.897803/>. 25.5.2020. Luettu 12.9.2023.

10 Chauhan, Divya; Bansal, Kartik. 2017. Using the Advantages of NOSQL: A Case Study on MongoDB. Verkkoaineisto. ResearchGate 2/2017. <https://www.researchgate.net/profile/Divya-Chauhan-

4/publication/349110376_Using_the_Advantages_of_NOSQL_A_Case_Study_on_MongoDB/links/6021154d92851c4ed5580298/Using-the-Advantages-of-NOSQL-A-Case-Study-on-MongoDB.pdf>. 2.2017. Luettu 16.4.2024.

11 Copeland, Rick. 2013. MongoDB Applied Design Patterns. O'Reilly. Luettu 28.4.2024.