TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

Lõppkasutaja arhitektuuri väljatöötamine ning kasutus veebirakenduste kontekstis Dinkory näitel

magistritöö

Üliõpilane: Kristof Mikael Rosenberg

Üliõpilaskood: 143811IAPM

Juhendaja: Mart Roost, MSc

Tallinn
2016

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.

(*kuupäev*) (*allkiri*)

Annotatsioon

Antud töö eesmärk on uurida lõppkasutaja arhitektuuri kasutamise mõistlikkust veebirakenduste kontekstis. Töös uuritakse andmete, väljanägemise ning funktsionaalsuse kohandamise võimaldamist. Pakutakse mitmeid lahendusi koos selgitustega ning nende seast valitakse mõned, mis realiseeritakse prototüübis Dinkory. Töös antakse ülevaade erinevate lähenemiste turvariskidest.

Prototüübi abil sai näidatud, et on võimalik kasutada lõppkasutaja arhitektuuri veebirakenduste kontekstis. Prototüübis ei saanud funktsionaalsuse laiendamise tuge piisavalt arendada – selgus, et funktsionaalsuse laiendamiseks oleks vaja spetsiifiliselt lõppkasutaja arenduses kasutatavat skriptikeelt. Olemasolevate vahenditega pole võimalik tagada turvalisust või on loodavad võimalused liiga limiteeritud.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 86 leheküljel, 14 peatükki, 14 joonist, 14 tabelit.

Abstract

The aim of the thesis is to evaluate the validity of using a end-user architecture in the context of web applications. I explore how to allow customization of saved data, the visual representation and how to expand functionality. Several solutions are proposed together with explanations. Some of these solutions are put to the test in the prototype named Dinkory. The potential security risks are given for each proposed solution.

The prototype is a proof-of-concept to prove it is possible to use an end-user architecture in the context of web applications. Functionality was not expanded much – it appeared, that there is a need for a scripting language specifically for end-user development in web applications. The available tools are either incapable of ensuring safety or are too limited in capability.

The thesis is in Estonian and contains 86 pages of text, 14 chapters, 14 figures, 14 tables.

Lühendite ja mõistete sõnastik

ES6	***ECMAScript 2015/6th Edition of ECMAScript*** ECMAScript on standardiseeritud programmeerimiskeel, mille tuntuim implementatsioon on JavaScript. ES6 tähendab selle keele 2015 aasta ehk 6ndat versiooni.
HTML	***HyperText Markup Language*** Veebilehtede märgendamiseks kasutatud keel. HTML veebilehtede kuvamiseks kasutatakse veebilehitsejaid.
XSS	***Cross-site scripting*** Teatud sorti küberrünnakud. Pahatahliku koodi lisamine infosüsteemi eesmärgiga see käivitada lõppkasutajatel.
API	***Application Programming Interface*** Rakendusliides. See tähendab defineeritud reeglistikku, mida saab kasutada loodud programmiga suhtlemiseks (teiste programmide poolt).
CSS	***Cascading Style Sheets*** Keel, mille abil HTML elementide kujundust ning asukohta määratakse.
JSON	***JavaScript Object Notation*** Vorming andmete salvestamiseks ning vahetamiseks. JSONit saab kasutada programmeerimiskeelest sõltumatult.

Jooniste nimekiri

Tabelite nimekiri

Sisukord

1. Sissejuhatus

1.1 Taust ja probleem

Lõppkasutajate arhitektuuri – kus lubatakse lõppkasutajate poolne süsteemi edasi disainimine – saab kasutada veebirakenduse loomiseks, kuid sellega kaasnevad omad turvariskid ning kasutatavuse probleemid. Kui leidub turvaline viis väga laialdase kohandatavuse toeks, saaks lõppkasutajate arenduse teoreetilisi tulemeid ära kasutada veebirakenduste kontekstis.

Veebikontekstis on uued väljakutsed turvalisuse tagamine ning andmemahud. Kasutajatele suure kontrolli andmiseks tuleb tagada murdskriptimise (inglise keeles „cross site scripting“) vältimine. Vastasel juhul ei saa veebirakendust pidada kasutajatele turvaliseks. Andmemahud ei saa liiga suured olla, kuna on kulukas nii nende serverilt kliendile kandmine kui ka serveripoolne töötlemine.

1.2 Ülesande püstitus

Eesmärk on leida viise, kuidas saaks toetada veebirakenduse kontekstis lõppkasutajate arendust. Selleks

1) kirjeldatakse teoreetilisi alusi;
2) püsitatakse nõuded;
3) nõuete rahuldamiseks projekteeritakse ning realiseeritakse prototüüp;
4) hinnatakse tulemust ning antakse viited edasiseks arenguks.

Idee tõestamiseks tuleb luua prototüüp, kus illustreeritakse lahenduste valiidsust. Prototüüp luuakse väheste vahenditega, mis omakorda näitab, kuidas lõppkasutajate arendust tuua veebikeskkonda limiteeritud võimalustele tuginedes. Prototüübi turvaaukude põhjalik uurimine ei mahu töö skoopi.

1.3 Metoodika

Teooriale tuginedes pakutakse välja andmemudeli, kujunduse ning funktsionaalsuse laiendamiseks sobivad teoreetilised variandid. Nende variantide seast valitakse selle prototüübi jaoks realistlik lahendus ning realiseeritakse see.

1.4 Ülevaade tööst

Alustatakse teoreetilise tausta selgitusega. Kogu töö vältel lähtutakse seal kirjeldatud printsiipidest. Seejärel kirjeldatakse loodava prototüübi infosüsteemi nõuded. Järgnevas peatükis kirjeldatakse lahti, missugune on prototüübi arhitektuur ning millised tehnoloogiad selle loomiseks valiti.

Järgnevad kolm peatükki, mis kirjeldavad üht lõppkasutajate arhitektuuriga seotud teemat – andmemudeli loomine, kujunduse määramine ning funktsionaalsuse laiendamine. Hinnatakse prototüübi vastavust määratud testidele ning läbitakse retrospektiiv. Retrospektiivis hinnatakse nii tulemust ning pakutakse, mida võiks edasi arendada.

Peatükis „Lisa 2: Prototüübi veebimaterjalid“ jaotis 8614 lk 86 antakse veebilingid, mille abil on võimalik tutvuda prototüübiga ning loodud lähtekoodi Git keskkonnaga.

2. Arendusel kasutatavad printsiibid

2.1 Metadisain ning lõppkasutaja arhitektuur ja arendus

Metadisaini raames eristatakse lõppkasutaja kasutust disainimisest. [1] Süsteemi arendajad ei soovi kitsendada süsteemi kasutust selgelt defineeritud aladele. Selle asemel luuakse tööriistad ning tugi, et tarkvara saaksid kasutada inimesed erinevatelt aladelt omade probleemide, mõistete ning lahendustega.

Süsteemi tööajal saavad kasutajatest omasorti disainerid. Nad loovad süsteemi uusi disainilahendusi ilma algsete metadisainerite kaasluseta. Tähtis on julgustada ning toetada lõppkasutajate aktiivset osalust süsteemi evolutsioonis. Samas ei saa eeldada, et kõik kasutajad tahavad süsteemi arengusse panustada.

Lõppkasutajad on need, kes toovad süsteemi omi mõisteid. Tavalise infosüsteemi loomise jaoks tehakse probleemvaldkonna analüüs, kuid antud projektis on vaja luua vaid keskkond ning tööriistad, et erinevate valdkondade inimesed saaksid endale relevantseid probleeme lahendada. Süsteemi loomiseks tehtud analüüsis ei kajastu mõisted, mida kasutavad lõppkasutajate grupid.

Kasutajakeskne disain (ingl. k. „User Centered Design“) kaasab kasutajaid arenduse käigus võimalikult vara, et ennetada täpselt vajadusi ja puudusi. Metadisain laiendab kasutajakeskset disaini, lisades sinna juurde areneva süsteemi uute nõuete ja vajaduste ennetamise. [2]

2.2 Tüüpprobleemid

2.2.1 Andmemahud on suured, töötluseks vajalik jõudlus suur

Lõppkasutajaarenduses antakse mingisugused baasandmed lõppkasutaja kätte töötlemiseks, et sealt saaks leida endale vajalik või esitada andmed mingil soovitud viisil. Veebirakenduste puhul on andmed tsentraalselt salvestatud ning seega nendega keerulistemate operatsioonide tegemine potentsiaalselt väga koormav, kui piisavalt palju kasutajaid seda korraga pärivad.

2.2.2 Süsteem võib näida liigselt keerukana

Üldjuhul peavad lõppkasutajad õppima uue süsteemi kasutust. Kui süsteem luuakse tervete probleemide gruppide lahendamiseks või süsteem koosneb pelgalt tööriistadest, peavad kasutajad alguses õppima süsteemi enda loogikat ning tööriistade kasutust. Nõnda on loodu küll laiade kasutusvõimalustega, kuid nõuab kasutajatelt rohkem teadmisi. Süsteem pole loodud kavandatud lõppkasutajale spetsiifilise probleemvaldkonna lahendamiseks.

2.3 Printsiibid probleemide vältimiseks

2.3.1 Eristatakse „stsenaariumeid“ ning „kasutuslugusid“

Üldjuhul on infosüsteemide projekteerimisel piisav nõuete ja/või kasutuslugude kirjeldamisest ning nende realiseerimisest, kuid lõppkasutajaarendamise ning metadisaini puhul erineb abstraktsioonitase kasutuslugude ning lõppkasutajate soovitud kasutamise stsenaariumite vahel. Süsteemi arendaja ning lõppkasutaja kasutavad erinevaid mõisteid, et sisuliselt sama probleemi lahendada.

Sellist kahe staadiumi eristamist nõuab nii metadisain kui ka lõppkasutajaarendus. Süsteemidisainerid peavad võimaldama hilisemat lõppkasutajate disainitööd luues neile vajalikud võimalused.

Näiteks võib lõppkasutaja oma stsenaariumi kirjeldamiseks kasutada mõistet „raamat“. Sellise tüübi võimaldamiseks aga süsteemi endasse raamatu mõistet kuhugi sisse ei tooda. Andmetüüpe luuakse lõppkasutajate endi poolt ning süsteemi arendamise kontekstis omavad sellised mõisted väärtust vaid testimise ja lõppkasutajatega suhtlemise raames.

„Stsenaarium“ tähendab lugu sellest, kuidas lõppkasutaja, kes pole arendusega kursis, tahaks loodavat süsteemi kasutada. „Kasutuslugu“ kirjeldab neid spetsiifilisi funktsionaalsuseid, mis stsenaariumite toetamiseks on vajalikud.

