Juho Leinonen

Tietokantasovellus

Sisällysluettelo:

1. Johdanto

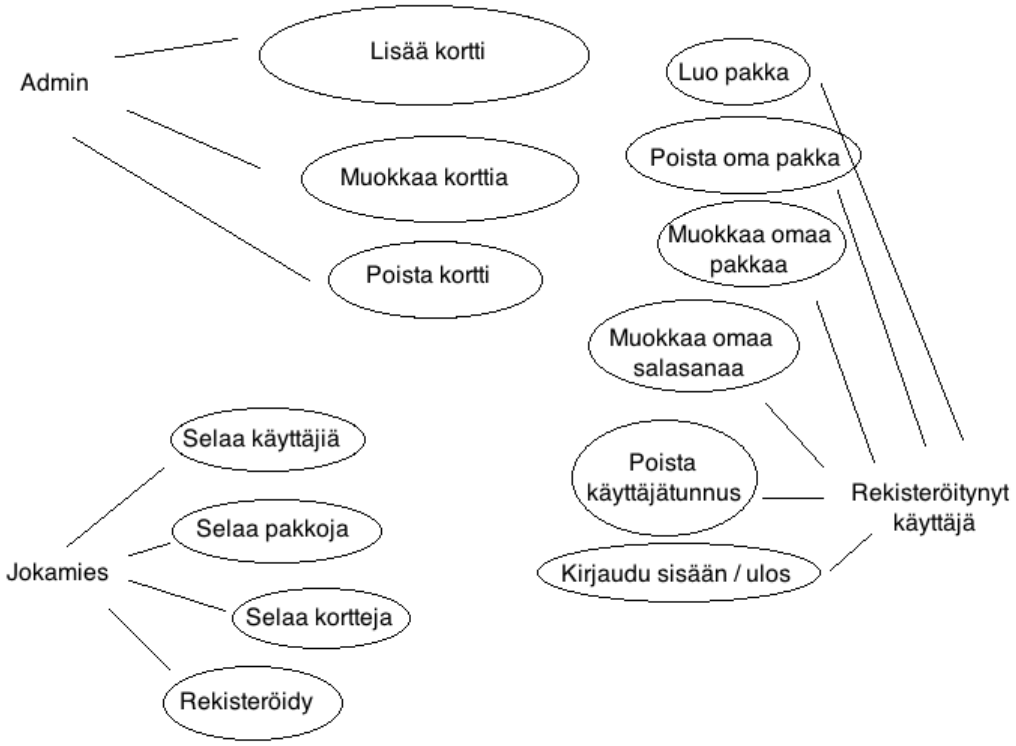
Työni aihe on ns. ”deck builder” Blizzardin uuteen peliin Hearthstone. Hearthstone on tietokonepeli, joka jäljittelee TCG-tyylistä korttipeliä. Sovelluksessani käyttäjät voivat selata kortteja, tehdä itselleen pakkoja ja katsoa muiden käyttäjien tekemiä pakkoja.

Kortteja on kolmen tyylisiä: loitsuja (spell), aseita (weapon) ja kätyreitä (minion). Lisäksi korteilla on luokka – kortin luokka voi olla neutral, warrior, paladin, hunter, shaman, rogue, druid, priest, warlock tai mage.

Aion käyttää sovelluslogiikkaan PHP:tä ja laittaa sovelluksen toimimaan laitoksen users-palvelimella. Tietokannaksi aion ottaa PostgreSQL:n. Sovelluksen alustajärjestelmän tulisi siis tukea PHP:tä. En ainakaan alustavasti aio tukea muita kuin PostgreSQL -tietokantoja. Lisäksi optimaaliseen toimintaan käyttäjän selaimen tulisi tukea javascriptiä (esim. korttilistan filtteröinti ei onnistu ilman sitä). Sivusto toimii ainakin Google Chromen uusimmalla versiolla javascriptit päällä.

