TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kalev Riivik 184950IADB

Sööjate märkimise süsteem üldhariduskoolidele

Bakalaureusetöö

Juhendaja: Einar Kivisalu
Magistrikraad

Tallinn 2022

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Kalev Riivik

16.05.2022

Annotatsioon

Käesoleva lõputöö eesmärgiks oli luua Eesti üldhariduskoolidele sööjate märkimise rakendus. Selle rakenduse abil saavad klassijuhatajad lihtsalt märkida õpilasi sööma. Rakenduse hulka kuulub ka kohalolu registreerimise süsteem, mis võimaldab koolil näha kui palju sööjaid tegelikult kohal käib (see aitab säästa nii toitu kui ka raha).

Sarnaseid rakendusi oli juba varem olemas aga ükski ei neist ei lahendanud probleemi täielikult. Olemasolevates rakendustes oli klassijuhataja roll jäetud tahaplaanile või puudus üldse. Kuna üldhariduskoolides on enamasti klassijuhatajad need, kes sööjaid märgivad, siis on vaja rakendust, mis teeks nende töö lihtsamaks.

Lõputöö käigus valmiv rakendus sobib väga hästi näiteks väikestele maakoolidele, kes ei taha palju raha kulutada, et paarisadat õpilast sööma märkida.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 31 leheküljel, 6 peatükki, 11 joonist.

Abstract

Diner Registration System for Comprehensive Schools

The purpose of this graduation thesis was to create web-app for comprehensive (municipality owned) schools in Estonia, that would allow them to manage school lunch registrations. This web-app allows homeroom teachers to register their students for school lunch (or any other meal). Part of the application is an attendance check, that allows the school to see how many students actually show up. The problem often faced in comprehensive schools is that many students are registered for meal, but not all of them will show up (which means wasting food and thus money).

Similar applications did already exist, but they didn't solve the problem very well. In most of them, the homeroom teacher's role was almost (or in some cases – entirely) non-existent. Since in Estonia, homeroom teachers are usually the ones who sign students up for meals, a solution was needed, that would make this task as fast and easy as possible.

Since author failed to find a web-app that would satisfy this (and other) requirements, he had to create one himself. This web-app is not meant to be "one size fits all" (or the "perfect app"). That's why the author decided to make it open source so anyone who wishes, can use it (and make adjustments to fit their needs). That being said, the author takes no responsibility and is not required to provide support for it.

It should be a good fit for smaller schools in countryside, with few hundred students, who may not wish to spend a lot of money on big complicated solutions.

The thesis is in Estonian and contains 31 pages of text, 6 chapters, 11 figures.

Lühendite ja mõistete sõnastik

AJAX	*Asynchronous JavaScript And XML* ehk asünkroonne JavaScript ja XML – võimaldab andmevahetust veebilehitseja ja serveri vahel ilma, et peaks tervet veebilehte iga kord uuesti alla laadima
backend	Toetab kaudselt *frontend*'i (tavaliselt kasutajatel otse ei lubata andmebaasi serveriga suhelda vaid see käib läbi *backend*'i)
CSV	*Comma Separated Values* ehk komaeraldusega väärtused (.csv fail sisaldab andmeid, kus rida koosneb komaga eraldatud väärtustest ja iga rida on uus kirje)
cross-platform	Platvormist sõltumatu või mitmeplatvormiline – rakendus või seade, mis võib töötada mitmel platvormil (näiteks Java rakendused, mis töötavad Windowsi, Linuxi ja Mac OS-i peal)
framework	Raamistik (loodavat platvormi, tarkvara, riistvara, protokolli vms. toetav struktuur
frontend	Veebilehe kasutajaliides, mille abil saab kasutaja igasugu toiminguid veebilehel teha
hash	Räsi - vastava algoritmi e. räsifunktsiooni abil tekstist genereeritud fikseeritud pikkusega arv
HTML	*HyperText Markup Language* ehk hüperteksti märgistuskeel (enamlevinud tekstivorming veebidokumentide loomiseks)
IDE	*Integrated Development Environment* ehk integreeritud programmeerimiskeskkond (tavaliselt sisaldab lähtekoodi redaktorit, kompilaatorit/interpretaatorit ja igasuguseid abivahendeid)
JVM	*Java Virtual Machine* (tarkvara, mis interpreteerib Java keeles kirjutatud programme)
KISS	*Keep It Simple, Stupid* – lihtsuse printsiip (universaalne põhimõte, et mida lihtsam on lahendus, seda parem see on)
LDAP	*Lightweight Directory Access Protocol* ehk lihtsustatud kataloogisirvimise protokoll (komplekt protokolle, mis võimaldavad ligipääsu infokataloogidele)
localhost	Kohalik host – see sama arvuti, kust päring tehti (tavaliselt 127.0.0.1)
MVC	*Model – View – Controller* (Mudel – Vaade – Kontroller. Tarkvara arhitektuurimuster, mis jagab rakenduse 3 omavahel seotud osaks).
MVCC	*Multi-Version Concurrency Control* ehk multiversioon-konkurentsjuhtimine (meetod ühiskasutusega andmebaaside jõudluse tõstmiseks).

MVP	*Minimum Viable Product* ehk minimaalne töötav toode (toote esimene versioon, mille eesmärk on katsetada, kas idee on jätkusuutlik).
nested loop	Mitmetasemeline tsükkel (ehk kui üks tsükkel asub teise sees)
pepper	Pipar (põhimõtteliselt teine sool/*salt*, aga seda ei hoita andmebaasis vaid näiteks rakenduse enda sees. Erinevalt soolast, kasutatakse igal pool sama andmejada)
plaintext	Lahtine tekst (või avatekst) – tekst, mis on n-ö inimloetav (kas siis enne krüpteerimist või pärast dekrüpteerimist).
rainbow table	Vikerkaaretabel (levinud algoritmi põhjal koostatud andmebaas, kus on paljudele räsidele vastavad salasõnad kergesti leitavad)
reverse proxy	Pöördproksi (võtab klientidelt päringuid vastu ning edastab need serveritele ning serveri vastuse saadab kliendile tagasi).
RFID	*Radio-Frequency Identification* (raadiosagedustuvastus)
runtime environment	Käituskeskkond (keskkond, mis võimaldab tarkvaral käivituda – näiteks JVM)
salt	Sool (suvaline admejada, mis lisatakse andmetele/paroolile enne räsifunktsiooni käivitamist. Tavaliselt hoitakse koos räsiga andmebaasis)
skin	Kest ehk kujundus (graafilise kasutajaliidese konkreetne väljanägemine)
strongly typed	Kasutatakse rangemaid muutuja tüübi määramise reegleid (tüübivead tulevad kompileerimise käigus välja).
SQL	*Structured Query Language* ehk struktuurpäringukeel (reeglite kogum, mille alusel konstrueeritakse päringud andmete otsimiseks andmebaasist)
SQL Injection	SQL injektsioon ehk SQL-i süstimine (turvanõrkus, mis lubab ründajal veebivormi abil SQL koodi sisestada)
tag	Märgend (näiteks HTML, XML jms. märgendid)
URL	*Uniform Resource Locator* (internetiaadress)
WAMP	Windows, Apache, MySql ja PHP (arenduskeskkond)

Sisukord

Jooniste loetelu

1 Sissejuhatus

Praktiliselt igas üldhariduskoolis on toitlustamine mingil kujul olemas. Olgu see siis kooli enda söökla, sisseostetud teenus või mõni muu lahendus. Kuna sööma läheb korraga üsna suur grupp inimesi, siis on vaja söök eelnevalt valmis teha. Selleks, et söök (ja seeläbi raha) raisku ei läheks, on hea teada kui palju inimesi korraga sööma läheb. Väga pikalt on kasutatud sellise info talletamiseks kõige tavalisemat paberit ja pliiatsit ning tõenäoliselt on asutusi, mis kasutavad neid endiselt.

Paberi kasutamine ei ole tänapäeval enam otstarbekas. Tavaliselt on sööklas kaustik, kuhu sööjaid märgitakse, mis tähendab, et iga muudatuse jaoks peab klassijuhataja sinna jalutama. Samamoodi ei ole võimalik kindlaks teha, et kes mida muutis ning millal. Isegi kui sööjate märkimisega probleeme ei ole, ei tähenda see, et sööjad õigel ajal ka kohale ilmuvad (eriti kui nad ise toidu eest maksma ei pea).

Käesoleva lõputöö raames analüüsib autor erinevaid võimalusi antud probleemi lahendamiseks, mis sobiks Eesti üldhariduskoolides kasutamiseks. Sobiv lahendus võimaldab elektrooniliselt sööjaid märkida ning ka registreerida, et kes reaalselt söömas käib. See lubab koolidel toidu pealt raha kokku hoida, sest ületootmist on vähem. Sammuti annab see juhtkonnale toimuvast hea ülevaate (kui palju sööma märgitakse, kes kohal ei käi jne.). Rõhku on pööratud sellele, et klassijuhataja saaks võimalikult lihtsalt oma klassi sööjatega tegeleda.

Autor on ise varasemalt Alutaguse vallas üritanud elektroonilist sööjate märkimise süsteemi teha ja see võeti kahes koolis ka kasutusele. See oli aga enne erialase hariduse omandamist ning sisaldab palju vigu. Käesoleva lõputöö raames valmib MVP ehk minimaalne töötav toode [1]. Seda toodet edasi arendades loodab autor Alutaguse vallas kasutatavad puudulikud süsteemid välja vahetada.

2 Loodava rakenduse analüüs

Rakenduse tegemiseks tuleb kõigepealt täpselt kirja panna, et millistele nõuetele antud rakendus vastama peab. Kui nõuded on paigas, siis tuleb otsustada millist tehnoloogiat kasutada. Tehnoloogia valikust omakorda sõltub arenduskeskkonna valik.

2.1 Probleemi ülevaade

Paljude klassijuhatajate jaoks on sööjate märkimine üks tüütu lisakohustus, mida nad peavad tegema. Tihti juhtub ka seda, et mõni õpilane mõtleb mitu korda ümber (või ei saa kooli tulla võistluse/haiguse/olümpiaadi tõttu) ning ka need muudatused tuleb üles märkida. Vahel juhtub ka seda, et klassijuhataja ei saa kohe seda muudatust ära märkida (ei ole aega lihtsalt kohe tegeleda) ja seetõttu ei saa ta hiljem kindel olla, et kas ta viis parandused sisse või mitte. Seega oleks vaja ka sellist võimalust, et kõik huvitatud osapooled (õpilane, klassijuhataja, köök jne.) saaksid kiiresti ja mugavalt vaadata, et kes on sööma märgitud ja kes mitte.

Sööjate märkimiseks on võimalik ka muid rakendusi leida aga need on üldjuhul tasulised ning raskesti kasutatavad. Tihti osutub probleemiks ka see, et rakendus ei ole saadaval eesti keeles. Mõned sööjate märkimise rakendused on osa mingist palju suuremast rakendusest. Suurem osa sellistest rakendustest on üles ehitatud eeldusega, et lapse paneb sööma lapsevanem. Osaliselt on see ka õige, sest vanem määrab ära, et kas ta laps sööb või mitte. Sellistes koolides, kus lapsevanemad (või õpilased) maksavad ise söögi eest on selline lahendus väga hea. Eesti üldhariduskoolides saavad lapsed aga tihtipeale n-ö tasuta toitu (mida vanemad kinni ei maksa). See tähendab aga seda, et lapsevanemad (ega õpilased) selle pärast eriti ei muretse kui laps on sööma märgitud ka sellistel päevadel, millal teda koolis ei ole (kuna nemad ei kaota selle arvelt midagi). Seega on lihtsam laps terveks veerandiks/trimestriks sööma märkida ja sellega rohkem mitte tegeleda. Tänu sellele on paljudes üldhariduskoolides just klassijuhatajad need, kes sööjaid märgivad. On muidugi ka selliseid klassijuhatajaid, kes unustavad (või ei

viitsi) sööjaid söömast maha võtta aga neid saab kool lihtsamalt mõjutada (lapsevanemate puhul on see raske). Sel põhjusel paneb autor oma rakenduses rõhku sellele, et klassijuhatajad saaksid võimalikult vähese vaevaga sööjaid märkida (ja muudatusi teha).