2.3.2 Kasutajaliides peab toetama ning suunama kasutajaid

Süsteemi keerukuse maandamiseks saab aidata kaasa kasutajaliidesele suure rõhu panemine. Kasutajaliides võib anda lõppkasutajatele kiiret tagasisidet, et tekiks kohene side tema sisendite ning oodatava väljundi vahel. See on eriti tähtis süsteemi tehniliselt nõudlike osade puhul.

Vormide täitmisel saab esimeste sisendite abil muuta ülejäänud vormi, et kuvataks kasutajale kõik relevantsed väljad. See vähendab visuaalset müra ning ei lase tähelepanul hajuda.

2.3.3 Kõik kasutajad ei pea olema arendajad

Süsteemidisainijad ei oska ette aimata, kui keeruline võib olla mõnele võhikule loodud funktsionaalsuse kasutamine ning süsteemis orienteerumine. Kui kasutada OECD uuringus kasutatud tasemete notatsiooni (vt [3]), süsteem võiks nõuda võimalikult vähe taseme 3 ja 2 oskusi. Lõppkasutajate arenduse oskus ja seega võimalus võib jääda muidu liiga väheste inimeste kanda – uuringu kohaselt ~5% populatsioonist.

Inimeste oskuste korvamiseks saab tükeldada funktsionaalsust nõnda, et teadlikumad kasutajad produtseerivad lahendusi, mida saavad kasutada ka kõik teised. See tähendab ühtse nimeruumi kasutamist ning loodud lahenduste kasutamise piiramist vaid autorile vältimist. Mõne tulemi kasutamine nõuab märgatavalt vähem oskusi, kui selle tulemi originaalne loomine.

3. Nõuded

Käesolevas peatükis tuuakse välja lõppkasutajatelt kogutud süsteemi kasutamise stsenaariumid, kasutuslood ning lühidalt nõuded. Kasutuslood on infosüsteemide projekteerimisel hädavajalikud, kuid lõppkasutajaarendusest lähtuvalt pole süsteemi kasutuslood otseses kooskõlas lõppkasutajate endi soovidega. Stsenaariumite abil saab kirjeldada, mida lõppkasutajad tahavad süsteemis teha ning kasutuslugude kaudu on võimalik need soovid formaliseerida ning seejärel realiseerida.

3.1 Arendaja enda poolt seatud piirangud

Süsteemi tuleb luua vastavalt nõuetele, mis mida soovitakse täita. Kuid teatud piirangud saab projekti juht ise seada, et määrata järgitavad piirid. Väljatoodud stsenaariumid peavad arvestama seatud piire.

3.1.1 Infosüsteem peab olema veebipõhine

Infosüsteemile pääseb ligi kasutades veebilehitsejat. Lõppkasutaja kasutab süsteemiga suhtlemiseks oma veebilehitseja abi. Seda võimaldav veebirakendus suhtleb ühe või enama andmebaasiga. Kasutaja ei suhtle otse andmebaasi(de)ga.

3.1.2 Info jaotatakse kaustadesse

Üheks põhiliseks kontseptiks on kaustad (ingl. k. „Folder“). Kogu informatsioon kogutakse kaustadesse. Sedasi saab tagada parema ülevaate ning võimaldada kasutajale sobivat info organiseerimist. Lisaks vähendab see suurte päringute arvu, kuna eeldatakse, et kuvatakse vaid valitud kausta informatsioon.

3.1.3 Infot salvestatakse kirjetena

Kaustade kõrval on teiseks tähtsaks kontseptiks kirjed (ehk sissekanded; ingl. k. „Entry“). Andmed salvestatakse sissekannete kaupa. See tähendab, et luuakse objekt, millega seotud selle sissekande andmed.

3.2 Stsenaariumid

Stsenaariumeid koguti potentsiaalsetelt lõppkasutajatelt, kelle hulka kuulus ka infosüsteemi autor ise. Kõik stsenaariumite kirjeldajad teadsid infosüsteemile rakendatavatest piirangutest (vt „3.1 Arendaja enda poolt seatud piirangud“ lk 19) ning pidasid neist kinni.

3.2.1 Stsenaarium 01

Soovin organiseerida oma järjehoidjaid. Teeksin kaustad, kuhu organiseeriksin oma järjehoidjad ning märgiks iga külge tema pealkirja, URLi ja märksõnad, et nende järgi saaks neid filtreerida.

Soovin jagada oma järjehoidjate kaustu ehk mitmeid samateemalisi järjehoidjaid korraga läbi lihtsa URLi. Sedasi on võimalik jagada kellelegi terve teema kohta olemasolevaid viited.

Otsingusse teksti sisestamisel filtreeritakse kausta sisu järjest väiksemaks ning kuvatakse vaid neid objekte, mille väljades leidub otsitav tekst.

3.2.2 Stsenaarium 02

Soovin organiseerida YouTube kanalite nimekirju. Määraksin kanali nime, mis töötab samaaegselt lingina kanali lehele, ning näitaks selle kõrval mängitavat YouTube videot, mille URLi määrasin sissekande külge. Sedasi saan tutvustada inimestele oma kausta jagades YouTube kanaleid, mis mulle meeldivad. Külastajad ei peaks otsima kanalilt mingit videot, mida vaadata, vaid neile näidataks sisseehitatud YouTube videomängija abil ühte kanali näidet.

3.2.3 Stsenaarium 03

Tahan oma infot sisse panna kasutades JSON formaati. Ettemääratud JSON struktuuri järgi lisan ühe tekstivälja kaudu palju sissekandeid. Ma sisestaksin 3D ruumis objektide koordinaadid ja nende vaatenurgad. Igat kirjet saaks muuta ning pärast ka JSON formaadis väljastada.

3.2.4 Stsenaarium 04

Juhendan lõputöid. Soovin salvestada lõputööde tegemisega seotud märkmeid ning suhtlust. Osa suhtlusest oleks piiratud vaid minu ja juhendatava vahele nii, et teised seda ei näeks. Soovin ka võimalust oma kommentaare ja märkusi jätta vaid endale nähtavaks. Mõned mõttearendused võivad osutuda kasulikuks ka teistele töödele.

Iga kontekst oleks eraldi kaustana.

3.2.5 Stsenaarium 05

Tahan organiseerida oma kudumismaterjale. Sinna kuuluvad näiteks lõngad, vardad, konksud, haagid ja muud vidinad. Need tüübid erinevad osaliselt, kuid kõigil oleks küljes, kas tegemist on uue või taaskasutatud materjaliga ning kui palju seda järel on.

Mõnel objektil on küljes näiteks tema värv või muster, tootja ning päritolu riik. Saaks kuidagi hinnata kvaliteeti.

Nende seast tahaks leida omale sobivaid objekte näiteks värvi järgi otsides.

3.3 Stsenaariumite põhjal üldistatud nõuded

3.3.1 Funktsionaalsed nõuded

1) Kasutaja saab luua ning kustutada kaustu. (Stsenaariumid: 01)
2) Kasutaja saab määrata kaustale ise unikaalse ID, mida kasutatakse kausta jagamise URLis. (Stsenaariumid: 01)
3) Kasutaja saab luua sissekande andmetüüpi nii nullist kui ka olemasoleva andmetüübi põhjal. (Stsenaariumid: 01, 05)
4) Kasutaja saab luua olemasolevate andmetüüpide põhjal sissekandeid. (Stsenaariumid: 01)
5) Kasutaja saab muuta ning kustutada olemasolevaid sissekandeid. (Stsenaariumid: 03)
6) Kontoga kasutaja saab peita oma loodud kausta teiste eest. (Stsenaariumid: 04)
7) Kasutaja saab sisestada sissekandeid mitme kaupa ettemääratud JSON formaati järgides. (Stsenaariumid: 03)
8) Kasutaja saab määrata ühele sissekandele ka teisi kaustu, kuhu see kuulub. (Stsenaariumid: 04)
9) Kasutaja saab kausta sisu filtreerida ehk sealt otsida endale huvitavaid sissekandeid. (Stsenaariumid: 01, 05)
10) Kasutaja saab määrata sissekande andmetüübile kujundust ehk missugune see veebilehel välja näeb. (Stsenaariumid: 06)

3.3.2 Mittefunktsionaalsed nõuded

- Kasutajate info kaitsmiseks on lubatud kasutajatel kontode kasutamine.
- Igale uuele kasutajale luuakse kodukaust, mille omanik nad ise on.
- Rakendus on veebipõhine ehk ilma lisarakendusteta kasutatav, kui olemas veebilehitseja.

3.4 Kasutuslood

3.4.1 Kasutuslugu 01: Kasutaja loob kausta

Tabel 1 Kasutuslugu 01

KASUTAJA LOOB KAUSTA

EESMÄRK	Kasutaja tahab luua kausta, et organiseerida informatsiooni sissekandeid kindlatesse kogumitesse.
EELTINGIMUSED	1. Kasutaja on vanemkausta omanik.
JÄRELTINGIMUSED	1. Uus kaust on loodud kasutaja poolt ettenähtud kausta.
KIRJELDUS	1. Kasutaja valib kausta, kuhu uus kaust luua. 2. Kasutaja täidab kausta kohta vajalikud andmed. 3. Kasutaja edastab kausta loomise. 4. Süsteem loob kausta. 5. Süsteem vastab kausta loojale loomise edukusest ning kuvab uue kausta kaustapuus.
ERANDID	4a. Kausta ei saanud luua, kuna selle ID-ga kaust juba eksisteerib või kaustal oli mõni andmetest puudu. 4a1. Süsteem teatab kasutajale veateatest.

3.4.2 Kasutuslugu 02: Kasutaja kustutab kausta

Tabel 2 Kasutuslugu 02

KASUTAJA KUSTUTAB KAUSTA

EESMÄRK	Kasutaja tahab loodud kaustu kustutada.
EELTINGIMUSED	1. Kasutaja on kausta omanik.
JÄRELTINGIMUSED	1. Kasutaja poolt määratud kaust on kustutatud.

KIRJELDUS	1. Kasutaja valib kausta, mida soovib kustutada. 2. Kasutaja edastab soovi seda kustutada. 3. Süsteem kustutab kausta. 4. Süsteem teatab kasutajale, et kaust on kustutatud.
ERANDID	-

3.4.3 Kasutuslugu 03: Kasutaja võtab kaustale otselingi

Tabel 3 Kasutuslugu 03

KASUTAJA VÕTAB KAUSTALE OTSELINGI	
EESMÄRK	Kasutaja on määranud kaustale unikaalse ID ja tahab selle abil luua/omandada linki, mis avamisel viib kaustani.
EELTINGIMUSED	1. Kasutajal on ligipääs kaustale.
JÄRELTINGIMUSED	-
KIRJELDUS	1. Kasutaja valib kausta. 2. Kasutaja pärib otselingi aadressi. 3. Süsteem kuvab kausta otselingi aadressi.
ERANDID	-

3.4.4 Kasutuslugu 04: Kasutaja loob sissekande andmetüübi

Tabel 4 Kasutuslugu 04

KASUTAJA LOOB SISSEKANDE ANDMETÜÜBI	
EESMÄRK	Kasutaja tahab luua uut sissekande andmetüüpi, sest olemasolevad ei pruugi tema vajadusi rahuldada.

