TAMPEREEN
AMMATTIKORKEAKOULU

Mobiilipelin palvelinsovelluksen arkkitehtuurin suunnittelu ja toteutus

MurderApp

Jarkko Virtanen

Opinnäytetyö
Toukokuu 2016
Tietojenkäsittely
Pelituotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Pelituotanto

VIRTANEN, JARKKO:
Mobiilipelin palvelinsovelluksen arkkitehtuurin suunnittelu ja toteutus
MurderApp
Opinnäytetyö 38 sivua, joista liitteitä 1 sivu
Toukokuu 2016

Tämän opinnäytetyön tarkoituksena oli toteuttaa palvelinsovellus, joka kommunikoi mobiililaitteessa toimivan MurderApp-mobiilisovelluksen käyttöliittymäsovelluksen kanssa. Tavoitteena oli tutkia toteutuksessa käytettäviä teknologioita ja selvittää, onko projekti mahdollinen ja järkevä toteuttaa kyseisillä teknologioilla. Tarkoituksena oli myös kuvata käytettävien teknologioiden liittämiset projektiin sekä projektin rakenne.

Opinnäyteyön tuloksena toteutettiin teknologioiden valinta ja näiden avulla palvelinsovelluksen toteutus. Työssä tarvittavat teknologiat todettiin toimiviksi ja tämän kokoiseen työhön soveltuviksi. Teknologiat eivät tuottaneet kehitysvaiheessa kustannuksia ja ovat tarvittaessa laajennettavissa.

Palvelinsovelluksen vaatimuksina olivat mobiilisovelluksen avulla pelattavan pelin koordinointi, pelaajien väliset viestinnät sekä tietokannan ylläpito. Palvelinsovellus välittää myös sähköpostiviestit, joita mobiilisovellus lähettää silloin, kun pelaaja kutsutaan peliin. Tiukkoja pelin sisäisen viestinnän reaaliaikaisuuden vaatimuksia ei ollut, sillä peli ei ole nopeatempoinen.

Työ onnistui hyvin ottaen huomioon lähtökohtaisen teoreettisen tiedon ja käytännön taidon vajauden palvelinsovelluksen toteutuksesta. Työssä käytettäviä teknologioita olivat Ruby-ohjelmointikieli, Rails-kirjasto, Git-versionhallintajärjestelmä, Heroku-ajoalusta sekä SendGrid SMTP -palvelu. Palvelin- ja käyttöliittymäsovellusten välinen kommunikointi tapahtui HTTP-rajapinnan ylitse REST-konventiota käyttäen.

Asiasanat: palvelinsovellus, mobiilisovellus, ruby, rails, git, heroku, sendgrid

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Game Development

VIRTANEN JARKKO:
Planning and Implementation of Server Application Architecture for a Mobile Game MurderApp

Bachelor's thesis 38 pages, appendices 1 page
May 2016

The purpose of this thesis was to implement a server application that communicates with the user interface application of the mobile application MurderApp. The objective was to study the technologies used in the implementation and to find out whether it would be possible and sensible to use these technologies in the project. The purpose was also to describe the structure of the project and the integration of the technologies used in the project.

The thesis resulted in a successful selection of technologies and implementation of a server application using these technologies. The technologies needed for the implementation proved to be functional and compatible with the project of this size, and they did not produce development costs and are expandable if necessary.

The server application was required to coordinate the game played via a mobile application, and to take care of the communications between the players and maintenance of the database. The server application's job is to forward emails that the mobile application sends when a player is invited to the game. There is no requirement for real-time internal communication because the game is not fast-paced.

The project was a success, taking into account the author's lack of existing theoretical knowledge and practical skill in implementing a server application. The technologies used in the project were Ruby programming language, Rails library, Git version control system, Heroku PaaS service and SendGrid SMTP service. Communication between the server and user interface applications is done through a HTTP interface, following the REST convention.

Key words: server application, mobile application, ruby, rails, git, heroku, sendgrid

SISÄLLYS

ERITYISSANASTO tai LYHENTEET JA TERMIT (valitse jompikumpi)

SMTP	Simple Mail Transfer Protocol on protokolla, jota käytetään viestien välittämiseen sähköpostipalvelujen välillä.
HTTP	Hypertext Transfer Protocol on protokolla, jota selaimet ja palvelimet käyttävät tiedonsiirtoon.
npm-komennot	Node Package Manager-komennot ovat komentoja joita käytetään Node.js komentokehotteessa
unix-komennot	Ovat käyttöjärjestelmän komentokehotteessa toimivia komentoja
virtuaalikone	Virtuaalikone on ohjelmallisesti tuotettu tietokone, jossa voidaan suorittaa ohjelmia aidon koneen tavalla.
REST	on HTTP-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen.
JSON	JavaScript Object Notation on yksinkertainen vakioitu tiedonvälityksen tiedostomuoto.
PaaS	Platform as a Service tarkoittaa palvelualustan ulkoistamista.

1 JOHDANTO

Tämän työn tarkoituksena oli toteuttaa palvelinsovellus, joka kommunikoi mobiililaitteessa toimivan MurderApp-mobiilisovelluksen käyttöliittymäsovelluksen kanssa. Tarkoituksena oli myös kuvata projektissa käytettävien teknologioiden liittämiset sekä projektin rakenne. Tavoitteena oli tutkia toteutuksessa käytettäviä teknologioita ja selvittää onko projekti mahdollinen ja järkevä toteuttaa kyseisillä teknologioilla.

Työn tuloksena toteutettiin teknologioiden valinta ja näiden avulla palvelinsovelluksen toteutus. Teknologioiden valinta perustui tiedonhakuun sekä yrityksen aikaisempaan tietoon palvelinsovelluksen toteutusmahdollisuuksista. Palvelinsovelluksen vaatimuksina olivat mobiilisovelluksen avulla pelattavan pelin koordinointi, pelaajien väliset viestinnät sekä tietokannan ylläpito. Palvelinsovellus välittää myös sähköpostiviestit, joita mobiilisovellus lähettää silloin, kun pelaaja kutsutaan peliin. Tiukkoja pelin sisäisen viestinnän reaaliaikaisuuden vaatimuksia ei ollut, sillä peli ei ole nopeatempoinen. Palvelinsovelluksen täytyy olla kuitenkin toimintavalmiina siten, että kaikille pelaajille saadaan haluamilla ajanhetkillä tarkastettua tilanteet sekä synkronoitua pelin kulku. Palvelinsovelluksessa olevassa tietokannassa tarvitsee olla luotujen pelitapahtumien tiedot, pelitapahtumaan kutsutut käyttäjät sekä käyttäjiin sidotut hahmot. Hahmon kautta löydetään pelaajan tiedot.

Opinnäytetyönä toteutettava palvelinsovellus tulee osaksi MurderApp-mobiilisovelluksen kokonaisuutta, jossa sen päätavoite on tukea käyttäjälle näkyvää ja käyttäjän kanssa kommunikoivaa käyttöliittymäsovellusta. Palvelinsovellus ja käyttöliittymäsovellus toimivat omina kokonaisuuksinaan. Sovellusten välinen keskustelu kulkee HTTP-rajapinnan kautta verkon välityksellä. Sovelluksessa käytettävät teknologiat kartoitetaan ja valitaan siten, että vastaavan kokoisessa ja lähes samoja määrittelyjä vaativa projekti olisi mahdollinen toteuttaa kyseisillä teknologioilla. Teknologioiden valintakriteereinä ovat myös, että kehitysvaiheessa ei synny kustannuksia ja ne ovat tarvittaessa laajennettavissa.

2 YRITYS

MurderApp-projekti toteutetaan kahden tamperelaisen yrityksen Tuoni Studiot Oy:n ja Kaski Studios:n yhteistyönä. Tuoni Studiot Oy on Tampereelta käsin maanlaajuisesti toimiva elämystoimisto, joka toteuttaa erilaisia elämystapahtumia yksityisille, yhteisöille ja yrityksille. Elämystapahtumia on useita ja ne voidaan järjestää suoraan sellaisinaan, hieman muokattuina tai täysin personoituna asiakkaan maun mukaan. Suurin osa Tuoni Studioiden järjestämistä elämystapahtumista ovat niin sanottuna murhamysteereitä. Kaski Studios on Tamperelainen pelistudio, joka on aloittanut toimintansa vuonna 2015. Kaski Studios työllistää 3 henkilöä, lisäksi yritys työllistää projektiluontoisesti muun muassa opiskelijoita. MurderApp-mobiilisovellus toteutetaan Kaski Studios:n sekä Tuoni Studioiden yhteistyönä. Projekti on Kaski Studios:n pilottituote sekä -julkaisu. (Pirinen 2015.)

Yritysten yhteistyö alkoi, kun Kaski Studios:n Janne Pirinen osallistui Tuoni Studioiden järjestämään murhamysteeriin keväällä 2015, missä idea sovelluksesta syntyi. Kaski Studios ehdotti ja ideoi toteutusta Tuoni Studioille kesäkuussa samana vuonna. Päätös kehityshankkeen aloittamisesta MurderApp-mobiilisovelluksen suhteen tehtiin heinäkuussa 2015. Kehitystyöstä päätettiin, että Tuoni Studiot vastaavat sovelluksen graafisesta ulkoasusta ja teemasta sekä tuottavat ja käsikirjoittavat eksklusiivisen murhamysteerin mobiilisovellusta varten. Kaski Studios vastaa sovelluksen kehitystyöstä sekä implementaatiosta monialustatuella iOS, android ja Windows Phone käyttöjärjestelmille. Kaski Studios vastaa sovelluskehityksestä yhteistyössä Tampereen Ammattikorkeakoulun tietojenkäsittelyn pelituotannon harjoittelijoiden kanssa. Sovelluksen tulisi pystyä tarjoamaan mahdollisuus kenelle tahansa järjestää pienimuotoinen murhamysteeripeli täysin ilman aiempaa kokemusta tämän tyyppisen pelin pelaamisesta, järjestämisestä tai hallinnoimisesta. (Pirinen 2015.)

Peli tukee nykyistä käytössä olevaa konseptia mysteeritapahtumista, joita Tuoni Studiot ovat järjestäneet. Yleiskuva tällaisista mysteereistä on mysteerin alustaminen ja hahmoidentiteettien jako osallistujille. Hahmoilla on käytössä kykyjä ja esineitä, joilla he voivat kerätä tietoa muista hahmoista ja näin selvittää tai salata mysteeriä. Esimerkkitilanteena työpaikan järjestämät pikkujoulut, joissa ohjelmaan kuuluu kyseinen tapahtuma. Tapahtuma voidaan järjestää omana ohjelmanumeronaan tai yhdistää esimerkiksi ruokailun yhteyteen. Jokaiselle osallistujalle jaetaan hahmon identiteettikortti, jossa lukee hahmon tiedot ja omat salaisuudet. Osallistujalle annetaan myös hahmolle kuuluvat kyvyt ja esineet, joita osallistuja voi käyttää muihin henkilöihin. Tapahtuma toteutuu näyttelemällä eli

eläytymällä omaan roolihahmoon, konkreettisesti keskustelemalla ja näyttelemällä muiden osallistujien kanssa. Osallistujat voivat käyttää kykyjä ja esineitä selvittääkseen tai salatakseen kyseisen mysteerin. (Pirinen 2015.)

3 PALVELINSOVELLUS ELI BACKEND

3.1 ESITTELY

Tämä opinnäytetyönä suoritettu projekti koostuu kahdesta osasta, joita ovat käyttäjän kanssa graafisen käyttöliittymän kautta kommunikoiva käyttöliittymäsovellus sekä käyttäjältä piilossa toimiva palvelinsovellus. Palvelinsovellus on sovellus, joka koostuu tietokannasta sekä sovelluksen toiminnasta vastaavista ohjelmoiduista osista. Tätä sovellusta ylläpitää sovellusten ajoalustana toimiva pilvipalvelu Heroku. Heroku huolehtii palvelinsovellusta ylläpitävästä infrastruktuurista sekä sovelluksen suorittamisesta. Palvelinsovelluksen tärkeimpiä tehtäviä ovat käyttöliittymäsovelluksen kanssa kommunikointi, tietokannan hallinta, sähköpostipalvelun käyttäminen. Käyttöliittymäsovellus lähettää palvelinsovellukselle pyyntöjä, jotka voivat olla tietokannan tietojen päivittämistä, lisäämistä, näyttämistä, poistamista tai sähköpostikutsun lähettämistä. Näiden pyyntöjen suorittamiseen palvelinsovelluksen tehtävänä on hallita tietokannan rakennetta, sisältöä sekä tiedonsiirtoa. Sähköpostipalvelua palvelinsovellus käyttää lähettämään sähköpostikutsun henkilöille, jotka kutsutaan pelitapahtumaan. Palvelinsovellus hallitsee taustalla toimivia palveluita ja niiden yhteyttä käyttöliittymäsovellukseen.

