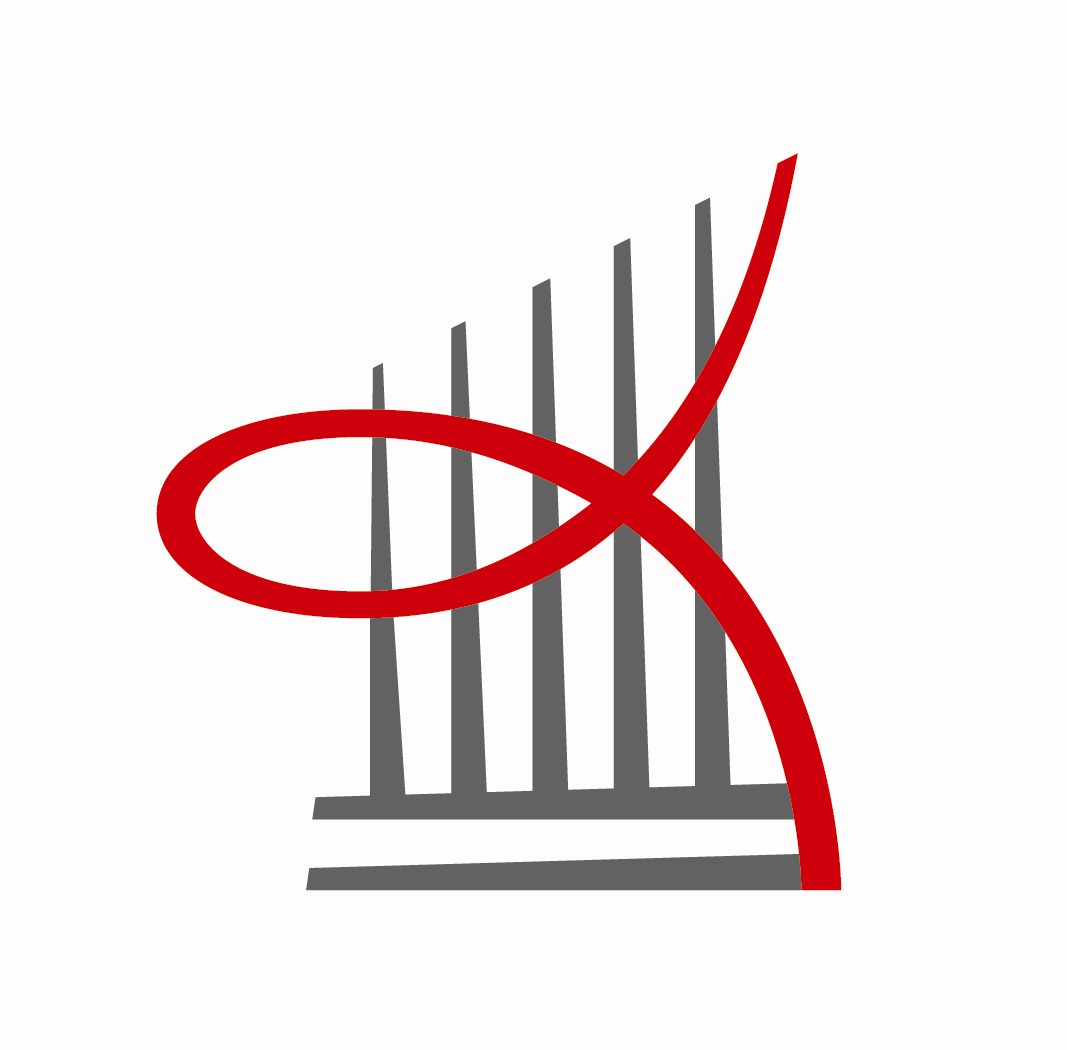
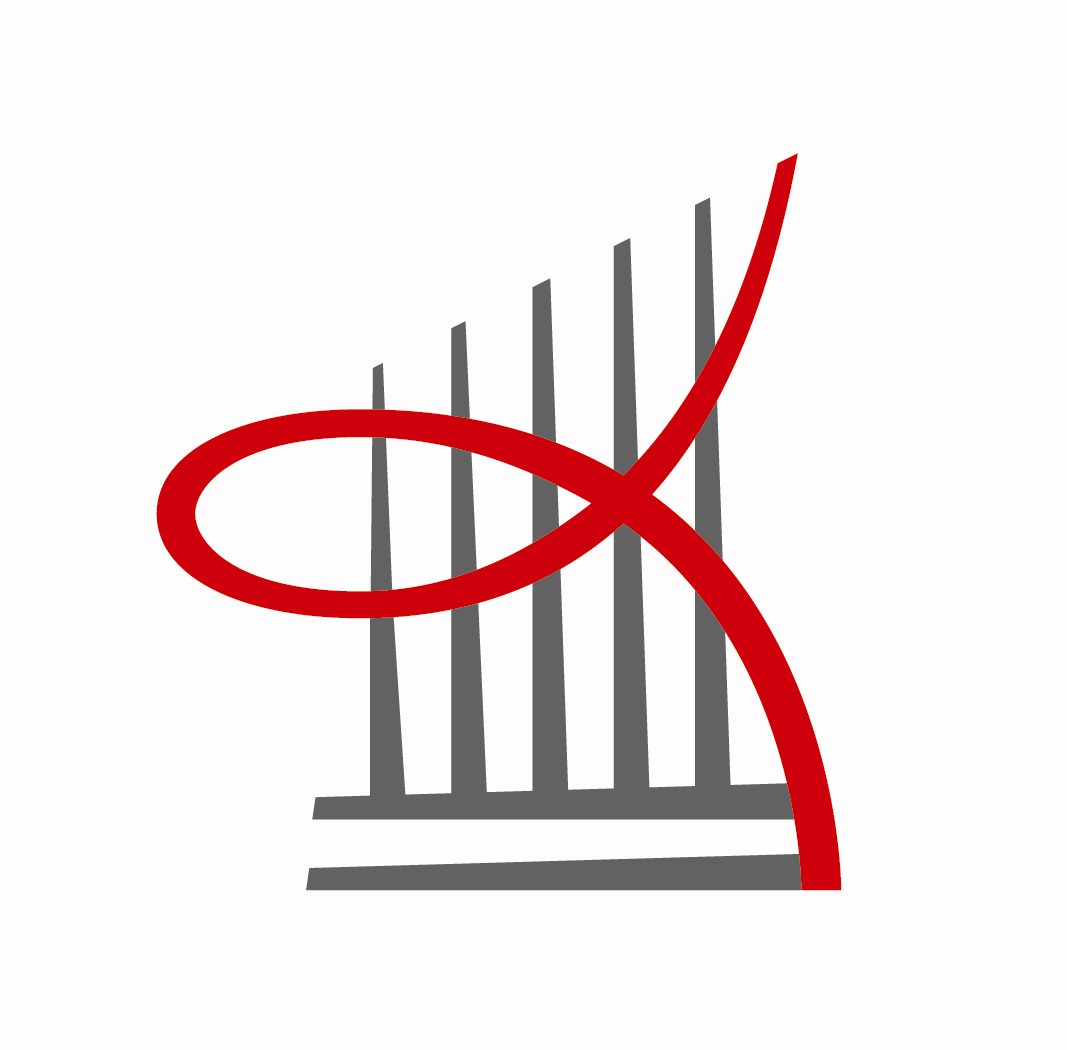
