TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutiteaduse instituut

KOOLITUSTE HALDAMISE SÜSTEEM KEELES JAVA

Bakalaureusetöö
ITV40LT

Üliõpilane: Maarja Lutsar
Üliõpilaskood: 103689
Juhendaja: Jaagup Irve

Tallinn 2015

Autorideklaratsioon

Olen käesoleva töö koostanud iseseisvalt. Kõik koostamisel kasutatud teiste autorite tööd, kirjandusallikatest pärinevad viited ning mujalt pärinevad andmed on viidatud. Tööd ei ole varem kusagil esitatud kaitsmisele.

Autor: Maarja Lutsar

25.05.2015

Annotatsioon

Käesolev töö käsitleb keeles Java kirjutatud koolituste haldamise süsteemi. Süsteem loodi laiendusena juba olemas olevale veebirakendusele, mis koosnes kahest omavahel tihedalt seotud süsteemist.

Süsteem järgib teenusepõhise arhitektuuri põhimõtteid ning kasutab DAO ja MVC disainimustreid. Süsteemi veebiosa on kirjutatud HTMLis, CSSis ning Javascriptis, kasutades AngularJS ja Bootstrap raamistikke.

Süsteemis on neli põhilist objekti: koolitus, koolitaja, moodul ja sertifikaat ning hulk väiksemaid objekte. Nendega sooritatakse CRUD operatsioone.

Süsteem on olnud kasutusel üle aasta, sellest osa rahvusvaheliselt. Selle aja jooksul on süsteemile tehtud üks tõsisem ümberehitus ning hulk väiksemaid muudatusi.

Abstract

This thesis disserts a system for managing trainings written in Java language. The system was created as an expansion of an already existing Web application comprised of two tightly interconnected systems.

The system follows the principles of service-oriented architecture and uses DAO and MVC design patterns. The Web part of the application is written in HTML, CSS and Javascript, using AngularJS and Bootstrap frameworks.

The system has four principal objects: training, trainer, module and certificate and various smaller objects. It performs CRUD operations on them.

The system has been in use for over a year, some of the time internationally. During this time it has been seriously rebuilt once and modified in numerous small ways.

Lühendite ja mõistete sõnastik

HRIS – Human Resources Integration Service, põhiline arendatud süsteem. Moodustab kahest seotud süsteemist koosneva rakenduse back-endi.

EMPL – Employee Portal, HRISi kasutav front-end süsteem.

API – Application Programming Interface, tarkvarakomponendi liides, mille abil teised programmid seda kasutavad; võimaldab komponenti kasutada ilma selle sisemist loogikat teadmata.

AJAX – asynchronous Javascipt and XML, veebiarendustehnoloogia, mis lubab serveriga suhelda ning andmeid muuta ilma lehte uuesti laadimata

CSS – Cascading Style Sheets, veebilehe välimuse ja formaadi kirjeldamiseks kasutatav keel

Sisukord

Jooniste nimekiri

1. Sissejuhatus

Firmas, kus töötan, ei eksisteerinud enne antud projekti algust süsteemi koolituste haldamiseks. Personaliosakond töötles infot töötajate koolituste kohta suuremalt jaolt käsitsi, kasutades andmete hoidmiseks harilikke arvutustabeleid.

Selline süsteem on mõistlik, kui koolitusi on vähe ning infomaht seega väike. Kuna meie vajadused olid suuremad, jõudsime järeldusele, et vaja on kasutada koolituste haldamisele pühendatud süsteemi.

Üks võimalus olnuks kasutada vabavaralist koolituste haldamise süsteemi. Leiti aga, et selle haldamine oleks pikas plaanis keerukam kui täpselt firma vajaduste järgi kirjutatud rakenduse kasutamine. Varasemast eksisteerisid juba Human Resources Integration Service (edaspidi HRIS) ning Employee Portal (edaspidi EMPL): kaks omavahel tihedalt seotud süsteemi, mis haldasid töötajate töötõendeid jm infot. Õigeim tundus neile süsteemidele laiendus kirjutada.

Antud laiendust nimetatakse Training Portaliks. Kuna tegu on EMPLi osaga, on sellele, nagu ka EMPLile võimalik ligi pääseda veebist. See muudab koolituste haldamise lihtsamaks, kuna süsteemi kasutamiseks on vaja internetiühendust ning süsteem hoiab andmeid ühes kohas. Vanas olukorras, kus haldamiseks kasutati põhiliselt arvutustabeleid, pidid haldajad hoolitsema ka andmete sünkroniseerimise eest.

Training Portal hoiab infot koolituste, õpetatava materjali, koolitajate, osalejate ning muu seotu kohta. Kuna HRISis on ka töötajate info, on koolituste ning osalejate seostamine süsteemis triviaalne. Samuti on koolitusel omandatud sertifikaatide töötajatega seostamine üksühese vastavuse tõttu väga lihtne.

Projekti alguses olin ma HRISiga juba mõned kuud töötanud. Oli loogiline, et sain ülesandeks koolituste haldamise süsteemi loomise.

Kogu HRISi poolel olev funktsionaalsuse kirjutasin ma ise, küsides vahel nõu kogenumatelt kolleegidelt. EMPLi poolel olev funktsionaalsus on segu minu ning teiste arendajate tööst.

Rakenduse algusperioodil oli arendamisega väga kiire, seega tegelesid EMPLiga

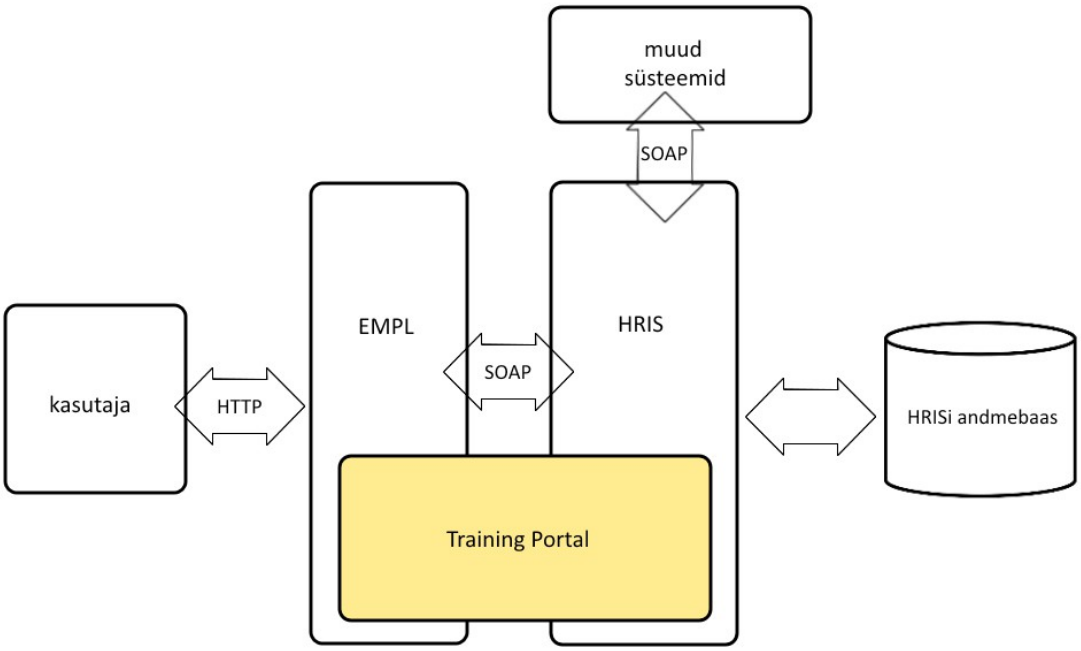
peamiselt kogenumad frontendi arendajad. Lõin kaasa väiksemate funktsionaalsuste loomisel. Kui rakendus funktsionaalseks loeti, jäin mina ainsaks aktiivseks arendajaks projektis.

Esimese tõsise ümberehituse ajal olin juba piisavalt kogenud arendaja, et kirjutada ümber eksisteeriv Java kood ning koos teise arendajaga arendada Javascripti funktsionaalsust.

Sellest ajast peale olen tegelenud EMPLi vigade parandamise ja pisemate funktsionaalsuste lisamisega. Suuremaid ümberehitusi pole rohkem toimunud.

2. Metoodika

Seni haldas süsteem töötajate töötõendeid, sooduskuponge ja isiklikke andmeid. Koolitustega võrreldes oli infomaht väiksem ning info muutus harvemini. Raamistikud, keeled ning praktikad olid süsteemis juba olemas. Töötasin etteantud suhteliselt kitsastes piirides.



Joonis 1. Training Portal HRISi ning EMPLi kontekstis.