2.1.1 Olemasolevad lahendused

Siinkohal peab autor tähtsaks mainida, et kuigi ta katsus võimalikult täpset infot saada, puudus enamasti otsene juurdepääs vaadeldavatele lahendustele. Seega suurem osa kirjutatust põhineb ekraanipiltidel ja dokumentatsioonil ning mõnel juhul ka kolleegide tagasisidel. Seega on soovitav arvesse võtta, et järgnev info ei pruugi olemasolevate lahenduste tegelikku olukorda täpselt kajastada (dokumentatsioon ja ekraanipildid võivad olla ebatäpsed või vananenud).

Arno on Piksel OÜ poolt arendatav pilvepõhine omavalitsuse haridusteenuste haldamise süsteem [2]. Sinna hulka kuulub ka koolilõuna moodul. Antud lahendus ei sobinud selle pärast, et seal on põhirõhk lapsevanema ja kooli (ja/või toitlustaja) vahelises suhtluses. Lapse registreerib sööma lapsevanem ja kui laps puudub, siis selle märgib ka lapsevanem. Kui eesmärgiks on lapsevanemaga arveldamise lihtsustamine, siis sobib see lahendus hästi. Selles lõputöös käsiteldava probleemi lahendamiseks see aga eriti hea lahendus ei ole.

EduPage on pilvepõhine koolihaldamise süsteem [3]. Sellega saab teha igasuguseid kooli haldamisega seotud asju aga üks enimkasutatavaid osasid sellest on ascTimetables (tunniplaani koostamise rakendus). EduPage hulka kuulub ka söökla haldamise moodul aga selle kasutamiseks peab tellima kolmest paketist kõige kallima (16 € kuu). Ka siin on sööja registreerimine lapsevanema (või lapse enda) ülesanne. Sarnaselt Arnole on ka siin maksmine sisse ehitatud (kui hästi see Eestis töötab ei oska autor öelda). Pilte vaadates tundub aga, et kasutamine on üsnagi ebamugavaks tehtud (näiteks söögikorra tühistamiseks peab administraator kolmest alammenüüst läbi käima). Söökla moodul paistis küll suuremalt jaolt eesti keelde olevat tõlgitud aga kahjuks dokumentatsiooni eesti keeles leida ei õnnestunud. Autor küsis ka ühelt tuttavalt koolijuhilt, et miks nad EduPage söökla moodulit ei kasuta (neil on EduPage keskmine pakett tellitud).

Vastuseks oli, et kuu hind muutuks kahekordseks ja see ei õigustaks ennast lihtsalt ära (eriti arvestades, et nad ei kasuta isegi hetkel kõiki olemasolevaid võimalusi)

IDNetwork pakub „lojaalsuskaartide" lahendust [4]. Seal küll sööjaks (ette) registreerimist ei ole aga seal saab kaardiga registreerida, et kes söömas käis. Autor teab ühte kooli, mis kasutab seda süsteemi. Selle kooli IT juht näitas lühidalt kuidas asi töötab. Süsteem registreerib ära, et kes ja millal oma kaarti sööklas kasutas. Kahjuks ei ole sööjate märkimine seal prioriteet olnud, seega on süsteemist raske infot mugaval kujul kätte saada (tuleb palju käsitööd teha). Saab vaadata logi, et kes, millal ning millise kaardilugeja juures kaarti kasutas. Seega kui on vaja teada, kes näiteks eile lõunat käis söömas, siis tuleks ükshaaval logi läbi vaadata ja käsitsi kirja panna, et kes kohal käis. Positiivse küljena võib välja tuua selle, et kasutajaliides paistis olevat eestikeelne.

Internetist otsides leiab muidugi veel igasuguseid lahendusi mis on (või sisaldavad) sööjate registreerimise võimalust. Näiteks LunchTime [5], HotLunch [6], Vanco [7] ja MySchoolAccount [8] (mis kasutab ka Vanco't). Need lahendused kahjuks ei sobi erinevatel põhjustel. Peamisteks probleemideks on, et need lahendused ei ole klassijuhatajale orienteeritud, on üsnagi ebamugavad kasutada, on tasulised ja enamasti ei ole eesti keelsed.

2.1.2 Lahenduse skoop

Henning Mankell kirjutas *„What doesn't exist you have to create yourself. Even a dream can be plucked out of your head and shaped for a purpose."* [9] („Mida pole olemas, selle pead sa ise looma. Isegi unistuse võib su peast noppida ning eesmärgipäraseks vormida"). Seega tuli sobiv lahendus autoril endal valmis teha.

Probleemi lahendamiseks otsustas autor ehitada veebirakenduse, mille põhieesmärkideks on sööjate märkimise lihtsustamine, juhtkonnale ülevaate andmine ning sööjate kohalolu märkimine.

Klassijuhataja jaoks muutub sööjate märkimine lihtsamaks, kui nad näevad terve klassi sööjaid korraga ning neil on võimalik rea (või tulba) kaupa sööjaid lisada või eemaldada. Näiteks kui on homseks vaja terve klass sööma märkida, siis terve tulba

korraga märgistamine on palju lihtsam kui iga õpilase eraldi sööma märkimine. Kui on paar õpilast, kes ei lähe homme sööma (näiteks puuduvad), siis on ikkagi kiirem terve tulp ära märkida ja need paar õpilast eemaldada.

Õpilane saab ka ennast sööma märkida (ja end sealt eemaldada). Erinevalt klassijuhatajast näevad sööjad ainult enda söömisi (klassikaaslaste omasid ei näe). Nii sööjatele kui ka klassijuhatajatele kehtib sööjate märkimisel ajaline piirang. Samamoodi ei saa nad tagantjärgi sööjaid muuta. Administraator saab määrata, et mitu tundi enne söögikorda registreerimine lukku pannakse (näiteks eelmise päeva lõunast) kuid tema (ning vaatleja rolliga kasutaja) saavad ka tagantjärgi muutusi teha.

Kohalolu kontroll tuleb RFID (*Radio-Frequency Identification*) kaartidega. Sööja näitab kaardilugejale oma kaarti ning süsteemis märgitakse, et ta on söömas käinud. Kui sööja mingil põhjusel kaarti kasutada ei saa (unustas koju, läks katki jne.), siis saab söökla personal ta kohe kirja panna. See võimalus peab olema ka tagantjärgi juhuks kui näiteks Internet ära kaob.

Aruandluse kohapealt sai lisatud ainult kõige tähtsamad aruanded. Eelkõige sellised aruanded, mis näitavad kui palju sööjaid oli klasside kaupa ja kokkuvõtted. Koka jaoks sai tehtud ka leht, kust ta näeb kui palju sööjaid igale söögikorrale iga päev registreeritud on (Joonis 1). Hiljem on võimalik lisada juurde keerulisemaid aruandeid vastavalt sellele, mis infot (ja mis kujul) tarvis on.

Lõunasöök

mai 2022	2	3	4	5	6	9	10	11	12	13	16	17	18	19	20	23	24	25	26	27	30	31	
← →	E	T	K	N	R	E	T	K	N	R	E	T	K	N	R	E	T	K	N	R	E	T	Kokku
1. - 4. kl	0	0	0	0	0	0	0	0	0	0	0	0	2	0	1	0	2	0	0	0	0	0	5
5. - 12. kl	0	0	0	0	1	1	1	1	2	1	3	0	0	1	2	1	1	1	1	1	1	1	20
Kokku	0	0	0	0	1	1	1	1	2	1	3	0	2	1	3	1	3	1	1	1	1	1	25

Joonis 1. Koka vaade.

Administraator tegeleb kasutajate lisamisega ning haldab söögikordasid, gruppe ja erandeid. Rakenduse seadetes on määratud, et mis päevadel asutus tavaliselt lahti on (koolide puhul tavaliselt E – R). Administraator võib lisada erandeid, et teatud kuupäevaks ei ole võimalik sööjaid märkida (pühad jms.) või vastupidi, et kui mingil põhjusel tulevad õpilased laupäeval kooli.

2.2 Rakenduse nõuded

Rakenduse nõuded jagunevad funktsionaalseteks ja mittefunktsionaalseteks. Funktsionaalsed nõuded täpsustavad tarkvara funktsionaalseid aspekte (ehk mida rakendus peab tegema). Mittefunktsionaalsed nõuded, milline rakendus peab olema (üldised omadused, mitte konkreetsed funktsioonid) [10].

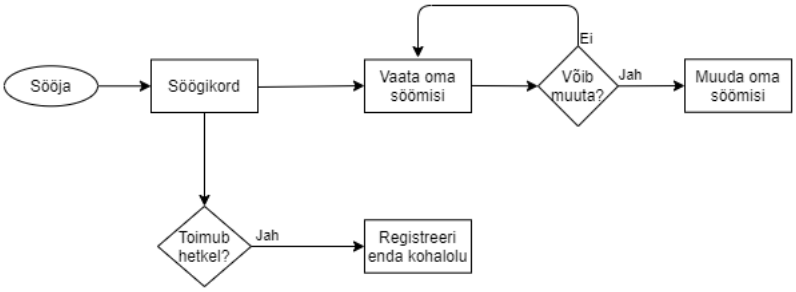
Rakendust kasutavad eri rollidega inimesed. Seetõttu toob autor nõuded välja rollide kaupa. Rollid on järgnevad: sööja (nii õpetaja kui ka õpilane), klassijuhataja, kokk, vaatleja ja administraator

Nõuete koostamisel ja täpsustamisel osales autor erinevatel koosolekutel. Räägitud sai nii koolijuhtide, klassijuhatajate, õpetajate kui ka köögitöölistega. Paljud nõuded tulevad ka autori isiklikus kogemusest, sest autor on üle kümne aasta üldhariduskoolis töötanud ning antud probleemiga kokku puutunud.

2.2.1 Funktsionaalsed nõuded

Sööja nõuded (Joonis 2):

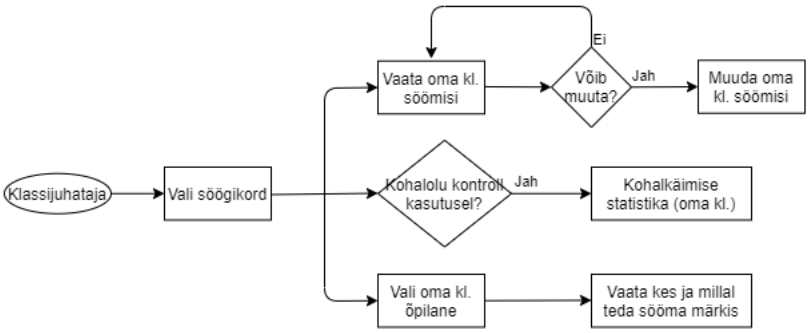
- Sööjana tahan ma sisse ja välja logida.
- Sööjana tahan ma näha kõiki erinevaid söögikorra tüüpe, kuhu mind sööma saab märkida (hommikusöök, lõuna jne.).
- Sööjana tahan ma näha mis ajaks mind sööma on märgitud.
- Sööjana tahan ma ennast ise sööma ja söömast maha märkida.
- Sööjana tahan ma näha kes mind sööjaks märkis ja millal.
- Sööjana tahan ma sööklas RFID kaartiga oma kohaloleku registreerida.