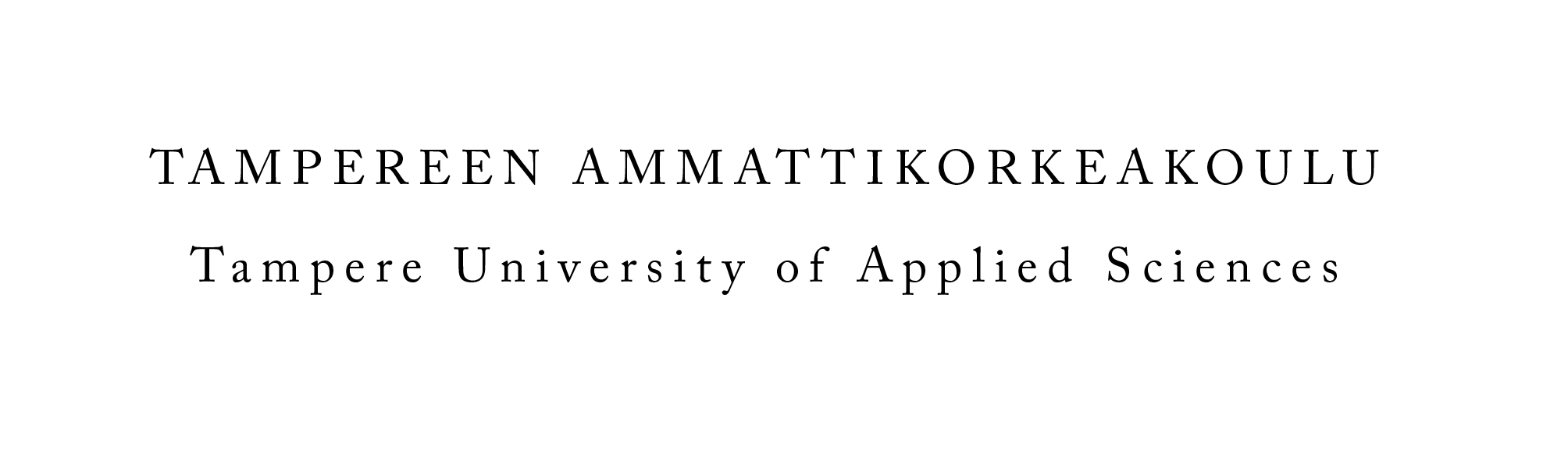
2