Käyttöliittymäsovellus on käyttäjälle näkyvä projektin sovellus, jonka kanssa käyttäjä keskustelee graafisen käyttöliittymän avulla. Käyttäjän ja käyttöliittymäsovelluksen keskustelun pohjalta sovellus lähettää palvelinsovellukselle pyyntöjä tarvitsemistaan tiedoista. Palvelinsovellus hakee tiedot tietokannasta, muokkaa niitä tarvittaessa ja palauttaa nämä tiedot oikeassa muodossa käyttöliittymäsovellukselle. Käyttöliittymäsovellus tuo graafisen käyttöliittymän kautta tiedot käyttäjän nähtäväksi. Käyttöliittymäsovelluksen ja palvelinsovelluksen yhteistyötä kuvaavana esimerkkinä toimii hyvin ravintola. Ravintolassa käyttöliittymäsovellus on ravintolan ruokasali, missä asiakas (sovelluksen käyttäjä) keskustelee käyttöliittymäsovelluksen (tarjoilija) kanssa. Tarjoilija toimittaa ruokasalista asiakkaan tilauksen keittiön (palvelinsovelluksen) puolelle. Keittiö on piilossa asiakkaalta, mutta paikka, josta asiakkaan tilausta varten haetaan ainekset varastosta (tietokannasta). Keittiössä valmistetaan ainekset tarvittavaan muotoon ja toimitetaan tarjoilijan kautta annos asiakkaalle, joka vastaa hänen tilaustaan.

3.2 PALVELINSOVELLUKSEN VAATIMUKSET

Tässä projektissa palvelinsovelluksen vaatimukset jakautuvat toiminnallisiin ja ei-toiminnallisiin vaatimuksiin. Toiminnallisia vaatimuksia ovat esimerkiksi tietokannan ylläpito, käyttöliittymäsovelluksen kanssa kommunikointi, sähköpostipalvelun hallinta ja sovelluksen suorituskyky. Ei-toiminnallisia vaatimuksia ovat muun muassa teknologioiden laajennettavuus myöhemmin. Alla on tarkempi jaottelu toiminnallisista ja ei-toiminnallisista vaatimuksista. (Pirinen 2015.)

- Toiminnalliset vaatimukset
    - tietokannan ylläpito
        - tietokannan rakenne
        - tiedonsiirto
        - tietojen muokkaus
    - käyttöliittymäsovelluksen kanssa kommunikointi
        - käyttöliittymäsovelluksen lähettämien pyyntöjen mahdollistaminen ja määrittäminen
        - käyttöliittymäsovelluksen pyyntöihin vastaaminen
    - sähköpostipalvelun hallinta
        - SMTP – välityspalvelun hyödyntäminen
        - sähköpostikutsujen lähettäminen
        - sähköpostikutsun muokkaaminen vastaanottajakohtaiseksi
    - sovelluksen suorituskyky
        - kykyä ylläpitää kymmeniä pelitapahtumia viikossa
        - kykyä suoriutua useista samanaikaisista pelitapahtumista
- Ei-toiminnalliset vaatimukset
    - teknologiat eivät saa kehitysvaiheessa tuoda kustannuksia
    - käytettävät teknologiat tarvittaessa laajennettavissa

Palvelinsovelluksen tulee mahdollistaa ja määrittää kuinka käyttöliittymäsovellus voi kommunikoida palvelinsovelluksen kanssa, miten käyttöliittymäsovellus saa yhteyden palvelinsovellukseen ja mitä toimintoja käyttöliittymäsovellus voi pyytää palvelinsovellusta tekemään. Näitä toimintoja ovat tietokannan tietojen muokkaus sekä tiedonsiirto sovellusten välillä. Käyttöliittymäsovellus voi myös pyytää palvelinsovellusta lähettämään sähköpostikutsun käyttäjille, jotka kutsutaan pelitapahtumaan.

Sähköpostikutsun lähettämiseen palvelinsovellus hyödyntää SMTP-välityspalvelua. Palvelinsovellus muokkaa sähköpostikutsun vastaanottajakohtaiseksi. Palvelinsovelluksen täytyy myös kyetä ylläpitämään tietokannassaan kymmeniä pelitapahtumia viikossa sekä suoriutua useasta samanaikaisesti pelattavasta pelitapahtumasta. Palvelinsovelluksessa käytettävät teknologiat valitaan siten, että kehitysvaiheessa ei synny kustannuksia ja teknologiat ovat tarvittaessa laajennettavissa. (Pirinen 2015.)

4 VALITUT TOTEUTUSTEKNOLOGIAT

4.1 CLOUD9IDE

Cloud9IDE on tässä projektissa käytettävä työympäristö. Työympäristön tehtävä on helpottaa projektin toteutusta luomalla yhteydet teknologioiden välille, hallita projektin tiedostoja ja antaa visuaalinen työpöytä projektin tekemistä varten. Työympäristöjä on useita ja käytettävä työympäristö valitaan usein joko ohjelmoijan itsensä mieltymyksen mukaan tai useamman tekijän projektissa voidaan sopia yhteinen käytettävä työympäristö. Samaakin projektia on mahdollista työstää eri työympäristöissä. Tärkeintä on, että muut teknologiat pystyvät toimimaan yhdessä, joita työympäristössä käytetään. (Kehoe 2015; Starnes 2015.)

Tämän projektin kannalta tärkeimmät Cloud9IDE:n ominaisuudet ovat sen yhteensopivuus muihin teknologioihin ja tekijän oma mieltymys. Työympäristö ei myöskään maksa mitään, kun projekti ei ylitä 512MB muistin ja 1GB levytilan rajaa. Cloud9IDE tarjoaa pilvessä toimivan työympäristön, joten projektitiedostot pysyvät tallessa vaikka työpistettä ja konetta vaihtaisi. Pilvipalvelussa toimiva työympäristö on käytännössä virtuaalikone, jolla on vähintään 512MB muistia ja 1GB levytilaa. Työympäristö on laajennettavissa, jos projekti niin vaatii, rahallista summaa vastaan. Cloud9IDE:ssä on sisäänrakennettu komentokehote, joka tukee npm- sekä unix-komentoja. Muita ominaisuuksia ovat selainpohjainen tekstieditori sekä tiedostojen hallinta ja selaimen kautta toimiva projektin esikatselu. Cloud9IDE ei tue varsinaisesti tabletteja tai puhelimia, sillä se vaatii suuremman näytön sekä näppäimistön. Se toimii kaikilla käyttöjärjestelmillä, jotka tukevat verkkoselaimen käyttöä ja internet-yhteyden. (Kehoe 2015; Starnes 2015.)

4.2 RUBY ON RAILS

Ruby on Rails nimi tulee Rails-kirjaston ja Ruby-ohjelmointikielen yhdistelmästä. Rails-kirjastokin on kirjoitettu juuri Ruby-ohjelmointikielellä. Ruby on Rails:sta käytetään usein lyhennystä Rails, sillä se on vakiinnuttanut paikkaansa Rubyllä tuotettavien web-sovellusten ja sivustojen teossa. Rails on pääsääntöisesti web-pohjaisten sovellusten kehitykseen tarkoitettu ohjelmistokehys. Rails julkaistiin joulukuussa 2005 ja se on kehittynyt tehokkaasti ja sen käyttö on yleistynyt dynaamisten verkkosovelluksien ja sivustojen toteutuksessa. Rails on täysin avointa lähdekoodia, joten sen käyttäminen on vapaata mihin tahansa tarkoitukseen. Rails myös noudattaa MIT-lisenssiä, joka

mahdollistaa Rails:n käytön ilmaiseksi myös kaupallisiin tarkoituksiin. Eli lataaminen, käyttö ja julkaisu eivät maksa mitään. Rails mahdollistaa koodiosuuksien generoimisen, joka helpottaa ja nopeuttaa esimerkiksi tietokannan taulukoiden sekä html-sivustojen luomista. Esimerkiksi scaffold-komennolla voidaan luoda yhdellä komennolla tietokannan taulu ja sen sisältämät tiedot. Rails osaa luoda tämän komennon pohjalta html-sivuston, jonka kautta pääsee lisäämään, lukemaan, päivittämään ja poistamaan taulun tietoja selaimessa html-sivuston niin sanotun graafisen käyttöliittymän kautta. Rails alustaa tällaisen scaffold- komennolla generoidun taulun siten että, tyypillisten tietueiden kuten luomisen, lukemisen, päivittämisen ja tuhoamisen mahdollistavat metodit ovat automaattisesti määriteltyjä. Tämä mahdollistaa tietojen hakemisen myös Rails:n ulkopuolelta, kuten tabletissa tai puhelimessa olevasta MurderApp-sovelluksesta. Rails tukee myös sähköpostin lähettämisen HTML-pohjaisen mailto-komennon avulla sekä SMTP-palvelua käyttäen. (Hartl 2015, 55–60; Kehoe 2013, Bigg, R., Katz, Y., Klabnik, S. & Skinner, R. 2015.)

Ruby on Rails käyttää model-view-controller (MVC) ohjelmistoarkkitehtuuria, joka jakaa sovelluksen kolmeen loogiseen osaan: malli(model), näkymä(view) ja ohjain(controller). Käytännössä tämä tarkoittaa, että Rails-sovelluksissa käyttöliittymän komponentit eli näkymät, toiminnalliset osat eli ohjaimet sekä käytettävä data eli mallit, on määritelty toisistaan erillään. Mallit määrittelevät millaista tietoa ohjelma käyttää ja miten annettuja tietoja voidaan muutella. Malleihin määritellään myös eri toiminnot, joita tiedolta vaaditaan, ja miten eri mallit liittyvät toisiinsa. Näkymät määrittelevät sovelluksen käyttöliittymän, jotka Rails-ohjelmissa ovat pääsääntöisesti selaimessa toimivia html-tiedostoja. Näihin html-tiedostoihin on kirjoitettu Ruby-kielellä toiminallisuuksia, joiden avulla ohjelma käsittelee käyttäjän toiminnat siten, että saadaan haluttu lopputulos. Ohjaimet vastaanottavat tulevat syötteet ja käyttävät malleja tehdäkseen niistä käytettävää dataa. Lisäksi ohjaimet vastaavat mallien mukaisen tiedon toimittamisesta näkymille sekä yleisesti erilaisten parametrien käsittelystä. Käyttäjä keskustelee näkymän kanssa, josta ohjain kerää käyttäjän valinnat sekä toiminnot ja mallien avulla määrittelee tarvittavan datan ja palauttaa halutun lopputuloksen takaisin näkymän kautta käyttäjälle. MVC arkkitehtuurin ansioista suunnittelu ja ylläpito yksinkertaistuvat, sillä osat eivät ole riippuvaisia toisistaan. (Kasurinen 2011, 149–151.)

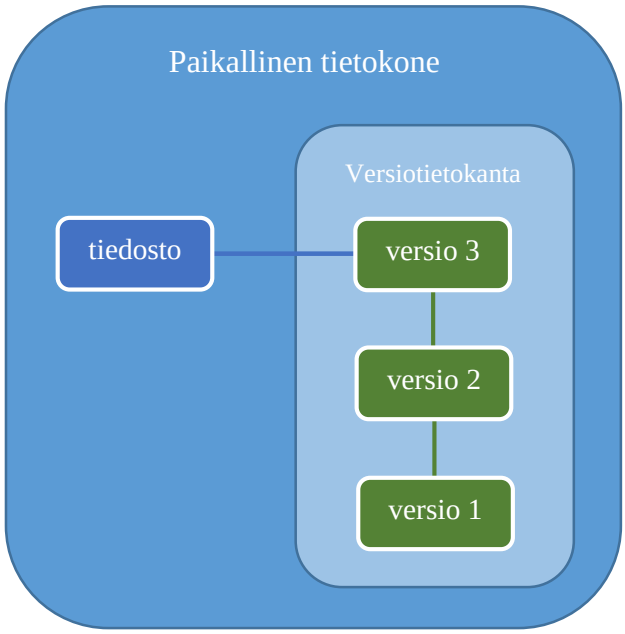
4.3 GIT

Nykyään kukaan ei aloita suurempaa projektia ilman jonkinlaista suunnitelmaa varmuuskopioinnista tai versionhallinnasta. Projekteissa on tiedostoja ja dataa niin paljon, että se voi hukkua