EELTINGIMUSED	-
JÄRELTINGIMUSED	1. Kasutaja poolt loodud andmetüüp on salvestatud.
KIRJELDUS	1. Kasutaja avab andmetüübi loomise. 2. Süsteem kuvab andmetüübi loomise võimaluse. 3. Kasutaja täidab andmetüübi loomiseks vajaliku info. 4. Kasutaja edastab andmetüübi loomise soovi. 5. Süsteem kontrollib edastatu valiidsust. 6. Süsteem salvestab uue andmetüübi. 7. Süsteem teatab andmetüübi salvestuse edukusest.
ERANDID	1a. Kasutaja võib ka valida mõne olemasoleva andmetüübi. 1a1. Süsteem kuvab olemasolevad andmetüübid. 1a2. Kasutaja valib olemasoleva andmetüübi. 1a3. Süsteem viib sammule 2, kuid osa informatsioonist on täidetud valitud olemasoleva andmetüübi põhjal.

3.4.5 Kasutuslugu 05: Kasutaja loob sissekande

Tabel 5 Kasutuslugu 05

KASUTAJA LOOB SISSEKANDE

EESMÄRK	Kasutaja tahab oma infot salvestada. Seda tehakse sissekannete abil.
EELTINGIMUSED	1. Kasutajal on õigus kausta informatsiooni muuta.
JÄRELTINGIMUSED	1. Kasutaja loodud sissekanne on salvestatud.
KIRJELDUS	1. Kasutaja avab kausta, kuhu soovib sissekannet teha. 2. Kasutaja avab sissekande loomise. 3. Kasutaja valib sissekande tüübi ja sisestab andmed. 4. Kasutaja edastab uue sissekande. 5. Süsteem salvestab sissekande.

	6. Süsteem teatab kasutajale edukast salvestamisest.
ERANDID	-

3.4.6 Kasutuslugu 06: Kasutaja muudab sissekannet

Tabel 6 Kasutuslugu 06

KASUTAJA MUUDAB SISSEKANNET

EESMÄRK	Kasutaja tahab loodud sissekande välju muuta.
EELTINGIMUSED	1. Kasutajal on õigus kausta informatsiooni muuta.
JÄRELTINGIMUSED	1. Kasutaja poolt tehtud muudatused on salvestatud.
KIRJELDUS	1. Kasutaja valib sissekande, mida muuta. 2. Kasutaja muudab väljade väärtuseid. 3. Kasutaja edastab muutused. 4. Süsteem salvestab muutused. 5. Süsteem teatab edukast salvestusest.
ERANDID	-

3.4.7 Kasutuslugu 07: Kasutaja kustutab sissekande

Tabel 7 Kasutuslugu 07

KASUTAJA KUSTUTAB SISSEKANDE

EESMÄRK	Kasutaja tahab loodud sissekannet kustutada.
EELTINGIMUSED	1. Kasutajal on õigus kausta informatsiooni muuta.
JÄRELTINGIMUSED	1. Kasutaja poolt määratud sissekanne on kustutatud.
KIRJELDUS	1. Kasutaja valib sissekande, mida soovib kustutada. 2. Kasutaja edastab soovi seda kustutada. 3. Süsteem kustutab sissekande.

	4. Süsteem teatab kasutajale, et sissekanne on kustutatud.
ERANDID	-

3.4.8 Kasutuslugu 08: Kasutaja peidab kausta

Tabel 8 Kasutuslugu 08

KASUTAJA PEIDAB KAUSTA

EESMÄRK	Kasutaja tahab peita olemasolevat kausta (ja selle sisu) teiste kasutajate eest, et võimaldada privaatse info olemasolu.
EELTINGIMUSED	1. Kasutajal on õigus kausta informatsiooni muuta.
JÄRELTINGIMUSED	1. Kaust on peidetud.
KIRJELDUS	1. Kasutaja valib kasuta, mida peita. 2. Kasutaja edastab kausta peitmise soovi. 3. Süsteem märgib kausta peidetuks.
ERANDID	-

3.4.9 Kasutuslugu 09: Kasutaja sisestab JSON formaadis sissekanded

Tabel 9 Kasutuslugu 09

KASUTAJA SISESTAB JSON FORMAADIS SISSEKANDEID

EESMÄRK	Kasutaja tahab mitmeid sissekandeid korraga luua kasutades selleks JSON formaati.
EELTINGIMUSED	1. Kasutajal on õigus kausta informatsiooni muuta.
JÄRELTINGIMUSED	1. JSON formaadis antud sissekanded on salvestatud.
KIRJELDUS	1. Kasutaja valib kausta, kuhu soovib salvestada. 2. Kasutaja algatab JSON formaadis info sisestamise.

	3. Süsteem pärib kasutajalt sissekannete infot JSON formaadis tekstivälja abil. 4. Kasutaja sisestab info. 5. Kasutaja edastab sisendi. 6. Süsteem kontrollib andmete sobivust. 7. Süsteem töötleb andmed ning salvestab sissekande(d). 8. Süsteem teatab kasutajat loodud sissekannete kohta.
ERANDID	6a. Edastatud sisend ei ole valiidne JSON formaat või andmed ei vasta sissekannete loomise nõuetele. 6a1. Kasutajat teavitatakse veateatest ning suunatakse tagasi 4. sammu juurde.

3.4.10 Kasutuslugu 10: Kasutaja võtab JSON formaadis kausta sissekannete andmed

Tabel 10 Kasutuslugu 10

	KASUTAJA VÕTAB JSON FORMAADIS KAUSTA SISSEKANNETE ANDMED
EESMÄRK	Kasutaja saab süsteemist eksportida kaustade kaupa andmeid JSON formaadis, et saaks sissekannetena salvestatud andmeid kusagil mujal kasutada.
EELTINGIMUSED	1. Kasutajal on õigus kausta sisu näha.
JÄRELTINGIMUSED	-
KIRJELDUS	1. Kasutaja valib kausta, mille sisu soovib eksportida. 2. Kasutaja edastab soovi JSON formaadi koostamiseks. 3. Süsteem kuvab kausta sisu tekstiväljas JSON formaadis.
ERANDID	-

3.4.11 Kasutuslugu 11: Kasutaja määrab sissekandele teise kausta

Tabel 11 Kasutuslugu 11

KASUTAJA MÄÄRAB SISSEKANDELE TEISE KAUSTA

EESMÄRK	Kasutaja tahab loodud kaustu kustutada.
EELTINGIMUSED	1. Kasutajal on õigus sissekannet (selle kausta) näha.
JÄRELTINGIMUSED	1. Sissekanne kajastub ka uues kaustas.
KIRJELDUS	1. Kasutaja valib sissekande. 2. Kasutaja algatab selle mujal kajastamise. 3. Süsteem kuvab võimaluse kaustadest, kuhu tõsta. 4. Kasutaja valib kausta ning see edastatakse süsteemile. 5. Süsteem salvestab sissekande kajastamise teises kaustas.
ERANDID	4a. Kasutajal pole õigust valitud kaustas sisu muuta. 4a1. Kasutajat teavitatakse veateatest.

3.4.12 Kasutuslugu 12: Kasutaja filtreerib kuvatavat kausta sisu

Tabel 12 Kasutuslugu 12

KASUTAJA FILTREERIB KUVATAVA KAUSTA SISU

EESMÄRK	Kasutaja tahab olemasolevast infost leida neid infokilde, mis teda antud hetkel huvitavad. Selleni on võimalik jõuda lubades kuvatava info filtreerimist märksõnade abil.
EELTINGIMUSED	1. Kasutajal on õigus kausta näha.
JÄRELTINGIMUSED	-
KIRJELDUS	1. Kasutaja avab kausta, mille infot tahab filteerida. 2. Kasutaja sisestab märksõnad otsingulahtrisse. 3. Süsteem kuvab kaustast vaid märksõnadele vastavad sissekanded.

ERANDID	-

3.4.13 Kasutuslugu 13: Kasutaja loob andmetüübile kujunduse

Tabel 13 Kasutuslugu 13

	KASUTAJA LOOB ANDMETÜÜBILE KUJUNDUSE
EESMÄRK	Kasutaja tahab luua olemasolevale andmetüübile väljanägemist. Selleks tuleb defineerida malli ning CSS abil kujundus.
EELTINGIMUSED	-
JÄRELTINGIMUSED	1. Kasutaja loodud kujundus on salvestatud.
KIRJELDUS	1. Kasutaja algatab uue kujunduse loomist. 2. Süsteem kuvab kasutajale uue kujunduse loomise võimaluse. 3. Kasutaja valib andmetüübi, millele kujundus luuakse. 4. Kasutaja sisestab uue kujunduse andmed. 5. Kasutaja edastab uue kujunduse. 6. Süsteem salvestab kujunduse. 7. Süsteem teatab kasutajale kujunduse salvestamisest.
ERANDID	-

3.4.14 Kasutuslugu 14: Kasutaja määrab kaustale vaikimisi kujundusvalikud

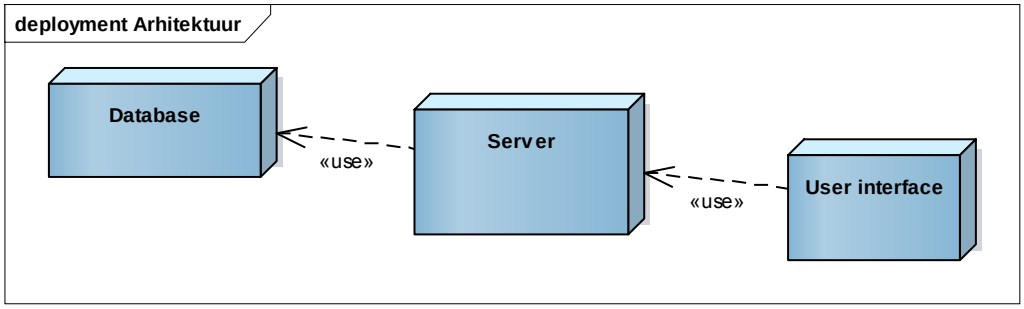
Tabel 14 Kasutuslugu 14

	KASUTAJA MÄÄRAB KAUSTALE VAIKIMISI KUJUNDUSVALIKUD
EESMÄRK	Kasutaja tahab valida kausta avamisel vaikimisi valitud kujunduse.