Nagu näidatud joonisel 1, koosneb kogu süsteem kahest omavahel tihedalt seotud, ent eraldi süsteemist: HRIS ning EMPL. HRIS käsitab andmebaasi ning sellesse tehtavaid muudatusi. (HRIS suhtleb ka teiste personaliinfosüsteemidega, ent Training Portali osas pole seda tarvis.) Infot väljastatakse SOAP protokolli abil. EMPL kasutab HRISi APIt andmetöötluseks ning töötleb saadud info kasutajatele loetavateks veebilehtedeks.

HRIS on kirjutatud Java programmeerimiskeeles, kasutades Spring raamistikku ning Hibernate teeki. EMPL koosneb Javas kirjutatud back-endist ning Javascriptis AngularJS raamistikku kasutades kirjutatud front-endist.

HRIS suhtleb andmebaasiga, kasutades JDBC ühendust ning Hibernate teeki andmete käsitamiseks. EMPLi (ning ka teiste süsteemidega) suhtleb ta kasutades SOAP protokolli.

Kuna EMPL on veebirakendus, suhtleb kasutaja sellega HTTP abil, ehk kasutab

andmete vaatamiseks harilikku veebilehitsejat.

2.1 Tehnoloogiad

2.1.1 Teenusepõhine arhitektuur

Teenusepõhine arhitektuur (ingl. k *service oriented architecture*) on arhitektuuristiil, mis toetab teenusepõhisust. Teenus tähendab siinkohal kindla lõpptulemusega korratava tegevuse loogilist esitust, mis on autonoomne (self-contained) ning kasutajate jaoks „must kast“. [11]

See tähendab et süsteemi loogika on jagatud teenustesse, mitte ei ole monoliitne blokk. Koolituste süsteemi puhul on teenuseks näiteks koolitaja andmete salvestamine ning lugemine. Teenuse kasutajad suhtlevad selle avalike meetoditega ning ei pea teadma kuidas andmeid teenuse sees töödeldakse, teenus on nende jaoks must kast.

Üksik teenus on piisavalt autonoomne, et seda saaks taaskasutada erinevas kontekstis, kas erinevates kohtades samas süsteemis või eri süsteemides. See tähendab, et kood on paindlikum kui teenusteks jagamata arhitektuur. Ka koodi hooldamine on nii lihtsam.

Teenused suhtlevad omavahel kindlaksmääratud protokolli abil. Koolituste süsteemis on kasutusel nii Java objektid, mille abil suhtlevad teenused omavahel HRISi sees, kui SOAP, mille abil suhtlevad EMPL ning muud personalisüsteemid HRISiga.

2.1.2 Teenusekiht

„Teenusekiht (ingl. k. *service layer*) defineerib rakenduse piiri ning selle poolt pakutavad operatsioonid liidestatud klientkihtide vaatepunktist. See kapseldab rakenduse äriloogika, kontrollides transaktsioone ja koordineerides vastuseid operatsioonide implementatsioonis.“ [3, lk 133-134]

HRISis on kasutatud *operation script* lähenemist teenusekihi implementeerimisele. See tähendab, et teenusekiht on võrdlemisi mahukas ning suurem osa loogikast on implementeeritud just selles. Teenusekihi all olevasse domeenimudelisse on jäetud võimalikult vähe loogikat.

2.1.3 CRUD

CRUD (Create, Read, Update, Delete) on lühend neljast põhilisest funktsioonist, mis

andmebaasirakenduses implementeeritakse. Need lubavad kasutajal andmeid vastavalt luua, lugeda, muuta ja kustutada. [10]

HRIS ei järgi CRUDi täielikult. HRISis sooviti kogu info arhiveerimispõhjustel säilitada. Seetõttu ei ole süsteemis implementeeritud andmete kustutamise meetodit. Objektidel, mis võivad kiirelt vananeda või mille puhul on tõenäoline, et neid soovitakse eemaldada, on olemas võimalus objekt deaktiveerida.

Nendel objektidel on väli `valid`, mis on vaikimisi tõene. Kui objekti soovitakse „kustutada“, määratakse `valid` väärtuseks väär. Selleks on eraldi meetod `Deactivate[Name]`, kui objekti deaktiveerimist läheb vaja sageli. Kui objekte nii tihti ei deaktiveerita, tehakse seda update meetodi abil.

```
public DeactivateCostItemResponse
deactivateCostItem(DeactivateCostItemRequest request) {
      //validation not shown

      CostItem costItem =
costItemDao.fetch(request.getCostItemId());
      costItem.setValid(false);
      costItemDao.update(costItem);
      costItem = costItemDao.fetch(request.getCostItemId());

      DeactivateCostItemResponse response = new
DeactivateCostItemResponse();
      response.setCostItem(mapper.map(costItem,
hris.api.CostItem.class));
      return response;
}
```

Nagu kuluobjekti deaktiveerimismeetodist näha, on deaktiveerimismeetod reaalselt uuendusmeetodi versioon. See valideerib objekti, seab välja `valid` väärtuseks `false`, uuendab info andmebaasis ning tagastab vastuse.

Kuigi deaktiveeritud objektid on andmebaasis olemas, käitub HRISi API, nagu neid ei oleks. Kui deaktiveeritud objekt peaks objekti loomisel unikaalsuskitsendust segama, see reaktiveeritakse (`valid = true`) ning kirjutatakse üle uute andmetega. Enamasti deaktiveeritud objektid eemaldatakse otsingutulemustest enne tulemuste tagastamist.

Juhul kui deaktiveeritud objekte peaks mingil põhjusel olema tarvis leida, on otsingumeetodil eraldi parameeter, mis näitab, kas tagastada ka deaktiveeritud objektid või mitte.

2.1.4 Mudel-vaade-kontroller

Mudel-vaade-kontroller on üks levinumaid kasutajaliidese arendamise mustreid. See jagab kasutajaliidese kolmeks eraldiseisvaks osaks: mudel, vaade ja kontroller.

Mudel on mittevisuaalne objekt, mis sisaldab kõiki andmeid ja käitumist peale selle, mis tegeleb kasutajaliidesega. Vaade on informatsiooni esitus. Vaade ise on staatiline, muudatusi informatsioonis haldab kolmas osa: kontroller.

Mudel-vaade-kontrolleri juures on äärmiselt oluline, et see eraldab andmete kasutajale näitamise mudelist. Samuti see, et esitus sõltub mudelist, kuid mitte vastupidi. Põhjuseid selleks on mitmeid: mudel ja esitus nõuavad erinevat lähenemist, sama mudeli peale võib olla ehitatud erinevaid esitusi ning mudeli testimine on lihtsam kui visuaalse esituse testimine.

Vähem oluline on, et kontroller ja vaade oleksid rangelt eraldatud. Need mõlemad tegelevad andmete kasutajale esitamisega. [3, lk 330-332]

2.1.5 DAO

„Data access object ehk lühemalt DAO on laialt levinud mehhanism püsivuse (ingl. k. persistence) detailide abstraheerimiseks rakenduses. Äriloogika suhtleb DAO kihiga, selle asemel, et suhelda otse andmebaasi, failisüsteemi, veebiteenuse või muu püsivusmehhanismiga, mida rakendus kasutab. DAO kiht suhtleb siis aluseks oleva teenuse või püsivusmehhanismiga.“ [7]

Püsivus tähendab, et andmed säilivad pärast neid loonud protsessi lõppemist. Andmebaasisüsteemis tähendab see üldiselt, et need salvestatakse andmebaasi.

DAOd võimaldavad eraldada püsivuse implementatsiooni äriloogikast. Püsivuse implementatsioon on sageli väga spetsiifiline, kuna see sõltub alloleva mehhanismi detailidest ning seetõttu on selle loogika äriloogikast erinev. Püsivuse implementatsiooni eraldamine DAOdesse tähendab, et kood on kergemini mõistetav. Kui on vajalik püsivusmehhanismi muuta, saab seda teha, ilma et peaks muutma ka äriloogikat.

2.1.6 Kontrollipöördus

„Kontrollipöördus (ingl. k. *inversion of control*) on võtmeosa erinevustes raamistiku ja teegi vahel. Teek on põhimõtteliselt kogum funktsioone, mida välja kutsuda saab ning mis on tänapäeval harilikult klassidesse organiseeritud. Iga väljakutse teeb pisut tööd ning tagastab kontrolli kliendile.