2





SuntikkaRSS

Anssi Vaakanainen

Janne Väisänen

Sisällys

[Ohjelman vaatimukset 3](#_Toc402868100)

[Ohjelman arkkitehtuuri 4](#_Toc402868101)

[Ohjelman suunnitteluratkaisu 5](#_Toc402868102)

[Keskeiset toteutusratkaisut 6](#_Toc402868103)

[Ohjelman testaus 7](#_Toc402868104)

[Ohjelman käyttöönotto 8](#_Toc402868105)

[MEAN-pinon sovellus 9](#_Toc402868106)

[Työhön käytetty aika eriteltynä eri vaiheisiin (esim. opiskelu, Angular.js-mallisovelluksen kokeilu jne.). Tätä varten toteutuneita työaikoja kannattaa tallettaa vaikka Excel-tiedostoon riveinä tyyliin pvm, aika, mitä tehtiin. 10](#_Toc402868107)

[Yhteenveto 11](#_Toc402868108)

# Ohjelman vaatimukset

Ohjelmisto vaatii toimiakseen käyttöjärjestelmän lisäksi node.js taustan, mongodb palvelimen ja javascriptiä tukevan modernin selaimen. Ohjelmisto voidaan ajaa keskitetysti palvelimelta jolloin loppukäyttäjältä vaaditaan vai sopiva selain. Toinen vaihtoehto on ajaa koko järjestelmä lokaalisti jolloin loppukäyttäjän järjestelmän vaatimuksiin kuuluu kaikki ohjelmiston vaatimukset.

Näistä ensinmainittu on suunniteltu käyttötapaus, sillä oman kannan ja node.js taustan vaatiminen loppukäyttäjältä ei ole hyvien suunnittelumallien mukaista.

# Ohjelman arkkitehtuuri

Ohjelman arkkitehtuuriksi valikoitui jo aikaisessa suunnitteluvaiheessa MVC-malli. Tämä malli on jo aikaisemmissa projekteissa todettu erittäin toimivaksi ja käytössä olevat menetelmät on rakennettu voimakkaasti MVC-mallia ajatellen. Erityisesti käytössä ollut angular.js on kehitetty käyttämään MVC-mallia kaikessa toiminnassaan.

Tämän päätöksen jälkeen muut arkkitehtuuriset ratkaisut olivat varsin triviaaleja, sillä käytössä olleet menetelmät antoivat selkeät osviitat siitä miten ohjelmisto niiden päälle tulee rakentaa. Tässä näkyi selvästi nykyaikaisten menetelmien selkeä vahvuus ja heikkous. Kun käytettävät työkalut ovat valitut tulevat ne ohjaamaan voimakkaasti ohjelmiston suunnittelua, joka vähentää arkkitehtuurin miettimiseen käytettyä aikaa. Toisaalta tämä on myös heikkous sillä rakentaakseen toisenlaisen rakenteen omaavan ohjelmiston joutuu usein vaihtamaan käytettäviä työkaluja.

# Ohjelman suunnitteluratkaisu

Ohjelma on hyvin yksinkertainen single-page applikaatio, jonka pohjalla on MEAN-pino.

Node ajaa server.js-tiedoston, johon määritelty:

Mongooselle on osoite MangoDB kantaan. Mongoosen tietuemodelit kaupungeille(todo.js) ja etäisyyksille(distance.js) on määritelty Models kansiossa. Routes.js tiedoston perusteella päätellään kumpaa modelia käyttävä vastaus tietokannasta saapuu.

Expressille kansio (/public), josta varsinainen applikaatio suoritetaan.