EELTINGIMUSED	1. Kasutaja on valitud kausta omanik. 2. Kaustas olevatel sissekannete tüüpidel on loodud visuaalsed kujundused.
JÄRELTINGIMUSED	1. Kausta külge on salvestatud JSON formaadis vaikimisi kujundusvalikud.
KIRJELDUS	1. Kasutaja valib kausta. 2. Kasutaja valib kujundusvalikud. 3. Kasutaja edastab süsteemile soovi valitud kujundused salvestada. 4. Süsteem salvestab JSON formaadis valitud kausta külge valitud kujundused. 5. Süsteem teatab kasutajale vaikimisi sätete salvestamisest.
ERANDID	-

4. Prototüübi arhitektuur ja kasutatavad tehnoloogiad

4.1 Arhitektuur



Joonis 1 Arhitektuuri diagramm

Kasutatakse lihtsat kolmest komponendist koosnevat arhitektuuri: andmebaas, sellega suhtlev server ning kasutajaliides serveri kliendina.

4.2 Arhitektuuri kompondendid

4.2.1 Server

4.2.1.1 Nõuded

Serveri tehnoloogia peab olema tugevalt orienteeritud API-de loomiseks, et selle abil toetada lisarakenduste loomist. Kasutajaliides ehitatakse ülesse tugevalt REST teenuseid kasutades, mitte vaadete loomise abil. Seetõttu pole vajalik valida mõnda Model-View-Controller või Hierarchical Model-View-Controller tüüpi raamistikku.

Kuna projekti raames on töö autor tarkvaraarendusprojekti kõigis rollides ise, peab eelistama raamistikke, mis arenduse tööd kiirendavad. Seetõttu võib teha järeleandmisi teekide ja raamistike valikutes jõudluses või stabiilsuses.

4.2.1.2 Valitud Nodal.js

Nodal.js on Node.js põhjal loodud API teenuste loomise raamistik. [4] Sellega on lihtne luua mudeleid ning nende kohta käivaid kontrollereid.

Nodal.js arendamiseks saab kasutada Javascripti ECMAScript 6 versiooni. ES6-ga loodud võimalusi üldkasutatud veebilehitsejad nagu Firefox ja Chrome veel täielikult ei toeta [5], kuid selle võimalusi saab kasutada Node keskkonnas töötavate rakenduste raames. [6]

Nodal võimaldab luua mudeleid ehk realiseerida andmestruktuuri omavahel seotud klasside näol. Nende mudelite loomisel määratakse väljad ning neile saab lisada meetodeid.

Mudelitega seotakse andmebaasitabelid, kuhu andmed salvestatakse. Suhtluse jaoks on raamistikus oma liides olemas, seega on andmete salvestamine, otsimine, uuendamine ja kustutamine koheselt võimaldatud.

Mudelite jaoks luuakse kontrollerid, mis genereeritult kohe tüüpilisi CRUD meetodeid toetavad. Kuna ühendust andmebaasiga on mudelitele lihtne konfigureerida ning kontrollerid genereeritakse põhjalikud, saab Nodal.js abil kiiresti töötava prototüübini jõuda.

Nodal.js on teenuste loomisele orienteeritud ning ei oma mingisugust (hierarhiliste) vaadete loomise süsteemi. Seetõttu tuleb kasutajaliides koostada staatiliste failide abil või kasutada lisaserverit vaadete loomiseks kasutajaliidese jaoks.

Ei leidu põhjuseid lisaserveri loomiseks, kuna kasutajaliidese infoga täitmiseks kasutatakse REST teenuseid. Kasutajaliideseks vajalikud staatilised failid – lehe HTML, käivitatav JavaScript, stiililehed – saab Nodal.js server kliendile edastada. Tegemist on staatiliste failide edastamisega serverilt kliendile.

4.2.2 Andmebaas

4.2.2.1 Nõuded

Andmebaasile rangeid nõudeid pole. Sobivad üldtuntud relatsioonilised andmebaasisüsteemid. Soovitav oleks JSON tüüpi väljade tugi, kuna osa andmetest salvestatakse JSON kujul.

4.2.2.2 Valitud PostgreSQL

Valitud serveri raamistik Nodal.js omab PostgreSQL andmebaaside adapterit ning PostgreSQL toetab JSON tüüpi andmete salvestamist.

Lisaks arendatakse Nodal.js raamistikus spetsiifiliselt PostgreSQLis salvestatud JSON väljade info pärimist. [7] Antud hetkel pole skoopi kavandatud serveripoolset sissekannete andmete filtreerimist, kuid see võib uue funktsionaalsuse realiseerimise raames muutuda vajalikuks.

4.2.3 Kasutajaliides

4.2.3.1 Nõuded

Kasutajaliidese loomisel kasutatakse REST teenuseid veebilehe infoga täitmiseks. Tuleb kasutada teeki, mis vähendaks arendusaega, et nende teekidega infot pärida ning lehel kuvada.

Antud projekti arendusse pole kaasatud eraldi veebilehe kujundajat, seega on soovitav kasutada mõnd teeki kasutajaliidese kergeks laiendamiseks.

4.2.3.2 Valitud AngularJS, Bootstrap, ES6 Babel abil

AngularJS on teek HTMLis dünaamiliste vaadete loomiseks. [8] Selle abil saab lehekülge automaatselt täita määrates andmed JavaScripti tasemel – „two way binding“ [9]. AngularJS abil saab HTMLi kompileerida nii, et muutujate märgendid asendatakse väärtustega. See aitab laiendada kujundust või funktsionaalsust.

Bootstrapi kasutades saab arendaja ligipääsu suurele hulgale lisafunktsionaalsusele ning parema väljanägemisega tuntud HTML võimalustele. [10]

AngularJSi ja Bootstrapi kooskasutamiseks on loodud eraldi teek, et need paremini koos töötaksid. [11] Bootstrapi komponendid kirjutati ümber AngularJSi abil kasutatavaks.

Kasutajaliidese arendamise raames ES6 funktsionaalsuse kasutamiseks tuleb koostatud ES6 kood kompileerida JavaScripti versiooniks, mida tuntud veebilehitsejad toetavad. [5] Kompileerimiseks sobib Babel, mille tööd saab automatiseerida käsurealiidese abil. [12]

5. Andmemudel

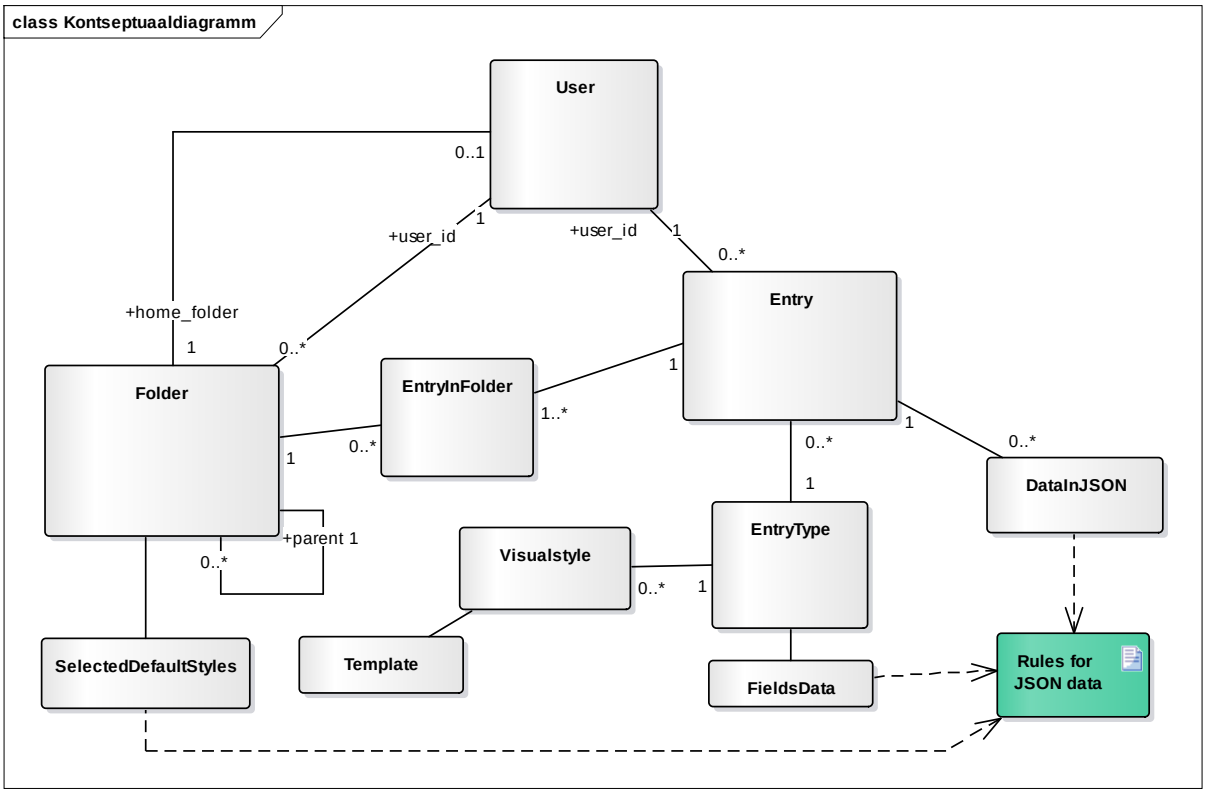
5.1 Eesmärk ja nõuded

Andmemudel peab olema võimeline toetama sisuliselt pea kõiksugu erinevat informatsiooni, kuna salvestatavate objektide andmeväljad defineeritakse kasutajate poolt, mitte süsteemidisainimise etapis.