2. Yleiskuva järjestelmästä

2.1 Käyttötapauskaavio:



2.2 Käyttäjäryhmät:

Jokamies: Kuka tahansa satunnainen henkilö, joka eksyy sivulle. Kaikki muut käyttäjäryhmät kuuluvat myös tähän ryhmään.

Rekisteröitynyt käyttäjä: Henkilö, joka on luonut käyttäjätunnuksen sivulle. Adminit ovat myös rekisteröityneitä käyttäjiä.

Admin: Henkilö, jolla on etuoikeutettu käyttäjätunnus.

2.3 Käyttötapauskuvaukset:
Jokamies:

- selata käyttäjiä
  eli katsoa kaikkien käyttäjien listaa ja katsoa tietyn käyttäjän omaa sivua
- selata pakkoja
  eli selata kaikkien pakkojen listaa ja katsoa tiettyä pakkaa tarkemmin
- selata kortteja
  eli selata kaikkien korttien listaa ja katsoa tiettyä korttia tarkemmin
- rekisteröityä

Rekisteröitynyt käyttäjä:

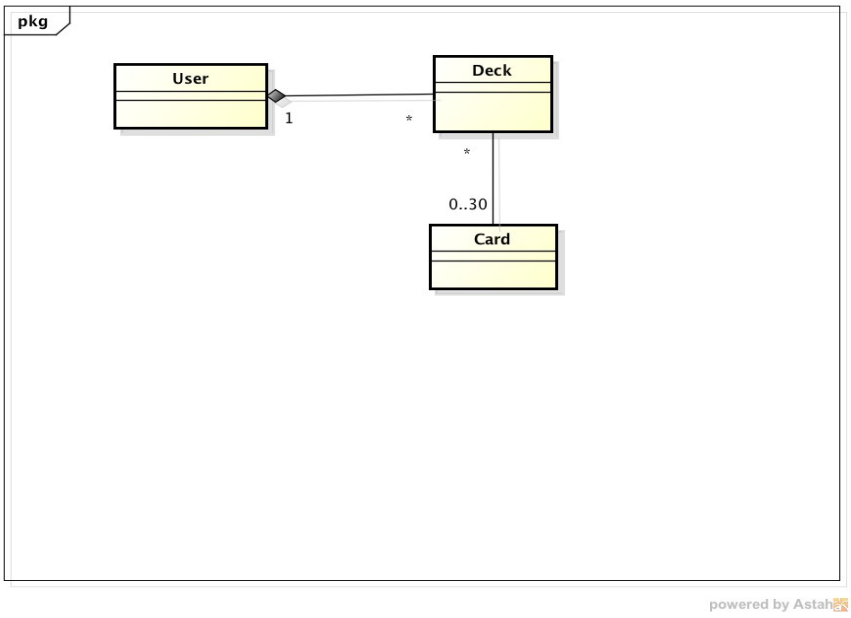
- luoda pakan
  eli valita 30 korttia, joista luodaan uusi pakka
- muokata jotain omaa pakkaansa
  eli vaihtaa pakan nykyisiä kortteja joihinkin muihin
- poistaa jonkin oman pakan
- kirjautua sisään tai ulos
- poistaa oman käyttäjätunnuksensa
- muokata omaa salasanaansa
- kaikki jokamiehen käyttötapaukset (paitsi rekisteröityminen)

Admin:

- lisätä kortteja
  jos pelin päivityksissä tulee lisää kortteja, admin voi lisätä nämä järjestelmään
- muokata kortteja
  jos pelin päivityksissä kortteja muokataan, admin voi korjata järjestelmässä olevien korttien tietoja
- poistaa kortteja
- kaikki jokamiehen ja rekisteröityneen käyttäjän käyttötapaukset (paitsi rekisteröityminen)

3. Järjestelmän tietosisältö

3.1 Käsitekaavio:



3.2 Tietokohteet:
User (tietokannassa nimellä player, koska PostgreSQL:ssä user on reserved keyword..):

Attribuutti	**Arvojoukko**	**Kuvailu**
Name	Merkkijono, max. 20 merkkiä	Käyttäjänimi
Password	Merkkijono, max. 20 merkkiä	Salasana
Admin	boolean	Onko käyttäjä admin vai ei.

