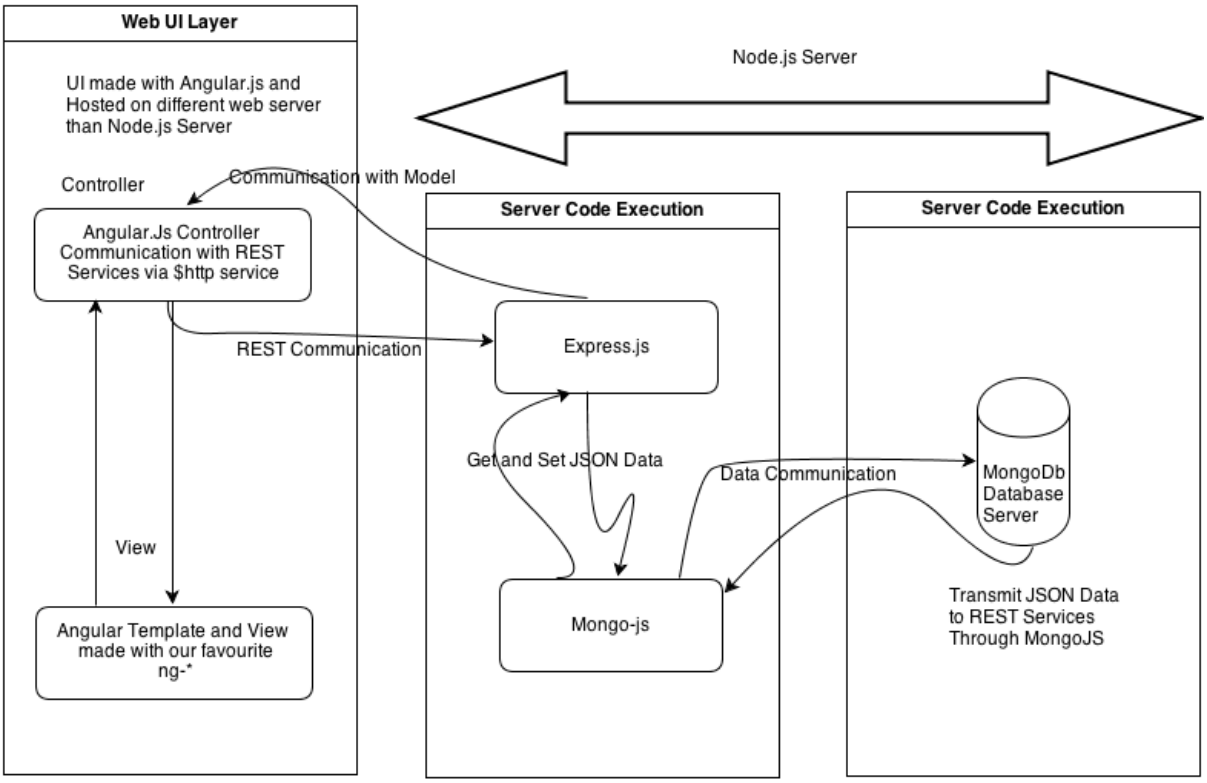
1. Syksyn 2014 Ohjelmistotekniikan työkurssin JavaScript-työ

Tehtävänäsi on tehdä yksinkertainen navigaattoriohjelma, joka laskee pienimmän etäisyyden kahden suomalaisen kaupungin välillä maanteitse. Vaatimuksia ohjelmalle:

- oltava selainpohjainen
- hyödynnettävä JavaScript:iä ja sen päälle rakennettuja kirjastoja
- suositellut kirjastot: MEAN-pino:
    - MongoDB-tietokanta tietovarastona
    - Express.js-kirjasto kutsurajapintana Node.js -palvelimeen päin
    - AngularJS-kirjasto apuna selainpään toimintojen tekemisessä ja MVC-mallin hyödyntämisessä
    - Node.js-kirjasto palvelinpään toimintojen alustana
- Muita tutustuttavia teknologioita (ks. alla oleva kuva): REST-rajapinnat
- Muitakin kirjastoja voi käyttää mutta olisi suositeltavaa, että ne olisivat valtavirtateknologioita. Perustele selostuksessasi tällaiset valintasi. Käyttöliittymän pitää toimia selaimessa, palvelinpäässä on oltava sovelluksen älykkyys ja käytetyt pohjatiedot on oltava tietokannassa. Lisäksi on sovellettava JavaScript:iä mahdollisimman paljon.
- Tämä työ on etupäässä tutustumista uusiin teknologioihin; ei niinkään työ valmiin tuotteen tekemiseksi. Työn eteneminen, aikaansaaminen ja ongelmien ratkaiseminen ovat tärkeämpiä tavoitteita kuin täysin oikein toimiva stilisoitu loppukäyttäjän sovellus. Ja etenemisen yhteydessä tärkeintä on siinä saatava oppiminen. Muistetaan myös ryhmätyön voima eli kaveria autetaan ongelmatilanteissa puolin ja toisin sekä työnjakoa harrastetaan. Etenemisen vaiheet ja lopputuotos kuvataan työstä tehtävässä raportissa.

2. Arkkitehtuuri

Järjestelmän arkkitehtuurin mallina voisi käyttää seuraavaa netistä löytyvää kuvaa (http://a3ab771892fd198a96736e50.javacodegeeks.netdna-cdn.com/wp-content/uploads/2014/03/Angularjs-Nodejs-Angular.png):



3. Vaatimuksia

Ohjelman käyttäjä valitsee kaksi suomalaista kaupunkia ja ohjelma kertoo niiden välisen lyhyimmän etäisyyden kilometreinä maanteitä pitkin.

Tietokannassa on tallessa tiedot kustakin kaupungista sekä etäisyydet niistä naapurikaupunkeihin.