Küll aga rakenduvad mõned piirangud või printsiibid:

- Sissekannete andmeid ei pea andmebaasi/serveri tasemel filtreerima. See tähendab, et andmeid võib salvestada JSON formaadis.
- Salvestatud andmetele rakendatakse mahupiiranguid. Kuna andmete uuendamiseks võib vaja minna kogu andmete ülekandmist, peab julgustama väiksemate sissekannete kasutamist. Selleks saaks piirata iga sissekande kogu JSON andmemahu suurust.
- Regulaaravaldiste implementeerimine aitaks kasutajatel oma andmete õigsust kontrollida ning tüüpide õiget kasutust tagada. Kahjuks võib regulaaravaldiste kasutus jõudluse mõttes väga kalliks osutuda. [13]

5.2 Kontseptuaalmudel



Joonis 2 Kontseptuaalmudel

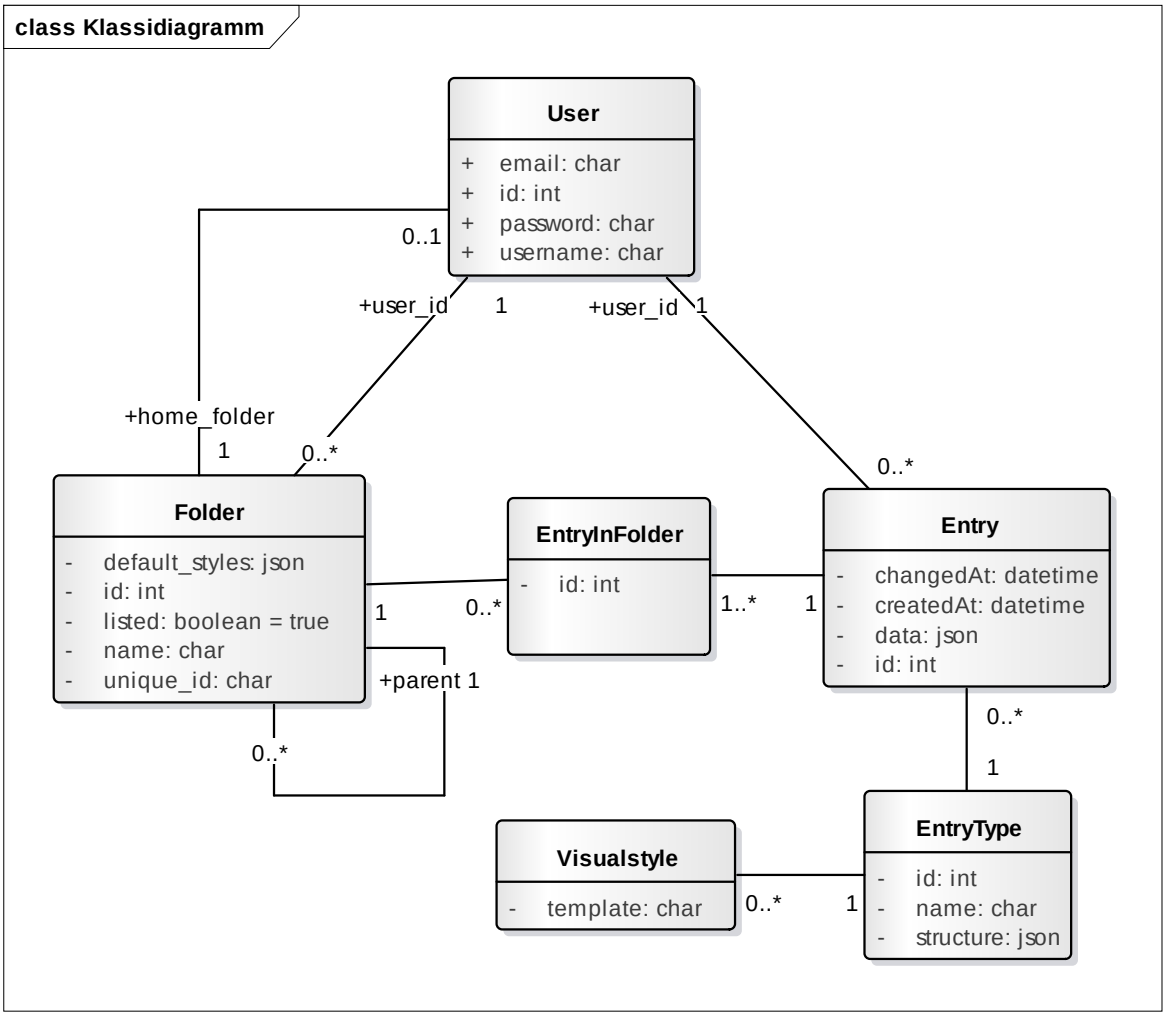
Keskseks kontseptiks on Sissekanded (inglise keeles „Entry“). Sissekandel leidub alati kasutajaga autor. Sissekanded organiseeritakse kaustadesse, kuid üks sissekanne võib esineda mitmes kaustas. Ning et anda kasutajatele võimalus oma kausta ning oma sissekandeid administreerida, kõigile kasutajatele luuakse kodukaust (inglise keeles „home folder“).

Sissekanded sisestatakse alati mingi tüübiga. Tüübist sõltub, missugused andmed sellel sissekandel salvestatakse JSON formaadis. JSON andmed vastavad alati kindlale tüübile.

On võimalus luua visuaalseid kujundusi. Iga kujundus käib kindla sissekande tüübi kohta, kuid neid võib ühe tüübi kohta olla mitmeid. Kujunduse määrab defineeritud mall (inglise keeles „Template“), kuhu kindla sissekande kuvamisel sisestatakse selle spetsiifilised andmed.

Kausta külge võib salvestada vaikimisi valitud visuaalsed kujundused. Sel puhul kasutatakse jällegi JSON formaadis andmete salvestamist, mis peab käima ettemääratud reegleid järgides.

5.3 Klassidiagramm



Joonis 3 Klassidiagramm

Klassidiagrammi keskmeks on 3 tähtsaimat klassi:

1. Kasutaja („User“)
2. Kaust („Folder“)
3. Sissekanne („Entry“)

Kausta ning sissekande looja saab alati objekti omanikuks. See annab võimaluse defineerida õiguste süsteemi, millega defineeritakse kasutajate võimalused teatud andmete suhtes – kas saab neid lugeda/kirjutada/muuta/kustutada?

Igale kasutajale luuakse oma kodukaust, mille omanik nad on. Sedasi saab globaalsesse kaustapuusse anda kuskil kasutajale endale võimaluse sissekandeid luua ning soovi korral peita.

Kaustapuu tähendab, et kõik kaustad, peale kõige ülemise „root“ kausta, omavad vanemkausta. Sedasi saab kaustade ligipääsu organiseerida.

Kausta ning Sissekannet ühendab „EntryInFolder“ vahetabel. Selle abil saab sama sissekannet kuvada mitmes kaustas.

Sissekande välimuse realiseerimiseks on Sissekande tüüpidele defineeritud visuaalsed stiilid. See tähendab, et spetsiifiliste sissekannete kuvamiseks saab valida loodud visuaalseid stiile.

5.4 Reeglid JSON andmetele

JSON andmetele reeglite rakendamine võib käia kasutades näiteks JSON Schemat, [14] kuid antud projektiks ei tundu see vajalik. Kui kindlat teeki andmete valiidsuse verifitseerimiseks ei kasuta, siis tuleb teha andmetekontroll käsitsi või otsustada ignoreerida ebakorrektseid andmeid.

Antud projektiks määratakse JSON formaadis salvestatud andmete struktuur lahtikirjutatult ning näidetega. Lõppkasutajad aga võivad andmete salvestamisel vigu teha, kui otsustavad kasutada andmete salvestamiseks/muutmiseks otse serveri APIt ning eirata loodud kasutajaliidest. API kasutamisega saab luua enda oma kasutajaliides või ühendada loodud süsteem mõne muuga. Tuleb arvestada, et andmete valiidsust süsteem ei kontrolli praeguse skoobi kohaselt.

5.4.1 Sissekande tüübi andmed

Sissekande tüüp salvestab JSON formaadis massiivi objektidest. Iga objekt omab kahte võtit ning igal võtmel on väärtus.

1. „name“ – Välja nimi
2. „type“ – Välja tüüp, mille võimalikud väärtused eelnevalt defineeritud

Näide:

```
[
        {
                "name" : "url"
                "type" : "string"
        },
        {
                "name" : "url_name"
                "type" : "string"
        }
]
```

5.4.2 Sissekande andmed

Sissekannete andmed salvestatakse tavaliste võti-väärtus paaridena ühes objektis. Võtmed on väljade nimetused – need defineeritud sissekande tüübi kaudu.

Näide:

```
{
    "url" : "http://www.google.com",
    "url_name" : "Google Search Engine"
}
```

5.4.3 Kausta vaikimisi visuaalse stiili valikud

Kausta vaikimisi visuaalseid stiile saab salvestada selleks, et sissekandeid soovitud viisil näidataks kõigile kausta avajatele. JSON formaadis selleks salvestatakse valikud ühte objekti võti-väärtus paaridena, kus

- võti: sissekande tüübi ID
- väärtus: võtmes defineeritud sissekande tüübi kohta käiva valitud visuaalse stiili ID

Näide:

```
{
    "1" : 2,
    "2" : 2,
    "10" : 4
}
```

5.5 Kasutajaliides andmete jaoks

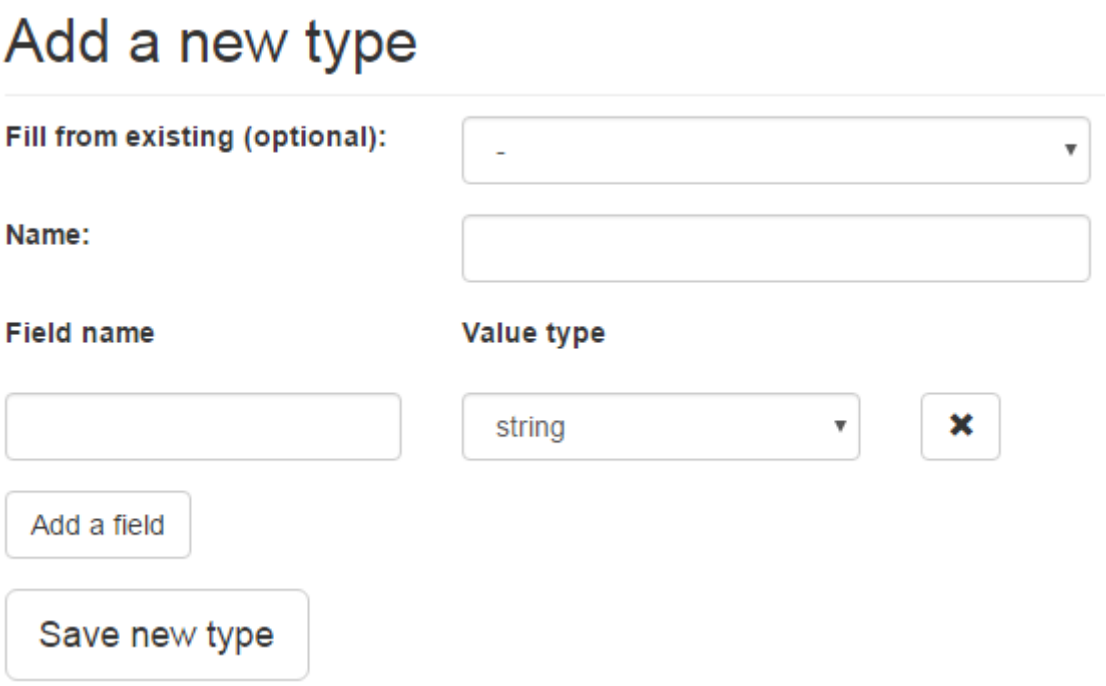
5.5.1 Sissejuhatus

Tüübi loomine tehti eraldiseisvana sissekannete loomisest, et need kaks osa lahutada. Sedasi on süsteemis salvestatud erinevad tüübid ning kasutajad saavad neid kasutada isegi, kui pole ise neid loonud. Kindlalt defineeritud tüübid annavad võimaluse kasutajaliidest täpselt juhtida ning lihtsustada. See tähendab, et pole vaja sissekande loomise hetkel läbi mõelda, missuguseid välju kasutaja soovib salvestada, vaid kasutaja valib tüübi ning kasutajaliides kuvab sellega seotud väljadest lähtuva vormi.