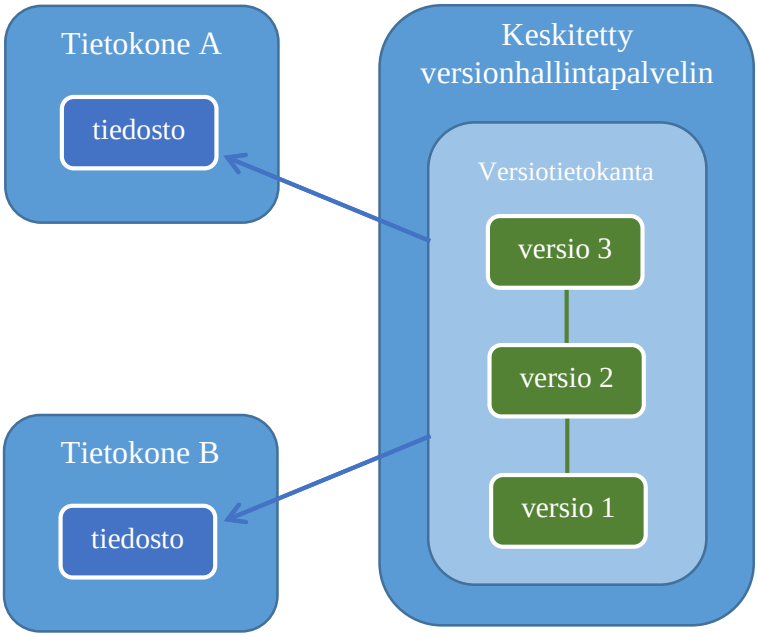
koodimuutosten tai katastrofisen levyn tai koneen kaatumisen tai korruptoitumisen takia. On viisasta pitää arkistoa kaikesta työstä, jotta mahdollisten ulkopuolisten tekijöiden tai yksinkertaisien vahinkojen aiheuttamat projektin osuuksien katoamiselta vältyttäisiin. Tähän avuksi on kannattavaa ottaa käyttöön jokin versionhallintaohjelma. Versionhallintaohjelma hallitsee ja ylläpitää seurantaa projektin eri versioista. Versionhallintaohjelman avulla voidaan muun muassa palauttaa yksittäisen tiedoston tai koko projektin takaisin viimeksi tallennettuun toimivaan tilaan, katselmoida ajan kuluessa tehtyjä muutoksia ja nähdä kuka on viimeksi muokannut jotain, mikä voi olla ongelman aiheuttaja. Versionhallinta tarjoaa myös virheidenhallintaan kommenttikentän, johon voidaan ilmaista milloin ja millaista ongelmaa on mahdollisesti havaittu ja kuka sen on huomannut. Versionhallinnan merkitys käytännössä on, että jos jotain sotkeutuu tai tiedostoja katoaa, voi helposti palauttaa projektin edelliseen toimivaan tilaan. Eikä tämän kaiken ylläpitoon tarvitse suurtakaan panostamista. Tärkeintä on olla huolellinen, kun uutta versiota päivitetään käytettävään versionhallintajärjestelmään. (Loeliger 2009,1-4.)

Useamman tekijän projekteissa yksi suurimmista ongelmien aiheuttavista tekijöistä on huolimaton versionhallintajärjestelmän käyttö, kun vanhaa ja uutta versiota yhdistetään toisiinsa. Tällaisessa projektissa jokainen henkilö voi yhdenkin päivän aikana tehdä useita muutoksia projektiin ja jokainen muutos täytyy saada päivitettyä kaikille projektissa mukana oleville toimivassa muodossa. Jokainen tällainen muutos voi aiheuttaa projektin sotkeentumisen, jos versiot yhdistetään huolimattomasti tai vahingossa epätarkasti. Kun useampi tekijä työstää projektia samanaikaisesti, on versionhallinnassa otettava huomioon mahdollinen versioiden yhtäaikainen päivittäminen. Ongelmien välttämiseksi onkin järkevä varmistaa ennen uuden version viemistä versionhallintaan, että itsellä on viimeksi päivitetty projektin versio. (Loeliger 2009,1-4.)

Olemassa on kolmentyyppisiä versiohallintajärjestelmiä: paikallinen versionhallintajärjestelmä, keskitetty versionhallintajärjestelmä sekä hajautettu versionhallintajärjestelmä. Paikallisen ja keskitetyn versionhallintajärjestelmän toiminta perustuu yhteiseen, paikalliseen tai palvelimella toimivaan, versiotietokantaan. Tämä versiotietokanta toimii yhteisenä versioiden tallennuspaikkana, jonne uudet versiot tallennetaan ja päivitetyt versiot haetaan. Paikallisessa ja keskitetyssä versionhallintajärjestelmässä on kuitenkin yksi kriittinen haittapuoli. Kaikki versiot ovat tallennettuna yhteen sijaintiin, joka syystä tai toisesta kaatuu tai korruptoituu, eikä varmuuskopiointia ole suoritettu, voidaan osa tai koko projekti menettää. Paikallisen versionhallinta on kuvattu kuviossa 1 ja keskitetty versionhallinta on kuvattu kuviossa 2. (Kehoe 2015; Starnes 2015.)



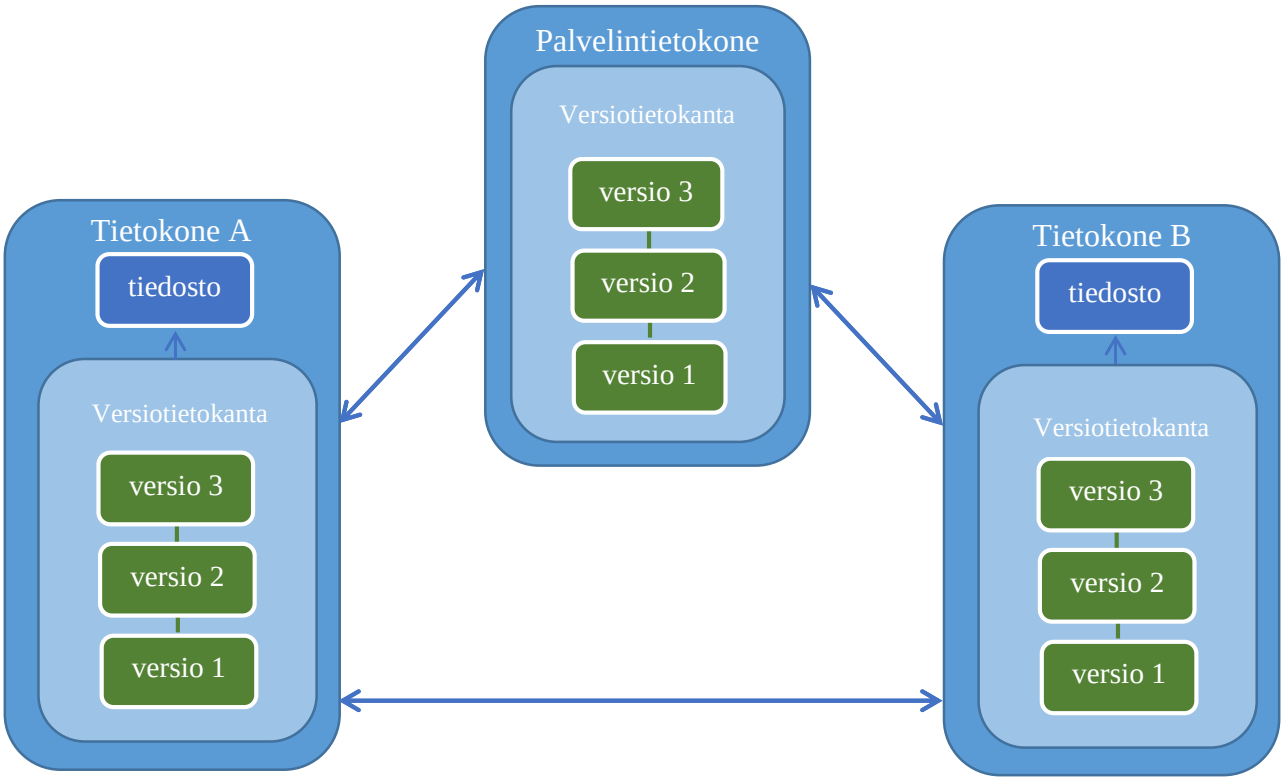
KUVIO 1. Paikallinen versionhallinta.



KUVIO 2. Keskitetty versionhallinta.

Kolmantena ja viimeisenä versionhallintajärjestelmän tyyppinä on hajautettu versionhallinta johon Git-versiohallintajärjestelmä kuuluu. Tämän tyyppisen versiohallinnan ehdottomasti hyödyllisin puoli on, että kehittäjät eivät hae ainoastaan viimeisintä versiota palvelimella olevasta versiotietokannasta, vaan kopioivat täysin kokonaan versiotietokannassa olevat tiedot. Tästä on merkittävä hyöty, sillä minkä tahansa palvelimen kaaduttua tai tietojen korruptoiduttua, jonka kautta järjestelmät tekivät yhteistyötä, voidaan tiedostot kopioida takaisin palvelimelle tiedon

palauttamiseksi. Jokainen projektin osallinen toimii käytännössä versionhallintajärjestelmän paikallisena varmuuskopiona, josta tiedot voidaan palauttaa tarvittaessa. Tämä takaa projektin sulavan ja huolettoman tekemisen, joten projektin työstämiseen voidaan panostaa eikä versionhallinta tuota ylimääräistä päänvaivaa tai hukkatunteja. Hajautettu versionhallinta on kuvattu kuviossa 3. (Chacon & Straub 2014.)



KUVIO 3. Hajautettu versionhallinta.

Git koostuu keskustietovarastosta, paikallisesta tietovarastosta, indeksistä sekä työkopiosta. Tietovarastolla tarkoitetaan versionhallintajärjestelmän paikkaa, joka sisältää varmuuskopiot projektin vaiheista. Keskustietovarastolla tarkoitetaan hajautetun versionhallintajärjestelmän palvelimella toimivaa tietovarastoa, joka on kaikkien projektin kehittäjien yhteisessä käytössä. Tähän tietovarastoon kehittäjät tallentavat muutokset, jotka tulevat muiden nähtäväksi ja käytettäväksi. Paikallinen tietovarasto on kehittäjän alussa omaan käyttöön tekemä kopio keskustietovarastosta. Paikalliseen tietovarastoon kehittäjä voi tallentaa versioita oman työpanoksensa koodiin tuottamista muutoksista ilman, että muut projektin jäsenet näkevät niitä. Kun paikallinen tietovarasto sisältää lisäyksiä, muutoksia tai korjauksia, jotka on tarkoituksenmukaista jakaa muiden projektin kehittäjien kesken suoritetaan push-operaatio, joka siirtää paikallisen tietovaraston sisällön muutoksineen keskustietovarastoon ja tämän kautta muiden käyttöön. Työkopiolla tarkoitetaan kehittäjän työn alla

olevaa versiota. Kehittäjä muokkaa, lisää tai poistaa tarpeen vaatiessa työkopion tiedostoja. Aina kun työkopioon on tehty sopiva määrä muutoksia, siitä tallennetaan versio paikalliseen tietovarastoon komennoilla add ja commit. Indeksi on Git:ssä eräänlainen lisävarasto työkopion ja paikallisen tietovaraston välissä. Aina, kun kehittäjä haluaa tallentaa projektin tilasta uuden version paikalliseen tietovarastoon, on hänen ensin kopioitava halutut tiedosto indeksiin add-komennolla. Kun paikalliseen tietovarastoon tallennetaan uusi versio commit-komennolla, vain ne tiedostot, jotka ovat kopioitu indeksiin, muuttuvat. (Tampereen teknillinen yliopisto: Git ja versionhallinta 2015.)

4.4 HEROKU

Heroku on pilvessä toimiva sovellusten ajoalustana toimiva palvelu. Se tarjoaa palveluna muun muassa sovelluksen toiminnasta pilvessä, tietokannan ylläpidon, yhteydet sekä serverien hallinnan. Herokun arkkitehtuuri koostuu monia ohjelmointikieliä tukevasta alustasta, monipuolisista kirjastoista, käyttöjärjestelmästä sekä taustalla toimivasta infrastruktuurista, joka tukee skaalautuvien verkkosovellusten kehittämistä. Herokussa ajettava sovellus jakautuu niin sanottuihin säiliöihin, joita kutsutaan dynoiksi. Dynon vastuulla on sille määritellyn sovelluksen osan prosessien suorittaminen. Jos projekti on suuri, voi siihen esimerkiksi sisällyttää useamman dynon, joista osa huolehtii verkkoliikenteen toiminnasta ja osa sovelluksen sisäisistä prosesseista. Dynoja on vain yhtä kokoa (512MB RAM), mikä tuokin Herokun hyvän puolen esiin. Jos projekti käy suureksi, dynoja lisäämällä voidaan laajentaa projektia ja jakaa projektin eri osa-alueet eri dynoihin. Projektin skaalaus tapahtuu siis dynojen määrää lisäämällä, eikä ainoastaan yhden dynon kokoa kasvattamalla. Tämä on siitä järkevä tapa skaalata, koska dynot eivät ole riippuvaisia toisistaan. Jos yksi dyno kaatuu, vain se kaatuu ja muut dynot voivat jatkaa toimintaansa. Tämä antaa projektille toimivan tavan laajentua, sillä järkevän projektin osioiden jakamisen avulla siitä saadaan hyvin toimintavarma kokonaisuus. (Hartl 2015, 41–45; Middleton & Schneeman 2013, 7-9; Hanjura 2014.)