Raamistik kehastab mingit abstraktset disaini ning sellesse on sisse ehitatud rohkem käitumist. Selle kasutamiseks peab kasutaja sisestama endapoolse käitumise erinevatesse kohtadesse raamistikus, kas klasse laiendades või omaenda klasse juurde lisades. Raamistiku kood kutsub neis punktides välja kasutaja koodi.“ [2]

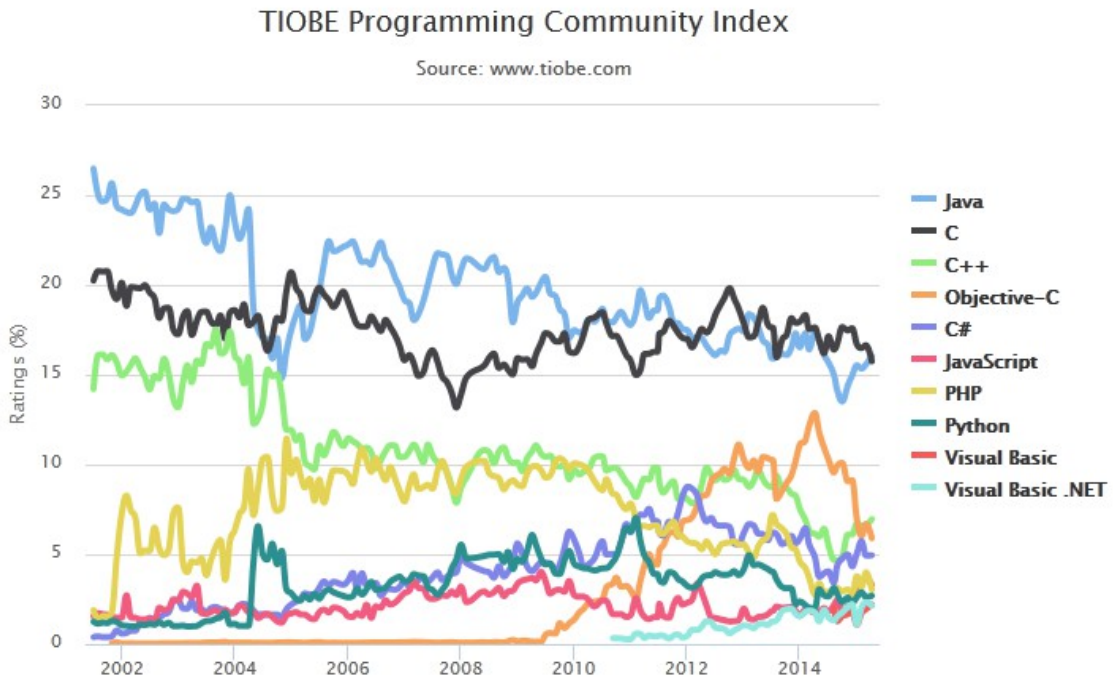
Kontrollipöördus tähendab konfiguratsiooni (kuidas täpselt teha) eraldamist äriloogikast (mida täpselt teha). See tähendab, et konfiguratsioon on koos ühes kohas, mitte laiali üle terve koodi. Seega, kui konfiguratsiooni on tarvis muuta (näiteks süsteemi teise keskkonda üle viimisel), on selle muutmine lihtsam. Samuti on disain ise lihtsam ning kergemini testitav.

Koolituste süsteemis kasutusel olevas raamistikus on kasutatud *dependency injection* versiooni kontrollipöördusest. See tähendab, et klassid ise ei leia oma sõltuvusi, vaid nende ette andmise eest hoolitseb mujal asuv konfiguratsioon.

2.2 Tehnikad

Back-end on nii HRISis kui EMPLis programmeeritud Javas, kasutades muuhulgas Spring raamistikku ning Hibernate teeki.

Nagu näha jooniselt 2, on Java üks populaarsemad programmeerimiskeeli maailmas. See jookseb kõigis levinumates operatsioonisüsteemides ning Javas kasutamiseks on loodud palju erinevaid tööriistu (serverid, raamistikud jne). Seetõttu on see kasutusel suuremas osas firma süsteemidest.



Joonis 2. Populaarsemad programmeerimiskeeled Tiobe indeksi järgi. [8]

Andmeid hoitakse MySQL andmebaasis. Tegu on vabavaralise üsna levinud andmebaasi haldamise süsteemiga. MySQLi kasutamine on samuti firma tava.

Kontrollerid, mis vahendavad front-endis infot mudeli ja vaadete vahel, kirjutati Javas. Süsteemi vanemates osades on kasutusel ka Java Server Page'id ning jQuery, ent koolituste osas kasutatakse pea täielikult AngularJSi.

Veebilehed on kirjutatud HTMLis, kasutades nende vormindamiseks CSSi.

2.2.1 Spring raamistik

Spring oli juba varem firma süsteemides kasutusel. Tegu on vabavaralise Java raamistikuga, mis võimaldab peale kontrollipöörduse näiteks ka integratsiooni andmebaasi ning veebiga. Springi peetakse üheks paremaks vabavaraliseks seda tüüpi raamistikuks.

Koolituste haldamise süsteemis on võimalusel kasutatud annotatsioone, kuna see tegi koodi arendaja jaoks lihtsamini mõistetavaks, kuid konfiguratsiooni võib hoida ka XML-failides.

Kuigi Springil on olemas moodulid andmebaasiga suhtlemiseks, kasutati antud projektis

selleks Hibernate'i.

2.2.2 Hibernate teek

Hibernate on vabavaraline objekt-relatsioonilise kaardistamise teek Java keeles. Hibernate'i abil on võimalik siduda Java objektid andmebaasi tabelitega. Antud projektis on sidumine sooritatud Java annotatsioonidega, ent sidumiseks võib kasutada ka XML-faili.

Hibernate'i eesmärk on vähendada andmete talletamisega seotud ülesandeid, mida arendaja sooritama peab, 95% võrra. See elimineerib vajaduse andmete käsitsi töötlemiseks SQLi ja JDBCd kasutades. [5]

Hibernate'i halbadeks külgedeks peetakse raskusi koodi silumisel ning keerukust. Väga suurte andmehulkade puhul on see ka aeglasem kui puhas JDBC. [6] Samas võimaldab see arendajal vältida puhta JDBC tarvitamist ning keskenduda äriloogikale, mitte andmebaasi haldamisele. Hibernate'i kasutades ei sõltu arendaja enam spetsiifilisest andmebaasisüsteemist ning erinevale andmebaasile üle minemine on võrdlemisi lihtne.

Hibernate genereerib ise SQL päringud ning võimaldab andmeid andmebaasist pärida ning neid sinna kirjutada. See tähendab, et arendaja peab kirjutama minimaalselt SQLi. Kuigi Hibernate'il on eraldi SQLi dialekt (HQL- *Hibernate Query Language*, Hibernate Päringute Keel), ei olnud antud projektis päringute SQLis kirjutamine absoluutselt vajalik. Kõik otsingud sooritati, kasutades kriteeriumipäringuid (ingl. k. *criteria query*), objekt-orienteeritud võimalust päringuid kitsendada.

2.2.3 AngularJS

AngularJS on vabavaraline Javascripti raamistik. Selle abil on võimalik veebilehte dünaamiliselt muuta lihtsamalt kui hariliku Javascripti või jQuery abil, kuna AngularJS toetab kahesuunalist andmete sidumist (ingl. k. *two-way data binding*). AngularJS võimaldab ka veebilehe enda sees kasutada mudel-vaade-kontroller mustrit. Nii on veebirakenduse loomine lihtsam, kiirem ning rakendus ise on selgem. [4, lk vii, 3, 4]

AngularJS ei toeta täielikult vanemaid brausereid. Rakenduse algusaegadel pidi see töötama ka Internet Explorer 8s, mida Angularis enam ei toetatud. Seega võib AngularJSi kasutades tekkida raskusi vanemate Internet Exploreri versioonidega.

Kuna tegu on Javascriptil põhineva raamistikuga, on see tundlik Javascripti tõrgetele.

Kui lehe laadimisel juhtub viga või brauseris on Javascript üldse välja lülitatud, on AngularJSis kirjutatud lehed mittefunktsionaalsed ning inetud. Harilikus Javascriptis või jQuerys kirjutatud veebilehed on selliste tõrgete puhul vastupidavamad ning ei pruugi kogu funktsionaalsust kaotada.