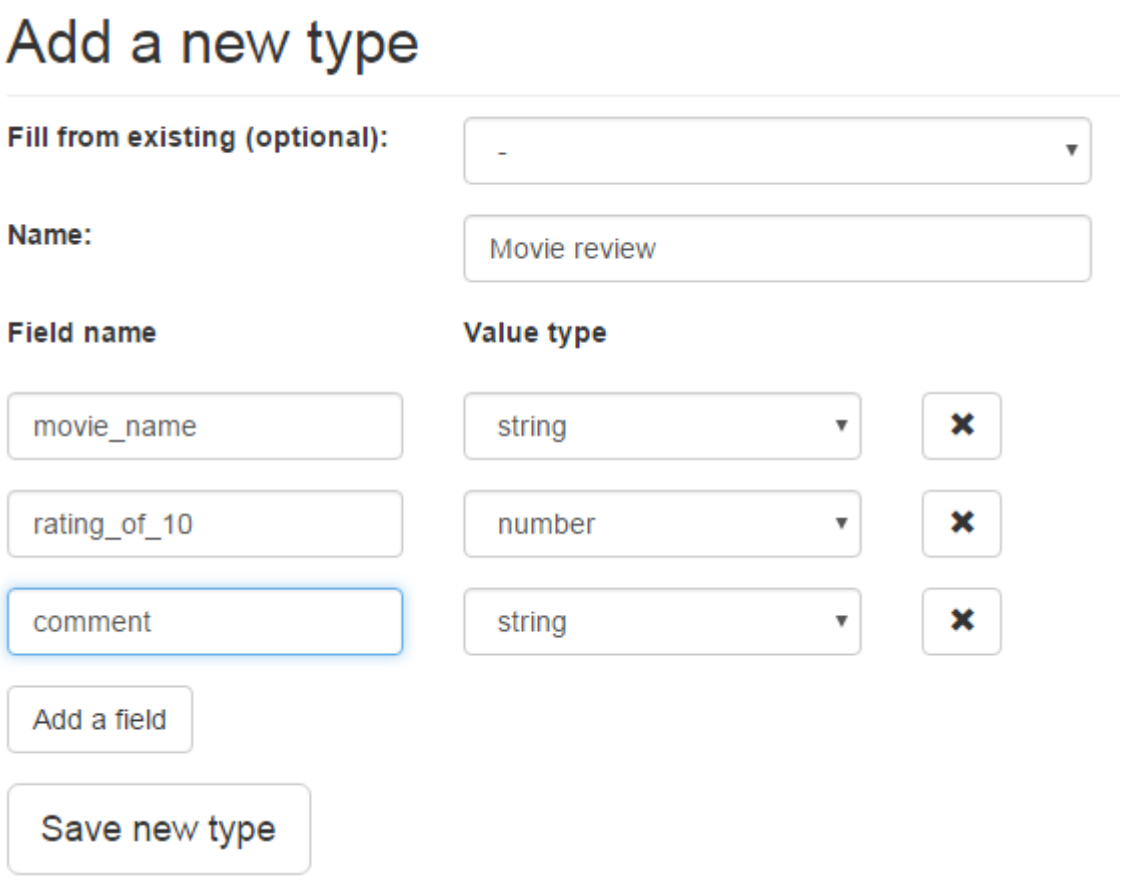
Antud peatükis on erinevad andmete manipuleerimise etapid läbitud – antud nende kohta kirjeldus ning mõned ekraanitõmmised. Kirjelduses selgitatakse, miks valitud lähenemine on kasulik lähtudes teoreetilisest alustest.

5.5.2 Kasutajaliides uue sissekande tüübi loomiseks

Sissekande tüübi loomiseks peab kasutajalt saama uue tüübi nime ning sisestatavad väljad koos nimede ning tüüpidega (kas tekst/sõne ehk „string“ või number ehk „number“). Tüübid on relevantsed funktsionaalsuse laiendamise kontekstis. Välju saab lisada ning likvideerida. Alternatiivselt saaks lubada kasutajatel otse JSON formaadis defineerida tüüp nii, nagu see antud kasutajaliidese tulemusena peidetult koostatakse. Realiseeritud kasutajaliidese puhul on ebatõenäolisem kasutaja vea esinemine ning kindlasti kergem algajatele kasutajatele.



Joonis 4 Uue sissekande tüübi loomine (tühi)

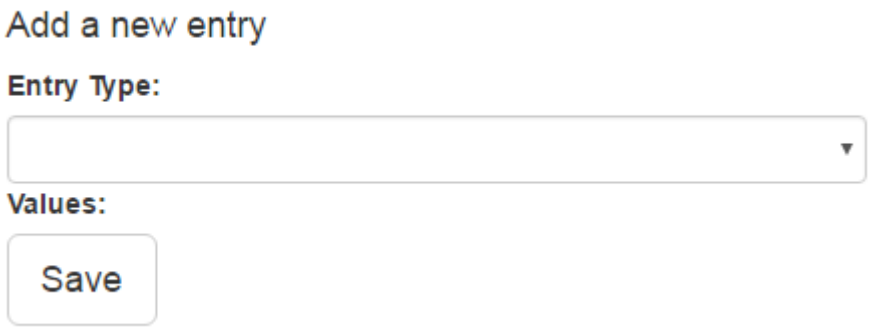


Joonis 5 Uue sissekande tüübi loomine (täidetud)

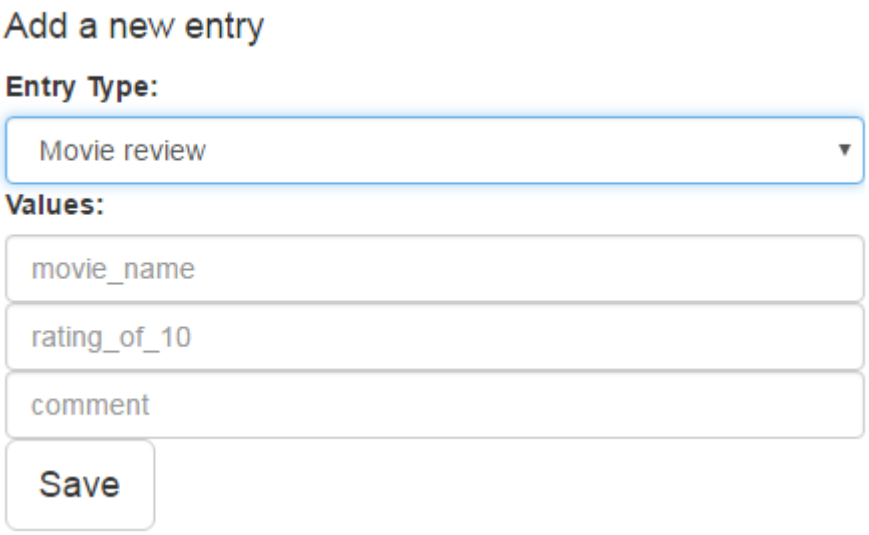
5.5.3 Kasutajaliides sissekande loomiseks

Sissekande loomiseks tuleb esmaselt valida tüüp. Seejärel kuvatakse kasutajale kõik selle tüübiga kaasnevad väljad.

Antud lahenduse puhul ei kuvata konstantselt iga sisendi välja juures sisendi silti, kasutatakse vaid kohatäidet (HTML „placeholder“ atribuut). See võib kohati segadust tekitada, kui läheb vormi täitmisel meelest ära, kuid üldiselt on julgustatud väiksemate sisendipikkustega tüüpide loomine. See tähendab lühikest välja täitmise aega ning seega lühimälu kasutust. Nõnda jääb kasutajaliides võimalikult minimalistlikuks.



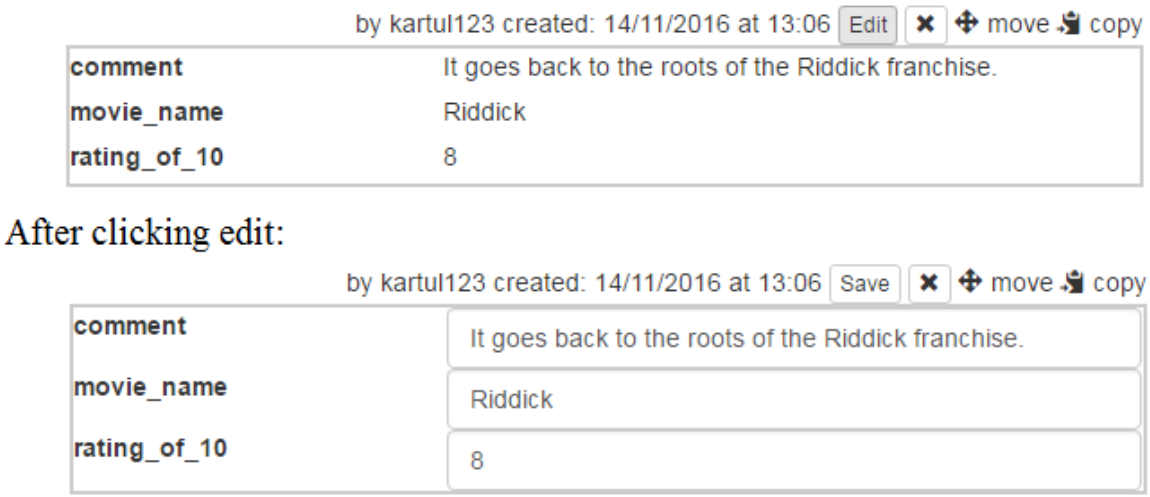
Joonis 6 Uue sissekande loomine



Joonis 7 Uue sissekande loomine pärast tüübi valikut

5.5.4 Kasutajaliides sissekande info muutmiseks

Sissekande muutmiseks tuleb valida nende nimekirjas õige sissekande muutmiseks mõeldud nupp („Edit“). See teeb sissekande väljad muudetavaks. Kui aga on valitud juba sissekandele tüüp, siis lisatakse need väljad. Muutmise nupp vahetub välja salvestamise nupuks. Alternatiivselt saaks uue salvestamise nupu lisada ettekande lõppu, et püsiks üldtuntud ülevalt-alla loogiline järjekord.