Joonis 2. "Sööja" rolli kasutajakogemuse diagramm.

Klassijuhataja nõuded (Joonis 3):

- Klassijuhatajana tahan ma oma klassi sööjaid lisada ja eemaldada.
- Klassijuhatajana tahan ma näha iga õpilase kohta, et kes ja millal teda sööma märkis või söömast maha võttis.
- Klassijuhatajana tahan ma näha kes mu õpilastest käis söömas ja kes mitte.

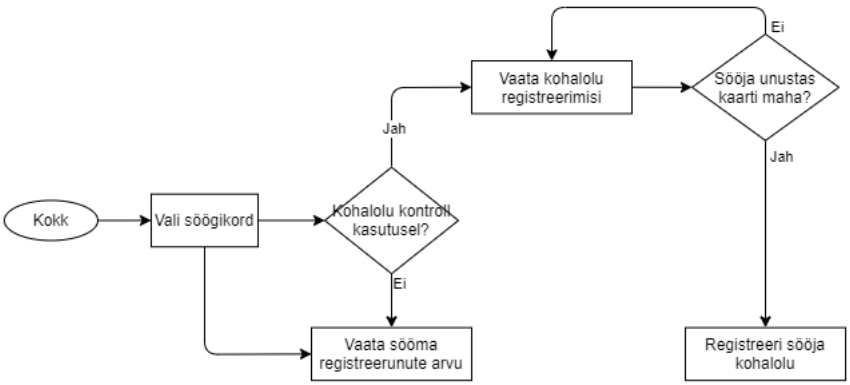


Joonis 3. "Klassijuhataja" rolli kasutajakogemuse diagramm.

Koka nõuded (Joonis 4):

- Kokana tahan ma ülevaadet, et kui palju sööjaid antud söögikorrale registreerunud on.
- Kokana tahan ma näha klasside kaupa sööma registreeritud kasutajaid, et vajadusel kontrollida kas kasutaja on ennast kirja pannud või mitte.

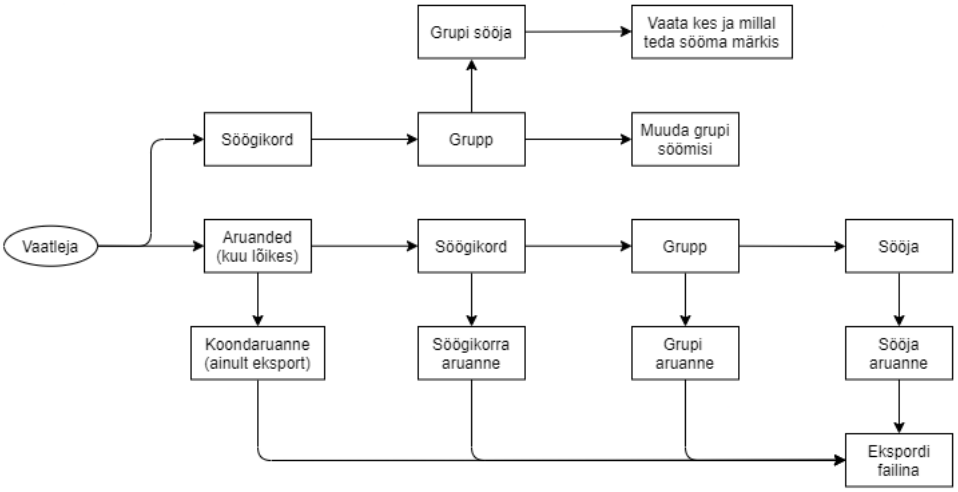
- Kokana tahan ma vaadata, kes sööjatest on oma kohalolu registreerinud (ka tagantjärgi).
- Kokana tahan ma sööja kohalolu registreerida, kui sööja unustas oma kaarti maha (kui kool kasutab kohalolu kontrolli).



Joonis 4. "Koka" rolli kasutajakogemuse diagramm.

Vaatleja nõuded (vaatleja on administraatori alamliik. Joonis 5):

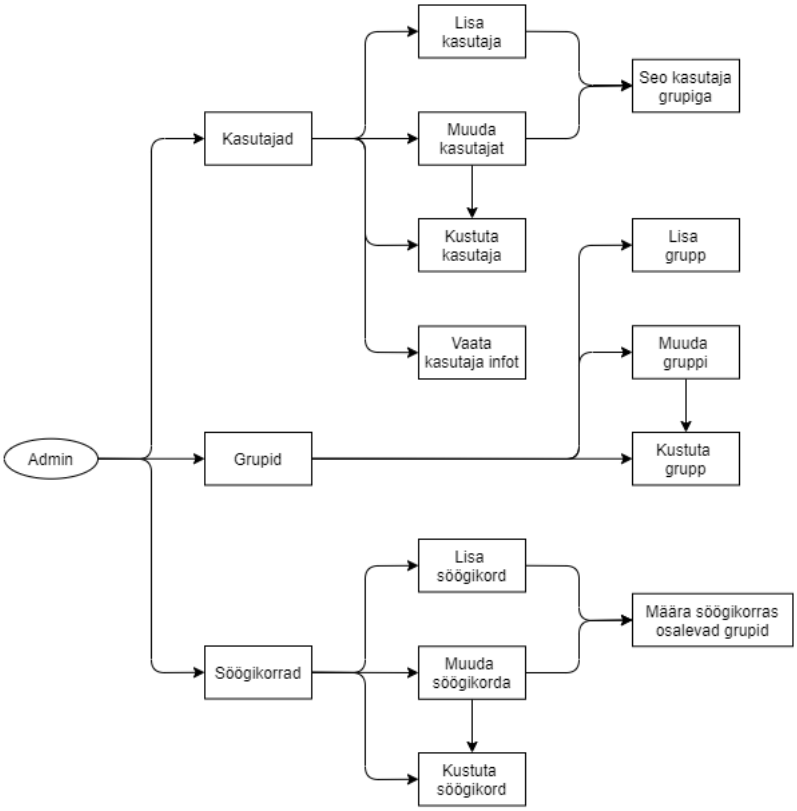
- Vaatlejana tahan ma teha samu asju mida kokk.
- Vaatlejana tahan ma, et söögikorra ajalimiit mulle ei kehtiks (et ma saan sööjaid lisada ning eemaldada enne sööki, söögi ajal ja ka peale sööki).
- Vaatlejana tahan ma olla võimeline ükskõik millise sööja kohta vaatama kes ja millal teda lisas/eemaldas söömast.
- Vaatlejana tahan ma näha kuu lõikes kokkuvõtteid/aruandeid söömise kohta grupi kaupa.
- Vaatlejana tahan ma eksportida Exceli tabelina aruandeid söömise kohta.



Joonis 5. "Vaatleja" rolli kasutajakogemuse diagramm.

Administraatori nõuded (Joonis 6):

- Administraatorina tahan ma teha samu asju mis vaatleja.
- Administraatorina tahan ma söögikorra tüüpe lisada, muuta ja kustutada (k.a. söögikorra kellaaeg ja mitu tundi enne söögikorda sööjate ja klassijuhatajate jaoks registreerimine lukku pannakse).
- Administraatorina tahan ma kasutajaid lisada .
- Administraatorina tahan ma kasutajainfot muuta (kasutajanimi/email, klass, kas kasutajal on õigus sisse logida jne.)
- Administraatorina tahan ma kasutajat märkida „kustutatuks” (ehk ära kustutada nii, et temaga seotud andmed jäävad alles).
- Administraatorina tahan ma, et oleks võimalik juba kustutatud kasutajat taastada.
- Administraatorina tahan ma sööjate gruppe lisada/muuta/kustutada (näit. „1. klass” ja „Õpetajad” grupp jne.).
- Administraatorina tahan ma sööjate gruppe söögikordadega siduda (ja lahti siduda).



Joonis 6. "Administraatori" rolli kasutajakogemuse diagramm.

2.2.2 Mittefunktsionaalsed nõuded

Kuna mittefunktsionaalsed nõuded kehtivad enamasti rohkem kui ühele rollile korraga, siis ei ole mõtet neid rolli kaupa grupeerida.

- Veebilehe kasutamine peab olema lihtne ja loogiline.
- Asutus peab saama otsustada kas ta soovib sööjate kohalolu kontrolli kasutada või mitte.
- Suurem osa kasutajaid tohib näha ainult enda andmeid.
- Veebileht peab korralikult töötama nii arvuti kui ka mobiili peal.
- Andmete lisamisest/muutmisest/kustutamisest peab kuskile jälg maha jääma.
- Kõik kasutajad peavad olema autenditud.

- Rakenduse koodi hoitakse avalikus repositooriumis ja kõik soovijad võivad seda vabalt kasutada.

2.3 Töövahendite valik

Töövahendite valik osutus küllaltki keeruliseks. Et valimine kergemalt läheks otsustas autor keskenduda populaarsematele tehnoloogiatele ja eelkõige sellistele, millega tal kokkupuude juba olemas oli.

Kõigepealt tuli otsustada, et kas ehitada üks suur terviklik veebirakendus või teha eraldi eesrakendus (*frontend* [11]) ja tagarakendus (*backend* [12]). Kuna antud veebirakendus väga keerukas ei tule, siis otsustas autor, et on lihtsam ehitada üks terviklik veebirakendus. Vajadusel saab sinna lisada üksikuid elemente ka *frontend/backend* tehnoloogiast (näiteks *AJAX* päring teha JavaScriptiga, mis toob serverist infot veebivormi jaoks vms.).

2.3.1 Keelte valik

Enne kui programmeerimiskeele valimiseks läks, tuli otsustada, et kas kasutada mõnda raamistikku *(framework)* või mitte [13]. Kuna rakendus ise väga keeruline ei ole, otsustas autor, et raamistiku kasutuselevõtt muudab asja keerulisemaks kui see peab olema. Kui selle projektiga oleks mitu arendajat tegelenud või kui autor oleks mõne raamistiku kasutamises juba vilunud, siis oleks asjal veel ehk mõtet olnud aga antud juhul oleks suur osa ajast läinud raamistiku enda (ning selle eripära) õppimise peale.

Kuigi veebilehte on võimalik teha paljude eri keelte abil, valis autor programmeerimise keele nende keelte hulgast, millega ta varem oli kokku puutunud. Täiesti uue keele õppimine (isegi kesk-tasemel) oleks väga ajakulukaks osutunud. Seega valikus olid järgmised keeled:

- C# - Microsofti poolt arendatud objekt-orienteeritud keel. Selle arhitektuur ühendab Java ja C++'i parimad omadused. Seda keelt kasutatakse peamiselt veebirakenduste ja Windowsi rakenduste tegemiseks (ka osad mängumootorid kasutavad seda) [14].

- Java – Sun Microsystems poolt välja lastud platvormist sõltumatu (*cross-platform*) objekt-orienteeritud programmeerimiskeel. Java programmid kompileeritakse baitkoodideks (.class failideks) ning neid interpreteerib JVM (*Java Virtual Machine* ehk Java Virtuaalmasin), mis muudab baitkoodi masinkeeleks (protsessorile arusaadavateks juhisteks). Seda keelt saab kasutada paljude asjade valmistamiseks (veebirakendused, Androidi rakendused, arvuti rakendused jne.) [15].

- JavaScript – Netscape Corporation'i poolt arendatud programmeerimiskeel. Tegemist on interpreteeritava keelega (ei kompileerita tervet programmi korraga masinkeelde vaid transleeritakse üks rida korraga ja seejärel see rida täidetakse ning nii iga reaga [16]). Kõige sagedamini kasutatakse seda selleks, et muuta veebileht dünaamiliseks (*frontend*'is). JavaScripti saab kasutada ka *backend*'i ehitamiseks [17].