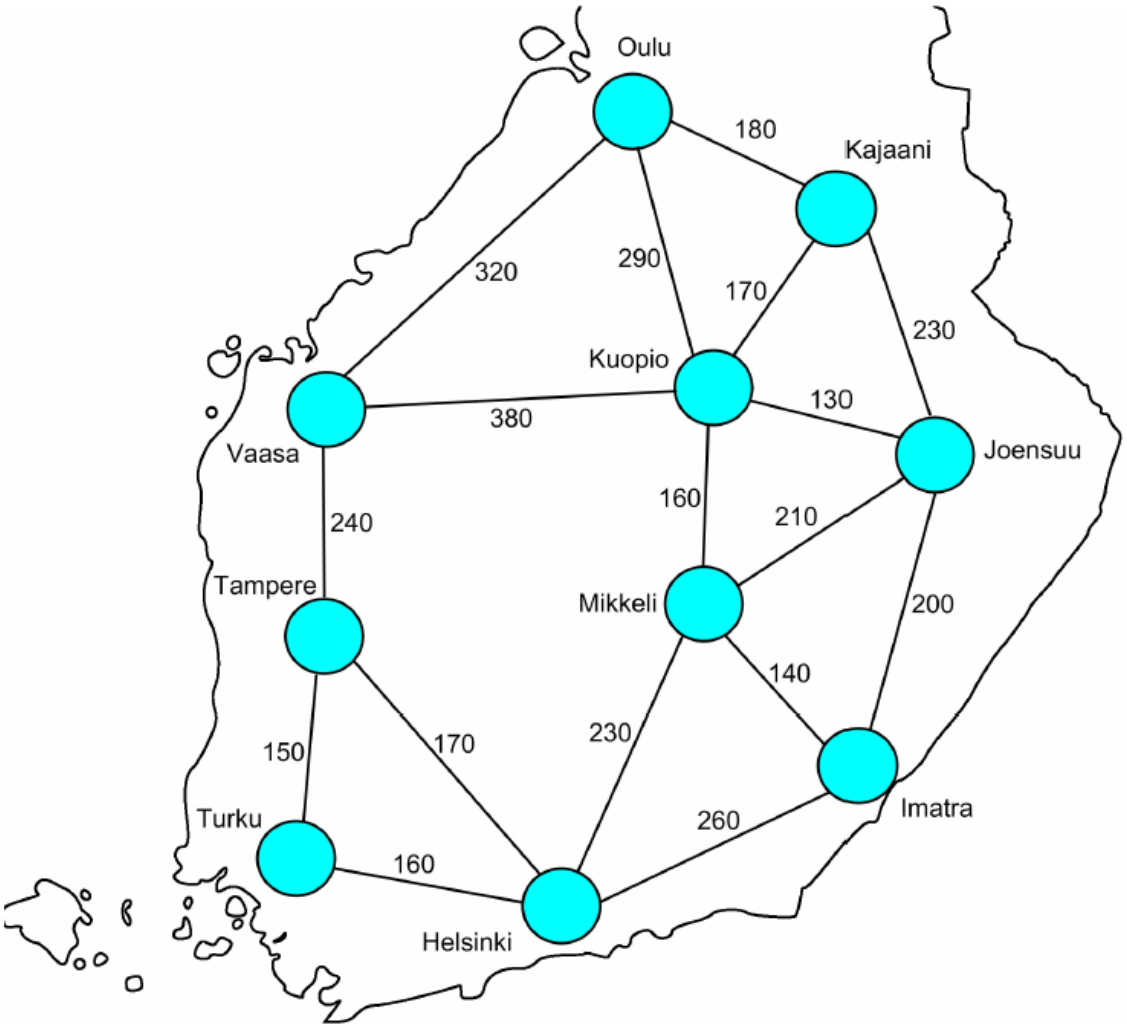
Etäisyyden laskennassa suosittelen käyttämään Dijkstran algoritmia (löytyy esim. Mark Allen Weiss: Data Structures and Algorithm Analysis in C++, 3rd edition, Pearson Education Inc. 2006, sivut 351 – 360). Kirjan kyseisiltä sivuilta löytyy myös kuva esimerkkiverkosta, jota voi käyttää testauksessa. Kirjassa esitellään etäisyysalgoritmin toimintaa vaihe vaiheelta ko. esimerkkiverkkoon sovellettuna. Verkossa kaupunkien niminä on ”v1”, ”v2” jne. mutta tällä ei ole merkitystä testauksen ja algoritmin toiminnan kannalta. Lisäksi verkossa solmujen väliset kytkennät ovat yksisuuntaisia; meidän tapauksessa lisäämme verkkoon kytkentöjä niin, että siitä tulee kaksisuuntainen. Esim. jos v1:stä on suora 2 km:n pituinen tie v2:een niin asia on myös toisinpäin (v2 -> v1). Käytännössä käytettyyn tietorakenteeseen lisätään tuplamäärä yhteyksiä. Algoritmi selviää myös tästä.

Käyttöliittymä voi olla aluksi tekstipohjainen, esim. vedetään kahdesta alasvetovalikosta alkupiste- ja loppupistekaupungit tai vielä yksinkertaisemmassa tapauksessa ne kirjoitetaan kahteen tekstikenttään (edellinen vaihtoehtohan vaatii sen, että selainkäyttöliittymä on alustuksissaan lukenut kaikkien kaupunkien nimet tietokannasta => enemmän pakerrettavaa). Jos aikaa jää voi tavoitella vaikkapa seuraavan kaltaista graafista käyttöliittymää, jossa hiirellä osoitetaan alku- ja loppupisteet (vrt. Google Maps). Tätä varten joudut todennäköisesti selvittelemään sopivan 2D-grafiikkaa tukevan JavaScript-kirjaston käyttöä. Kartta voi olla hyvin vakioitu eli sitä ei tarvitse

generoida tietokantasisällön perusteella mutta jos tähänkin on aikaa voi automaattisen kartan generoinnin tehdä.