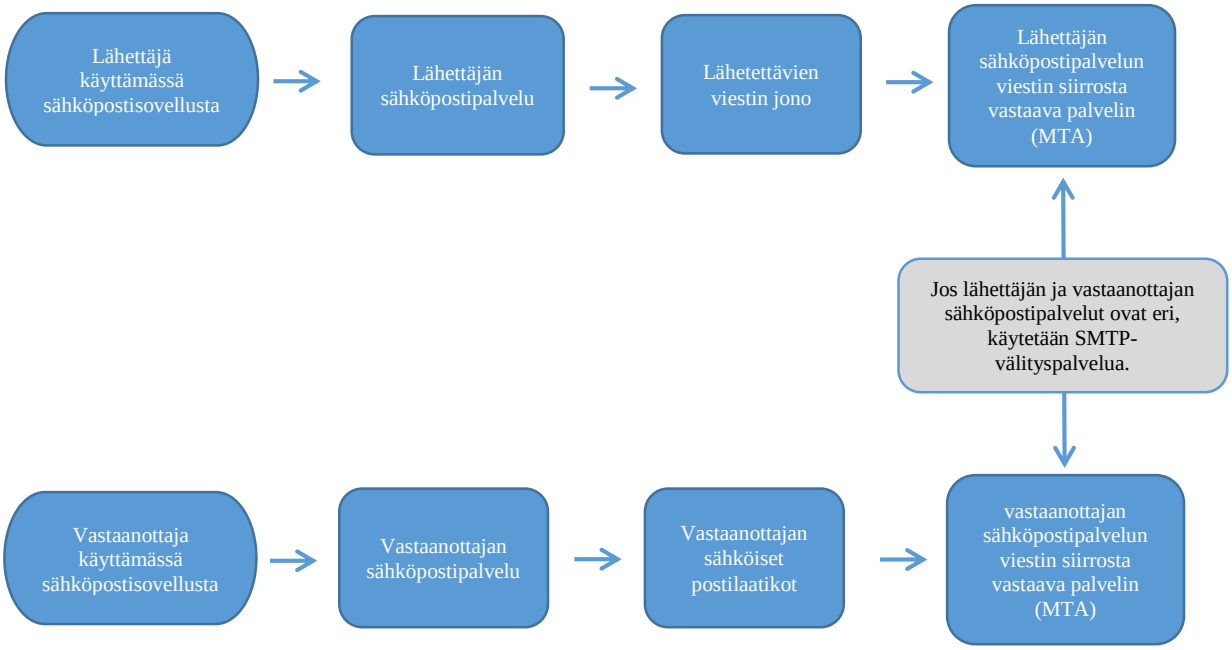
Herokuun on myös saatavilla liitännäisiä, joita on tarjolla yli 150. Tässä projektissa on käytössä postgresSQL tietokanta ja SendGrid SMPT-palvelu Herokun liitännäisten kautta. Heroku tukee yhtä liitännäistä suoraan tilin luotua, mutta jos projektiin haluaa liittää useamman liitännäisen, täytyy Heroku-tiliin liittää luottokorttitiedot. Herokulla on tämä käytäntö, jotta tili voidaan vahvistaa käyttäjälle ja voidaan hallita liitettävien liitännäisten määrää. Heroku on luotu juurikin Rails ja muiden web-sovellusten käyttöönottoa varten. Heroku hyödyntää Git-versionhallintajärjestelmää, jonka avulla sovellusten julkaisu ja käyttöönotto ovat erittäin helppoa ja vaivatonta. Tällaisen PaaS-

palvelun kuten Herokun käyttäminen antaa projektin tekijöille mahdollisuuden keskittyä vain sovelluksen tuottamiseen, joka tuo tehokkuutta projektin etenemiselle. PaaS tulee sanoista Platform as a Service, mikä tarkoittaa, että palvelu huolehtii servereiden ylläpidosta, infrastruktuurista ja huoltamisesta. Näin kehittäjät voivat keskittyä tehokkaammin itse sovelluksen toimintaa ja kehittämiseen. (Hartl 2015, 41–45; Middleton & Schneeman 2013, 7-9; Hanjura 2014.)

4.5 SMTP ja SendGrid-palvelu

Useimmat sähköpostipalvelut, kuten Outlook ja Gmail, lähettävät sähköpostiviestit verkon yli hyödyntäen SMTP-protokollaa eri sähköpostipalvelimien välillä. SMTP tulee sanoista Simple Mail Transfer Protocol, joka on TCP-pohjainen protokolla. TCP-protokolla on tietoliikenneprotokolla, jolla luodaan tietokoneiden välille yhteydet tietoliikennettä varten. SMTP protokolla hyödyntää TCP-yhteyttä viestien välittämiseen sähköpostipalveluiden kesken. SMTP:tä käytetään, kun sähköpostia lähetetään eri sähköpostipalveluiden välillä. Esimerkiksi Outlook ja Gmail käyttävät omia protokolliaan päästäkseen käsiksi sähköpostitileihin heidän omilla sähköpostipalvelun palvelimilla, mutta kaikki sähköpostipalvelut käyttävät hyväkseen SMTP protokollaa, kun sähköposteja lähetetään tai vastaanotetaan heidän järjestelmänsä ulkopuolelta. SMTP:tä voidaan käyttää viestien lähettämiseen sekä vastaanottamiseen tai hyödyntää sitä vain toiseen. Esimerkiksi, jotkin sähköpostisovellukset hyödyntävät sitä vain lähettämiseen, mutta vastaanottamiseen he käyttävät joko POP3 tai IMAP protokollia, jotka hakevat saapuneet sähköpostit palvelimelta ja tuovat käyttäjän sovellukseen vastaanotettuina sähköposteina. (Riabov 2005.)

Sähköpostien lähettäminen tapahtuu siten, että käyttäjä keskustelee sähköpostisovelluksen kanssa (esimerkiksi Outlook), josta viesti menee lähetettyjen sähköpostien jonon kautta sähköpostisovelluksen viestin siirrosta vastaavalle osalle, joka on MTA (Message Transfer Agent). Sähköpostisovelluksen MTA välittää viestin TCP-yhteyden avulla vastaanottajan sähköpostipalvelun palvelimelle. Jos viesti täytyy välittää eri sähköpostijärjestelmien palvelimien välillä, tässä kohtaa SMTP tulee käyttöön. lähettäjän MTA muuntaa sähköpostin SMTP muotoon ennen lähetystä. SMTP toimittaa viestin vastaanottajan sähköpostisovelluksen palvelimelle toimivalle MTA:lle, joka muuntaa viestin vastaanottajan sähköpostisovelluksen muotoon. Tästä viesti siirtyy IMAP, POP tai sähköpostipalvelun omalla protokollalla vastaanottajan sähköpostisovelluksen vastaanotettuihin sähköposteihin. Sähköpostilähetyksen toiminta yleisesti on kuvattu kuviossa 4. (Riabov 2005.)



KUVIO 4. Yleinen sähköpostin lähetyksen prosessikaavio.

SendGrid on yksi suosituimmista SMTP-palvelun tarjoajista, joka on perustettu vuonna 2009. SendGrid on pilvipohjainen SMTP-palveluntarjoaja, joka mahdollistaa sähköpostien lähettämisen heidän ylläpidossaan olevien palvelimien kautta. Palvelu on helposti laajennettavissa omien tarpeiden mukaisesti ja se tarjoaa teknisten asioiden seurannan, kuten pyyntöjen, lähetysten, vastaanottojen seurannan ja ongelmien kategorioinnin. Se kertoo käyttäjälle reaaliaikaista tietoa palvelun käytöstä, jonka perusteella on helppo seurata ja analysoida palvelun käyttöä ja näin laajentaa sitä tarvittaessa. (SendGrid: SendGrid Overview 2016.)

SendGrid:n käyttö projektissa on helppoa. Ohjelmakoodiin lisätään sähköpostin lähetystä varten käytettävän SMTP- palvelun tiedot osoittamaan SendGrid palvelinta ja luomasi SendGrid tilin käyttäjätunnus sekä salasana. SendGrid tarjoaa ilmaisen version, joka kattaa jopa 12000 sähköpostia kuukaudessa, joka on rajoitettu 300:an sähköpostiin päivässä. SendGrid on liitettävissä projektiin Herokun liitännäisenä, joka oli projektin kannalta tärkeä kriteeri. Tämä rajasi SendGrid:n kilpailijat neljään SMTP-palveluun, jotka olivat mahdollista liitännäisinä: SparkPost, CloudMailIn, Mailgun sekä Postmark. Kaikki tarjoavat reaaliaikaista seurantaa sekä palvelun analysointia, mutta erityisesti SendGrid oli muita parempi vaihtoehto hintapakettien suhteen. Ilmainen paketti, mikä tarjoaa 12000 sähköpostia kuukaudessa, oli erittäin kattava verrattuna muihin ja jatkon kannalta paras vaihtoehto,

jos tarvitaan palvelun laajennusta. SendGrid tarjoaa 40000 sähköpostia kuukaudessa hintaan 9.95$/kuukausi, joka on edullinen verrattuna muihin palveluihin. SendGrid tarjosi myös ohjeet, kuinka se liitetään Ruby on Rails-projektiin. Muut kilpailevat SMTP-palvelut eivät tarjonnut tällaista ohjeistusta. (SendGrid: SendGrid Overview 2016.)

5 TYÖN TOTEUTTAMINEN

5.1 Vaatimusten määrittely

Työ alkoi palvelinsovelluksen arkkitehtuurin vaatimusten määrittelyllä. Pelin luomisesta ja toiminnoista vastaa käyttöliittymäsovellus, joten palvelinsovelluksen tehtävänä on koordinoida ja välittää pelaajien väliset viestinnät sekä ylläpitää tietokantaa. Palvelinsovellus välittää myös sähköpostiviestit, joita peli lähettää peliin kutsutuille henkilöille. Tiukkoja reaaliaikaisuus vaatimuksia ei ole, sillä peli ei ole nopeatempoinen, kuitenkin siten, että kaikille pelaajille saadaan haluamilla ajanhetkillä tarkastettua tilanteet sekä synkronoitua pelin kulku. Palvelinsovelluksessa olevassa tietokannassa tarvitsee vähintään olla tieto käyttäjästä ja tähän liitetystä sähköpostiosoitteesta. Tietokannasta täytyy löytyä myös sähköpostiosoitteeseen liitetty peli, kyseisen pelin tiedot ja pelin sisällä sähköpostiosoitteeseen liitetty hahmo, jonka kautta löydetään pelaajan hahmon tiedot.

5.2 Työn lähtökohta

Työn teko alkoi palvelinsovelluksen arkkitehtuurin teknologioiden tiedonhausta. Tärkeimpiä työn alkuvaiheessa olevia teknologioita ovat millä ohjelmointikielellä ja mitä kirjastoa hyväksikäyttäen työtä tehdään. Missä ympäristössä ohjelmallinen osa tuotetaan sekä missä julkaisuun menevä tuotos ylläpidetään eli tarvitaanko ulkopuolista palveluntarjoajaa vai onko järkevämpää ylläpitää omia servereitä. Sähköpostien välityspalvelimien tarjonta kartoitetaan ja valitaan projektiin sopivin ja muihin teknologioihin liitettävä palvelu. Näiden lisäksi tarvitaan myös versionhallinta, jotta projekti pysyy hallinnassa ja turvassa.

Lähtökohtana työssä oli edellisen työntekijän, jonka poisjäämisen vuoksi työ tehtiin opinnäytetyönä, alustava ajatus siitä, miten palvelinsovelluksen voisi toteuttaa. Hänen teknologioitansa olivat Ruby-ohjelmointikieli ja Rails-kirjasto. Palvelimen ja puhelimen välillä kommunikointi toteutetaan HTTP-rajapinnan ylitse REST-konventiota käyttäen. Tiedonsiirron muotona JSON ja tietovarastona PostgreSQL relaatiotietokanta. Ajoalustana toimii Heroku.

Vaihtoehtoina projektin alussa oli joko tutkia jo aloitettua suunnitelmaa tai valita uudet teknologiat, joilla työ toteutettaisiin. Koska aikaisempaa kokemusta palvelinsovelluksen toteutuksesta ei ollut,

päädyttiin tutkimaan jo aloitetusta työstä, ovatko siinä valitut teknologiat ajan tasalla. Teknologiat olisivat voineet olla esimerkiksi vanhentuneita tai laajennusmahdollisuuksia ei ole. Tämä oli järkevin tapa sillä, jos näillä teknologioilla palvelinsovelluksen toteutus on mahdollista, se säästäisi aikaa. Jos teknologioita täytyy vaihtaa tai lisätä, löytyisi teknologioihin tutustumalla tärkeitä asioita, joita ottaa huomioon uusien teknologioiden valinnassa, kuten millaisia osioita työstä jonkin sovelluksen täytyisi tehdä tai teknologian yleinen toiminta. Selvitettiin voisiko työn viedä näillä teknologioilla loppuun lisäämällä työn aikana mahdollisesti tarvittavia teknologioita, joita esimerkiksi on sähköpostin lähettämiseen tarvittavaa SMTP-välityspalvelu.