- PHP – Rasmus Lerdorfi poolt loodud keel [18]. Tegemist on serveripoolse skriptimiskeelega (skriptimiskeelt võib lugeda programmeerimiskeele alamliigiks), mida kasutatakse veebilehtede loomiseks. PHP on väga laialdaselt levinud keel. Peamiselt kasutatakse seda veebilehtedele dünaamilise sisu lisamiseks (näiteks kui veebileht peaks midagi arvutama või infot andmebaasiga vahetama jne.). Erinevalt JavaScriptist (kus tavaliselt muudetakse lehte otse brauseris), koostab PHP serveris vajaminema HTML (HyperText Markup Language) lehe ning saadab selle siis kasutajale [19].

Nendest keeltest otsustas autor koheselt loobuda Javast ning JavaScriptist. Asi ei olnud selles, et need keeled halvad oleksid olnud, lihtsalt Java ja JavaScriptiga (terve rakenduse) ehitamise kogemus autoril praktiliselt puudub. Seega peamine valik oli PHP ja C# vahel.

PHP plussiks oli järgnev. Esiteks oli autori poolt varem tehtud sööjate märkimise portaal tehtud PHP-ga. Kuigi esimesel versioonil programmeerimise häid tavasid ei jälgitud, on PHP kasutamine pisut lihtsam kui teiste keelte puhul. Suureks miinuseks oli aga see, et autor ei ole ühtegi PHP raamistikku kasutanud ning uue raamistiku õppimine oleks väga ajakulukas tegevus olnud. Raamistikest kaalus autor Laraveli [20], sest see on üks (kui mitte kõige) populaarsemaid PHP raamistikke.

C# plussiks oli see, et ülikoolis kasutas autor seda päris tihti. Teine pluss oli see, et hajussüsteemide aines autor katsetas sööjate registreerimise lehe loomist, mis tähendas, et osa koodi oleks saanud taaskasutada. Kuna ülikoolis kasutati peamiselt ASP.NET Core raamistikku, siis tundus see ka lõputöö jaoks loogiline valik olevat. See on küll pigem autori isiklik eelistus aga C# / ASP.NET Core tundus ka selle pärast parem, et see on *strongly typed* (see aitab suure osa tüübi vigadest juba kompileerimise käigus üles leida [21]).

ASP.NET Core üheks suurimaks miinuseks autori jaoks oli see, et kui PHP on suuremal osal veebiserveritest juba olemas (ning uue rakenduse lisamiseks tihti piisab failide kopeerimisest kausta), siis ASP.NET Core kasutamiseks tuleb *runtime environment* (käituskeskkond) kõigepealt paigaldada ning *reverse proxy* (pöördproksi) kasutusele võtta, sest Kestrel (ASP.NET Core kasutatav veebiserver) ei luba sama IP ja pordi peal mitut veebilehte hostida (isegi kui rohkem veebisaite ei ole, soovitatakse *reverse proxy*'t sellegi poolest kasutada *[22]*).

Väga raske oli otsustada, kas kasutada C# või PHP-d. Alguses proovis autor ASP.NET ja C# kasutades ülikoolis tehtud projekti edasi teha. Mida rohkem autor selle kallal töötas, seda rohkem ilmnesid probleemid. Peamiseks probleemiks oli see, et hajussüsteemide aines tehtud „algeline" versioon oli liiga keeruline ja autor ei saanud kõigest 100% aru. Näiteks kui tuli välja, et rakenduse tegemisel on mõni suur loogikaviga sisse tulnud, mis sunnib veidi suuremat muudatust tegema, siis selle muudatuse tegemine oli väga raske ja võttis palju aega (sest muudatusi tuli teha paljudes eri kohtades ning mõnes kohas läks tükk aega enne kui asi korralikult tööle hakkas). Autor otsustas kasutada KISS printsiipi [23], mis ütleb, et süsteemi disain peaks olema võimalikult lihtne.

Liigne keerukus oleks tekitanud veel ühe probleemi. Nimelt on tõenäosus väga suur, et kui autoril lõputöö kaitstud, siis rohkem ta selle rakendusega tegeleda ei oleks tahtnud. Seega otsustas autor lõpuks ikkagi PHP kasuks. Teine suur põhjus oli see, et PHP lehti on (üldjuhul) lihtsam paigaldada (sest enamusel veebiserveritest on PHP olemas). Raamistikke otsustas autor mitte kasutada. Üheks põhjuseks oli see, et autor ei ole PHP raamistikke varem kasutanud aga see ei olnud peamine põhjus. Palju tähtsam oli, et autori arvates õpib ta tavalist PHP koodi kirjutades palju rohkem kui raamistikku

kasutades. Raamistik sunnib asju teatud viisil tegema aga tihti jääb arusaamatuks, et miks/kuidas see viis parem on.

2.3.2 Andmebaasi valik

Andmebaasi valik ei olnud eriti raske. Esimeseks valikukriteeriumiks oli andmebaasi hind ja teiseks paigaldamise keerukus.

Vaatluse all olid järgnevad andmebaasid:

- MySQL – on 1995 aastal välja lastud (osaliselt avatud lähtekoodiga) relatsioonilise andmebaasi haldamise süsteem. Alates aastast 2010, kuulub see Oracle korporatsioonile (ning tänu sellele ei ole kogu lähtekood avalik) [24]. Tegemist on ühe populaarseima andmebaasitarkvaraga.

- MariaDB – on 2010 aastal MySQL-ist tekkinud haru. See on täielikult avaliku lähtekoodiga. Hetkel on veel väga sarnane MySQL-iga, kuid pakub rohkem funktsioone (võimalusi) [25].

- MS SQL Server – on 1989 aastal Microsofti poolt välja lastud relatsioonilise andmebaasi haldamise süsteem. See kasutab T-SQL'i (Transact-SQL), mis on Microsofti enda loodud keel ja põhineb SQL-il (*Structured Query Language*). Tegemist on tasulise tarkvaraga (Express versioon on tasuta aga üsnagi suurte piirangutega) [26].

- Oracle DB (*Oracle Database*) – on 1977 aastal Lawrence Ellisoni (ja teiste arendajate) poolt arendatud objekt-relatsioonilise andmebaasi haldamise süsteem (hübriid, mida saab kasutada nii objektorienteeritud andmebaasi haldurina kui ka relatsioonilise andmebaasi haldurina). Tegemist on ühe (kui mitte kõige) populaarseima andmebaasitarkvaraga [27]. Ka siin on tegemist tasulise tarkvaraga (sarnaselt MS SQL Serverile on olemas tasuta versioon, mis on üsnagi suurte piirangutega) [28].

- PostgreSQL – on UC Berkley poolt loodud (avatud lähtekoodiga) relatsioonilise andmebaasi haldamise süsteem (BSD litsentsiga), mis toetab mingil määral ka objekt-orienteeritud toiminguid. See kasutab MVCC-d (M*ulti-Version*

Concurrency Control), mis lubab mitmel kasutajal korraga samas süsteemis andmeid lugeda ja kirjutada [29].

MS SQL Server langes kohe välja, sest tasuta versioonil on terve rida piiranguid. Kõige silmatorkavam oli see, et andmebaasi maksimum suurus on 10 GB. On küll kaheldav, et söökla rakendus nii palju ruumi võtab aga milleks riskida. Samal põhjusel langes välja ka Oracle DB.

Seega jäi valik PostgreSQL ning MySQL-i/MariaDB vahele. PostgreSQL-i ei ole autor varem kasutanud ning selle paigaldamine ning seadistamine on keerulisem kui MySQL-i/MariaDB oma. Kuigi PostgreSQL-iga tuleb kaasa rohkem võimalusi (ja võimalus keerulisemaid päringuid teha), siis autor peab tähtsamaks seda, et andmebaas oleks kiire, stabiilne ja kergesti hallatav [30].

MySQL-i ja MariaDB vahel valides otsustas autor MariaDB kasuks. Hetkel on need kaks omavahel vahetatavad (sest MariaDB põhineb MySQL-i lähtekoodil ja mõlema autor on sama isik – Michael „Monty" Widenius). Mida aeg edasi läheb, seda suuremaks muutub tõenäosus, et see enam nii ei ole. Lõplik valik tuli MariaDB kasuks, sest tegemist on MySQL-i täiustatud versiooniga, mis on kiirem ning kasutab GPL litsentsi [31]. MySQL-i puhul heidutas natuke ka see, et seal võib tekkida huvide konflikt (Oracle DB ja MySQL kuuluvad mõlemad samale firmale).

2.3.3 Väliste teekide kasutamine

See oli üsnagi keeruline teema. Ühest küljest ei ole mõtet jalgratast leiutama hakata (ehk kui keegi teine on sobiva koodi juba valmis kirjutanud, siis miks mitte seda kasutada). Samas liigse teekide kasutamisega kaasnevad omad probleemid. Kõige ilmselgem on see, et ehitatav rakendus sõltub siis nendest teekidest. Kui mõne teegi arendaja otsustab, et ta enam ei uuenda oma teeki (või lihtsalt selle katki teeb), siis mõjutaks see ka autori tehtavat rakendust. Seega oli eesmärgiks leida tasakaal. Piisavalt teeke, et tähtsad asjad tehtud saaks, kuid mitte liiga palju.

Esimene teek, mille autor kasutusele võttis oli PHRoute [32]. See teek võtab kõik sissetulevad päringud (põhimõtteliselt URL-id ehk internetiaadressid) vastu ja vaatab (regulaaravaldisi kasutades), kas selline aadress on rakenduses kirjeldatud. See on hea,

sest seda teeki kasutades ei pääse kasutajad enamusele .php failidele otse ligi vaid kõik lehed serveeritakse läbi index.php faili (kus on lubatud aadressid ära kirjeldatud). Teine hea omadus sellel teegil on see, et seal saab aadressidele (sisemiselt kasutatavaid) lühinimesid anda ja koodi sees nende abil suunamisi teha (tuntud kui *reverse routing*). Näiteks võib tuua, et kui autor otsustas, et kõikide kasutajate vaatamiseks tuleb */users* aadressi lõppu kirjutada aga hiljem otsustatakse, et /kasutajad oleks ikka parem, siis on võimalik see lihtsalt ära muuta (ilma, et peaks rakenduses kõiki suunamisi ümber tegema).

Teine teek, mille autor kasutusele võttis oli Twig [33]. Veebirakendust kirjutades on hea tava hoida äriloogika ja kuva loogika lahus. Põhjusi selleks on mitmeid (näiteks Terence Parri kirjutatud artiklist [34] tooks ma välja järgmised põhjused).

- Esimese põhjusena võib välja tuua kapselduse (*encapsulation*), ehk rakenduse välimus asub täielikult mallides ning äriloogika asub mudelis (*model*) Üsnagi populaarne on näiteks MVC (*Model – View – Controller* ehk mudel – vaade - kontroller) lähenemine. Lihtsustatud seletus oleks, et mudel haldab andmeid, vaade kuvab neid andmeid kasutajale ning kontroller juhib suhtlust mudeli ja vaate vahel [35].

- Teiseks põhjuseks on selgus. Mall ei ole programm, mille tulemusena valmib HTML fail vaid see ongi ise põhimõtteliselt HTML fail (lihtsalt lüngad on sisse jäetud kuhu andmed tulevad ning minimaalselt loogikat).

- Kolmas põhjus on tööjaotus. Graafilised disainerid saavad malle ehitada samal ajal kui programmeerijad rakendust arendavad. Disaineril kerge lehe välimust muuta ilma programmeerijat segamata.

- Neljas põhjus on komponentide taaskasutus. Nagu programmeerijad jagavad suuri meetodeid väiksemateks tükkideks, saavad ka disainerid malle jagada erinevateks alam-mallideks. Näiteks kui disainer kasutab igal pool samasuguse välimusega nuppu, siis võib nupust alam-malli teha ja seda ühte malli mitmes kohas kasutada (mis tähendab ka seda, et saab kõiki nuppe ühest kohast muuta).