Sivuston käyttäjällä voi olla monta pakkaa. Käyttäjällä ei ole pakko olla ollenkaan pakkoja. Jos käyttäjä poistaa käyttäjätilinsä, myös hänen pakkansa poistetaan.

Card:

Attribuutti	**Arvojoukko**	**Kuvailu**
Name	Merkkijono, max. 50 merkkiä	Kortin nimi
Mana cost	Kokonaisluku	Paljonko manaa kortin pelaaminen maksaa
Class	Merkkijono, max. 15 merkkiä	Mihin luokkaan kortti kuuluu
Description	Merkkijono, max. 255 merkkiä	Kortin kuvaus
Attack	Kokonaisluku	Kortin hyökkäysarvo
Health	Kokonaisluku	Kortin health (elinvoima?)

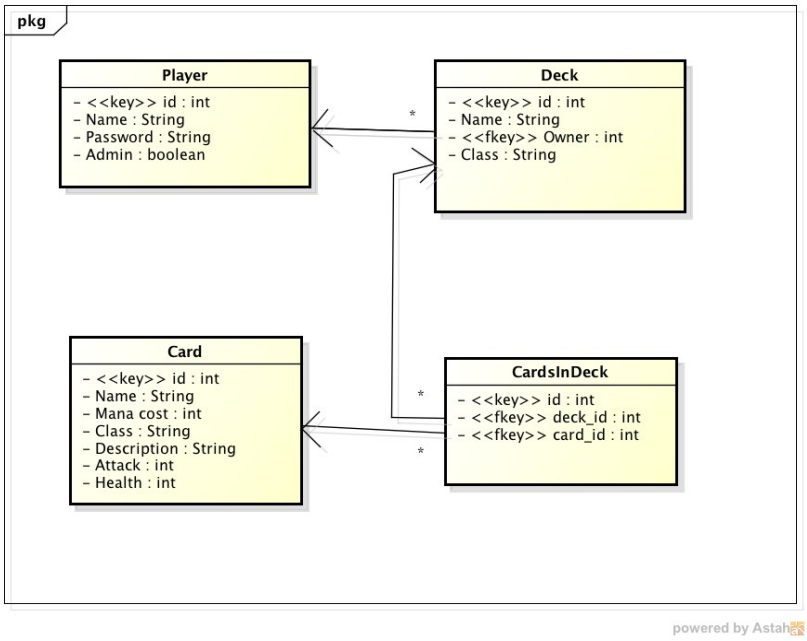
Kortti voi kuulua useaan pakkaan. Kortti voi luokastaan riippuen kuulua vain tietynlaisiin pakkoihin (neutral kortit voivat kuulua mihin vaan pakkaan, mutta esim. mage kortit voivat kuulu vain mage-luokkaisiin pakkoihin).

Deck:

Attribuutti	**Arvojoukko**	**Kuvailu**
Name	Merkkijono, max. 20 merkkiä	Käyttäjän pakalle antama nimi
Owner	Integer	Pakan tehneen käyttäjän id
Class	Merkkijono, max. 15 merkkiä	Mihin luokkaan pakka kuuluu

Pakan omistaa aina sen luonut käyttäjä. Jos käyttäjä poistaa käyttäjätilinsä, myös hänen tekemänsä pakat poistuvat järjestelmästä. Pakkaan kuuluu aina korkeintaan 30 korttia. Jos pakka poistetaan, siihen kuuluvat kortit jäävät yhä järjestelmään (sillä ne todennäköisesti kuuluvat jonkun muun pakkaan yhä, ja muutenkin niiden tulee poistua järjestelmästä vain, jos pelin tekijä Blizzard jossain päivityksessä poistaa kortin).