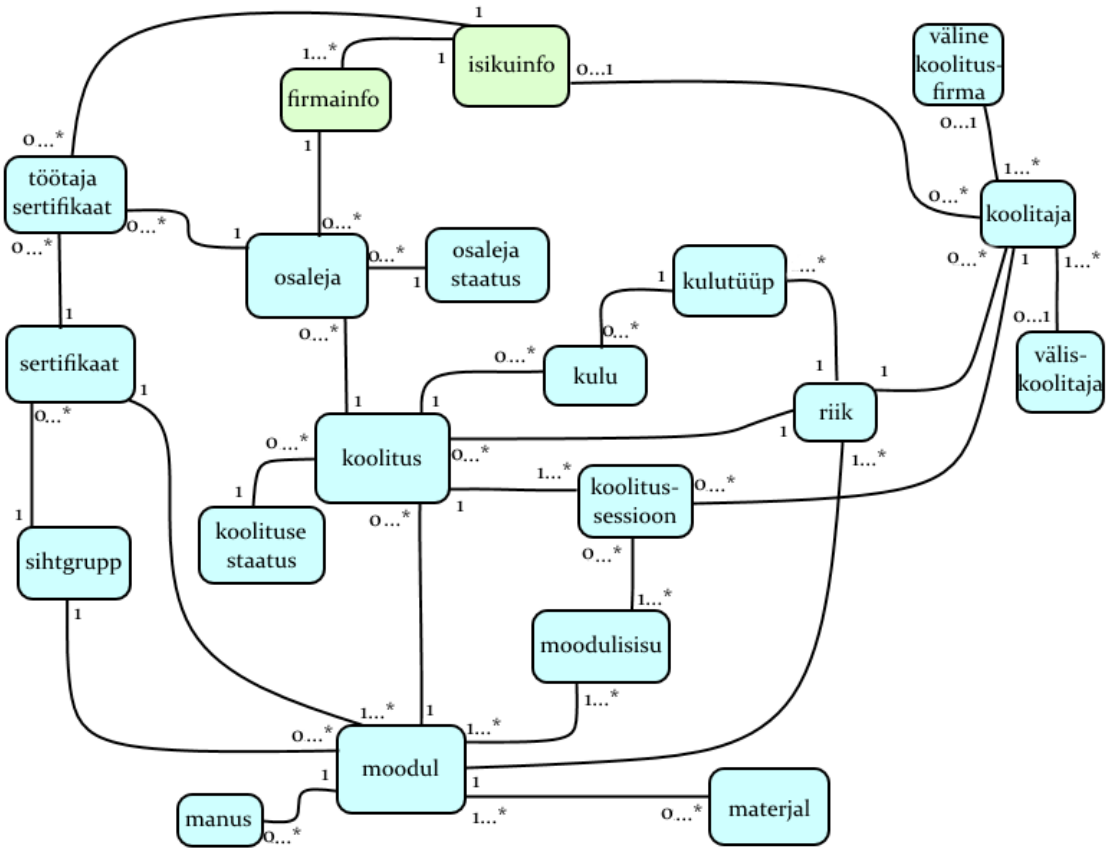
2.2.4 UI Bootstrap

Bootstrap on vabavaraline ning vabatarkvaraline HTMLi, CSSi ning Javascripti raamistik. See on mõeldud töötama sama hästi nii monitoridel kui väikestel ekraanidel (telefonid, tahvelarvutid). Tegu on ühe enimkasutatavama front-end raamistikuga maailmas.

Antud projektis kasutati UI Bootstrapi, ehk AngularJSis kirjutatud Bootstrapi komponente. See tähendas, et Bootstrapist endast oli kasutusel vaid selle CSS. [9]

UI Bootstrapi kasutamine tähendas, et arendajatel oli vaja kirjutada väga vähe CSSi. Arenduse käigus hiiglaslikuks paisunud põhiline CSS fail kahanes pärast Bootstrapi kasutusele võtmist enam kui suurusjärgu võrra, kuna varem täpselt defineeritud ning ümberdisainimiste käigus keerukaks muutunud koodi sai ümber kirjutada kasutades Bootstrapi komponente, näiteks sõnumeid (*alert*), kaarte (*tab*) ja modaale.

3. Rakendus



Joonis 3. Lihtsustatud domeenimudel.

Joonisel 3 on kujutatud rakenduse hallatavad objektid. Sellel on näidatud objektide nimed ning omavahelised suhted. Rohelise taustaga objektid ei kuulu koolituste süsteemi pädevusse.

3.1 DAOd

Kuna HRIS töötab CRUD põhimõttel, on DAOd võrdlemisi lihtsad. Reeglina eksisteerib iga domeeniobjekti kohta eksisteerib üks DAO, milles on implementeeritud andmete loomise, uuendamise ning lugemise meetodid, vastavalt `create`, `update` ja `fetch` ning `get[object]ByCriteria` (näiteks `getTrainerByCriteria`).

Erandiks on `manus` ning `materjal`, mida ei ole loogiline `moodulist` eraldada ning seega ei ole neil oma DAOd, kuigi nad on domeenis eraldiseisvad objektid. Nende infot käsitatakse koos `mooduliga`, mille sisse need kuuluvad.

Samuti on erinevad `koolituse staatus`, `osaleja staatus`, `sihtgrupp` ja `riik`, millel on ainult kõigi objektide tagastamise meetod `findAll`. Neid on harva tarvis kasutada eraldiseisvatena, mitte teiste sama tüüpi objektide kontekstis ning neil on alla kümne võimaliku väärtuse, seega pole neil muid meetodeid tarviski. Need objektid võiksid olla enumeraatorid, mitte tingimata domeeniobjektid. Samas oleks nende väärtusi siis keerukam muuta, mistõttu otsustati domeeniobjektide kasuks.

DAOd kasutavad Hibernate teeki kõikide meetodite jaoks. `Create` ning `update` on väga lihtsad. DAOsse jõudes peab objekt olema piisavalt valideeritud, et selle andmebaasi sisestamine ohutu oleks. Seega võtavad nad argumendina vastu salvestatava objekti, üritavad seda vastavalt andmebaasi salvestada või andmebaasis uuendada ning tegelevad heidetud erindiga (mille põhjuseks on harilikult unikaalsuskitsenduse rikkumine).

`Fetch` on veelgi lihtsam. See võtab argumendina numbri ning tagastab kas objekti, mille identifikaatoriks (unikaalseks id-ks) on see number või nulli, kui sellist objekti ei eksisteeri.

`Get[object]ByCriteria` on veidi keerulisem. See võtab argumentidena objekti, mille järgi otsida, ning lipu, mis näitab, kas tekstiväljade otsimisel tarvitada metamärke (wildcard) või ei. Etteantud objekti järgi konstrueerib see kriteeriumipäringu. Kui objektil on identifikaator, otsitakse objekte selle järgi. Kui identifikaatorit ei ole, otsitakse objekte kõigi ülejäänud mittetühjade väljade järgi.

3.2 Äriloogika ja teenused

Äriloogika on osa programmist, mis tegeleb andmetöötlusega. Kuna see implementeerib ärireegleid, võib äriloogikat nimetada programmi keskseks osaks. Programmi ülejäänud osad tegelevad tehniliste detailidega. [1]

HRISi äriloogika on implementeeritud, kasutades teenusekihi arendusmustrit. See tähendab, et äriloogika on jagatud domeenimudeliks ning selle peal olevaks teenusekihiks. Nii on erineva vastutusega kihid paremini eraldatud. Domeenimudel on lihtsustatult kujutatud joonisel 3.

Operatsioonid, mida andmetega sooritada tuleb, on üsna lihtsad. Peamiselt on tarvis sooritada andmetega CRUD operatsioone. Vaid vähestel juhtudel on tarvis muid, keerukamaid operatsioone.

Koolituste süsteemi teenusekiht on jagatud viieks teenuseks: TrainingModuleService, TrainingCourseService, TrainingService, TrainerService ja CompetenceService. Iga teenus vastutab keskse(te) objekti(de) ning nendega lähedalt seotud mõnevõrra vähemtähtsate objektide eest, sooritades nendel CRU operatsioone (Delete'i HRIS ei kasuta, nagu eelpool mainitud).

Kõige olulisemad neist on kaks esimest, TrainingModuleService ning TrainingCourseService, kuna nende kesksed objektid on süsteemi toimiseks kõige olulisemad. Ülejäänud kolm teenust toetavad kahte esimest ning on eraldatud pigem selguse huvides.

Algselt moodustasid TrainingCourseService, TrainingModuleService ning TrainingService ühe teenuse TrainingService, mis oli teistest märkimisväärselt (umbes kolm korda) suurem. Kui süsteemile esitatud nõuded muutusid ning TrainingService veelgi kasvas, jagati see kolmeks: kahe põhilise objekti põhised teenused TrainingModuleService ja TrainingCourseService ning TrainingService. TrainingService haldab nüüd objekte, mis olid liiga väikesed, et väärida omaette teenust, kuid ei kuulunud selgelt ühegi olemasoleva põhilise teenuse alla.

3.2.1 TrainingCourseService

TrainingCourseService'i põhiline objekt on `koolitus`. `Koolitus` on võrreldav õppeainega ühe semestri jooksul. See tähendab kindaid teadmisi, mis tuleb koolituse käigus omandada. See on väiksem kui õppekava ning sellesse kuuluvad teadmised on omavahel tihedalt seotud. Samuti on sellel ajaline paiknemine, see ei ole lihtsalt abstraktne teadmiste kogum.

Teine oluline TrainingCourseService'i objekt on `koolitussessioon`. Kui `koolitus` ise on võrreldav õppeainega, siis `sessioon` on üks loeng või harjutus selles aines. See on väiksem ning õpetab mingit alamosa teadmistest.