- Viies põhjus on erinevate kujunduste kasutamine. Kui muuta ainult välimust (mitte andmeid), siis võib disainer mitu kujundust ehk *skin*'i teha ja programmeerija ei pea asjasse sekkuma.

Autor oleks malle võinud ka puhta PHP-ga teha (ehk HTML faili kirjutada *<?php echo $muutuja ?>*) aga seda oleks väga raske lugeda olnud. Kui kasutada paari lihtsat PHP *tag*'i (märgendit), ei ole hullu, aga kui kasutada mitmetasemelisi tsükleid (*nested loop*) ja *if-else* tingimusi, muutub asi kiiresti loetamatuks. Peale Twig-i kaalus autor ka Smarty [36] kasutamist aga see tundus vanem ja rohkemate piirangutega olevat (näiteks ei luba Smarty PHP-d otse mallis kasutada, mida üldjuhul ei soovitatagi teha aga võimalus võiks igaks juhuks olemas olla).

2.4 Arenduskeskkonna valik

Kõigepealt tuli otsustada, et kus rakenduse koodi hoida. Seejärel tuli luua autori arvutisse arenduse jaoks sobilik keskkond (et koodi saaks samas arvutis kohe ka käivitada). Lõpuks tuli valida välja ka tarkvara, millega koodi kirjutada.

2.4.1 Versioonihaldus süsteem

Programmeerimine käib küll arvutis aga koodi hoidmiseks on soovitatav kasutada versioonihaldust (kuhu saab koodi jaoks koodivaramu e. repositooriumi teha) [37]. Peamiseks põhjuseks oli see, et kui arvutiga midagi peaks juhtuma, siis ei lähe andmed kaduma. Kasuks tuli ka, et kuna versioonihaldusega on väga lihtne varasemat seisu taastada, siis võis julgelt koodi kirjutada kartmata, et kood parandamatult katki läheb.

Kolm tuntumat versioonihalduse süsteemi, millega autor kokku puutunud oli, olid GitHub [38], Bitbucket [39] ja GitLab [40]. Ülikoolis oli autor kasutanud GitLab-i (TTÜ oma) ja Bitbucketit (mille konto sai ühe kursuse raames tehtud). Nende hulgast parima valimise peale autor eriti aega raiskama ei hakanud. Põhjuseid selleks oli kaks. Esimene oli see, et autori teadmised / kogemus versioonihalduse kasutamisel oli üsnagi algeline (ehk ta kasutas ainult põhifunktsionaalsust, mis oli kõigis kolmes olemas). Teine põhjus oli see, et autor kaalus ka seda, et kui rakendus on nii kaugel, et seda saaks

kuskil reaalselt kasutusele võtta, siis teha täiesti uus repositoorium (kus algse arenduse jälgi näha ei oleks).

Autor valis Bitbucketi oma repositooriumi hoidmiseks. Põhjusi oli kaks. Esiteks oli konto juba olemas ja teiseks saab seal tasuta ka privaatseid repositooriumeid teha (kuigi antud rakenduse lähtekood on avalik, soovis autor kõiki oma repositooriumeid samas kohas hoida). Põhimõtteliselt oli autoril konto ka GitLabis olemas aga kuna tegemist oli ülikooli serveriga, siis seda ei riskinud autor kasutada (sest sealt tõenäoliselt kustutatakse konto ära kui autor ülikooli lõpetab). GitHub-i ei hakanud autor isegi kaaluma, sest seal saab tasuta teha ainult avalikke repositooriumeid.

2.4.2 Kohaliku keskkonna loomine

Arenduskeskkonna paigaldamiseks kasutas autor tarkvaralahendust nimega Wampserver [41]. See on **W**indowsi tarkvara, mis sisaldab endas **A**pache-t, **M**ySql-i ja **P**HP-d (sealt ka nimi WAMP). Tegelikult sisaldab see nii MySql-i kui ka MariaDB-d (mida autor ka kasutas). Autor tegi Apache veebiserverisse virtuaalse hosti nimega canteen.demo ja määrast hosts failis selle IP aadressiks 127.0.0.1 (mis on sama arvuti ehk *localhost* [42] aadress). Tänu sellele saab autor rakendust otse enda arvutis käivitada, mitte ei pea iga muudatuse vaatamiseks seda kuskile Internetti üles laadima.

PHP juures otsustas autor kasutada versiooni 8.x, sest selliseid probleeme ei õnnestunud antud versiooni juures tuvastada, mis planeeritud rakendust otseselt mõjutanud oleks. Arenduse alguses paigaldatud WAMP versioon ei sisaldanud veel PHP 8.x versiooni aga selle lisamine ei olnud eriti keeruline.

2.4.3 Koodi kirjutamise tarkvara

Koodi kirjutamiseks on mõttekas kasutada IDE tarkvara (*Integrated Development Environment* ehk integreeritud programmeerimiskeskkonda) [43]. Siiamaani oli autor kasutanud PHP kirjutamiseks kahte erinevat IDE-t. Enne ülikooli (ja ka ülikoolis) kasutas autor PhpStorm-i [44] ja praktikal olles oli kasutusel Eclipse [45]. Autori jaoks oli tähtsal kohal kasutamise lihtsus ja mugavus. Tähtis oli ka see, et kui „tark” IDE on (ehk kui hästi ta oskab arvata mida kasutaja teha üritab ja selle põhjal soovitusi pakub). Autori jaoks oli mugavam kasutada PhpStorm-i ning suur osa oligi just sellel abistaval

funktsionaalsusel. Eclipse sisse oli ka sarnast tarkust ehitatud ja ta oskas soovitusi pakkuda, et mida kasutaja (tema arvates) kirjutada üritab aga PhpStorm tegi seda lihtsalt paremini. Siinkohal tuleks kindlasti mainida, et seda hinnangut mõjutas ka see, et PhpStormiga on autor kauem töötanud (ja oskas seda paremini kasutada kui Eclipse-i). Siiski oli üks suur asi, mille poolest Eclipse oli tükk maad parem ja selleks oli hind. Eclipse oli tasuta aga PhpStorm oli (individuaalseks kasutamiseks) 89 € aastas või 8.90 € kuus (organisatsioonide jaoks veel kallim). Autoril oli küll tasuta PhpStorm-i litsents aga selle kasutustingimused jäid natuke segaseks. Seega võttis autor kasutusele Eclipse'i.

Andmebaasiga suhtlemiseks otsustas autor kasutusele võtta HeidiSQL-i [46]. Tegemist tasuta tarkvaraga ja autor on selle kasutamisega juba harjunud. Põhiliselt sai seda kasutatud selleks, et proovida SQL käske otse andmebaasi vastu ning selleks, et tabelitest (ja nende sisust) hea ja lihtsa ülevaate saaks. Kui näiteks PHP koodi kirjutatud SQL lause andis veateate või käitus mingil muul viisil imelikult, siis piisas sellest, et SQL HeidiSQL-i kopeerida ning too andis veidi täpsema veateate või oli päringu vastusest näha, et vale info tuli tagasi. Kuna autor oli selle programmiga juba tuttav ja teadis, et see saab soovitud ülesannetega hakkama, siis ei olnud mõtet hakata aega kulutama alternatiivide uurimise peale.

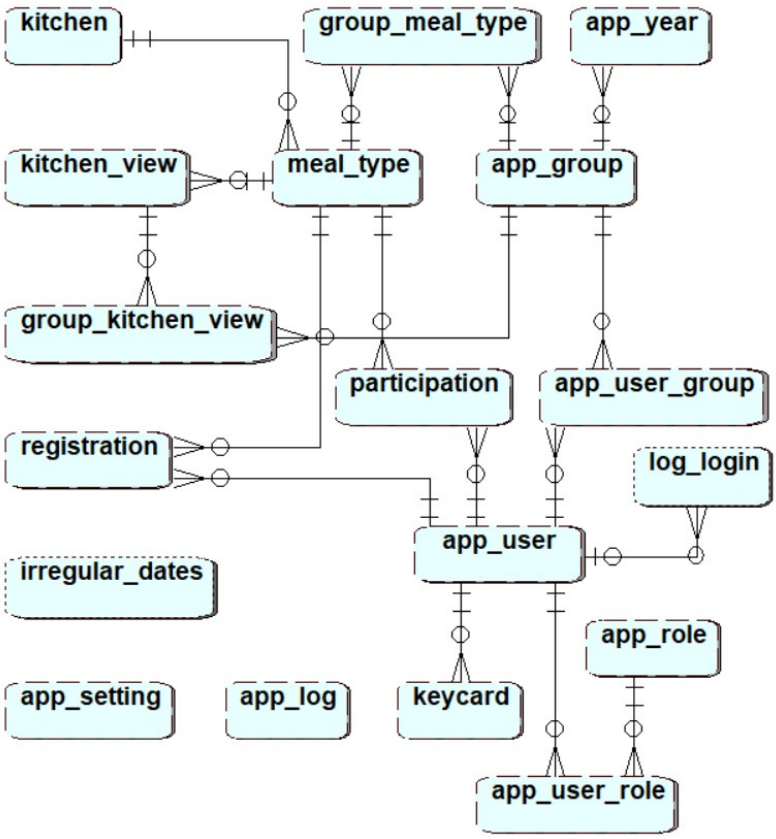
3 Veebirakenduse arendus

Autor ehitas veebirakenduse, mis lahendab käesolevas lõputöös püstitatud probleemi. Kui aus olla, siis oli rakenduse arendamine tunduvalt pikem (ja raskem) protsess kui autor algselt arvas.

3.1 Andmebaasi kasutusele võtmine

Enne kui andmebaasi sai reaalselt looma hakata tuli see valmis disainida. Seega tegi autor lihtsustatud kavandi, et milline projekti andmebaas võiks välja näha. Kuna tegemist oli prototüübiga, siis oli juba ette teada, et seda on vaja töö käigus muuta. Kõigepealt tuli luua tühi andmebaas ja sinna luua rakenduse jaoks kasutaja (koos vajalike õigustega).

Andmebaasi disainimiseks kasutas autor programmi nimega QSEE SuperLite [47]. Joonis 7 kujutab lihtsustatud versiooni andmebaasi olemi-suhte diagrammist. See diagramm on täiskujul nähtav Lisas 2. Andmebaasi tabelite ja väljade nimesid valides katsus autor vältida reserveeritud sõnu (mida kasutatakse SQL lausetes). Enamus tabeleid sisaldavad väljasid *created_by*, *created_at*, *deleted_by* ja *deleted_at*. Nende väljade kasutamine lubab kohe ära fikseerida, kes ja millal kirje sisestas (või kustutas). Andmebaasist tegelikult andmeid ära ei kustutata. Selle asemel lisatakse kuupäev (ja kellaaeg) *deleted_at* lahtrisse. Kasutaja näeb, et andmed kustutatakse ära aga andmebaasis on info alles. See võimaldab hiljem kindlaks teha, et kes kustutas ja vajaduse korral kustutatud infot otse andmebaasist vaadata.



Joonis 7. Lihtsustatud olemi-suhte diagramm.

Autor lisas projekti .sql faili, kuhu ta kirjutas tabelite loomise SQL käsud ja lisas ka natuke algandmeid. Arenduse esimeses pooles kopeeris autor need käsud HeidiSQL-i käsureale, et tabeleid teha ja andmeid sisestada (mis näitas kohe ka seda, kas käsud olid vigased või mitte).