Käyttöliittymäsovellus, jonka tueksi palvelinsovellus toteutetaan, oli vielä kehitysvaiheessa. Projektin edetessä pidettiin palavereja, joissa selvitetään koko projektin sen hetkinen tilanne ja neuvoteltiin mahdollisista toteutustavoista ja rakennemuutoksista. Sillä palvelinsovellus toteutettiin tukemaan projektin käyttöliittymäsovellusta, tapahtui toteutus niin sanotusti käyttöliittymäsovellus edellä. Prosessi eteni siten, että käyttöliittymäsovellukseen määriteltiin vaatimukset, miten sen pitää toimia ja millaisia ominaisuuksia tarvitaan. Tämän jälkeen selvitettiin, mitä ominaisuuksia palvelinsovellukseen tarvitaan, jotta käyttöliittymäsovellukseen vaaditut toiminnot saadaan toteutettua. Varsinainen testaustyö tapahtui käyttöliittymäsovelluksen toteuttajan toimesta, sillä näin saatiin varma testitulos käyttöliittymäsovelluksen ja palvelinsovelluksen välisestä kommunikaatioista ja oikeiden tietojen välittymisestä.

5.3 Teknologioiden selvittäminen

Teknologioiden selvittämisen ja niiden kartoittamisen alkoi niin sanotulla harjoitteluprojektilla, jonka tarkoituksena oli selvittää millainen on palvelinsovelluksen rakenne ja onko se mahdollinen tai järkevää toteuttaa aikaisemmin mietityillä teknologioilla. Samalla tutustuttiin näiden teknologioiden toteuttamiseen ja ohjelmoinnillisiin osiin. Tärkeimpiä asioita, joita teknologioiden selvittämisessä otettiin huomioon, olivat projektin mahdollinen toteuttaminen kyseisillä teknologioilla ja niiden kyky toimia yhdessä. Oli myös tärkeää, että teknologiat eivät kehitysvaiheessa tuottaisi kuluja ja ne olisivat laajennettavissa tarpeen mukaan.

Työympäristöksi valittiin Cloud9IDE. Valinta perustui Cloud9IDE:n tarjoamaan pilvessä toimivaan työympäristöön, joten projektitiedostot pysyvät tallessa vaikka työpistettä ja konetta vaihtaisi. Pilvipalveluna toimiva työympäristö on käytännössä virtuaalikone, jossa, ilmaisessa perusversiossa,

on 512MB muistia ja 1GB levytilaa. Työympäristö on laajennettavissa, jos projekti niin vaatii, rahallista summaa vastaan. Työympäristö ei ole työn kannalta oleellinen, enemmänkin työn tekijän mielipide, minkälaisessa ympäristössä työn haluaa tehdä. Cloud9IDE:n hyviä puolia ovat sen tarjoama pilvipalveluna toimiva työympäristö, koska näin projektia pystyy tekemään hyvin eri koneilta ja eri työpisteiltä. Projekti pysyy ajan tasalla ja viimeisimmät muutokset ovat aina synkronoituna riippumatta siitä, missä työpisteessä on viimeksi työskennellyt. Cloud9IDE tukee myös ”Ruby on Rails” kirjasto-ohjelmointikieli paria. Teknologioita selvitettäessä käytettiin useita sähköisessä muodossa olevia kirjoja sekä keskustelupalstoja. Jonkin verran käytettiin painettua kirjallisuutta sekä videomateriaaleja.

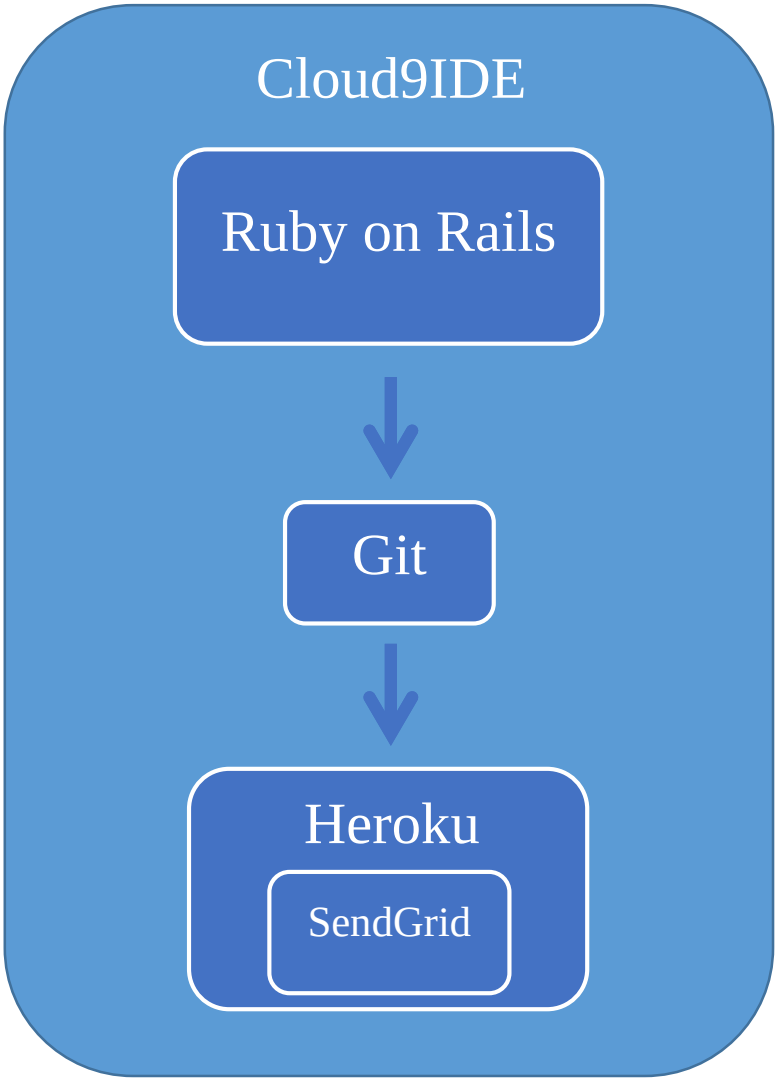
Teknologioiden selvittämisprosessin aikana selvisi, että harjoitusprojektissa käytettävät teknologiat soveltuvat hyvin MurderApp-mobiilisovellukselle. Näiden teknologioiden yhdistelmällä on aikaisemminkin toteutettu pienempiä ja suurempia palvelinsovelluksia, joista tunnetuimpia GitHub sekä twitch.tv. Teknologiat ovat ajankohtaisia sekä täyttävät projektille vaaditut teknologioiden kriteerit. Ruby-ohjelmointikielen ja Rails-kirjaston valinta teknologioiksi perustui tietoon, että palvelinsovelluksen toteutus pystytään niillä tämän tapaiseen projektiin tekemään. Ruby on Rails mahdollistaa käyttöliittymä- ja palvelinsovelluksen välisen kommunikoinnin hyödyntäen JSON tiedonsiirtotyyppiä sekä HTTP-rajapintaa ja REST-konventiota. Omien ylläpidettävien palvelimien idea jäi pois heti, kun Heroku-palveluun tutustuttiin. Heroku tarjoaa tehokkaan ja hyvin laajennettavissa olevan paketin, eikä se tuota kustannuksia, kun otetaan huomioon projektin tämän hetkinen suunniteltu laajuus. Herokusta itsestään ja siihen liitettävästä SendGrid SMTP-palvelusta on ilmaiset pakettiversiot. Paketit soveltuvat MurderApp-mobiilisovelluksen tämänhetkisiin vaatimuksiin ja niitä on jatkoa ajatellen mahdollisuus laajentaa sovelluksen suosion ja käyttötarpeen mukaisesti.

5.4. Teknologioiden liittäminen projektiin

5.4.1 Teknologioiden liittyminen toisiinsa

Työssä käytetyt teknologiat liittyvät toisiinsa seuraavin tavoin. Työympäristönä toimii Cloud9IDE, jonne aloitetaan Ruby on Rails projekti. Projektiin otetaan käyttöön Git-versionhallintajärjestelmä, joka hallinnoi projektin versioita. Git:n käyttöä taas hallinnoidaan Cloud9IDE:n komentokehotteen avulla. Ajoalustana projektissa toimii Heroku. SendGrid SMTP-palvelu lisätään projektiin Heroku-

palvelun liitännäisenä. Jotta Heroku ja Ruby on Rails projekti toimisivat yhdessä, niiden välille tehdään liitos. Ruby on Rails projekti viedään Git-versionhallintajärjestelmän avulla Herokuun ajettavaksi ja ylläpidettäväksi. Kuviossa 5 havainnollistetaan teknologioiden yhteydet toisiinsa.



KUVIO 5. Projektissa käytettävien teknologioiden yhteys toisiinsa.

5.4.2 Cloud9IDE

Teknologioiden liittämisprosessi aloitettiin Cloud9IDE työympäristön käyttöönottamisella ja alustamisella projektia varten. Cloud9IDE työympäristön luominen tapahtuu rekisteröitymällä sähköpostiosoitteella Cloud9 palveluun. Tämän jälkeen sivustolta voidaan luoda työympäristö. Työympäristölle on tehty mallipohjia, joihin on valmiiksi asennettu joitakin pieniä esiasennuksia.

Vaihtoehtoina on joko valita mallipohja tai sitten aloittaa niin sanotusti puhtaalta pöydältä. Tässä projektissa valittiin Ruby on Rails – pohja, joka oli valmiiksi alustettu Ruby on Rails projektia varten. Pohjan valinnan jälkeen työympäristö alustetaan automaattisesti ja hetken päästä se on käyttövalmiina aloittamaan projektin tekemisen Cloud9IDE:ssä. Cloud9IDE toimii pilvessä ja on käytettävissä suoraan selaimen kautta. (Cloud9: Getting Started with Cloud9 2016.)

5.4.3 Ruby on Rails

Ruby on Rails projektin alkuun pääsemisessä on muutama vaihe, jotka sisältävät itse Rails:n lataamisen projektiin, projektikansion luomisen sekä Gemfile- tiedoston muokkauksen omien tarpeiden mukaiseksi. Gemfile-tiedostossa määritellään, mitä liitännäisiä ja näiden versioita käytetään kehitys-, testaus- ja julkaisuversioissa. Tiedostoon muokataan esimerkiksi käytettävä Ruby- ja Rails-versio sekä käytettävä tietokantatyyppi. Koska projektiin valittiin Cloud9IDE-työympäristöä luodessa Ruby on Rails-pohja, on Gemfile-tiedosto alustettu suurimmalla osalla liitännäisistä, joita projektissa tarvitsemme. Tätä Gemfile-tiedostoa tulemme muokkaamaan, kun projektiin liitetään lisää teknologioita. Komento ”Bundle install” päivittää gemfile-tiedoston muutokset gemlock-tiedostoon, jotta valitsemamme versiot ja liitännäiset pysyvät samoina sovelluksen eri versioissa tai, jos projekti ladataan jollain toisella koneella. Nyt projekti on alustettu ja aloituskykyinen. Esikatselua voi suorittaa työympäristön komentorivillä komennolla rails server – b $IP –p $PORT. (Hartl 2015.)