Työssä kannattaa edetä yhä haastavampiin osuuksiin niin kauan kuin työtä ylipäätään tehdään. Jos jokin osuus jää kesken (esim. graafinen käyttöliittymä) on siihen liittyvistä teknologioista opittu ainakin jonkin verran ja niiden tutkimista voi jatkaa kurssin jälkeenkin. Mutta kannattaa toimia kuitenkin inkrementaalisesti: kevyempi versio työstä tehdään toimivaksi ja jäädytetään versionhallintaan ennen seuraavaan vaiheeseen etenemistä. Siis viimeisen työn alla olleen inkrementin ei tarvitse olla ”asiakaslaatua”.

Muista kuitenkin että tavoite tässä työssä on teknologioihin tutustuminen syvemmin eli esim. Google Maps –palvelun tarjoaminen ratkaisuna ei edesauta oppimista lainkaan eikä ole hyväksyttävä suoritus.



(kuva on lähteestä:
http://wanda.uef.fi/matematiikka/kurssit/MatematiikanMestariluokka/VerkotJaRelaatiot/Kurssimoniste/VerkotJaRelaatiot_PainotetutVerkot.pdf)

4. Tekemisestä

Alla kerrottu on eräs ehdotus työn tekemisen vaiheistamiseksi. Siinä ideana on tutustua pikkuhiljaa käytettäviin teknologioihin riittävän tarkasti sopivasti ryhmiteltynä, jotta työn tekeminen etenee ja asioita ymmärretään tarpeellisessa määrin työskentelyä varten. Toisenlaistakin vaiheistusta voi käyttää esim. jos jokin toinen lähestymistapa toimii osaltasi paremmin.

Aluksi kannattaa varmaan perehtyä jollain tasolla JavaScript:iin, lähteistä löytyy muutama käyttökelpoinen materiaali. Sen jälkeen kannattaa tutustua Angulariin siitä tehdyn opetusvideon / opetusdokumentin avulla.

Tämän jälkeen kannattaa aloittaa kokeilu vaikkapa esimerkin http://www.phloxblog.in/single-page-application-angular-js-node-js-mongodb-mongojs-module/#.U38bpPmSyE4 avulla; kokeiltaessa on tärkeää ottaa aina selvää siitä mitä on tekemässä eli copy-paste –työskentely ei auta oppimaan. Pidä siis itsesi aina kartalla eri tietolähteitä käyttämällä / kavereita konsultoimalla. Tämä edellyttää esitetyn materiaalin ymmärtämisen (sitä on tutkittava ja selviteltävä niin kauan että ymmärtää sen netin lähteitä käyttämällä). Lisäksi on hyvä tiedostaa sovelluksen ”iso rakenne” (= arkkitehtuuri): mitä keskeistä ajetaan missäkin päässä (selain, node-palvelin, tietokanta). Voi tietysti koittaa muuttaa hiukan sovellusta ja katsoa muutosten vaikutusta sen toimintaan.

Edellä kerrotussa lähteessä kuvataan pienen esimerkkisovelluksen tekemistä samoilla teknologioilla, joita tässä työssä sovelletaan. Eli kokeile esimerkkisovelluksen rakentamista loppuun asti; aja sitä ja tutki sen eri osia.

Tämän jälkeen tutkittua sovellusta mallina käyttäen teet oman navigaattoriohjelman. Nythän tietokannassa onkin kaupunkien nimipareja ja niihin liittyviä etäisyyksiä. Voit jättää sovellukseen mahdollisuuden lisätä näitä tietoja käyttöliittymällä lisää mutta aluksi tietokantaan on syytä luoda jokin pieni testisisältö. Esim. lisäät kymmenkunta eteläsuomalaista kaupunkia ja niiden etäisyydet kantaan. Tai sitten aivan aluksi lisäät kantaan saman datan kuin on kerrottu Weiss’in oppikirjassa (ks. edellä).