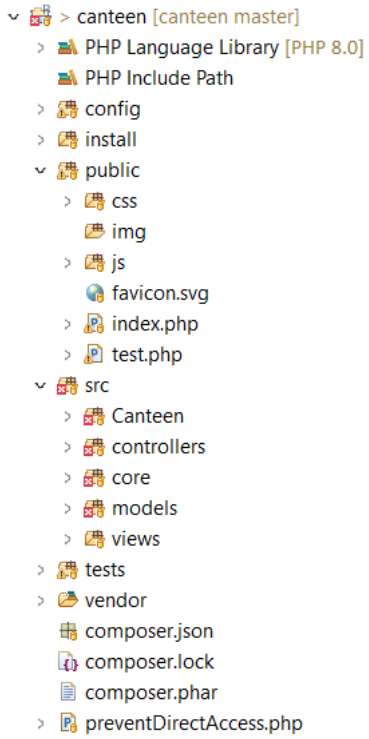
Kui rakendus piisavalt valmis saab, et seda reaalselt kasutama võiks hakata, siis tuleb sellega kindlasti kaasa ka paigaldamise skript. See skript küsib paigaldajalt seadistusi, kontrollib parameetrid üle (näiteks kas andmebaasiga saab ühenduse luua) ja loob andmebaasi vajalikud tabelid (koos algsete andmetega). Kahjuks lõputöö raames rakenduse arendus nii kaugele ei jõudnud, et installi skripti mõtet teha oleks olnud.

Ettevõttepraktikal õpitu põhjal otsustas autor andmebaasi jaoks teha ka versioonihalduse. Eesmärk oli see, et andmebaasis oleks rakenduse versioon kirjas. Kui autor mõne uuenduse teeb, siis lisatakse uus (versiooninumbriga) .sql fail, mis viib andmebaasi muudatused sisse ja muudab ka andmebaasis versiooni numbrit. Järgmise muudatusega lisatakse järgmine fail jne.

See on kasulik, kui mõni suvaline inimene otsustab autori tehtud rakendust kasutama hakata. Oletame, et ta kasutab aasta või kaks ära ja siis avastab, et vahepeal on mitu uuendust välja lastud. Kui ta nüüd uuendama hakkab, siis vaadatakse, et mis versiooninumber andmebaasis on, ning käivitatakse kõik .sql failid (õiges järjekorras), mis selle numbri ja kõige uuema versiooni vahel on. See tähendab, et ükskõik mis versiooni pealt on võimalik minna kõige viimasele versioonile.

3.2 Veebirakenduse ülesehitus

Rakenduse failid on jagatud erinevatesse kaustadesse (Joonis 8).



Joonis 8. Kaustapuu lühiversioon.

Kaust *config* sisaldab konfiguratsioonifaili. Seal on kirjas veebirakenduse seadistuse info. Näiteks andmebaasi detailid (kasutajanimi, parool, andmebaasi nimi jms.). Sinna on lisatud ka muutuja nimega *SESSION_PREFIX,* mis liidetakse automaatselt sessiooni muutujate ette. Kui juhtub, et keegi tahab sellest rakendusest mitut koopiat korraga käitada, siis ei lähe sessiooni muutujad omavahel konflikti.

Kaustas *public* on avalikult juurdepääsetavad failid. Näiteks staatilised failid (pildid, stiilifailid ja JavaScripti failid). Sammuti on seal *index.php* fail, mille kaudu rakenduse kasutamine käib. Samas kaustas .htaccess failis on määratud URL-i ümberkirjutamise reeglid, mis suurema osa päringutest suunavad index.php faili. See fail kasutab PHRoute teeki ja tegeleb päringute suunamisega. Näiteks kui (arendusarvutis) aadressireale kirjutada *http://canteen.demo/users/edit/1* siis *index.php* failis on kirjas suunamisega tegelev kood (Joonis 9).

```
$router->get(['/users/edit/{id:i}', 'users_edit'], function ($id) {
    global $paths;
    require $paths['users'];

    $controller = new UserController();
    return $controller->getEdit($id);
});
```

Joonis 9. Päringu suunamine PHRoute teegi abiga.

Selles koodis on avaldis *{id:i}*, kus *id* on muutuja nimi ja :i tähistab PHRoute teegis täisarvu. See täisarv edastatakse siis *UserController* klassi *getEdit()* meetodisse parameetrina. Sellest meetodist tagastatakse siis veebilehe kood ja saadetakse kasutajale. Sellel koodijupil on veel üks tähelepanuväärne asi. Nimelt on võimalik *users_edit* nime kasutada rakenduses sellele aadressile viitamiseks. Kui mingil hetkel otsustatakse, et URL-id võiksid eesti keelsed olla, siis võib *index.php* failis */users/edit/{id:i}* asemel kirjutada */kasutajad/muuda/{id:i}*. Kuna rakendus sisemiselt kasutab *users_edit* nime, siis ei pea hakkama koodis igal pool neid aadresse muutma hakkama. Igale sisestatud aadressile otsitakse *index.php* failist vaste ja kui vastet ei leita, siis suunatakse vealehele.

Src kaust on jagatud viieks alamkaustaks. Kolm esimest kausta on MVC kaustad (mudel – vaade - kontroller). *Views* kaustas on *.twig* failid (ehk veebilehe välimusega tegelevad HTML failid). *Models* kaustas on äriloogika ja *Controllers* kaust vahendab infot mudeli ja vaate vahel.

Siis on *core* kaust, kus on selline kood, mis ei ole ülejäänud rakendusest sõltuv. See kaust on disainitud nii, et selle võib ühest rakendusest teise tõsta ja kohe kasutusele võtta. Seal on näiteks andmebaasi klass (mis sisaldab üldiseid andmebaasi toiminguid),

kuupäeva klass (kuupäeva valideerimine, nädalapäeva hankimine, kuu kõigi päevade saamine jne.), sessiooni klass (sessiooni muutujate salvestamine/tagastamine nii ükshaaval kui hulgi) ja utiliitide klass (kus on muud väikesed abi funktsioonid, mille jaoks eraldi klassi luua mõtet ei ole).

Viimasena on *Canteen* kaust. Seal hoitakse klasse, mis ei sobi MVC kaustadesse ega ka *core* kausta. Näiteks andmeklassid jms. (andmeklassid on sellised klassid mille põhiülesandeks on andmete hoidmine).

3.3 Veebirakenduse turvalisus

Autor on läbinud andmeturbe sissejuhatava kursuse, mis andis talle baastasemel teadmised kaasaegsest andmeturbest ja krüptoloogiast. Neid teadmisi kasutades on üritatud veebirakendust turvalisemaks muuta.

3.3.1 Parooli turvamine

Kõige tavalisem asi, mille pärast muret tuntakse on see, et kuidas rakendus paroole salvestab. Lahtise tekstina (*plaintext*) parooli salvestamine on muidugi mõista halb idee. Esiteks ei tohiks parooli teada peale parooli omaniku (kasutaja) mitte keegi (ka rakenduse administraator mitte). Teiseks on see ohtlik selle pärast, et kui mõnel kurjategijal peaks õnnestuma (näiteks *SQL Injection*'it [48] ehk SQL-i süstimist kasutades) andmebaasile ligi pääseda, siis saab ta kohe kõik kasutajanimed ja paroolid teada. Kuigi juba ammu on soovitatud, et igal pool peaks erinev parool olema, siis kogemus näitab, et väga paljud inimesed kasutavad samu paroole mitmes kohas (mis tähendab, et parooli lekkimine on palju suurem probleem kui esijalgu paistab).

Seega otsustas autor kasutada paroolide salvestamiseks PHP-sse sisseehitatud funktsiooni nimega *password_hash* [49]. Üks peamisi põhjusi oli, et see kasutab parooli räsi (*hash*) tekitamisel ühesuunalist räsifunktsiooni (mis tähendab, et räsi on praktiliselt võimatu tagasi tekstiks/parooliks muuta).

Teine põhjus oli see, et *password_hash* kasutab „soolamist" (*salt*), kus lisatakse paroolile enne räsimist veel üks juhuslik andmejada (see sama andmejada salvestatakse

koos räsiga andmebaasi). Siis ei saa ründajad vikerkaaretabelit (*rainbow table*) kasutada.

„Soolamisel" on ka oma nõrkus. Kui kellelgi õnnestub teie andmebaasile lubamatult juurde pääseda (näiteks *SQL Injection*), siis saab ta ka „soola" teada ning see lihtsustab ründamist.

„Soola" nõrkuse ravimiseks soovitatakse kasutada „pipart" (*pepper*), mis sarnaneb „soolale" aga iga parool kasutab sama „pipart" ning seda ei hoita mitte andmebaasis vaid kuskil mujal (failis, lähtekoodis, pilves jne.). See tähendab, et kui kurjategijal õnnestub (ainult) andmebaasile ligi pääseda, siis „pipra" väärtust ta ikkagi ei tea.

Kuigi „pipar" tundub olevat suurepärane lahendus, on ka sellel omad puudused. Kõige tähtsam neist on, et selle väärtust ei ole võimalik hiljem muuta (sest seda kasutati ühesuunalises räsifunktsioonis). Kui „pipar" kuidagi peaks lekkima, siis ei ole võimalik seda uue vastu vahetada.

Seega otsustas autor kasutada järgnevat lähenemist. Kõigepealt võetakse kasutaja parool ja lastakse läbi *passwd_hash* funktsiooni (kus lisatakse automaatselt „sool"). Peale seda aga kasutatakse Sodium teegi [50] *sodium_crypto_secretbox* funktsiooni, mis kasutab sümmeetrilist võtit (sama võtmega saab ka dekrüpteerida), et eelmises sammus saadud räsi veelkord ära krüpteerida. Saadud tulemus salvestatakse siis andmebaasi.

Sisse logimisel otsitakse kasutajanime (e-maili) põhjal andmebaasist krüpteeritud parooli väärtus ja pakitakse see *sodium_crypto_secretbox_open* funktsiooniga lahti (kasutades sama võtit, millega see krüpteeriti). Seejärel kontrollib PHP-sse sisseehitatud funktsioon *password_verify*, kas kasutaja sisestatud paroolist tekib (sama „soola" kasutades) identne räsi (ehk kas on õige parool).

See lähenemine võimaldab võtmete vahetamist (graafiku alusel või turvaintsidendi korral), mis suurendab turvalisust. Muidugi kui kurjategijad serveri failisüsteemile juurde pääsevad, siis ei ole ka sellest lähenemisest palju abi. Sodiumi sümmeetrilise võtme loomise lisab autor installi skripti (et igal rakenduse koopial oleks erinev ja automaatselt genereeritud võti).

3.3.2 Ligipääs PHP failidele

Et tavakasutaja ei saaks lihtsalt suvalisi PHP faile käivitada (näiteks otse .php faili juurde viivat aadressi trükkides) võttis autor kasutusele kaks lähenemist.

Esimene neist oli see, et kogu rakendus käib läbi *index.php* faili. Sellest oli lähemalt juttu peatükis „3.2 Veebirakenduse ülesehitus" (*public* kataloogi juures). Lühidalt võib kokku võtta selle nii, et *.htaccess* failis kirjutatakse URL-id ümber ja suunatakse *index.php* faili, ning too otsustab millist PHP faili kasutada. Täpsustuseks võib lisada, et oma arendusmasinas seadistas autor Apache veebiserveri nii, et *canteen.demo* URL suunaks otse *public* kausta (mitte rakenduse juurkausta).

Teine lähenemine on failipõhine. Võimatu on teada kuidas teised inimesed seda rakendust kasutavad. Mõni võib näiteks lihtsalt kogu rakenduse visata *public_html* kausta ja seal tegutseda (Apache võib ka *.htaccess* faili lihtsalt ignoreerida). Et ka siis ei saaks suvalisi PHP faile otse käivitada, deklareeritakse *index* failis üks konstant. PHP failide alguses kontrollitakse, et kas see konstant on olemas ja kui ei ole, siis saadetakse vastuseks veakood 403 ja peatatakse PHP skripti täitmine. Seega kui kasutaja proovib otse mõnele PHP failile ligi pääseda, siis konstanti ei ole ja kasutaja saab veakoodi (juurdepääs keelatud).