4. Relaatiotietokantakaavio



5. Järjestelmän yleisrakenne

Sovelluksen rakenne perustuu MVC-malliin. Kaikki kontrollerit löytyvät projektin juuresta (kaikki juuressa olevat php-tiedostot ovat kontrollereja). Näkymät löytyvät hakemistosta views. Modelit löytyvät hakemistosta libs/models/. Yleiskäyttöiset metodit löytyvät hakemistosta libs/.

Kaikki tiedostonnimet on kirjoitettu pienellä. Modeliin liittyvät kontrollerit sisältävät yleensä modelin nimen (poikkeuksena user, tähän palataan myöhemmin). CRUD:iin liittyvät kontrollerit ovat muotoa newmodel (create), models (listaus tietoalkioista), model (yksittäisen tietoalkion sivu), updatemodel (update) ja destroymodel (destroy). Poikkeuksena user-modelin create tapahtuu signup-kontrollerin avulla. Kirjautumisen käsittelevät kontrollerit ovat loogisesti login.php ja logout.php. Index.php:ssä konfiguroidaan, mikä sivu on projektin kotisivu (tällä hetkellä cards, eli index redirectaa cardsiin). Lisäksi kontrollereissa addcards.php ja removecard.php käsitellään pakkaa (lisätään tai poistetaan siitä kortteja).

Näkymät on yritetty nimetä mahdollisimman kuvaavasti. Yleinen käytäntö on se, että tietyn kontrollerin renderöimä näkymä löytyy samalla nimellä views/ kansiosta. Poikkeuksena lomakkeet, joissa on yleensä sana form lisätty perään.