Applikaatio muodostuu index.html-näkymästä ja core.js-kontrollerista ( angular ). Sen kummempaa kansiojakoa ei ole tehty, koska ohjelma on niin pieni. MEAN pinoon kuuluva AngularJS on kyseisen applikaation toiminnan kannalta täysin merkityksetöntä, sillä todellisuudessa ainut kirjasto, jota ohjelman toteutukseen tarvittiin on LeafletJS-kirjasto kartan piirtämiseen.

Ohjelma on suunniteltu toimimaan niin, että sivun ladattaessa tehdään MongoDB palvelimelle AJAX-kutsu kaupungeista ja näiden lähimmistä pisteistä.

Saadut kaupungit piirretään LeafletJSllä näkyviin.

Kartalla näkyviä kaupunkeja clickaamalla kutsutaan funktiota valitseKaupunki, joka päättelee onko hyväksytysti määritelty alku- ja loppupiste.

Alku- ja loppupisteen ollessa tiedossa valitseKaupunki kutsuu funktiota dijkstraAlgoritmi.

Tämä luo kaikista kartan pisteistä Kaupunki-oliot, jotka tietävät lähimmät kaupunkinsa etäisyyksineen.

dijkstraAlgoritmi käy läpi kaupunkeja alkupisteestä eteenpäin ja laskee jokaiselle kaupungille lyhintä reittiä alkupisteestä. Mikäli jollekkin kaupungille löydettiin lyhyempi reitti kuin aiemmin, se palautetaan dijkstraAlgoritmille uudestaan läpikäytäväksi.

funktio käy kaupunkeja läpi niin pitkään kuin kohdelistasta löytyy kohteita.

funktion suoritettua itsensä määränpääkaupunki tietää lyhimmän reittinsä ja sen kordinaattipisteet lähetetään drawPoly-funktiolle, joka piirtää kartalle kyseisen reitin.

ruudulle piirretään angularia käyttäen tiedot alku- ja loppupisteestä sekä reitin pituus.

## funktiot ja luokat:

tarkemmin lähdekoodissa core.js tiedostossa

ajaxkutsut

hakee kannasta kaupungit ja etäisyydet

kasitteleEtaisyydet(array)

Käsitellään saadut etäisyydet.

Etäisyydet-taulukkoon lisätään jokainen kaupunki ja niiden avaimiksi

kyseiseen kaupunkiin yhteydessä olevat kaupungit sekä niiden arvo

{"Helsinki":{"Kotka":132,"Lahti":105,"Hämeenlinna":101,"Salo":116} } yms

lisaaKaupungit()

lisää kaupungin tietokannasta saatuihin kordinaatteihin kartalle

dijkstraAlgoritmi( lähtökaupunki, määränpää )

lasketaan lyhin reitti JOKAISEEN pisteeseen

ja lopulta piirrä reitti lähtöpisteen ja loppupisteen välille

checkCity(leaflet\_objecti)

selvittää objektista minkä niminen kaupunki on kyseessä

valitseKaupunki(leaflet\_objecti)

kutsutaan kun karttapistettä klikataan. selvittää tiedetäänkö alku ja loppupiste

drawPoly(arr)

piirtää reitin

luokka Kaupunki(nimi, lähimmät\_kaupungit, djikstranAlkupiste )

luokka kaupunkien tietojen ylläpitoon ja etäisyyslaskuihin

-> checkPath(kaupungit)

yleinen apufunktio voisi olla myös funktio. käy läpi kaikki kaupungit ja tarkistaa niiden lähimmät reitit.

-> getPath()

hakee tämänhetkisen lyhimmän reitin lähtöpisteestä kyseessä olevaan pisteeseen, lisäten viimeiseksi kyseisen pisteen nimen

-> Name()

palauttaa kaupungin nimen

-> getPathLength()

palauttaa nykyisen reitin pituuden (laskettu alkupisteestä)

-> calcPath(town)

syötteenä tulee kaupunki

kaupungilta kysytään sen tämän hetkinen lyhin reitti ja etäisyys

palautetaan true if lyhyempi kuin aiempi reitti

# Keskeiset toteutusratkaisut

Dijkstra algoritmin sisäistäminen muodostui hieman ongelmalliseksi sillä sen hahmottaminen osoittautui haasteelliseksi. Toteuttaminen koodiin oli varsin simppeliä, mutta halusimme silti ymmärtää mitä koodimme tekee, joten siihen kului hieman aikaa. Muilta osin ohjelman käyttämät algoritmit ja rakenteet ovat varsin selkeitä eikä monimutkaisia spagetti ratkaisuja jouduttu käyttämään.