Joonis 8 Enne ning pärast sissekande muutmise avamist

6. Sissekande kujunduse määramise võimaldamine

6.1 Sissejuhatus

Antud peatüki eesmärk on leida lahendusi, kuidas lubada kasutajal andmete kuvamist ise defineerida. Selleks pakutakse välja erinevaid lahendusi ning valitakse antud projektis sobivaim. Alternatiivid võivad osutuda valituks mõne muu sarnase projekti raames, kui probleemid on lahendatud või aktsepteeritavad. Samuti pole välistatud, et saaks kasutada lahenduste kombinatsioone, sest need võivad üksteist täiendada.

6.2 Võimalikud lahendused

6.2.1 Lahendus 1: otsene HTML ja CSS kirjutamine

Üks lihtsamaid võimalusi oleks lasta kasutajatel defineerida ise CSS keeles stiili lisaks mingit sorti HTMLi defineerimisele. HTMLini võib jõuda kasutades BB-Code tehnoloogiat. CSS peaks käima kas spetsiifiliselt tüübi või kausta kohta. Kausta laadimisel või stiili valimisel laetakse alla stiili kohta käiv CSS ning käivitatakse see.

6.2.1.1 Lahenduse probleemid ja turvariskid

CSS abil on võimalik käivitada HTTP päringuid, kui kasutada CSS sees funktsiooni „url()“. Seetõttu tuleb alati vähemalt see funktsioon CSS seest välja filtreerida. [15] Samuti ei tohiks see CSS kuidagi mõjutada lehel olevaid muid elemente. Selleks on kaks võimalust:

1) CSS luuakse spetsiifiliselt elemendi jaoks ning lõppkasutajad ei kasuta ise CSS väljavalijaid (inglise keeles „selector“). Kui mitu elementi on sarnase stiiliga, tuleb lõppkasutajal sama asja korduvalt defineerida.

2) Limiteeritakse ja filtreeritakse, milliseid CSS väljavalijaid on õigus kasutada. Kõik üleliigsed definitsioonid likvideeritakse.

CSS jaoks peaks lõppkasutaja oskama seda keelt kirjutada ning haldama viiteid elementidele. Kui andmetüüp ja seda kuvav HTML tuleb keeruline ja/või pikk, võib CSS (või väljavalijad) samuti liigkeerukas olla. Sisuliselt nõutakse selle lahendusega lõppkasutajalt veebiarenduse põhilisi oskusi ning võib olla ületamatu neile, kes pole kunagi veebilehti koostanud ning puudub huvi selle õppimiseks.

See lahendus säilitab veebitehnoloogiates üldtuntud struktuuri ja kujunduse kaheks jaotamise, kuid tähendab, et kasutajal tuleb hallata ja luua mõlemad. See võib algajatele lõppkasutajatele keerulisem olla.

6.2.2 Lahendus 2: eeldefineeritud kujunduselemendid

Saaks defineerida kasutajatele hulga kujunduselemente, mille seast valitakse ning kombineeritakse. Tuleks luua uus raamistik või implementeerida mõne teegi kasutus, mille abil elementide asukohta ning stiili saaks defineerida, andmebaasi salvestada ning teistele kasutajatele kuvada.

6.2.2.1 Lahenduse probleemid ja turvariskid

Antud projektis see lahendus pole sobilik, kuna seda võib lugeda kõige töömahukamaks. Eeldefineeritud elementidega saaks limiteerida turvariske, kuid lõppkasutajate võimalused oleks piiratud neile pakutavaga. Kui tekivad uut sorti nõuded, peaks süsteemidisaini tasemel võimalusi laiendama.

Et teada, missuguseid kujunduselemente on vaja, peaks süsteemidisaini tasandil tegema uurimistööd (potentsiaalsete) lõppkasutajatega. Sedasi saab kindel olla, et loodavad elemendid on vajalikud ning midagi kriitilist kasutajatel puudu ei jää. Süsteemidisaini tasandil lihtsalt ei teata, missugused nõuded võivad lõppkasutajatel tekkida.

6.2.3 Lahendus 3: BBCode'ide kasutamine koos CSS sisestamisega

Lõppkasutajad saavad kujunduse luua kasutades üldkasutatud Bulletin Board Code'e. [16] BBCode kasutatakse just turvariskide maandamiseks ning HTML keerukuste peitmiseks lõppkasutajate eest.

BBCode'e saab ise defineerida ning muuta, kuid võib alustuseks võtta need, mida näeb tihti teistes infosüsteemides: paks kirjatüüp, kaldkiri, lingid, listid jt. Soovitatav oleks kasutada mõnda teeki, mida saab ise süsteemidisaini tasemel laiendada.

Isedefineeritud märgendile (inglise keeles „tag") saab külge panna stiili sisendi. See annab mugava valikulise võimaluse lõppkasutajal CSSi kirjutada nii, et see mõjutab täpselt defineeritud skoopi.

Lahenduse raames on sissekande kujunduse struktuur ja väljanägemine (piltlikult HTML ja CSS) ühes koos ning pole vaja neid eraldi hallata.

6.2.3.1 Lahenduse probleemid ja turvariskid

Lõppkasutajale antakse võimalus sisestada otseselt CSSi lehele, mida kuvatakse teistele kasutajatele. Sellega kaasnevad probleemid kattuvad nendega, mis kaetud peatükis „6.2.1.1 Lahenduse probleemid ja turvariskid." CSS kirjutamise lubamine ning salvestamine, taasesitamine tekitab XSS riski. [15]

6.2.4 Lahendus 4: kujunduste pakkumine ning kinnitamine

Oleks võimalik eelnevalt pakutud lahendusi kasutada laiendusega, et kõik lõppkasutajate pakutud kujundused vaadatakse üle mingit sorti administreeriva rolli esindaja poolt. Sedasi saaks tagada paindlikkust ning toetada tehnoloogias kõiki võimalusi, kuid samas lükata tagasi kujundused, mis näivad pahatahtlikena.

Algselt saaks kõikide loodud kujunduste kasutust piirata vaid autorile endale. Sedasi võib potentsiaalselt pahatahtlikku kujundust kasutada vaid selle looja.

6.2.4.1 Lahenduse probleemid ja turvariskid

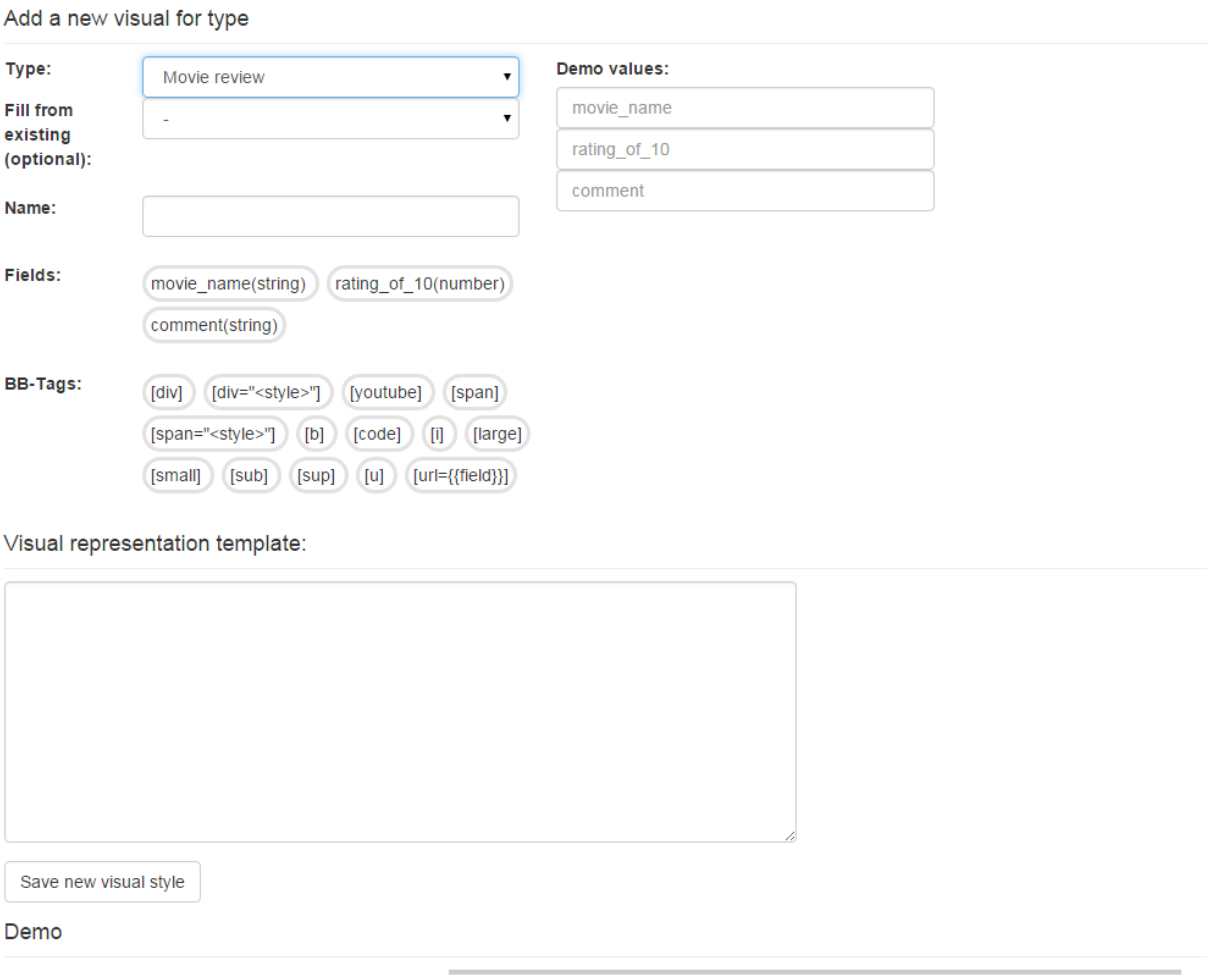
Pakutud kujunduste ülevaatamine ning kinnitamine sisaldab suuremat inimfaktorist lähtuvat riski. See tuleneb võimalusest, et tegelikult pahatahtlik kujundus kiidetakse süsteemis heaks ning tehakse kättesaadavaks ka teistele kasutajatele. Tuleb ka arvestada, et potentsiaalselt pahatahtlik lahendus on infosüsteemi salvestatud isegi kui otsustatakse, et praegusel hetkel pole see kättesaadav. Selline fakt muutub relevantseks infosüsteemi evolutsiooni ja suurtemate muudatuste kontekstis.

Isegi kui lõppkasutajatel on võimalus saada kujunduste ülevaatajaks, tekib süsteemidisanija ja lõppkasutaja vahepealne roll. See tähendab, et süsteem vajab märgatavalt suuremat administreerimist ning tuge ehk suuremat tööjõu hulka. Kui selliseid usaldusväärseid administraatoreid kogukonnast ei leita, peab siiski infosüsteemi arendajate meeskond tagama, et rolli ülesanded oleks täidetud.