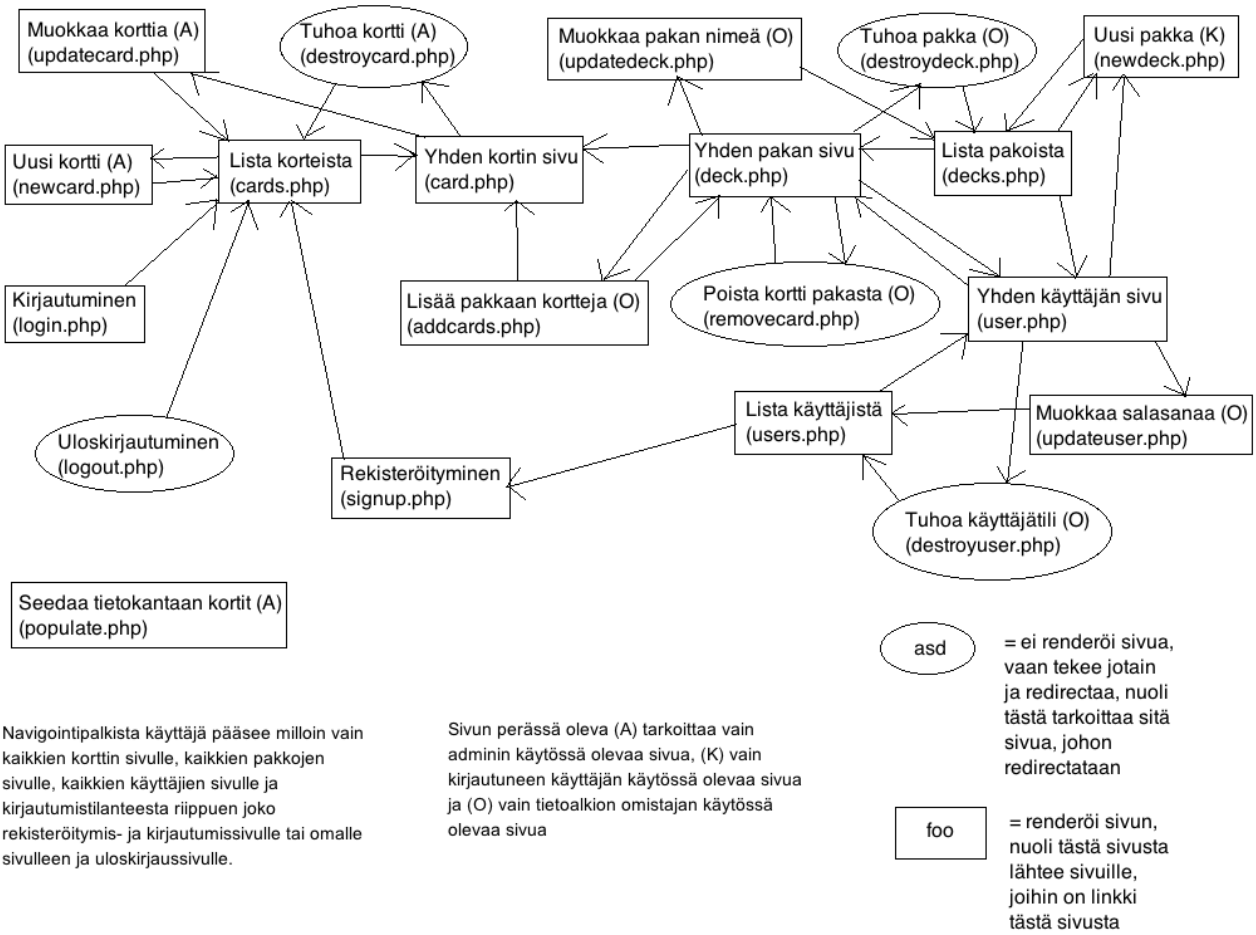
Modulaarisuutta on yritetty noudattaa erityisesti viewien kohdalla. Modeleissa ja kontrollereissa refaktorointiyritykset ovat pääsääntöisesti epäonnistuneet - php ei selvästi tykkää refaktoroimisesta (usein sellainen refaktorointi, joka muilla kielillä, esim. java tai ruby, toimisi mainiosti, onnistuu jotenkin rikkomaan sovelluksen toiminnan). Tämän takia modeleihin ja kontrollereihin jäi hiukan toisteista koodia.

Sovellus käyttää istuntoa kirjautumisen hallintaan. Sovelluksessa on käytetty melko paljon http getin avulla välitettyjä parametreja. Tämä on tehty tarkoituksella http postin sijaan, sillä getillä parametrit voi välittää linkkien kautta (toisin kuin postilla).

Perus bootstrap ja jquery javascript kirjastojen lisäksi sovellus käyttää javascriptiä kahteen tarkoitukseen. /js/confirmdelete.js -javascript-tiedostoa käytetään siihen, että käyttäjältä voidaan kysyä varmistus, ennen kuin hän poistaa pysyvästi tärkeää tietoa (esim. pakan, kortin tai tunnuksensa). /js/filter.js -javascript-tiedostoa käytetään listausten filtteröintiin.

6. Käyttöliittymä

6.1 Käyttöliittymäkaavio:



Navigointipalkista käyttäjä pääsee milloin vain kaikkien korttin sivulle, kaikkien pakkojen sivulle, kaikkien käyttäjien sivulle ja kirjautumistilanteesta riippuen joko rekisteröitymis- ja kirjautumissivulle tai omalle sivulleen ja uloskirjaussivulle.