Aluksi node-sovelluksen on järkevää vain välittää pyyntöjä tietokannalle selaimesta; emme lisää siihen business-logiikkaa tässä vaiheessa vaan tärkeää on saada vain kommunikointiyhteydet pelaamaan molempiin suuntiin (tynkäprosessilähestymistapa). Tämän jälkeen MVC-mallin periaatteiden mukaan M-osan toiminnallisuus (business-logiikka) eli etäisyyslaskenta sisällytetään node-osuuteen.

Selainkäyttöliittymässä voit minimissään kahden tekstikentän avulla syöttää haluttujen kaupunkien nimet ja sovellus laskee algoritmilla niiden optimaalisen etäisyyden. Etäisyyslaskennassa pitää käyttää jotain tunnettua reittialgoritmia, esim. Dijkstraa (kuvattu Weissin oppikirjassa).

Mikäli aikaa ja mielenkiintoa jää voi selainkäyttöliittymän tehdä graafiseksi edellä ehdotetulla tavalla. Älä hävitä mitään tekemääsi koodia vaan tee sovelluksestasi uusi versio tai uusi haara, jossa onkin karttakäyttöliittymä tekstikäyttöliittymän sijaan. Karttanäkymää varten on hyvä selvittää kaupunkien sijainnit (leveys-/pituuspiireinä, GPS-tietoina), jolloin kartasta tulee realistisempi; esim.

Tampere sijaitsee Helsingin pohjoispuolella. Kaupungista naapurikaupunkiin johtavan tien pituuden voit selvittää Google Maps:lla tai vastaavalla työkalulla lisätäksesi sen kantaan.

Alla vielä tiivistettynä työn vaiheistus ja työnjako (ehdotus):

1. Kaikki tutustuvat annettuja lähteitä tai itse löytämiään parempia sellaisia käyttäen JavaScript:in perusteisiin, lyhyesti HTML:ään, AngularJS:n perusteisiin, node.js:n perusteisiin ja vielä MongoDB:n perusteisiin. Jos ymmärrät yllä kerrotun esimerkkisovelluksen ohjelmakoodin olet hoitanut tämän kohdan sataprosenttisesti!
2. Näillä kullakin teknologialla kannattaa tehdä pieniä kokeiluja annettuja lähteitä hyödyntäen, jotta saa jotain toimivaa aikaiseksi. Edellä on ehdotettu erään esimerkkisovelluksen ajamista. W3School:in JavaScript-itseopiskelupaketti on hyvä tapa ottaa itse kieli haltuun riittävässä laajuudessa.
3. Tämän jälkeen kannattaa ryhmässä jakaa vastuualueet niin, että kaikki eivät selvitä juuri samaa asiaa tai samaan aihepiiriin liittyvää asiaa projektissa. Vastuualue-ehdotuksia:
    - Selainkäyttöliittymä ja AngularJS:n View-/Kontrolleriosuuden tekeminen
    - node.js:n toiminta ja etäisyyslaskenta-algoritmin toteutus omana moduulina / moduuleina sovituilla rajapinnoilla.
    - MongoDB-osuuden toimivaksi saattaminen. Tietokantaan tarvittavat tiedot, niiden noutaminen model-osuuteen sopivassa formaatissa sopivia luokkia käyttäen, jotta etsintäalgoritmiosuus voi niitä käyttää. Vaatii jälleen omien moduulien tekemistä sekä rajapinnoista sopimista.

    Vastuualueissa tehtävät moduulit kannattaa toteuttaa niin että niiden itsenäinen testaaminen on mahdollista mahdollisimman pitkälle. Esim. ensi vaiheessa reitinlaskenta-algoritmista voi tehdä itsenäisen Node.js –sovelluksen tai ihan vain JavaScript-sovelluksen niin että voit varmistua sen oikeellisuudesta itsenäisillä testeillä ennen moduulien integrointia.