`Koolitusel` on kindlalt pikkus minutites, `toimumisriik`, märked, kas osalejad tohivad ise registreeruda (kui ei, registreerib nad koolitusele juht) ning kas koolitust tuleb näidata firma intranetis, `staatus`(avatud, registreerumine lõppenud jne), seotud `moodul` ning vähemalt üks `sessioon`. Samuti võivad sellel olla nimekiri toimumiseks vajalikest materjalidest (nt projektor või kirjutusvahendid), maksimaalne osalejate arv (kui seda pole, võib osalejaid olla piiramatult) ning ajavahemik, mille jooksul töötaja

saab koolitusele registreeruda (kui see on talle lubatud).

`Koolitusel` on ka muid seotud objekte (`kulud`, `osalejad`). Neid käsitleb aga TrainingService, kuna need on piisavalt suured ning keerukad, et mitte kuuluda otse TrainingCourseService'i haldusse.

`Sessioonil` on kohustuslikult algus- ning lõppaeg, `koolitaja` ja toimumiskoht. Sellel on harilikult ka märkmed, keel, tüüp ning `moodulisisu`. Ühes `sessioonis` võib õpetada mitut `moodulisisu`, kuid peab õpetatama vähemalt ühte.

Igal `koolitusel` peab olema vähemalt üks `sessioon` ning iga `sessioon` peab kuuluma vähemalt ühe `koolituse` alla.

3.2.2 TrainingModuleService

TrainingModuleService'i põhiline objekt on `moodul`. `Moodul` on võrreldav õppekavaga. See tähistab kindlate teadmiste kogumit, mis tuleb omandada, kuid ei ole otseselt seotud õpetamise aja ning kohaga.

Teine oluline TrainingModuleService'i objekt on `moodulisisu`. `Moodulisisu` on võrreldav õppeainega õppekavas. See õpetab mingit alamosa `mooduli` teadmistest ning seda ei saa enam väiksemateks ühikuteks jagada.

`Moodulil` on nimi, kirjeldus, `sihtgrupp` (näiteks klienditeenindajad või juhipositsioonidel töötajad), eesmärk, eeldused, loomise ning viimase uuendamise aeg, märked, kas see on aktiivne, kas selle lõpus tuleb sooritada eksam, kas selle sooritamine annab sertifikaadi ning kas see on heaks kiidetud ning hindamise kriteerium. Kui see on heaks kiidetud, peab sellel olema heakskiiduprotokolli number. Kui moodul annab `sertifikaadi`, peab see olema seotud `sertifikaadiga`.

`Moodul` võib olla seotud `materjalide` ning `manustega`. Kumbagi neist võib olla null kuni mitu. `Materjalid` on välise lingi kujul, ning süsteemis säilitatakse vaid link ise, mitte selle sisu. `Manus` seevastu on süsteemis endas säilitatav dokument.

`Moodulisisul` on kindlasti pealkiri, pikkus minutites, järjekorranumber mooduli sees ning aktiivsuse märge. Sellel võib olla ka kirjeldus.

`Manuse` kohta säilitatakse failinimi, loomise kuupäev ning aadress failisüsteemis. Selle sisu ei hoita aga andmebaasis, kuna suurte failide saatmine oleks aeglane, vaid

eraldiseisvas kaustas samas serveris, kus jookseb EMPL.

Igal `moodulil` peab olema vähemalt üks `sisu` ning iga `sisu` on seotud `mooduliga`.

3.2.3 TrainingService

`Koolitusel` on ka `kulud`. `Kulul` on summa, vääring ning `kulutüüp`. `Kulutüübil` on hetkeseisuga kirjeldus (näiteks koolitaja palk, majutuskulud) ning `riik`, kus see tasuda tuleb. See tähendab, et erinevates riikides tekkinud sarnastel kuludel on erinevad `tüübid`, kuna nende eest ei tasuta samasse kohta. Tulevikus lisatakse `kulutüüpidele` arvenumbrid, kuhu kulu eest tasuda tuleb.

`Osaleja` tähistab töötajat, kes osaleb `koolitusel`. `Osalejal` on `staatus` (registreerunud, kinnitatud jne) ning seosed `koolituse` ja isikuinfoga süsteemis. `Osaleja` all salvestatakse ka info majutuse kohta (kas osaleja vajab seda, kui jah, siis majutuse algus- ning lõppkuupäev) ning kommentaar (nt erivajadused, allergiad vms).

TrainingService'i alla kuulub mitmesuguseid pisemaid objekte: `osaleja seisund`, `koolituse seisund`, `riik`, `sihtgrupp`. Üks võimalus olnuks teha neile koodis enumeraatorid ning vähendada nii tabelite arvu andmebaasis. Andmebaasi rea lisamine või muutmine on aga antud juhul arendajate seisukohalt lihtsam kui enumeraatori muutmine, seega otsustati väikeste objektide kasuks.

3.2.4 TrainerService

TrainerService'i põhiline objekt on `koolitaja`. `Koolitaja` võib olla nii firma enda töötaja kui väljastpoolt palgatud inimene (kuid mitte mõlemat korraga). `Koolitaja` on kindlalt seotud `riigiga`, kus koolitus läbi viiakse. Samuti on tal kindlalt märgitud, kas ta on aktiivne või mitte. Kui koolitaja on firmasisene, on ta seotud töötaja isikuandmetega süsteemis. Kui koolitaja on palgatud väljast, on ta seotud kahe objektiga: `väliskoolitaja` nng `väline koolitusfirma`.

`Väliskoolitaja` on lihtne objekt, kus salvestatakse koolitava isiku andmed: nimi, sünniaeg ja kontaktandmed. Kõik need andmed on kohustuslikud.

`Väline koolitusfirma` on samuti lihtne objekt, kus salvestatakse koolitusfirma andmed: nimi ja asukoht (aadress, linn ja riik). Samuti võib salvestada firma kontaktandmed, ent need ei ole kohustuslikud: tarvidusel võetakse ühendust koolitajaga,

mitte tema firmaga.

`Koolitaja` implementeerimisel oli kaks valikut: laiendada Trainer klassi ExternalTraineriks ning InternalTraineriks või võimaldada Trainer klassis hoida infot nii firmavälise kui -sisese koolitaja kohta, kontrollides, et üks koolitaja ei oleks samaaegselt sisene ja väline. Kuigi esimene valik vastaks paremini objektorienteeritud programmeerimise headele tavadele, on HRISis kasutusel teine valik. See oli algselt implementeerida mõnevõrra lihtsam ning on vähem võimalust, et tekib vigu, mis võiks süsteemi toimimist tugevalt häirida.

Kuigi TrainerService on koodimahu poolest suur (suuruselt teine süsteemis), on ta loogika poolest kõige lihtsam. Objektide vahelisi seoseid on vähe ning objektid ise on samuti lihtsad.

3.2.5 CompetenceService

CompetenceService'i põhiline objekt on `sertifikaat`. `Sertifikaat` on tunnistus, mille saab materjali ettenähtud määral omandanud (st kompetentne) `koolitusel` osalenu. `Sertifikaadil` on kindlalt nimi ja info selle kohta, kas see on aeguv ning kas sertifikaadi saaja saab unikaalse numbriga tunnistuse või ei. `Sertifikaat` on seotud ühe kindla `riigiga` ning tal võib olla kirjeldus. `Sertifikaat` võib, kuid ei pruugi aeguda. Kui `sertifikaat` aegub, võib see juhtuda kas kindlal kuupäeval või kindla arvu päevade järel.

`Sertifikaadil` võib, kuid ei pruugi olla eeldussertifikaate. Ilma eeldusi omandamata ei ole võimalik sertifikaati omandada. `Sertifikaat` peab olema seotud `mooduliga`. Üks ja sama `sertifikaat` võib mõistagi olla seotud mitme erineva `mooduliga` ning üks `moodul` võib anda enam kui ühe `sertifikaadi`.

Teine oluline objekt on `töötaja sertifikaat`, mis implementeerib seost `sertifikaadi` ja `osaleja` vahel. Siia alla kuulub ka sertifikaadi omandamise info: kindlasti omandamise kuupäev ning kas sertifikaat kehtib. Samuti hinne, kui koolituse läbimisel see saadi ning sertifikaadi unikaalne number, kui `sertifikaat` seda nõuab.

CompetenceService on kolmest abistavast teenusest kõige väiksem ning teistest mõnevõrra eraldatum. Teenuses on implementeeritud objektide loomise, muutmise ja lugemise meetodid ning valideeringud. Seega on iseenesest tegu üsna lihtsa klassiga,

mida komplitseerivad veidi sertifikaatide keerukad seosed teiste objektidega.

Töötaja info jaguneb HRISiks kaheks: isiku enda info ning info, mis puutub isiku töötamist kindlas firma osas. Ühel töötajal on ainult üks `isikuinfo`, ent tal võib olla rohkem kui üks `firmainfo` (kui ta töötab näiteks nii Eestis kui Soomes).