5.4.4 Git – versionhallinta

Ennen kuin itse projektin työstäminen aloitettiin, liitettiin projektiin Git-versionhallintajärjestelmä sekä yhteys Heroku-ajoalustaan. Heroku-ajoalustaan lisätään myös Herokun liitännäisenä SendGrid:n tarjoaman SMTP-palvelun, eli sähköpostiviestien välityspalvelu. Git:n käyttöönotto tapahtuu Git:n asentamisella ja henkilökohtaisen Git-tilin luomisella. Toimeksiantajayrityksellä oli oma GitLab-tili, jonne henkilökohtainen tili luotiin ja liitettiin yrityksen GitLab-projektiin. GitLab huolehtii projektin ylläpidosta ja jakamisesta muille projektissa mukana oleville käyttäjille. Jotta Git saadaan toimimaan, täytyy tietokone alustaa Git-tilin tiedoilla. Nämä ovat tietokonekohtaisia asennuksia ja tarvitaan tehdä vain kerran tietokonetta kohden. Suoritetaan kaksi komentoa, jotka tallentavat Git-tilin tiedot tietokoneeseen:

1. *$ git config --global user.name "Sinun Nimesi"*
2. *$ git config --global user.email sinun.sähköposti@esimerkki.com*

Komentojen suorituksen jälkeen voidaan aloittaa paikallinen Git:n käyttö. Jokaisen uuden projektin kohdalla on versionhallinta alustettava komennolla:

$ git init

Komento alustaa paikalliseen projektitiedostoon versionhallintaan tarvittavat tiedostot. Tämän jälkeen voidaan muutokset viedä paikalliseen versionhallintaan komennoin:

1. *git add –A*
2. *git commit –m "mitä muutoksia tehty?"*

Ensimmäinen komento lisää muokatut tiedostot Git:n indeksiin ja toinen komento vie muutokset paikalliseen versionhallintaan. Projektin vieminen paikallisen Git-versionhallinnan alle on nyt tehty, joten voimme liittää sen ulkoiseen Git-projektien ylläpitoa ja jakamista tarjoavaan palveluun, GitLab:in. Tietokoneen ja GitLab:n välille tarvitaan tehdä SSH-avainpari. Tämä tarkoittaa liitosta tietokoneen ja palvelun välillä, jotta versiot päätyvät oikeaan osoitteeseen. Liitosta varten tarvitsee luoda julkinen avain, joka Git:n asentamisen jälkeen saadaan komennolla:

$ cat ~/.ssh/id_rsa.pub

Kyseinen avain kopioidaan GitLab-tilin profiiliasetuksiin SSH-avainten osioon. Avainpari-liitoksen jälkeen luodaan GitLab:in versiohallinnan säilytyspaikka(repository). Säilytyspaikan tietoihin annetaan sen nimi, kuvaus, yksityisyys sekä säilytyspaikan tyyppi, joka tässä tapauksessa on Git. Säilytyspaikan nimeä tarvitaan seuraavassa vaiheessa, joka on GitLab:n liittäminen ja projektiversion vieminen säilytyspaikkaan ensimmäisen kerran työympäristöstä. Tämä tapahtuu kahdella komennolla, joista ensimmäinen liittää GitLab:n projektiin ja toinen vie projektiversion GitLab:n ylläpidettäväksi. Komennot ovat:

$ git remote add origin git@gitlab.com:<GitKäyttäjänimi>/<säilytyspaikanNimi>.git
$ git push -u origin –all

Nyt Git ja GitLab ovat liitetty projektiin ja versionhallintaa voidaan alkaa sujuvasti käyttämään. Versioiden tallentaminen paikalliseen versionhallintaan ja vieminen GitLab:in ylläpitoa ja jakamista varten tapahtuu tästä eteenpäin komennoin:

1. *git add –A*
2. *git commit –m "mitä muutoksia tehty?"*
3. *git push*

Kaksi ensimmäistä tallentavat muutokset ja vievät ne paikalliseen versionhallintaan ja kolmas komento ”puskee” version GitLab-liitoksen ja SSH-avainparin avulla palveluntarjoajan huostaan. (Hartl 2015.)

5.4.5 Heroku

Herokua liitettäessä Ruby on Rails projektiin, täytyy muutama vaihe ottaa huomioon. Heroku käyttää postgreSQL tietokantatyyppiä. Tästä johtuen Rails-projektin gemfile-tiedostoon täytyy lisätä tietokantaliitännäinen, joka käskee Herokun käyttää postgreSQL tietokantatyyppiä. Täytyy myös varmistaa, että tietokantatyypin SQLiten liitännäinen on poissa käytöstä, koska Heroku ei tue sitä. SQLite on Rails:n oletuksena lisäämä tietokantatyyppi. Gemfile-tiedoston muutosten jälkeen täytyy suorittaa komento ”bundle install”, jotta muutokset päivittyvät gemlock-tiedostoon. Tämän jälkeen luodaan Heroku-tili osoitteessa https://signup.heroku.com/. Pilvessä olevalla Cloud9IDE työympäristöllä Heroku CLI(Command-Line Interface), eli Herokun komentokehote, löytyy suoraan. Käytettäessä paikallista työympäristöä, on Heroku CLI asennettava Heroku Toolbelt-työkalun avulla. Kun Heroku CLI toimii, voidaan käyttää Herokun komentoja ja alustaa sekä liittää Heroku projektiin. Tämä tapahtuu kolmella komennolla:

1. *$ heroku login*
2. *$ heroku keys:add*
3. *$ heroku create*

Kaksi ensimmäistä komentoa liittävät Herokun projektiin. Komennolla ”heroku login” annetaan käyttäjänimi sekä salasana ja komento” heroku keys:add” lisää SSH-avainparin Herokun ja Git-

komentojen välille. Komento ”heroku create” luo tyhjän projektin Herokuun ja alustaa sen. Kun Rails projekti halutaan viedä Herokuun käyttöönotettavaksi, käytetään Git-versionhallinnan komentoa:

$ git push heroku master

Komento vie viimeksi paikalliseen versionhallintajärjestelmään tallennetun version Herokuun, jonka jälkeen sovellus on heti toimintavalmiina ja nähtävissä Herokun tarjoamalla nettisivulla osoitteessa <SovelluksesiNimi>.herokuapp.com. (Hartl 2015; Hanjura 2014.)

5.4.6 SendGrid – palvelu

Tässä vaiheessa projektiin liitettävistä teknologioista puuttuu vielä SendGrid SMTP-palvelu. SendGrid liittäminen ja toimimaan saaminen vaatii kaksi vaihetta. SendGrid liitetään Heroku-ajoalustaan ja Ruby on Rails projekti täytyy määrittää käyttämään sähköpostin lähettämisessä kyseistä SMTP-palvelua. SendGrid liitetään projektiin Herokun liitännäisenä. Tämä on projektin ainoa niin sanotusti ylimääräinen Herokun liitännäinen, mistä johtuen Heroku-tili täytyi varmentaa. Tilin varmentaminen ei tuota kustannuksia ja tapahtuu Herokun verkkosivujen kautta. Sendgrid:n liittäminen projektiin tapahtuu käyttämällä yhtä komentoa:

$ heroku addons:create sendgrid:starter.

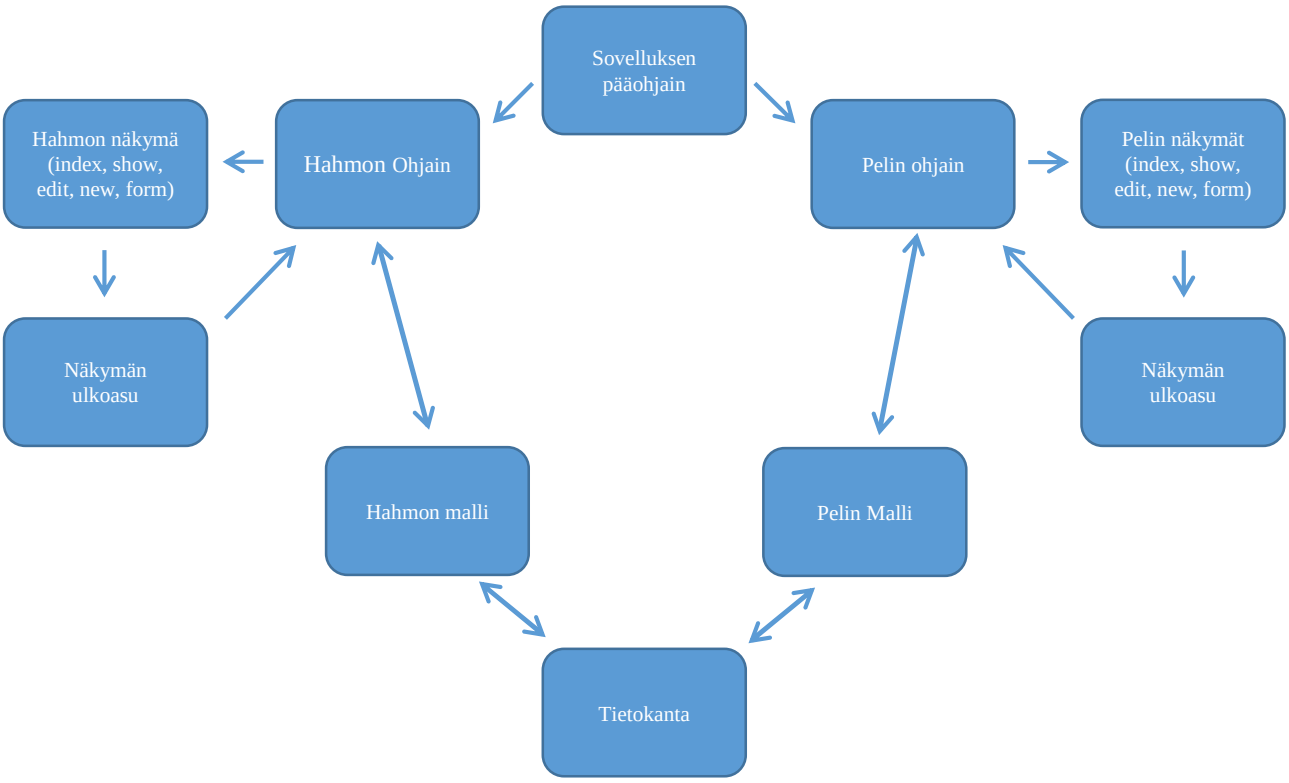
Komento luo SendGrid liitännäisen Heroku projektiin ja kyseinen “sendgrid:starter” kertoo, minkä paketin SendGrid palvelusta haluamme projektiin liittää. Starter-versio mahdollistaa 400 sähköpostin lähettämisen päivässä, joka soveltuu projektille hyvin. Versio on myös ilmainen, joten se ei tuota projektille kustannuksia. Ruby on Rails projektissa tarvitsee määritellä sähköpostin lähettämisessä käytettävän SMTP-palvelun tiedot vastaamaan SendGrid-palvelun vaatimia tietoja. Tiedostossa app/config/environments/production täytyy lisätä sekä muokata muutama komento. Nämä muutokset käskevät Rails-projektin käyttää sähköpostin lähettämiseen SMTP-palvelua, jota ylläpitää heroku-ajoalustan tarjoama domain. SMTP-asetukset alustetaan SendGrid:n käyttöön oikealla välityspalvelinosoitteella, portilla sekä käyttäjätunnuksella ja salasanalla. Asetuksiin vahvistetaan domain, josta sähköpostit lähtevät. (SendGrid: SendGrid Overview 2016; SendGrid: Integrate With SendGrid 2016; Heroku: SendGrid-addon 2016). Kuvassa 1 on kuvattu määritelmät, joita tarvitaan sähköpostiviestin välittymiseen SendGrid-palvelun kautta.

```
config.action_mailer.raise_delivery_errors = true
config.action_mailer.delivery_method = :smtp
host = '<HerokuSovelluksenNimi>.herokuapp.com'
config.action_mailer.default_url_options = { host: host }
ActionMailer::Base.smtp_settings = {
  :address        => 'smtp.sendgrid.net',
  :port           => '587',
  :authentication => :plain,
  :user_name      => ENV['SENDGRID_USERNAME'],
  :password       => ENV['SENDGRID_PASSWORD'],
  :domain         => 'heroku.com',
  :enable_starttls_auto => true
}
```

KUVA 1. Sähköpostiviestin välityspalvelinmääritykset SendGrid-palvelulle.