4. Kaikki vastaavat työhön liittyvän selostuksen tekemisestä, sovelluksen osien integroinnista sekä testaamisesta. Erityisesti jokainen vastaa oman vastuualueensa asioiden tuomisesta raporttiin.

Vaikka työn tekemisen aikana työryhmän jäsenten kannattaa erikoistua vastuualueisiinsa tämä ei tarkoita sitä, etteikö jokaisen tule oppia työn jokaisesta osa-alueesta riittävästi. Muiden selville ottamiin asioihin voi tutustua ohjelman integrointi-, testaus- tai raportointivaiheessa esimerkiksi. Mutta ihan samaa asiaa ei kannata monen selvitellä samaan aikaan projektin kestäessä vaan hommat jaetaan. Tästä tulee mm. se hyöty, että jos jollain osa-alueella tekeminen jumittaa voi se edetä muilla osa-alueilla ja siten koko projekti yhä etenee. Toki ”jumitus” pitää ratkaista tavalla tai toisella.

Jakamista kannattaa myös tarkentaa projektin kestäessä esim. siitä syystä että jokin homma osoittautuu nopeaksi hoitaa tai jokin toinen selvittely vaatii isoja ponnisteluja. Mutta isot suuntaviivat vastuualueissa kannattaa sopia jo alkuvaiheessa projektia.

Onnea tekemiseen!

PS. Varaudu etsimään erilaisia tietoja paljon netistä ja kyselemään asioita kavereiltasi / auttamaan heitä! Teknologiat ovat melko tuoreita ja ison kehittämisen kohteina joten kaikkitietävää henkilöä / oppikirjakokoelmaa niiden suhteen tuskin löydätte mistään … nämä tulevat sitten jälkijunassa. Myöskin versioyhteensopivuusongelmiin kannattaa varautua.

Esimerkkisovelluksen ja oman sovelluksen kaikkia moduuleita (selainpään toiminnot, Node.js, MongoDB) voi ajaa samalla työasemalla; sitä varten meillä on niihin admin-tunnukset. Useimmat ellei kaikki työkalut joutuu asentamaan verkosta; niihin löytyy verkosta / annetuista lähteistä hyvät asennusohjeet.

5. Työn raportointi

Työn arviointi perustuu itse tehtyyn ohjelmaan sekä siitä tehtyyn raporttiin. Ohjelman osalta on palautettava kaikki tehdyt ohjelmakoodit / HTML-sivut sekä ohje siitä, miten niistä saa aikaiseksi ajokelpoisen ohjelman testejä varten. Myös ohjeet ohjelman käynnistämiseksi ja ajamiseksi on kerrottava.

Testausympäristön muiden tarvittavien ohjelmien, tietokantojen jne. asentamiset pitää myös kuvata ja samoin niiden konfiguroinnit (esim. sopivan datan syöttäminen tietokantaan testejä varten pitää selostaa). Asennusskriptejä kannattaa suosia mahdollisuuksien mukaan; esim. ”MongoDB –skripti” luo tarvittavat taulut tietokantaan ja lisää niihin sopivan testidatan. Mikäli tällaisia skriptejä on tehty pitää ne myös palauttaa raportoinnin osana.