Koolitusel `osaleja` on seotud `firmainfoga`, kuna koolitusele saadetakse töötaja mingist kindlast firmaosast. `Töötaja sertifikaat` on seotud nii `osalejaga` (sest `koolituse` läbimine annab `sertifikaadi`) kui `isikuinfoga` (sest omandatud kompetentsust võib rakendada erinevates firmaosades).

3.3 HRIS API

HRIS suhtleb mitmete teiste firma süsteemidega, mis kasutavad töötajatega seotud infot. EMPL seevastu suhtleb ainult HRISiga. Teatavat vähest osa EMPList (koolituste osalist nimekirja) näidatakse firma intranetis, kuid see info on staatilises iframe's ning intraneti poolt ei ole võimalik kasutajal EMPLiga suhelda.

HRISi API, mida EMPL ning teised süsteemid kasutavad, on defineeritud SOAP protokolli abil, mis kasutab formaadina XMLi ning edastab andmeid üle HTTP ühenduse. Nimelt SOAPi kasutab HRIS, kuna oli tarvilik tagasiühilduvus (ingl. k *backwards compatibility*) teiste süsteemidega, mis kasutasid sama protokolli.

APIs on kirjeldatud domeeniobjektide, päringute ning vastuste formaat.

Domeeniobjektid on APIs enamasti samade väljadega, mis HRISi sisestes objektides. Suurim erinevus on, et kõik API objekti väljad on nullitavad, isegi kui sisese objekti väljad seda ei ole. Objektid valideeritakse HRISi teenuses ning kui mõni kohustuslik väli osutub tühjaks, saadetakse vastuses tagasi veateade.

Päringud ning vastused on enamasti formaadis `[meetod][objekt][Request|Response]`, näiteks `DeactivateParticipantRequest` või `GetTrainerResponse`. Nad sisaldavad objekti, päringu autentimist puudutavat infot ning vajadusel otsinguparameetreid (näiteks kas tagastada deaktiveeritud objekte).

Päringu näide: GetTrainerContactsRequest.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:hris="<!--url not shown-->">
      <soap:Header/>
```

```
      <soap:Body>
            <hris:GetTrainerContactsRequest>
                  <hris:Authentication>
                        <!--not shown-->
                  </hris:Authentication>
                  <hris:TrainerContact>
                        <!--Optional:-->
                        <hris:Id></hris:Id>
                        <!--Optional:-->
                        <hris:FirstName></hris:FirstName>
                        <!--Optional:-->
                        <hris:LastName></hris:LastName>
                        <!--Optional:-->
                        <hris:Birthdate></hris:Birthdate>
                        <!--Optional:-->
                        <hris:Phone></hris:Phone>
                        <!--Optional:-->
                        <hris:Email></hris:Email>
                  </hris:TrainerContact>
            </hris:GetTrainerContactsRequest>
      </soap:Body>
</soap:Envelope>
```

See päring on genereeritud programmi SoapUI abil, mida HRISi enda testimiseks kasutan. Reaalses päringus ei ole kommentaare ning tühjad väljad jäetakse täiesti välja, mitte tühjaks.

Näide vastusest: GetTrainerContactsResponse suvaliste andmetega. XML tagid erinevad pisut, seda põhjustab SoapUI.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
      <env:Header/>
      <env:Body>
            <ns2:GetTrainerContactsResponse xmlns:ns2="<!--url
not shown-->">
                  <ns2:Errors/>
                  <ns2:TrainerContacts>
                        <ns2:TrainerContact>
```

```
                        <ns2:Id>2</ns2:Id>
                        <ns2:FirstName>Tõnu</ns2:FirstName>
                        <ns2:LastName>Tamm</ns2:LastName>
                        <ns2:Birthdate>2015-01-
06</ns2:Birthdate>
                        <ns2:Phone>1234567</ns2:Phone>
                        <ns2:Email>tõnut@hot.ee</ns2:Email>
                    </ns2:TrainerContact>
                    <ns2:TrainerContact>
                        <ns2:Id>4</ns2:Id>
                        <ns2:FirstName>Maie</ns2:FirstName>
                        <ns2:LastName>Maasikas</ns2:LastName>
                        <ns2:Phone>1212121</ns2:Phone>
                        <ns2:Email>maie@gmail.com</ns2:Email>
                    </ns2:TrainerContact>
                    <ns2:TrainerContact>
                        <ns2:Id>6</ns2:Id>
                        <ns2:FirstName>Test</ns2:FirstName>
                        <ns2:LastName>Test2</ns2:LastName>
                        <ns2:Phone>9876543</ns2:Phone>
                        <ns2:Email>test@firma.ee</ns2:Email>
                    </ns2:TrainerContact>
                </ns2:TrainerContacts>
            </ns2:GetTrainerContactsResponse>
        </env:Body>
</env:Envelope>
```

3.4 Veebirakendus

Veebirakendus EMPL oli olemas enne koolituste haldamise süsteemi loomist. See võimaldas töötajatel teada saada mitmesugust nendega seotud infot, näiteks kui palju on neil puhkust saada või millises seisus on nende töötõendid.

Koolituste alamsüsteemi loomisel loodi EMPLile juurde koolituste haldamise osa. Selle abil on võimalik vaadata, luua ning muuta süsteemis olevaid andmeid koolituste jm kohta. Ligipääs koolituste alamsüsteemile on piiratud. Hetkel kasutavad seda vaid personaliosakonna töötajad.

Koolituste alamsüsteem nagu ülejäänud EMPL on kirjutatud kasutades mudel-vaade-kontroller arendusmustrit.

3.4.1 Mudel

Mudelikihi moodustavad kaht tüüpi teenused. Esimeste ülesanne on suhelda ekvivalentsete teenustega HRISis. Seda ülesannet täitev kood on harilikult väga lihtne: meetod saab parameetri, loob ning saadab päringu ning tagastab vastusest saadud info.

Teist tüüpi teenustel ei ole HRISis otsest ekvivalenti. Selliseid teenuseid on kolm: TrainingCourseParticipantsExcelService, TrainingCourseParticipantsNotificationService, TrainingCourseParticipantsPDFservice ning TrainingModulePDFService. Nagu nimedest näha, tegelevad need teenused peamiselt osalejatega ning kolm neist loovad eksporditavaid faile.

TrainingCourseParticipantsNotificationService võimaldab saata kõigile koolitusel osalejatele massmeili. Saajateks võivad olla kõik `osalejad`, `koolitaja` ise (et tal oleks ülevaade saadetud kirjadest) või muud meiliaadressid. See teenus kasutab HRISi poolel olevat meili saatmise funktsionaalsust, kuid HRISipoolne funktsionaalsus ei ole unikaalne vaid sellele teenusele.

TrainingCourseParticipantsExcelService loob koolitusel osalejate nimekirja Exceli failis. Hetkel on nimekirjaformaate, mida see loob, kaks. Üks sisaldab nende infot `koolituse` kontekstis (nimi, sünniaeg, majutus, staatus). Teine formaat on mõeldud osalejate haldamiseks, kui koolitus leiab aset paigas, kus osalejatel ei ole õigust omapäi viibida, ning sisaldab nende isikuandmeid (nimi, sünniaeg) ja infot selle kohta, kes nende eest vastutab, millal nad saabusid ja millal lahkusid. Selle formaat on väga täpselt määratud, kuna teised firma süsteemid peavad olema võimelised seda faili vigadeta sisse lugema.

TrainingCourseParticipantsPDFservice loob lihtsa PDF formaadis nimekirja `osalejatest` iga `sessiooni` kohta. Seda kasutatakse osalejalt allkirjade kogumiseks, et oleks teada, kes kohal viibis.

TrainingModulePDFService loob PDF formaadis nimekirja `mooduli` infost, jättes välja materjalid ning manused.

3.4.2 Kontroller

Kontrollerid vahendavad infot mudeli ning vaate vahel. EMPLi puhul on mõnevõrra keerukas tõmmata selget joont kontrolleri ning vaate vahel. Javas kirjutatud kontrollerid suhtlevad Javascriptis kirjutatud teenustega, kasutades AJAXit. Need teenused suhtlevad omakorda harilikke Javascripti objekte kasutades Javascriptis kirjutatud kontrolleritega, mis haldavad AngularJSi kasutades kirjutatud HTML lehti.

Java kontrollerid on siiski Javascripti omadega võrreldes lihtsamad. Nende ülesanne on vahendada infot mudeli ning Javascripti teenuste vahel, samas kui Javascripti kontrollerid on mõlemasuunaliselt seotud vaatega, reageerivad seega kasutaja tegevustele ning teostavad esmast valideerimist.