Tegemist on üsnagi algelise kaitsega, nii et tulevikus tuleb seda veel parandada.

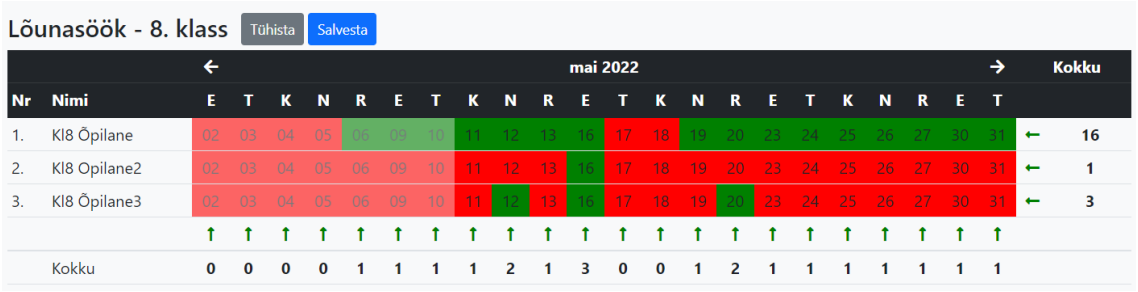
4 Rakenduse kasutusloogika

Kasutusloogikat katsus autor arendada nii, et tavakasutaja (eelkõige klassijuhataja ja sööja) suudaks ilma eelneva juhendamiseta hakata rakendust kasutama. Teised kasutajad vajavad tõenäoliselt veidi juhendamist.

4.1 Sööjate märkimine

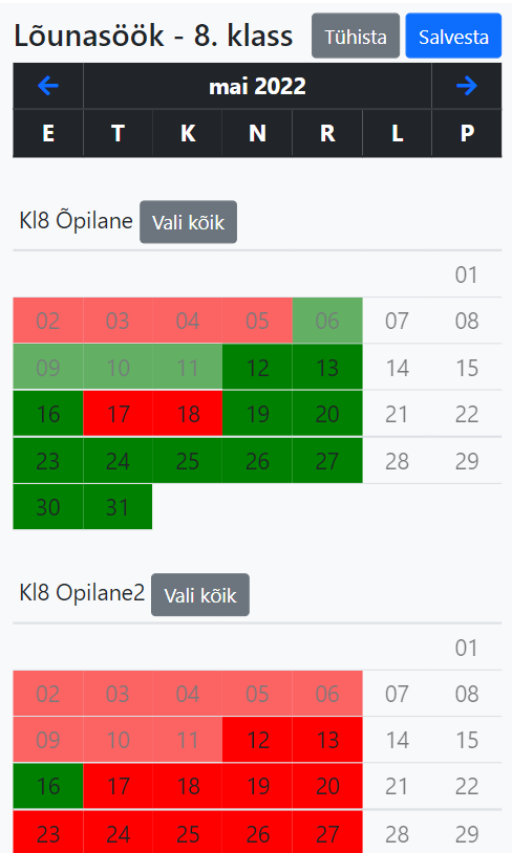
Sööjate märkimine on üles ehitatud nii, et iga sööja näeb enda infot (söömisi jms.). Klassijuhatajad näevad ainult oma klassi õpilaste andmeid ja kokk, vaatleja ning administraator näevad kõikide sööjate infot. Ehk teisisõnu, igaüks näeb ainult seda mida tal vaja näha on.

Klassijuhataja valib söögikorra ning tema klassi(de) nimekiri tuleb lahti (Joonis 10). Sööjate märkimine toimub kuu kaupa. Arvutis vaadates on vaadeldava kuu kuupäevad kõrvuti rivis. Neid kuupäevasid, millal sööki ei pakuta seal ei näita (koolides tavaliselt laupäevad ja pühapäevad). Klassijuhataja saab sööjaid märkida kas ühekaupa või rea/tulba kaupa. Nii rea kui ka tulba lõpus on loendurid. Rea lõpus on näha kui mitmeks päevaks antud isik sööma on märgitud (ja nupp, millega saab terve rea sööma märkida või eemaldada). Tulba lõpus näitab ära mitu sööjat antud kuupäeval on (ja nupp terve tulba märgistamiseks või märgistuse eemaldamiseks). Esijalgu oli plaanis kasutada pluss ja miinus märke sööjate näitamiseks (vastavalt kas „sööb” või „ei söö” aga õpetajatelt tuli tagasiside, et väikese ekraani pealt võib seda raske lugeda olla. Seetõttu otsustas autor värve kasutada (esijalgu nii, et roheline „sööb” ja punane „ei söö” aga need värvid tõenäoliselt muutuvad tulevikus, sest antud kombinatsioon ei ole värvipimedate jaoks eriti hea). Kuna pluss/miinus märki enam vaja ei olnud, sai selle asemel lahtrisse kirjutada kuupäeva (et ei pea klassijuhataja tulpa pidi järge ajama, et kas ta ikka muudab õiget kuupäeva või mitte).



Joonis 10. Sööjate märkimine klassijuhataja vaates.

Mobiili vaates (Joonis 11) kuvatakse iga inimese kohta kuu nädalate kaupa (üks rida on terve nädal – koos nende päevadega, kus sööma registreerida ei saa). Tänu sellele ei ole mõtet seal rea / tulba kaupa märgistamist rakendada. Selle asemel on iga nime juures nupp, mis märgistab terve kuu ära (ainult sellel sööjal). See kõige parem lahendus ei ole aga mobiiliga sööjate märkimine (klassi kaupa) ongi pigem tagavara variant (kui arvutit käeulatuses ei ole). Tulevikus võib seda edasi arendada.



Joonis 11. Sööjate märkimine mobiiliga.

4.2 Kohalolu registreerimine

Kohalolu registreerimine tavaolukorras käib läbi koka konto. Köögis (kuhu õpilased ei pääse) on arvuti kuhu kokk logib sisse. Selle arvuti külge on ühendatud RFID kaardilugeja, mis asub söögisaalis. Sööjad saavad oma kohalolu registreerida kaarti lugejale näidates ning kokk oma ekraanil näeb seda.

Kui mõni sööja jätab kaardi näiteks koju, siis võib kokk ka käsitsi kohaloleku kirja panna. Kokk (söökla töötaja) näeb antud söögikorra kõikide gruppide nimekirju. Need nimed, kes on kohalolu registreerinud, on ühte värvi ja kes ei ole, nende nimed on teist värvi (kes ennast sööma ei märkinud on hallid). Esijalgu oli plaanis teha kohal olevad sööjad rohelise värviga ja puuduvad sööjad punasega aga see ei sobi (värvipimedate tõttu). Samas saab kasutada ikoone (kohalolija ees on näiteks linnuke ja puuduva sööja ees X aga seda saab hiljem otsustada).

5 Töö tulemus

Loodud veebirakendust on raske objektiivselt hinnata, sest antud lõputöö tähtajaks ei olnud rakenduse programmeerimine veel täielikult valmis. Selgus, et töö maht oli suurem kui algselt arvatud. Palju aega kulus selliste otsuste tegemiseks, mis mõjutasid funktsionaalsust. Korrapärase kasutamise puhul ei olnud asi eriti keeruline aga läbi tuli mõelda ka nn äärejuhtumid. Näiteks kui administraator määrab, et kuupäeval X sööki ei pakuta aga keegi on jõudnud juba ennast sööma märkida, siis mida teha, kas näidata veateadet (et sel kuupäeval on vähemalt 1 sööja märgitud) või kustutada antud kuupäeva sööjad ära lihtsalt? Antud juhul valis autor, et tuleks hoiatust näidata ja kui administraator sellegi poolest jätkab, siis söömised ära kustutada.

Rakenduse lähtekood on kõigile kättesaadav Bitbucket-i keskkonnas [51]. Iga soovija võib selle tasuta alla laadida ning oma äranägemist järgi muudatusi teha.

Lõputöö esitamise ajaks olid valmis ainult rakenduse põhiosad. Nagu näiteks kasutajate (ja gruppide) haldus, söögipäevade erandite lisamine / eemaldamine, söögikordade haldamine, sööjate sööma märkimine (klasside kaupa), kohalolu registreerimine (RFID kaarti kasutades), sööjate aruanne jms.

Lõpetamata on veel päris mitmed tähtsad osad. Raamatupidamise aruandlus (kokkuvõtvad aruanded) on veel pooleli ja (aruande) failide eksportimine on puudu. Mitmed osad vajavad veel viimistlemist ja testimist (sest esijalgu on testitud ainult autori poolt käsitsi, et kas programm käitub nii nagu oodatud tavatoimingute käigus). Testitud sai mõningaid äärejuhtumeid aga neid on kindlasti veel.

Autoril on plaanis peale lõputööd jätkata rakenduse arendamist. See on plaanis kasutusele võtta Ida-Virumaa üldhariduskoolides (kui arendus on lõpuni viidud ja suuremad vead eemaldatud). See aitab kulusid kokku hoida. Näiteks Iisaku Gümnaasiumis on juba uksekaardi süsteem kasutusel ja sellega seonduvalt on õpilastel RFID kiibiga kaardid. Neid samu kaarte saab väga edukalt ära kasutada antud rakenduses. Kui autori valmistatud rakendus mõnda aega kasutusel on, siis on lihtsam

avastada, et milline klassijuhataja unustab pidevalt sööjaid söömast maha võtta. Samamoodi tuleb nähtavale ka see, kui mõni õpilane pidevalt sööma minemata jätab. See lubab juhtkonnal tegeleda nende probleemidega ja laseb köögil täpsemalt söögikogust arvestada, mis omakorda säästab raha (toitu ei toodeta nii palju üle). Sööjate eelistuste kohta on võimalik siis oletusi teha (sest pakutav söök mõjutab sööjate arvu mingil määral).

5.1 Võimalused edasisteks arendusteks

Edasi arendamise võimalusi on üsna palju. Võib teha mõnda järgnevatest arendustest.

- Lubada sisselogimist domeeni (LDAP) kontot kasutades.
- Kui LDAP-i struktuur on hästi disainitud, siis võiks ka otse sealt kasutajaid importida.
- Kasutajate massiline importimine CSV (Comma Separated Values) faili abil.
- Andmete eksportimise sellisel kujul, et neid saaks koolides / omavalitsustes kasutusel olevatesse raamatupidamise programmidesse importida.
- Kasutajatele teha seadete leht, kus saab rakenduse välimust muuta, otsustada mis leht avaneb sisse logimisel jne.
- Lisada rakendusele mitmekeelsuse tugi (see eeldab muudatusi ka andmebaasis).
- Õpetajad on avaldanud soovi, et kui sööja ennast sööma märgib, siis on hea kui menüü seal kohe kõrval on. See eeldaks köögi poolset lisatööd (sest menüü tuleb iga nädal süsteemi sisestada).
- Kohaloleku kontrolli arendusena saab lisada RFID kaartide asemel QR koodi kasutamise võimaluse.

6 Kokkuvõte

Käesoleva lõputöö eemärk oli luua Eesti üldhariduskoolidele veebirakendus, mis lubab neil elektrooniliselt sööjaid märkida ning kindlaks teha kui palju neist sööjatest ka reaalselt kohal käib. Sarnase suunitlusega rakendusi on küll juba olemas aga neil on alati mõni probleem (kas ei ole eesti keeles, on ebamugav kasutada, kallis jne.).