Raportista pitää ilmetä seuraavat asiat:

1. Ohjelman vaatimukset
2. Ohjelman arkkitehtuuriratkaisu: hyvä kuva ja selitystä siitä
3. Ohjelman suunnitteluratkaisu: moduulit, luokat, aliohjelmat, funktiot, ohjelmakoodin jakautuminen moduuleihin jne.
4. Keskeisimmät ja haastavimmat / vaikeimmat toteutusasiat kerrotaan ohjelmakoodiin tukeutuen
5. Ohjelman testaus
6. Ohjelman käyttöönotto-ohjeet (asennus). Tähän voit viitata edellä kun kerrot kuinka ohjelmaasi pääsee testaamaan.
7. Kerro omin sanoin MEAN-pinoa soveltavan ohjelman rakenteen ydinasiat: käsitteet, millainen on ohjelman rakenne, mitä tehdään missäkin jne. Tässä luvussa tavallaan tiivistät sen mitä opit MEAN:sta ja jos joku tietämätön kysyy millainen on tyypillinen MEAN-sovellus voit toimittaa hänelle tämän luvun luettavaksi. Selitä erityisesti omin sanoin keskeisten käytettyjen JavaScript-kirjastojen roolit kokemuksesi perusteella: mihin niitä tarvitaan, mitä ne tekevät?
8. 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.
9. Yhteenveto: mitä opittiin, mitä voisi vielä tehdä, hyvät jutut, kritiikki

Raportin muotoilun on oltava TAMKin kirjallisten töiden raportointiohjeen mukainen (https://intra.tamk.fi/documents/33617/0/Kirjallisen_raportoinnin_ohje2014_25_3.pdf/8cb101dc-e37c-473b-a4bf-c01a09a32142). Lukuotsikot laaditte itse; sen pohjana kannattaa käyttää edellistä listaa. Kaksitasoinen lukuotsikointi voisi olla tässä työssä sopiva; esim. 7. luku käsittelee kokonaan edellisen listan 7. kohdan ja siinä aliluku 7.1. vaikkapa asiakas(selain-)pään keskeiset käsitteet jne. Kuvia kannattaa käyttää; ne selventävät esitettyä asiaa huomattavasti. Toki kuvassa olevia asioita selitetään tekstissä syvemmin.

6. Lähteitä

AngularJS-opiskeluvideo (hieman yli tunnin pituinen): https://www.youtube.com/watch?v=i9MHigUZKEM . Tässä esitetyn pienen sovelluksen toimintaa kannattaa sen työryhmän jäsenen kokeilla & tutkia, jonka vastuualueeseen kuuluu selainpään toiminnot. AngularJS:n käsitteet ja käyttöliittymäosuuden osien rakenne (View & Controller) selvitetään.

Toinen hyvältä näyttävä AngularJS-lähde (uusi; tämän kirjoittaja ei ole vielä ehtinyt tutustua enempää): http://www.w3schools.com/angular/default.asp. Näyttäisi rakennetun niin, että ensin käsitellään mahdollisimman yksinkertaisin esimerkein perusasioita vaihe kerrallaan edeten monimutkaisempiin rakenteisiin mikä on hyvä lähestymistapa oppimista silmälläpitäen. Tsekatkaa tämä!

JavaScript:in haltuunottoon: http://www.cs.helsinki.fi/group/java/s12-weso/
Toinen lähde samaan tarkoitukseen: http://www.w3schools.com/js/

Verkosta löytyy näitä vaikka kuinka paljon mutta itse pidän edellisiä varsin perusteellisena esityksenä kielen yksittäisten ominaisuuksien haltuun ottamisessa.

Pieniä esimerkkisovelluksia, joissa on käytössä samat teknologiat kuin tässä työssä:

http://www.phloxblog.in/single-page-application-angular-js-node-js-mongodb-mongojs-module/#.U38bpPmSyE4

http://scotch.io/tutorials/javascript/creating-a-single-page-todo-app-with-node-and-angular

7. Havaintoja esimerkkiohjelmaa kokeillessa (http://www.phloxblog.in/single-page-application-angular-js-node-js-mongodb-mongojs-module/#.U38bpPmSyE4)

- express.js:stä syytä käyttää versiota 3.8.0; ei neljä-alkuista (ellei halua muuttaa ohjelman ohjelmakoodia enemmän; express.js:n rajapinta on muuttunut vähän ja ohjelmassa ei ole

otettu tätä huomioon => koodi ei suoraan toimi 4-alkuisten versioiden kanssa (configure-metodi pudotettu pois jostain kohtaa jne. …).