Sivun perässä oleva (A) tarkoittaa vain adminin käytössä olevaa sivua, (K) vain kirjautuneen käyttäjän käytössä olevaa sivua ja (O) vain tietoalkion omistajan käytössä olevaa sivua

6.2 Sivukartta:

index.php
cards.php
	card.php
		updatecard.php (admin only)
		destroycard.php (admin only, ei renderöidä, pelkkä redirect)
	newcard.php (admin only)
decks.php
	deck.php
		addcards.php (owner only)
		removecard.php (owner only)
		updatedeck.php (owner only)
		destroydeck.php (owner only, ei renderöidä, pelkkä redirect)
	newdeck.php (registered users only)
users.php
	user.php
		updateuser.php (owner only)
		destroyuser.php (owner only, ei renderöidä, pelkkä redirect)
	signup.php (kirjautumaton käyttäjä)
signup.php (kirjautumaton käyttäjä)
login.php (kirjautumaton käyttäjä)
user.php (kirjautuneen käyttäjän oma sivu)
logout.php (kirjautunut käyttäjä, ei renderöidä, pelkkä redirect)

Sivukartan lukemisohje:

1. taso == tämä on navigaatiopalkissa
	2. taso == 1. tasolta on linkki tänne
		3. taso == 2. tasolta on linkki tänne
			jne.
suluissa aina lisätietoa sivusta

7. Asennustiedot

Sovellus pyörii laitoksen users palvelimella. Sovellus käyttää tällä hetkellä PostgreSQL-tietokantaa, eikä tue muunlaisia tietokantoja. Asenna sovellus kopioimalla sen tiedostot palvelimen nettiin näkyvään hakemistoon (esim. usersin htdocs-hakemisto). Huom. palvelimen pitää tukea PostgreSQL:ää. Koska käytetään PostgreSQLää, tietokantayhteyteen ei tarvitse erikseen määritellä asetuksia (riittää, että käyttäjällä on oikeus kirjoittaa kansioihin, jotta tiedostot voidaan kopioida sinne).

8. Käynnistys- / käyttöohje
Harjoitustyön osoite laitoksen users-palvelimella:
http://leju.users.cs.helsinki.fi/hsdb/esittelysivu.html

Sovelluksen kirjautumissivu:
http://leju.users.cs.helsinki.fi/hsdb/login.php

Admin tunnukset:
Username: Admin
Password: salasana

Normaalin käyttäjän tunnukset:
Username: Pena
Password: qwerty

Selaimen tulisi tukea javascriptiä parasta käyttökokemusta varten. Sivuston pitäisi toimia ainakin Google Chromen uusimmalla versiolla ilman extensioneita (e.g. ilman noscriptiä).

9. Testaus, tunnetut bugit ja puutteet & jatkokehitysideat
En ole tehnyt automaattisia testejä sovellukseen. Olen kuitenkin mielestäni testannut kaikken sivujen kaikkia mahdollisia skenaarioita (toimiva syöte, väärä syöte, kirjautunut, kirjautumaton, jne.).

Varsinaisia bugeja en ole nykyisestä versiosta löytänyt. Puutteita on lähinnä toteutuksen ja sisällön laadussa - esim. nyt aseet, loitsut ja kätyrit erotellaan muiden fieldien avulla, niillä ei ole omaa columnia tietokantataulussa. Tämä olisi hyvä lisä. Lisäksi korttien harvinaisuutta (rarity) ei oteta nykyisessä sovelluksessa mitenkään huomioon. Tämä johtuu lähinnä siitä, että löytämässäni jsonissa, jota käytetään tietokannan alustamisessa korttien seedaukseen, harvinaisuutta ei ole, eli jokaiselle kortille pitäisi lisätä harvinaisuus (free, common, rare, epic, legendary) käsin. Tämän takia sovelluksessani yhteen pakkaan voidaan lisätä kaksi samanlaista legendary-korttia, vaikkei sen pitäisi olla mahdollista.

Sovellukseen voisi myös lisätä hyödyllistä tilastotietoa korteista, esim. kuinka monessa tietyn luokan deckissä juuri tämä kortti on.

10. Omat kokemukset
En ollut ennen tehnyt websovellusta php:llä. Olen kuitenkin käynyt Webpalvelinohjelmointi Ruby on Rails -kurssin, joten websovellukset yleensä eivät olleet aivan vieraita.