Üldiselt suhtleb iga vaatelega üks Javascripti kontroller ning üks või kaks Java kontrollerit. Erandiks on `koolitus`, mille detailivaade on teistest märgatavalt suurem ja keerulisem (peale andmete vaatamise saab selles ka meile saata ning PDF ja Excel formaadis faile luua), seega on sellel kontrollereid rohkem.

Javascripti teenused vastavad rohkem domeenimudelile. Eraldi teenused on näiteks ka `osalejatele` ja `kuludele`, millel pole eraldiseisvat vaadet.

3.4.3 Vaade

EMPLis on neli põhilist objekti: `koolitus`, `moodul`, `koolitaja` ning `sertifikaat`, millel igaühel on kaks vaadet: üldine vaade, mis sisaldab filtreeritavat ja sorditavat objektide nimekirja, ning detailivaade, milles saab kindla objekti andmeid vaadata ning muuta. Detailivaadet kasutatakse ka uute objektide loomisel.

Üldiselt hoolitsevad detailivaated samade objektide eest, mis vastava nimega teenused HRISis. `Koolitaja` detailivaates hallatakse ka `välise koolitaja` ja `koolitusfirma` infot. `Mooduli` detailivaates hoolitsetakse ka `moodulisisude` eest. Erandina `koolituse` detailivaates hallatakse peale `koolitussessionide kulusid`, `osalejaid` ning `töötaja sertifikaate`, kuna need kõik kuuluvad loogiliselt `koolitusega` kokku. Tulemusena on see märgatavalt suurem ja keerulisem kui ülejäänud detailivaated.

TrainingService'ile ei vasta ükski vaade. `Osaleja` ja `kulu`, selle kaks tähtsamat objekti kuuluvad `koolituse` detailivaate alla. Väiksematel objektidel pole oma vaadet tarvis.

Vaated on kirjutatud HTMLis, kasutades AngularJSi andmesidumise võimalusi. Lehtede vormindamiseks on kasutusel Bootstrapi CSS, mis annab neile selge lihtsasti arusaadava kujunduse.

3.5 Automaattestid

Ühiktest (inglise keeles *unit test*), on väike automaatselt käivitatav meetod, mis testib ühte diskreetset osa funktsionaalsuses. Neid kasutatakse äriloogika korrektsuse kontrollimiseks. Parimate tavade kohaselt ei kuulu süsteemi vastupidavuse või teiste komponentidega ühilduvuse testimine ühiktestide alla.

Ühiktestid on kasulikud näiteks funktsionaalsuse muutmisel. Kui koodile on ühiktestid juba kirjutatud, on võimalik need pärast muudatusi käivitada ning kontrollida, et muudatused ei ole rikkunud mingit varasemat loogikat.

3.5.1 Testide vajalikkus

Arendatav rakendus oli mahukas ning esialgsed tähtajad lühikesed. Koodis oli palju valideerimist (näiteks kas objekti kohustuslik väli on mitte null väärtus, kas koolituse lõppkellaaeg on pärast koolituse algust jne). Samuti oli kood tolles arendusfaasis muutuv, kuna lisandus uus funktsionaalsus ning ka nõuded muutusid.

Koodi testisid ka inimesed, kuid arvukate valideeringute käsitsi läbi testimine oli ajaliselt mahukas ning nüri töö. Lihtsam oli kirjutada ühiktestid ning nende abil kontrollida, et kõik vajalikud valideeringud on olemas, vigased andmed ei saa kahju teha ning süsteem tagastab just seda, mida peaks.

Näitena `koolituse` valideering koosnes teatud hetkel viiest meetodist, mille pikkus oli kokku üle saja koodirea. Valideering reageeris kümnetele olukordadele, mis võisid koolitusega valesti olla, üritades võimalusel andmeid parandada, ent enamasti lihtsalt erindit heites. Samas heidetakse kogu sellest valideeringust vaid kaht tüüpi erindeid: `MandatoryFieldMissingException` ning `CommonHrisException`.

Kuigi koolituse valideering on kõige mahukam, ei ole teistegi objektide valideeringud väikesed. Üritada neid kõiki käsitsi läbi testida ning leida ka olukorrad, kus viga ei ole intuitiivselt märgatav, on võimalik, ent ajamahukuse poolest ebarealistlik.

Ühiktestidest oli väga palju kasu ka uut funktsionaalsust lisades. Alati ei olnud sisse ilmunud vead kohe märgatavad, ent olemas olevad ühiktestid katsid 76% koolitustega

seotud süsteemiosast. Nende abil oli vigase koodi leidmine lihtsam.

Eriti palju oli testidest kasu ümberehituse käigus. Ilma nendeta oleks kogu funktsionaalsuse pidanud käsitsi läbi testima, mis oleks aga arendustempot tõsiselt aeglustanud.

Praeguse seisuga on koolituste süsteemile kirjutatud 500 ühiktesti. Need katavad kolm viiendikku koodiridadest.

3.5.2 Tekkinud probleemid

Vahel olid tähtajad väga lühikesed, kuna klient soovis tulemusi näha kiirelt. Sellistel puhkudel oli põhiline rõhk funktsionaalsuse valmis saamisel ning aega koodi läbi analüüsimiseks ja uute testide kirjutamiseks ei olnud.

Keerulisemate meetodite puhul võis juhtuda, et ühiktestid ei katnud tegelikult kogu koodi. Mitmed valideeringud heitsid sama erindi (`CommonHrisException`). Oli võimalik, et testandmed olid koostatud ebakorrektselt ning `CommonHrisException`i heitis mõni varasem valideering, mis kasutas sama erindit.

Kuna hea tava kohaselt ei kontrollita ühiktestidega ühilduvust teiste komponentidega, ei katnud ühiktestid andmebaasiga reaalselt tehtavaid muudatusi. Ühiktestidega oli kontrollitud, et andmed, mis andmebaasini jõuavad, on korrektsed. Vead andmebaasiga ühendumisel ning muud säärased eksimused ei kuulunud ühiktestide pädevusse.

3.5.3 Lahendused probleemidele

Lühikeste tähtaegadega ei olnud võimalik midagi otseselt teha. Kirjutasin testid tagantjärele, kontrollides hoolikalt, et vahepeal kontrollitud kood kaetud oleks ning et testid lähtuks nõuetest, mitte koodist endist.

Arenduskeskkond, mida ma kasutasin (IntelliJ IDEA 13.0), võimaldab graafiliselt demonstreerida, kas koodirida läbib mõni ühiktest või mitte. See funktsionaalsus oli väga kasulik valest kohast heidetud õigete erindite leidmisel.

Tulevikus on plaanis luua rohkem teste. Sealhulgas kirjutada ka integratsioonitestid, mille abil saab kontrollida reaalselt andmebaasiga tehtavaid muudatusi.

4. Edasine tegevus

4.1 Rakenduse esialgsest valmimisest praeguse hetkeni

Rakenduse esialgne versioon valmis 2014. aasta alguses. Selle äriloogika erines praegusest märgatavalt. TrainingCourseService'it ning TrainingModuleService'it ei eksisteerinud ning praegu nende alla kuuluvad objektid kuulusid TrainingService'i koosseisu.

Kuigi TrainingService oli suurem kui ülejäänud kolm teenust (umbes kolm korda rohkem koodi), ei tundunud selle väiksemateks teenusteks jagamine mõistlik. Selle alla kuuluvad objektid olid lihtsamad ning nende jagamine erinevatesse teenustesse oleks loonud tarbetult väikesed teenused.

4.1.1 Erinevused esialgses versioonis

Süsteemi varaseimas versioonis tähistas `koolitus` vaid ühte koolituskorda. `Koolitussessiooni` ei eksisteerinud üldse ning ühel koolitusel sai olla vaid üks algus- ning lõppkellaaeg. Selle pikkust mõõdeti minutites, mitte tundides. Muus osas sarnanes koolitus praeguse objektiga.

Enne süsteemi esialgse versiooni valmimist lisandusid `koolitusajad`. Need asendasid `koolituse` algus- ja lõppkellaaja ning toetasid võimalust, et ühel koolitusel on mitu algus- ja lõppaega (näiteks juhud, kus sama koolitust õpetatakse kahes osas järestikustel päevadel). Koolitusaeg oli väga väike objekt, mis sisaldas ainult algus- ning lõppaega ning oli koolitusest endast lahutamatu. Hiljem kasvas just sellest välja koolitussessioon.

`Kulud` ning `kulutüübid` lisandusid koolitusaegadega umbes samal ajal.

`Moodul` sarnanes pigem praeguse moodulisisu kui mooduliga. Seda võis võrrelda õppeainega (nagu moodulisisu nüüd). Moodulil ei olnud infot kinnitamise kohta ega ka manuseid, ent sellel oli minimaalne osalejate arv. Moodul võis olla seotud ainult ühe riigiga. Moodulisisusid ei eksisteerinud üldse ning koolitus oli otse mooduliga seotud.