- lataa verkosta Windows 7:lle MongoDB-tietokannan 64-bittinen versio sekä Apache Web Server.
- client-pään ohjelmien on syytä olla Apache Web Server:in DocumentRoot –hakemistossa tai sen alihakemistoissa.
- ohjelman ajamiseksi MongoDB:n on oltava ajossa ja myös node:n annetun ohjeen mukaisesti. node-sovellus ottaa vastaan asiakasohjelman pyynnöt ja välittää ne edelleen MongoDB:lle.
- MongoDB-tietokannan sulkeminen (jos ajat sitä cmd-työkalussa edustaprosessina):

  mongo

```
> use admin
> db.shutdownServer();
```

- edellisen sovelluksen käynnistys:
  1. MongoDB käyntiin:

     mongod –dbpath db

     , mikäli tietokanta on käynnistyshakemiston alla db-hakemistossa kuten esimerkissä on asian laita.

  2. node.js –palvelinpuoli käyntiin palvelinskriptillä:

     node appmongodbangular.js

     , olettaa että komennon käynnistyshakemistossa on mainittu JavaScript-tiedosto; esimerkissä pitää siis käynnistäminen tehdä server-hakemistossa, jossa mainittu skripti on.

  3. selaimen osoiteriville http://localhost/angular/angularnodeuser.html, olettaen että Apache Web Server:in DocumentRoot-hakemiston alla on hakemisto angular ja sen alla osoitteessa mainittu html-tiedosto.

- Tyhjän MongoDB-tietokannan luominen:

  http://www.mkyong.com/mongodb/how-to-create-database-or-collection-in-mongodb/

- MongoDB:n bin-hakemisto on hyvä lisätä PATH-ympäristömuuttujaan, jotta työkalujen ajamisessa ei tarvitse mainita polkua. Itselläni se on: C:\Program Files\MongoDB 2.6 Standard\bin. Ja ohje lisäämiseksi löytyy: http://geekswithblogs.net/renso/archive/2009/10/21/how-to-set-the-windows-path-in-windows-7.aspx
- ohjelman käyntiin selaimessa: http://localhost:80/angular/angularnodeuser.html . Apache Web Server –palvelin palvelee pyynnön.

Debuggaamisesta

Voit debugata sovelluksen client-osuutta (selainpään toiminnot) sekä Chromesta että Firefoxista löytyvällä debuggerilla. Chromessa se löytyy: Tools-valinta -> Developer Tools. Firefox:ssa sama asia löytyy: Developer-valinta -> Toggle Tools. AngularJS-osuuden debuggausta helpottaa vielä AngularJS Batarang-laajennoksen asentaminen Chromeen; sen avulla voit helpommin seurata keskeisen $scope-muuttujan sisältöä ohjelman debuggauksessa. Kirjoittajalla ei ole tästä omia kokemuksia vaan tieto perustuu muiden opettajien kokemukseen.

Node.js :llä pyörivää ohjelmaa voit debugata lähteestä http://stackoverflow.com/questions/1911015/how-to-debug-node-js-applications löytyvän vastauksen 486 mukaisesti. Siinä ohjeistetaan node-inspector –työkalun asentaminen ja käyttö. Varmasti muitakin työkaluja samaan asiaan löytyy mutta tämä on havaittu toimivaksi ja käyttöönotto ei vaadi paljoa vaivannäköä.

JavaScript-koodin kirjoittamisesta

Suositeltava IDE-työkalu ohjelmakoodin kirjoittamiseksi on WebStorm. TAMK on ostanut siihen lisenssin eli voi asentaa asentaja-tunnuksella. Toki voit käyttää mitä hyvänsä editoria, jonka koet tukevan riittävästi tekemistäsi.