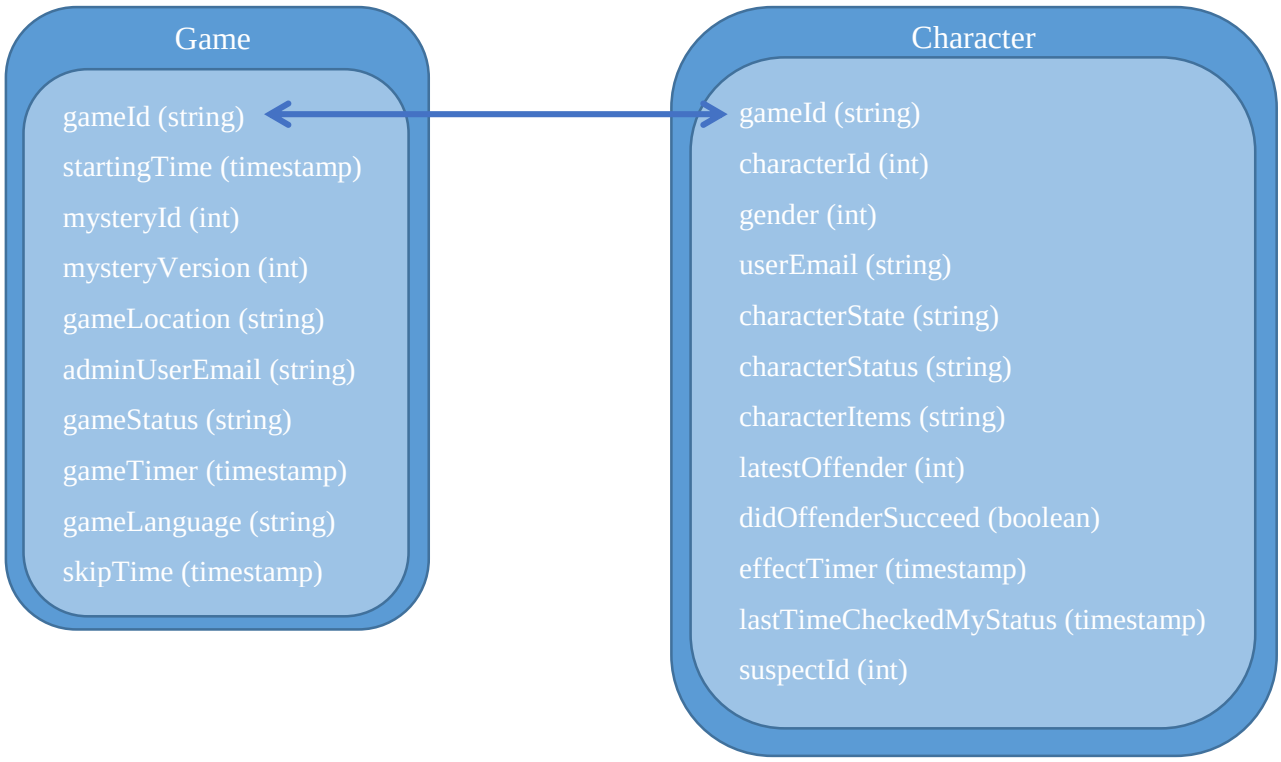
5.5. Varsinaisen sovelluksen rakenne

Projekti koostuu MVC-mallin mukaisesta rakenteesta eli malleista, näkymistä ja ohjaimista. Kuten teoriaosuudessa käytiin läpi, MVC-mallissa mallit, näkymät ja ohjaimet toimivat erillisinä osina. Ohjain on toiminnan keskuksena, joka määrää malleille ja näkymille käskyjä. Ruby on Rails projektissa ohjainten hierarkia koostuu järjestyksessä; toiminnoista vastaava ohjain, sovelluksen pääohjain sekä tietokannan taulukoiden toiminnasta vastaavat ohjaimet. Tietokannan taulukoiden toiminnasta vastaavat ohjaimet keskustelevat mallien avulla tietokannan kanssa. Jokaisella ohjaimella on oma mallinsa, jonka avulla tieto haetaan tietokannasta. Ohjain toimittaa haetut tiedot näkymälle, joka määrittää ulkoasun sisällön ja luo selaimessa näkyvän sivun. Näkymät koostuvat ulkoasusta ja sisällöstä. Näkymien ulkoasu on sama, mutta näkymien sisältö määritetään sen mukaan, mitkä tiedot ohjain on käskenyt näyttää. Kuviossa 6 on kuvattu mallien, näkymien ja ohjainten toiminta.



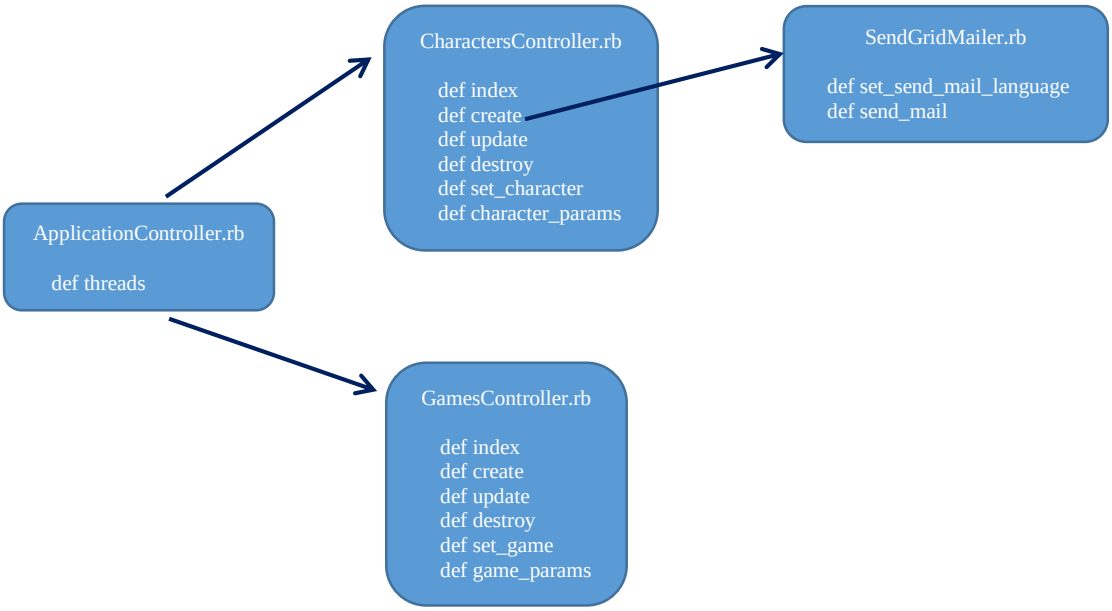
KUVIO 6. MVC-mallin mukaisten mallien, näkymien ja ohjainten toiminta projektissa.

Palvelinsovelluksen yksi tehtävistä on ylläpitää tietokantaa pelitapahtumista ja hahmoista. Ylläpidon lisäksi palvelinsovellus määrittää tietokannan rakenteen, tietojen muokkauksen sekä vastaa tiedonsiirrosta. Tietokannan rakenne on luotu kahdesta taulukosta: characters (hahmot) ja games (pelitapahtumat). Nämä taulukot yhdistetään toisiinsa pelitapahtuman tunnuksen (gameId) avulla. Taulukoiden tiedot ja tietotyypit ovat valittu tukemaan käyttöliittymäsovelluksen toimintaa. Pelitapahtuman aikana käyttöliittymäsovellus päivittää sekä hakee taulukoiden tietoja palvelinsovelluksen määrittämien parametriyhdistelmien avulla. Palvelinsovellus tunnistaa käyttöliittymäsovelluksen lähettämän pyynnön sisältämät parametrit ja osaa toimia sen mukaan. Palvelinsovellus hyödyntää taulukoiden tietoja sähköpostikutsun lähettämisessä muokatakseen kutsun vastaanottajakohtaiseksi. Kuviossa 7 on kuvattu taulukoiden rakenne sekä liitos toisiinsa.



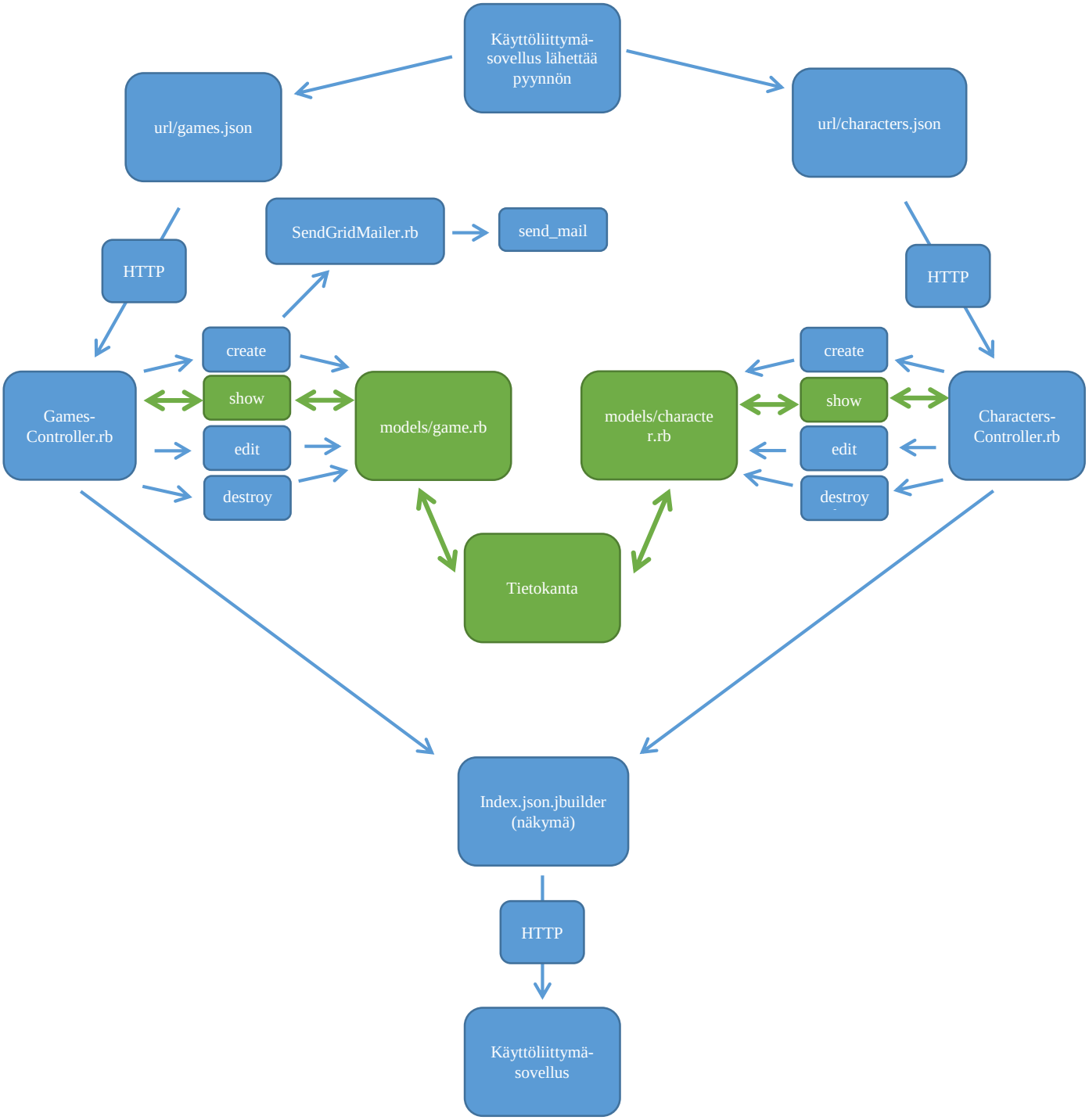
KUVIO 7. Tietokannan taulukoiden rakenne sekä liitos toisiinsa

Palvelinsovellus tarjoaa niin sanottuja palveluja, joita käyttöliittymäsovellus voi käyttää. Näitä ovat tietokannan taulukoihin liittyvät komennot sekä sähköpostiviestin lähetys. Tietokannan tietoihin liittyviä komentoja on neljä: show, create, edit ja destroy. Jokainen komento käy läpi käyttöliittymän pyynnön sisältämät parametrit. Parametrien mukaan komento joko lähettää käyttöliittymäsovellukselle kysytyt tiedot tietokannasta, luo uuden tietueen tietokantaan, muokkaa tietokannan tietuetta tai poistaa sen. Uutta hahmoa luotaessa, palvelinsovellus lähettää sähköpostikutsun hahmoon liitettyyn sähköpostiosoitteeseen. Kuviossa 8 on kuvattu projektin luokkakaavio luokista, jotka mahdollistavat palvelut. Hahmoja ja pelitapahtumia hallinnoivat luokat periytyvät sovelluksen pääohjaimesta.