Esialgses versioonis eksisteeris objekt `programm`, mida võis võrrelda õppekavaga. See sarnanes pisut praeguse `mooduliga`, ent oli lihtsama ehitusega, kujutades endast moodulite kogumit. Sellel oli nimi, kirjeldus, publiku kirjeldus ning märge, kas

programm on aktiivne. Samuti oli see seotud kindla sihtgrupi ning riigiga. Igas programmis oli vähemalt üks moodul ning iga moodul oli seotud vähemalt ühe programmiga.

`Osalejal` ei olnud majutuse infot ega võimalust kommentaare lisada, kuid see sisaldas hinnet. `Töötaja sertifikaadis` ei olnud hinnet ega sertifikaadi numbrit.

Varaseimas versioonis ei olnud koolituse ning osaleja `staatused` eraldiseisvad objektid, vaid atribuudid vastavalt koolituse ning osaleja tabelis. Arenduse käigus osutus see lahendus ebaselgeks ning staatused said eraldiseisvateks objektideks.

4.1.2 Esimesest versioonist ümberehituseni

Esialgne versioon oli kasutuskõlblik 2014. aasta kevadel, valmimisega mõnevõrra hilinedes. Seda kasutas kohalik personaliosakond, kes oli sellega enam-vähem rahul. Suve jooksul ei toimunud suuremat arendust. Parandasin vigu ning lisasin pisemaid funktsionaalsusi, ent tegelesin samal ajal ka nende osadega HRISist ja EMPList, mis ei kuulunud Training Portali alla.

Sama süsteemi sooviti kasutada ka teistes personaliosakondades. Osutus, et neil olid koolitustele väga erinevad nõudmised. Ka kohalik personaliosakond hakkas avastama funktsionaalsusi, mis nende meelest päris õiged polnud. Sügisel oli selge, et süsteem tuleb ümber ehitada.

Ümberehitus kestis novembrist 2014 jaanuari alguseni 2015. Selle käigus valmis süsteemi praegune versioon. Lisandusid moodulisisud ning koolitussessioonid, kadus programm ning mitmed objektid muutusid detailsemaks.

Oluline osa ümberehitusest oli ka HRISi API parandamine. Esialgses süsteemis olid objektid omavahel väga tihedalt seotud ning võis tekkida olukordi, kus objekt X sisaldas objekti Y, ent ka vastupidi Koodis ei põhjustanud see probleeme, ent API selliseid tsükleid ei lubanud.

Kogemuse puudumise tõttu lahendasin ma probleemi, luues objektist mitu versiooni. Enamasti olid probleemsed objektid asendatud id-dega, kuid esines ka juhtumeid, kus need välja jäeti. Kuigi see oli HRISi poolt vaadates lahendus, muutis see API kasutamise raskemaks, kuna süsteemi lähedalt tundmata oli API liiga keerukas. Aru saada, millist versiooni objektist kasutada millises kontekstis, ei olnud triviaalne ülesanne ning kellelgi peale HRISi arendaja ei olnud intuitiivset tunnetust, millal

kasutatakse millist versiooni. Kuna dokumentatsioon oli puudulik, tuli vastav info lihtsalt koodist endast välja lugeda.

Ümberehituse käigus tegin ma API kergemini arusaadavaks. Kuna objektide alternatiivversioone kasutati reaalselt vähe, oli võimalik nendest loobuda. Osutus, et sageli ei pidanudki üks objekt teist sisaldama, kui esimene teises juba sisaldus.

Pärast ümberehitust ei ole süsteemi arendamine katkenud. Kuna süsteemi kasutatakse mitmes erinevas riigis ning igaühes neist on nõuded süsteemile erinevad, tekib uusi vajadusi ning avastatakse seni märkamatuks jäänud vigu.

4.2 Tulevikuplaanid

Alati ei ole võimalik intuitiivselt aru saada, kas objekti deaktiveerimiseks on eraldi meetod või ei. Olemasolev dokumentatsioon on kirjutatud, mõeldes kasutajatele, mitte arendajatele ja seega tundub kiirem kasutada objekti deaktiveerimiseks uuendamise meetodit kui hakata uurima, kas antud objektil eksisteerib deactivate või ei. Samas võtab deaktiveerimismeetodi kasutamine vähem aega. Update meetodite requestis on tarvis kogu objekti, deactivate'is vaid deaktiveeritava objekti unikaalset id-d. Selle probleemi lahenduseks oleks tõenäoliselt korralik dokumentatsioon.

HRISi APIs ei ole hetkel kõikide andmete lugemisel implementeeritud leheküljestamine (ingl. k. *pagination*). Mõne mahukama vastuse tagastamiseks kulub HRISil seetõttu rohkem aega ning nõnda on EMPL mõnel juhul silmnähtavalt aeglane. See häirib kasutajaid. Lahenduseks oleks leheküljestamise implementeerimine kõigi andmete puhul.

Kogu kood ei ole hetkel testidega kaetud. Integratsoonitestide puudus on oluline puudus, kuna mõni viga võib tahes-tahtmata kahe silma vahele jääda. Nende testide kirjutamine on plaanis millalgi tulevikus.

Plaanis on ka viia HRISi ning EMPLi suhtlus üle RESTile. Üheks SOAPi puuduseks peetakse selle aeglust ning suurust, kuna edastatavate andmete kätte saamiseks tuleb parsida palju XMLi. REST oleks tõenäoliselt kiirem. Selleks tuleks HRISile lisada ka teine lõppsõlm, mis kasutaks suhtlemiseks RESTi, mitte SOAPi.

5.Kokkuvõte

Töö käigus laiendasin olemasolevat rakendust, lisades sellele koolituste haldamise alamsüsteemi. Rakendus koosnes kahest tihedalt seotud süsteemist: back-end HRIS, mis tegeleb andmetöötlusega ning front-end EMPL, mis tegeleb veebirakenduse haldamisega. Koolituste haldamise süsteem on nende kahe vahel jaotatud.

Süsteem on kirjutatud keeles Java, kasutades Spring raamistikku ning Hibernate teeki. See järgib teenusepõhise arhitektuuri põhimõtteid ning kasutab DAO ja MVC disainimustreid.

Koolituste haldamise alamsüsteemil on neli põhilist objekti: koolitus, koolitaja, moodul ja sertifikaat. Neile lisandub veel väiksemaid objekte. Objektidega sooritatakse CRU operatsioone (D ei ole antud süsteemis kasutusel).

Süsteemi veebiosa on kirjutatud HTMLis, CSSis ning Javascriptis, kasutades AngularJS ja Bootstrap raamistikke.

Üle poole süsteemist on kaetud testidega See muudab süsteemi arendamise ning hooldamise hõlpsamaks, kuna tekkinud vead on kiiremini märgatavad.

Süsteem on olnud kasutusel üle aasta, sellest osa rahvusvaheliselt. Selle aja jooksul on süsteemile tehtud üks tõsisem ümberehitus ning hulk väiksemaid muudatusi.

Süsteemil on veel arenemis- ning paranemisruumi (näiteks API või dokumentatsiooni osas). Pärast ümberehitust näib see töökindel ning seega võib projekti õnnestunuks nimetada.

Kasutatud kirjandus

1. e-teatmik: IT ja sidetehnika seletav sõnaraamat: äriloogika [WWW] http://vallaste.ee/index.htm?Type=UserId&otsing=%203824 (5.06.2014)
2. Fowler, M. InversionOfControl [WWW] http://martinfowler.com/bliki/InversionOfControl.html (24.05.2015)
3. Fowler, M. Patterns of Enterprise Application Architecture. 19. trükk Crawfordsville : Addison-Wesley, 2013
4. Green, B., Seshadri, S. AngularJS. Sebastopol : O'Reilly Media, Inc., 2013
5. Hibernate Getting Started Guide 4.2.13.Final. Preface [WWW] http://docs.jboss.org/hibernate/orm/4.2/quickstart/en-US/html/pr01.html (25.05. 2015)
6. Hibernate vs JDBC performance [WWW] http://phpdao.com/hibernate_vs_jdbc/ (23.05.2015)
7. Jenkov, J. The DAO Design Pattern [WWW] http://tutorials.jenkov.com/java-persistence/dao-design-pattern.html (5.06.2014)
8. Tiobe Software: Tiobe Index [WWW] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (22.04.2015)
9. UI Bootstrap [WWW] https://angular-ui.github.io/bootstrap/ (25.05.2015)
10. What is Create, Read, Update and Delete (CRUD)? Definition from Techopedia [WWW] http://www.techopedia.com/definition/25949/create-retrieve-update-and-delete-crud (5.06.2014)
11. What is SOA? [WWW] http://www.opengroup.org/soa/source-book/soa/soa.htm#soa_definition (24.05.2015)