Toinen haasteellisempi kohta oli MongoDB:n pystytys sillä se oli ryhmän kaikille jäsenille uusi asia. Tästä huolimatta siitäkin selvittiin ilman suurempia ongelmia sillä internetistä löytyy lukuisia hyviä tutoriaaleja aiheesta.

# Ohjelman testaus

Ohjelmiston testaukseen käytettiin funktionaalista käyttäjätestausta. Ohjelman toiminnan yksinkertaisuuden kannalta emme lähteneet rakentamaan sen suurempaa testaus suunnitelmaa vaan toteutimme testauksen ns. lennossa. Testin aikana määritimme muutamia reittejä joita ohjelman on tarkoitus tarkastella. Katsoimme ohjelman antamat tulosteet ja varmistimme niiden oikeellisuuden laskemalla samat reitit käsin Wolfram Alphaa hyödyntäen. Katsomme että tämä on ohjelman luonteen huomioiden riittävä kattavuus toiminnalliselle testille.

Tämän lisäksi suoritimme pienimuotoista kuormitustestiä, mutta siinä emme saaneet havaittavaa tulosta ohjelman pienestä koosta johtuen. Lisäksi suoritimme virheellisen käytön testauksen ja totesimme että järjestelmän väärinkäyttämisen riski on erittäin pieni ja ohjelma ohjaa käyttäjän oikealle reitille huomaamattomasti mutta tehokkaasti.

# Ohjelman käyttöönotto

Ohjelman käyttöönoton kannalta on oleellista pystyttää ensinmäisenä Mongodb kanta. Tässä dokumentissa emme mene syvällisemmin MongoDB:n asennukseen ja kannan luomiseen sillä näistä aiheista on kirjoitettu useita hyviä ohjeistuksia.

MongoDB asennuksen jälkeen mongo.exe ohjelmalla tulee kantaan lisätä kaupungit.txt tiedoston sisältö (config/kaupungit.txt).

Tämän jälkeen asenf

# MEAN-pinon sovellus

Ohjelmassa on käytetty pohjana standardia MEAN-pinoa. MEAN koostuu MongoDB kannasta, Express.js web kehyksestä, Angular.js Javascript kehyksestä sekä node.js serveri päädystä. Mean on verrattavissa LAMP, MAMP ja WAMP pinoihin, mutta koostuu modernimmista osista ja noudattaa nykyaikaista MVC-mallia ohjelmiston toteutuksen kannalta. Normaalin MEAN-pinon lisänä olemme tässä työssä käyttäneet Mongoose javascript kirjastoa helpottamaan MongoDB kannan kanssa keskustelua.

MEANin vahvuutena on käytettävien kielien skaalautuvuus ja mahdollisuus toteuttaa useita osa-alueita samalla tutulla kielellä, jolloin taustan ja edustan välinen raja häilyy ja ohjelmoija ei joudu siirtymään kielestä toiseen, näin hidastaen kehitystyötä. Pinon käyttöönotto on myös tehty helpoksi sillä pinon suunnittelu on tapahtunut kokonaan aikakaudella jolloin pakettien koostaminen ja asentaminen on yksinkertaista. Aiheesta löytyy myös lukuisia korkeatasoisia tutoriaaleja netistä, joten tässä dokumentissa emme mene sen syvemmälle tässä dokumentissa. On turhaa kirjoittaa huono ohje johonkin josta on jo olemassa parempi.

# Työhön käytetty aika

asdasdasdasdasdasd 5h

koodaus 4h

dokumentointi 2h

# Yhteenveto

Työn tekeminen antoi hyvän läpileikkauksen nykyaikaisten web teknologioiden käytöstä. Erityinen oppimisen paikka oli eri kerrosten välisessä saumattomassa kommunikaatiossa ja nykyaikaisten työkalujen helppokäyttöisyydessä. Oli mielenkiintoista huomata miten helposti monimutkaisetkin asiat saa tehtyä kun etukäteen miettii käytettävät työkalut ja valitsee ne yhteensopiviksi keskenään.

Ongelmallisia kohtia tässä työssä ei juurikaan ollut ja suurimmaksi haasteeksi muodostui ryhmän sisäinen työnjako sillä toinen ryhmän jäsenistä joutui tekemään töitä odotettua enemään syksyn aikana.