KUVIO 8. Palvelinsovelluksen luokkakaavio

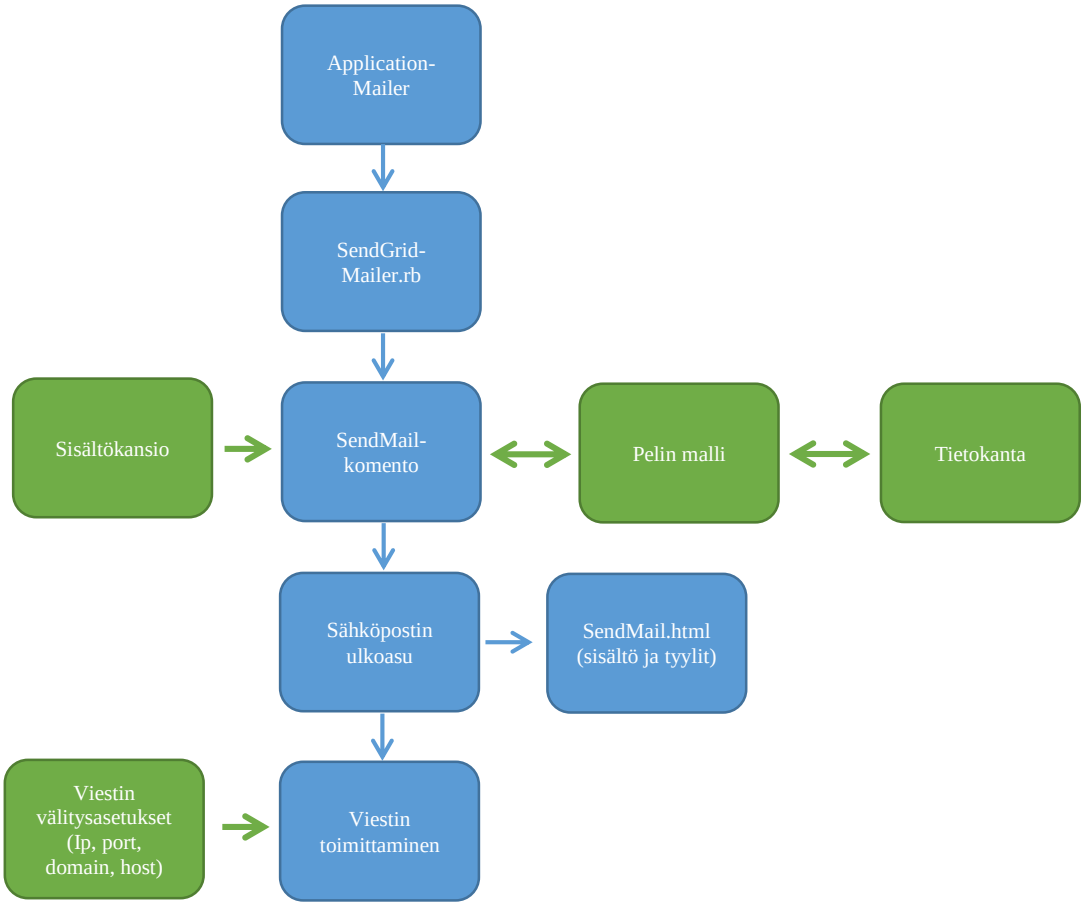
Käyttöliittymä- ja palvelinsovelluksen kommunikointi ja yhteys palvelinsovelluksen tietokannan tietoihin toimii seuraavalla tavalla. Käyttöliittymäsovellus lähettää HTTP-rajapinnan ylitse pyynnön oikeaan URL-osoitteeseen. Pyyntö voi koskea tietokannan tietojen hakemista, uusien tietojen lähettämistä tai nykyisten tietojen muokkausta. Otetaan esimerkkinä käyttöliittymäsovelluksen pyyntö hakea tietoja palvelinsovelluksen tietokannasta. Riippuen siitä haluaako käyttöliittymäsovellus tiedon hahmoista vai peleistä, on osoitteena url/characters.json tai url/games.json. Url-osoitteen perässä “.json” ilmoittaa palvelinsovellukselle tiedostotyypin, jona käyttöliittymäsovellus haluaa kysytyn tiedon palautuvan. Käyttöliittymäsovelluksen pyyntö tulee HTTP-rajapinnan ylitse palvelinsovelluksen ohjaimelle. Kun kyseessä on tiedon haku tietokannasta, ohjain valitsee ”show”- komennon eli vaihtoehdon näyttää käyttöliittymäsovellukselle tietoja. Mallin kautta tiedot haetaan tietokannasta, joita pyynnössä tarvitaan. Tiedot palautuvat ohjaimelle, joka toimittaa ne näkymälle index.json.jbuilder. Tämä näkymä muuntaa tietokannasta haetut tiedot JSON muotoon, jotta ne voidaan HTTP-rajapinnan ylitse toimittaa käyttöliittymäsovellukseen halutussa muodossa. Kuviossa 9 on kuvattu käyttöliittymä- ja palvelinsovelluksen kommunikointi.



KUVIO 9. Käyttöliittymä- ja palvelinsovelluksen kommunikointi projektissa.

Projektin toinen iso kokonaisuus on sähköpostin lähettäminen käyttäjälle, kun tämä kutsutaan peliin. Kutsu lähtee käytännössä siinä vaiheessa, kun pelin luoja liittää peliin hahmoja ja määrittelee kenet liitetään tiettyyn hahmoon. Kun hahmo, johon käyttäjän sähköpostiosoite on liitetty, palvelinsovellus lähettää käyttäjäkohtaisen sähköpostin liitettyyn osoitteeseen. Malli lähetettävästä sähköpostista on liitteessä 1.

Sähköpostikutsun lähetyksen käskee hahmoista vastaava ohjain. Sähköpostin lähettämisen prosessi projektissa on seuraavan kaltainen. Ruby on Rails projektissa on olemassa pohjatiedosto ”ActionMailer”. ActionMailer hallitsee sähköpostiviestintää projektissa. “ApplicationMailer”, joka periytyy ActionMailer:sta, on tiedosto, jonka kautta sähköpostien lähetys alkaa. Tämän tiedoston alle luomme ohjelmointitiedoston, jonka avulla määrittelemme haluamamme sähköpostilähetyksen. Tämän tiedoston nimenä käytämme SendGridMailer. Tässä tiedostossa on olemassa toiminto SendMail, jonka tehtävänä on muokata sähköpostiviesti käyttäjäkohtaiseksi, sekä määrittää viestin lähettäjä, vastaanottaja sekä aihe. Toiminto hakee tietokannasta käyttäjän pelin ja hahmon tiedot sekä sisältökansiosta kuvia ja tekstejä, joita käytetään sähköpostin ulkoasuun ja sisältöön. Kun sähköposti on muokattu toiminnon avulla käyttäjäkohtaiseksi, voidaan viesti toimittaa vastaanottajalle. Viestin toimittamisen mahdollistamiseksi SendGrid-palvelun kautta, on ActionMailer:lle määritelty käytettävä sähköpostinvälitysmuoto, välityspalvelun asetukset sekä lähettävän palvelun ylläpidosta vastaava URL-osoite. Kuviossa 10 on kuvattu sähköpostiviestin lähetys projektissa.



KUVIO 10. Sähköpostin lähettämisen prosessi projektissa.

6 POHDINTA

Työn tarkoituksena oli toteuttaa MurderApp-mobiilipelin käyttöliittymäsovelluksen tueksi palvelinsovellus. Palvelinsovellus hallinnoi pelaajien väliset keskustelut, ylläpitää tietokantaa sekä välittää sähköpostikutsut pelaajille. Palvelinsovelluksen toteutuksessa käytettyjen teknologioiden soveltuvuus todettiin tämän tyyppiseen ja kokoiseen projektiin. Teknologiat valittiin siten, että kehitysvaiheessa ei synny kustannuksia ja teknologiat ovat tarvittaessa myöhemmin laajennettavissa. Työssä käytettäviksi teknologioiksi valikoitui Ruby-ohjelmointikieli, Rails-kirjasto, Git-versionhallintajärjestelmä, Heroku-ajoalusta sekä SendGrid SMTP-palvelu. Käyttöliittymä- ja palvelinsovelluksen kommunikointi tapahtui HTTP-rajapinnan ylitse REST-konventiota käyttäen. Tiedonsiirtomuotona kommunikoinnissa oli JSON. Työn tarkoitus saavutettiin, sillä työn tuloksena saatiin valittua toimiva teknologioiden kokonaisuus, joilla palvelinsovelluksen toteutus onnistui todella hyvin. Teknologiat soveltuvat myös samankaltaisiin ja suunnilleen samoja vaatimuksia sisältävien projektien toteuttamiseen.

Palvelinsovelluksen toteutuksessa tarvitsee hallita moni itsenäinen teknologia ja näiden liittäminen toisiinsa. Teknologioita ja näiden yhdistelmiä, joilla palvelinsovelluksen toteutus onnistuu, on monia. Aikaisempi palvelinsovelluksen toteutuksen kokemus helpottaa paljon teknologioiden kartoittamista ja valitsemista. Tässä projektissa aikaisemman teoreettisen tiedon puute koitui jonkin verran aikataululliseksi ongelmaksi. Palvelinsovelluksen rakenteen oppiminen ja käytettävien teknologien haltuunotto vei odotettua enemmän aikaa. Teknologiat tutkittiin huolellisesti alkuvaiheessa, jotta saatiin varmuus, että valituilla teknologioilla palvelinsovelluksen toteutus MurderApp-mobiilisovelluksen asettamien vaatimusten täyttämiseksi onnistuu. Tiedonhaussa ja ongelmakohtien ratkaisujen löytämisessä kului aikaa, tärkeätä oli löytää ja varmistaa tieto oikeaksi juurikin tässä työssä käytettävän teknologiayhdistelmän ongelmiin. Esimerkiksi palvelinsovelluksen lähettämän sähköpostikutsun ulkoasun toteutukseen vaikuttavia tekijöitä oli monta. Toteutuksessa aiheuttivat ongelmia esimerkiksi HTML-tyylien osalta itse SendGrid SMTP-palvelun vaatimukset, selainkohtaiset vaatimukset, laitekohtaiset vaatimukset sekä sähköpostipalvelun vaatimukset. Työn aikana kuitenkin onnistuttiin oppimaan ja omaksumaan teknologioiden käyttö ja toteuttamaan toimiva sovellus.

Työn toimimisen varmuuteen vaikuttaa palvelinsovelluksen aikaisemman toteutuskokemuksen puute. Epävarmuustekijöitä ovat muun muassa se, että löydettiinkö mahdolliset puutteet teknologioista tai onko niitä sovellettu mahdollisimman optimaalisesti. Testausta suoritettiin

käyttöliittymäsovelluksen kanssa projektiryhmän yhteistyönä. Testauksesta saatiin tietoa palvelinsovelluksen toiminnasta ja kommunikoinnista käyttöliittymäsovelluksen kanssa todellisessa käyttötilanteessa. Palvelinsovellus kykenee ylläpitämään tietokannassaan useita pelitapahtumia samanaikaisesti. Laajamittaista testausta ei kuitenkaan testilaitteiden vähäisyyden takia päästy suorittamaan, jossa monia pelejä pelattaisiin samanaikaisesti. Tästä syystä palvelinsovelluksen suorituskykyä ja skaalautuvuutta suuressa kuormitustilanteessa ei päästy testaamaan. Suoritetut pelitapahtumien testaukset onnistuivat ongelmitta, joten voidaan päätellä, että palvelinsovellus kykenee suoriutumaan useastakin samanaikaisesta pelitapahtumasta. On kuitenkin vaikea sanoa ilman tarkempaa testausta, missä vaiheessa mahdollinen skaalautuvuusongelma ilmenee.

Jatkokehitysehdotuksina antaisin työn ohjelmoinnillisen optimoinnin tarkastamisen. Lähtökohtaisen Ruby-ohjelmointikielen ja Rails-kirjaston kokemus oli vähäinen, joten joitakin asioita olisi mahdollisesti voinut tehdä optimaalisemmin. Jatkotutkimusaiheena projektin mahdollisen myöhemmän laajentumisen myötä olisi hyvä toteuttaa palvelinsovelluksen skaalautuvuustestaus. Tarkemman testauksen avulla saataisiin tietää kykenisikö palvelinsovellus suoriutumaan kymmenistä tai jopa sadoista samanaikaisista peleistä. Testauksen myötä saataisiin tärkeää tietoa, missä vaiheessa teknologioiden pakettiversioita täytyisi laajentaa. Selviäisi myös tarvitseeko kaikkia paketteja laajentaa vai saadaanko mahdollinen skaalautuvuusongelma korjattua vain yhden tai muutaman teknologian osien pakettien laajentamisella.

LÄHTEET

Hartl, M. 2015. Ruby on Rails Tutoria (3rd ed.). 3.painos. Addison-Wesley.

Bigg, R., Katz, Y., Klabnik, S. & Skinner, R. 2015. Rails 4 in Action. Manning Publications Co.

Kasurinen, J.P. 2011. Ruby on Rails-ohjelmointi. Jyväskylä: Docendo.

Chacon, S. & Straub, B. 2014. Pro Git (2nd edition). 2.painos. Apress.

Loeliger, J. & McCullough, M. 2012. Version Control with Git, 2nd Edition. 2.painos. O'Reilly Media.

Middleton, N. & Schneeman, R. 2013. Heroku: Up and Running. O'Reilly Media.

Riabov, V. 2005. SMTP (Simple Mail Transfer Protocol). Rivier College.

Hanjura, A. 2014. Heroku Cloud Application Development. Packt Publishing Ltd.

Starnes, D. 2015. Create a Simple Flask Application with Cloud9, Heroku and MongoDB. Leanpub.

Kehoe, D. 2015. Ruby on Rails with Cloud9. Luettu 10.3.2016. http://railsapps.github.io/rubyonrails-cloud9.html.

Cloud9. 2016. Getting Started with Cloud9. Luettu 10.3.2016. https://docs.c9.io/docs/getting-started

Kehoe, D. 2013. What is Ruby on Rails?. Luettu 10.3.2016. http://railsapps.github.io/what-is-ruby-rails.html.

Tampereen teknillinen yliopisto. 2015. Git ja versionhallinta. Luettu 10.3.2016. http://www.cs.tut.fi/~opersk/S2015/@wrapper.shtml?materiaalit/git.

SendGrid. 2016. SendGrid Overview. Luettu 10.3.2016. https://sendgrid.com/docs/User_Guide/index.html.

SendGrid. 2016. Integrate With SendGrid. Luettu 10.3.2016. https://sendgrid.com/docs/Integrate/index.html.

Heroku. 2016. Sendgrid-addon. Luettu 10.3.2016. https://elements.heroku.com/addons/sendgrid.

Pirinen, J. Hallituksen puheenjohtaja. 2016. Opinnäytetyön taustaa. Sähköpostiviesti. janne.pirinen@kaskistudios.fi. Luettu 27.9.2015.

LIITTEET

Liite 1. Sähköpostikutsu