Php:llä sivuston tekeminen oli tuskastuttavaa verrattuna Ruby on Railsiin, esim. jo 5. kohdassa maintsemani refaktoroinnin vaikeuden takia. Lisäksi laitoksen users-palvelimen PostgreSQL-kanta kaatuili (joka johti sellaiseen bugiin, että kaikki sivut, jotka käyttävät jotenkin tietokantaa, renderöityivät php:n "white screen of death":ina).

Itse koodaaminen oli vaivatonta ja helppoa. Dokumentaation tekeminen oli myös helppoa, mutta silti työlästä. Tämä ei ole mielestäni "hyvä combo", sillä helppojen, mutta työläiden, asioiden tekemisestä ei yleensä jää mitään käteen. Mielestäni kurssi nykyisessä muodossaan on vanhanaikainen - on jännä kokemus käydä Ohjelmistotuotantoa ja tätä kurssia yhtä aikaa.

Ensimmäisellä toitotetaan, että big design up front ja vesiputousmalli ovat vanhoja ja erittäin, erittäin huonoja (Luukkaisen sanoin "paskoja") software developing malleja. Tällä kurssilla niitä tunnutaan noudattavan säntillisesti. Esim. ensimmäisellä parilla viikolla suurin osa ajasta meni pelkkään dokumentointiin. Aluksi luodaan html-demo, ei mitään "aitoa" toiminnallisuutta.

Ehdottaisinkin, että kurssia uudistettaisiin edes sen verran, että dokumentointi tehdään koodauksen jälkeen, eikä ennen. Tämä olisi jo suuri parannus. Nyt (ainakin minulla, ja kuulemani perusteella monella muullakin) kävi niin, että aluksi tehdään suuret suunnitelmat, jotka kuitenkin vaihtuvat koodatessa. Tämän jälkeen dokumentointi on kuitenkin pakko tehdä uudelleen, jotta se

heijastaisi sovelluksen nykyistä tilaa.

Toinen parannus, joka menisi käsi kädessä dokumentaatiomuutoksen kanssa, olisi se, että jo ensimmäiseltä viikolta alkaen aletaan tehdä toimivaa sovellusta. Ensimmäisellä viikolla voitaisiin tehdä vaikka kirjautuminen ja uloskirjautuminen. Seuraavalla tietoalkioiden listaus, jne. Tämä olisi ns. modernien developing designien mukaista, eli yritettäisiin tehdä mahdollisimman nopeasti toimiva osa sovelluksesta, joka voitaisiin näyttää asiakkaalle.

En myöskään oikein ymmärrä, miksi kurssilla ei saanut käyttää tietokanta-abstraktioita. Tietääkseni ennen kurssilla nämä on sallittu? SQL-syntaksia opitaan kuitenkin jo Tietokantojen perusteet -kurssilla. Tietokanta-abstraktioita käytetään kuitenkin sitten työelämässä melkein aina, jos jotain tietokantasovellusta tehdään. Lisäksi kurssi olisi vähemmän pelottava ensimmäisen vuoden opiskelijoille (esim. itse en ottanut sitä viime vuonna, koska php:n syntaksi näytti niin vaikealle, ja Javan servletit vielä vaikeammille). Kuitenkin muuten laitoksella suositaan sellaista menetelmää, että aluksi käytetään abstraktimpia asioita (esim. Java-ohjelmointikurssit ennen Tietokoneen toimintaa), ja sitten opitaan "pellin alla" tapahtuvat jutut.

Kurssilla opitut asiat jäivät lähinnä php:n syntaksin tasolle. Toisaalta tämä on osittain oma vikani, kun käyn kurssia vasta nyt, vaikka mallilukujärjestyksen perusteella se olisi pitänyt käydä vuosi sitten.

11. Liitteet

Ohjelmakoodi, sql-tiedostot ja kaikki muukin projektiin liittyvä tieto löytyy osoitteesta https://github.com/Mession/Tsoha