Tänu valmivale rakendusele saavad klassijuhatajad mugavalt sööjad märkida. Kui kool soovib, saavad õpilased ka ennast ka ise sööma märkida, mis on eriti tähtis sellistel juhtudel, kus õpilane ise peab oma söögi eest maksma. See aitab vältida selliseid olukordi, kus õpilane keeldub maksmast öeldes, et tema ennast sööma ei ole märkinud. Kui rakenduse hulka kuulub ka kohalolu kontroll, siis saab kooli juhtkond hea ülevaate sellest, kas sööma registreerunute arv vastab ka reaalselt söömas käivate inimeste arvule. See lubab koolil kindlasti raha säästa, sest aruandlusest tuleb kohe välja kui mõni õpetaja unustab näiteks võistlusele sõitvad õpilased söögilt maha võtta või mõni õpilane lihtsalt ei viitsi söömas käia. Kuna juhtkonnal on sellistest asjadest ülevaade olemas, siis saavad nad probleemiga tegeleda. Majandusjuhataja töö muutub ka kergemaks, sest enam ei pea ta täis soditud paberilehelt käsitsi hakkama sööjaid Exceli tabelisse ümber trükkima. Raamatupidamisse saatmiseks võib rakendusest otse faili eksportida.

Esialgu sai tehtud MVP ehk minimaalne töötav toode, kuid ruumi jäi ka edasi arendamiseks. Üks asi, mille võib tulevikus lisada on menüü lisamise/vaatamise võimalus. Kasuks tuleb ka võimalus rakendusse domeeni (LDAP) kontoga sisse logida.

Kasutatud kirjandus

[1] O. Loit, „Mis on MVP ja milleks seda tarvitada?“, *Veebimajutuse blogi*, 9. mai 2018. https://www.veebimajutus.ee/blogi/mvp-toode (vaadatud 12. mai 2022).

[2] Piksel OÜ, „Ülevaade“, *Arno*. https://arno.ee/ulevaade (vaadatud 21. veebruar 2022).

[3] Applied Software Consultants, „aSc EduPage“. https://www.edupage.org/? (vaadatud 21. veebruar 2022).

[4] IN Deal OÜ, „Mida me teeme“, 15. august 2019. https://www.idnetwork.eu/et/what-we-do/ (vaadatud 6. märts 2022).

[5] LunchTime School Lunch Software, „School Lunch Software-School Food Service Software-LunchTime Cafeteria Data Management-School Cafeteria Software“. https://www.lunchtimesoftware.com/ (vaadatud 6. märts 2022).

[6] Hotlunch.com, „School Lunch Software - Order Hot Lunch Online - School Lunch Ordering System“, *Hotlunch.com*. https://hotlunch.com/ (vaadatud 6. märts 2022).

[7] Vanco Payment Solutions Inc, „School Lunch Pre-ordering Software - Vanco Payments“. https://www.vancopayments.com/education/school-lunch-pre-order-software/ (vaadatud 6. märts 2022).

[8] Food Service Solutions, „School payment software for lunch accounts & fee payment“. https://www.myschoolaccount.com/ (vaadatud 6. märts 2022).

[9] H. Mankell, *Chronicler of the Winds*, First Vintage Books Edition, June 2007. USA: Vintage Books, 2007.

[10] V. Vorteil ja J. Laanpere, „Tarkvara arendusnõuded“, *Tarkvara analüüs ja testimine*, Tallinna Ülikool. Vaadatud: 9. mai 2021. [Online]. Available at: https://web.htk.tlu.ee/digitaru/testimine/chapter/tarkvara-arendusnouded/

[11] Airfocus GmbH, „What is a Front End (In a Website) - Definition & Development“. https://airfocus.com/glossary/what-is-a-front-end/ (vaadatud 14. juuni 2021).

[12] D. Urmann, „What is a Website Backend? A Beginner’s Guide | Learn with Diib®“, *diib®*, 2. detsember 2020. https://diib.com/learn/website-backend/ (vaadatud 14. juuni 2021).

[13] A. Judd, „Why should I use a framework?“, *Medium*, 10. aprill 2018. https://medium.com/quick-and-easy/why-should-i-use-a-framework-3b17189b349a (vaadatud 14. juuni 2021).

[14] M. Watson, „What is C# used for?“, *Stackify*, 18. september 2020. https://stackify.com/what-is-c-used-for/ (vaadatud 15. juuni 2021).

[15] A. Johari, „What is Java? A Beginner’s Guide to Java and its Evolution“, *Edureka*, 19. aprill 2017. https://www.edureka.co/blog/what-is-java/ (vaadatud 16. juuni 2021).

[16] H. Vallaste, „e-Teatmik: IT ja sidetehnika seletav sõnaraamat“. http://vallaste.ee/ (vaadatud 16. juuni 2021).

[17] J. Gallagher, „What is JavaScript Used For?: A Complete Guide“, *Career Karma*, 10. jaanuar 2021. https://careerkarma.com/blog/what-is-javascript-used-for/ (vaadatud 16. juuni 2021).

[18] The PHP Group, „PHP: History of PHP - Manual“. https://www.php.net/manual/en/history.php.php (vaadatud 21. juuni 2021).

[19] A. Bradley, „What Is PHP Used For?“, *ThoughtCo*. https://www.thoughtco.com/what-is-php-used-for-2694011 (vaadatud 21. juuni 2021).

[20] S. Tawde, „What is Laravel? | Learn 13 Key Features in the Laravel Framework“, *EDUCBA*, 4. oktoober 2019. https://www.educba.com/what-is-laravel/ (vaadatud 21. juuni 2021).

[21] A. Jain, „How To Understand The Difference Between Statically - Dynamically - Strongly - Weakly Typed Language | Hacker Noon“. https://hackernoon.com/actually-understand-statically-dynamically-strongly-weakly-typed-languages-axbpi3za2 (vaadatud 21. juuni 2021).

[22] Rick-Anderson, „When to use a reverse proxy with the ASP.NET Core Kestrel web server“. https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel/when-to-use-a-reverse-proxy (vaadatud 21. juuni 2021).

[23] The Interaction Design Foundation, „KISS (Keep it Simple, Stupid) - A Design Principle“, *The Interaction Design Foundation*. https://www.interaction-design.org/literature/article/kiss-keep-it-simple-stupid-a-design-principle (vaadatud 25. aprill 2022).

[24] Kinsta Inc, „What is MySQL? A Beginner-Friendly Explanation“, *Kinsta*. https://kinsta.com/knowledgebase/what-is-mysql/ (vaadatud 7. juuli 2021).

[25] P. Gera, „MariaDB vs MySQL: Key Performance Differences“, *Coding Ninjas Blog*, 30. aprill 2021. https://www.codingninjas.com/blog/2021/04/30/mariadb-vs-mysql-key-performance-differences/ (vaadatud 7. juuli 2021).

[26] R. Peterson, „What is SQL Server? Introduction, Version History“. https://www.guru99.com/sql-server-introduction.html (vaadatud 7. juuli 2021).

[27] Statista, „Most popular relational DBMS 2022“, *Statista*. https://www.statista.com/statistics/1131568/worldwide-popularity-ranking-relational-database-management-systems/ (vaadatud 9. mai 2022).

[28] www.javapoint.com, „What is Oracle - javatpoint“, *www.javatpoint.com*. https://www.javatpoint.com/what-is-oracle (vaadatud 9. mai 2022).

[29] K. Sharma, „What is PostgreSQL and why do enterprise developers and start-ups love it?“, *Ubuntu*. https://ubuntu.com/blog/what-is-postgresql (vaadatud 7. juuli 2021).

[30] M. Berga ja R. Ferreira, „PostgreSQL vs MySQL: how to choose?“, *Blog | Imaginary Cloud*, 1. juuli 2021. https://www.imaginarycloud.com/blog/postgresql-vs-mysql/ (vaadatud 7. juuli 2021).

[31] R. Peterson, „MariaDB vs MySQL: What is the Difference Between MariaDB and MySQL“. https://www.guru99.com/mariadb-vs-mysql.html (vaadatud 7. juuli 2021).

[32] J. Green, *PHRoute - Fast request router for PHP*. 2022. Vaadatud: 20. märts 2022. [Online]. Available at: https://github.com/mrjgreen/phroute

[33] Symfony, „Home - Twig - The flexible, fast, and secure PHP template engine“. https://twig.symfony.com/ (vaadatud 2. aprill 2022).

[34] T. J. Parr, „Enforcing strict model-view separation in template engines“, *Proceedings of the 13th international conference on World Wide Web*, New York, NY, USA, mai 2004, lk 224–233. doi: 10.1145/988672.988703.

[35] W3schools, „What is MVC Architecture?“ https://www.w3schools.in/mvc-architecture (vaadatud 2. aprill 2022).

[36] „PHP Template Engine | Smarty“. https://www.smarty.net/ (vaadatud 11. detsember 2021).

[37] I. Gaba, „What is Version Control and What Are Its Benefits?“, *Simplilearn.com*. https://www.simplilearn.com/tutorials/devops-tutorial/version-control (vaadatud 25. aprill 2022).

[38] GitHub Inc, „GitHub: Where the world builds software“, *GitHub*. https://github.com/ (vaadatud 6. veebruar 2022).

[39] Atlassian Pty Ltd, „Bitbucket | The Git solution for professional teams“, *Bitbucket*. https://bitbucket.org/product (vaadatud 6. veebruar 2022).

[40] GitLab B.V., „Iterate faster, innovate together“, *GitLab*. https://about.gitlab.com/ (vaadatud 6. veebruar 2022).

[41] R. Bourdon, „WampServer“, *WampServer*. https://www.wampserver.com/ (vaadatud 6. veebruar 2022).

[42] CGP Holdings Inc, „What is a localhost?“, *WhatIsMyIPAddress*, 28. august 2019. https://whatismyipaddress.com/localhost (vaadatud 6. veebruar 2022).

[43] M. Nagathan, „What is IDE? | How It Works | Need & Scope | Skill And Advantages of IDE“, *EDUCBA*, 13. mai 2019. https://www.educba.com/what-is-ide/ (vaadatud 6. veebruar 2022).

[44] JetBrains s.r.o., „PhpStorm: The Lightning-Smart IDE for PHP Programming by JetBrains“, *JetBrains*. https://www.jetbrains.com/phpstorm/ (vaadatud 6. veebruar 2022).

[45] C. Guindon, „Eclipse Project | The Eclipse Foundation“. https://www.eclipse.org/eclipse/ (vaadatud 6. veebruar 2022).

[46] A. Becker, „HeidiSQL - MariaDB, MySQL, MSSQL, PostgreSQL and SQLite made easy“. https://www.heidisql.com/ (vaadatud 6. veebruar 2022).

[47] Leeds Beckett University, „QSEE Technologies | Leeds Beckett University“. https://www.leedsbeckett.ac.uk/qsee-technologies/ (vaadatud 11. mai 2022).

[48] Refsnes Data, „SQL Injection“. https://www.w3schools.com/sql/sql_injection.asp (vaadatud 9. aprill 2022).

[49] The PHP Group, „PHP: password_hash - Manual“. https://www.php.net/manual/en/function.password-hash.php (vaadatud 6. aprill 2022).

[50] F. Denis, *jedisct1/libsodium*. 2022. Vaadatud: 9. aprill 2022. [Online]. Available at: https://github.com/jedisct1/libsodium

[51] R. Kalev, „kalevr / canteen — Bitbucket“. https://bitbucket.org/kalevr/canteen/src/master/ (vaadatud 4. mai 2022).

Lisa 1– Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Kalev Riivik

1 Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Sööjate märkimise süsteem üldhariduskoolidele," mille juhendaja on Einar Kivisalu

1.1 reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2 üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2 Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3 Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

16.05.2022

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Olemi-suhte diagramm