6.3 Valitud lahendus ja selle realisatsioon prototüübis

Valiti lahendus 3: BBCode'ide kasutamine koos CSS sisestamisega. See annab tahetud paindlikkuse, kuid omab ka lihtsustavat kihti. BBCode'ide kasutamine pole algajatest lõppkasutajatele liiga keeruline, kuna nende jaoks on võimalik teha abistav kasutajaliides, mis sisestab kasutaja valitud märgendi ise.

6.3.1 Kujunduse loomine



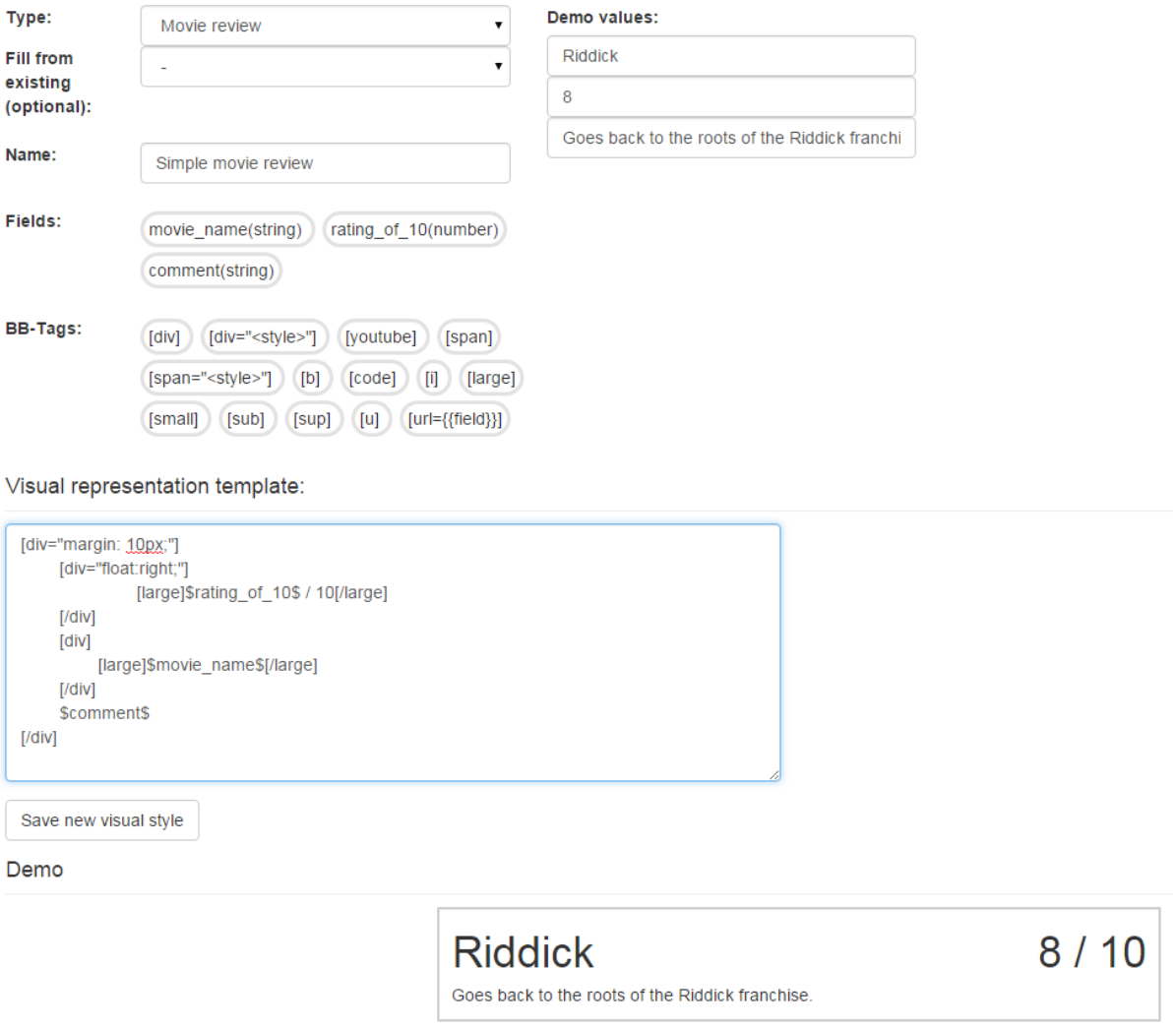
Joonis 9 Uue kujunduse loomise täitmata vorm

Vormil tuleb esmaselt valida tüüp, millele kujundus luuakse. Sedasi vorm muutub, et kuvada informatsiooni, mis aitab „Template" – sisult tekstiala – koostamisel. Kasutajale antakse abistavad nupud, et teha tekstivälja täitmist lihtsamaks, kuid nende kasutamine pole kohustuslik.

Tekivad nupud sektsioonis „Fields“, mis sisestavad vajutamisel tekstialasse väljadele määratud sümbolite jada. Näiteks, kui välja nimi on „movie_name“, siis sisestatakse kokkuleppeliselt välja nimi kahe $-märgi vahel – „$movie_name$“. Individuaalse sissekande vaatamisel asendatakse see sissekande selle välja väärtusega.

„BB-Tags“ sektsioonis on nupud, et sisestada toega BBCode’i märgendid. Märgendite nimekirja saab laiendada, kui süsteemis kavandatakse uute märgendite tuge.

Loodud tüübi valimise järgselt kuvatakse kasutajale vormis koht, kuhu panna proovitavad väärtused (inglise keeles „Demo values“), et näidis tuleks sisukas ning võimalikult elutruu – kasutajal oleks hea aimdus, kuidas selliselt sisestatud kujundus välja näeb (näide vt „Joonis 10 Täidetud kujunduse lisamise vorm“). Kohese tagasiside abil saab keerulisemad süsteemi osad kasutajatele selgemaks teha.



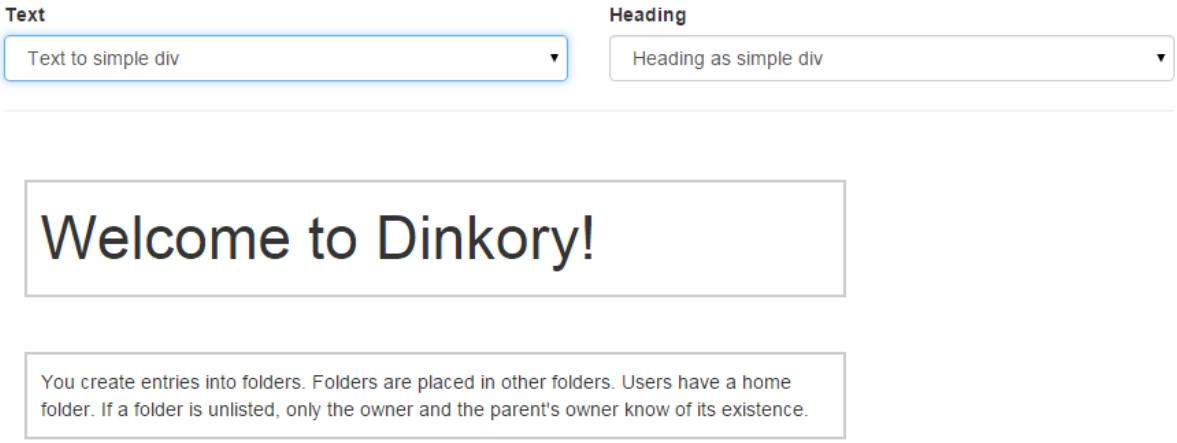
Joonis 10 Täidetud kujunduse lisamise vorm

6.3.2 Kujunduse valimine ja muutmine vaadates sissekandeid

text	Welcome to Dinkory!
text	You create entries into folders. Folders are placed in other folders. Users have a home folder. If a folder is unlisted, only the owner and the parent's owner know of its existence.
text	Entries have a type. Type defines what fields each entry of this type has.
text	Visual styles define what a type looks like. You can save default visual styles for a folder. You are currently seeing them.
text	Folders can have unique ID-s. For example, the root folder has 'root' and therefore it can be reached by adding "?root" to the url. http://52.50.182.93:3000/?root

Joonis 11 Sissekanded ilma kujunduseta

Sissekanded omavad vaikimisi väljanägemist, et isegi ilma kujunduseta oleks andmeid võimalik vaadata (vt „Joonis 11 Sissekanded ilma kujunduseta“ lk 52). Kujunduse valiku tegemisel koheselt selle tüübi sissekannete väljanägemine muutub. (Samade sissekannete puhul erinevuse nägemiseks vt „Joonis 12 Sissekanded valitud kujundustega“ lk 53.)

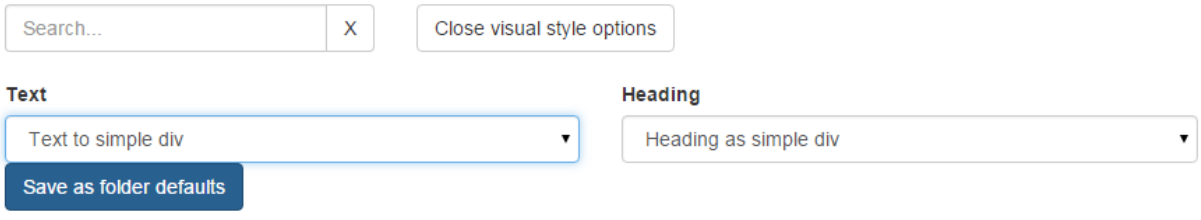


Joonis 12 Sissekanded valitud kujundustega

6.3.3 Kujunduse valiku salvestamine kausta tasemel

Pole mõistlik eeldada, et kasutaja igal kausta avamise korral valib uuesti, missugused kaustas olevad sissekanded välja peavad nägema. Kui kasutaja on omale valinud mõne meelepärase väljanägemise, oleks hea see kuidagi infosüsteemi salvestada. Lisaks aitab see kasutajal kuvada enda loodud sissekandeid teistele kasutajatele just nii, nagu soovitud. Informatsiooni edastamise viisist sõltub, kui hästi lugeja seda hoomab.

Korduvalt samade kujunduste valimise asemel on süsteemis kasutajal võimalus salvestada tüüpide vaikimisi kujundusvalikud kausta tasemel. Valikud salvestuvad kausta objekti külge, seega nende määramiseks peab kasutaja olema kausta omanik.



Joonis 13 Kausta vaikimisi valikute salvestamine

7. Funktsionaalsuse laiendamise võimaldamine

Funktsionaalsus on kolmas aspekt, mis lahendada. Universaalse andmemudeli loomine ning veebikeskkonna kujunduselementide kasutatavaks tegemine jääb projektis realiseeritavate osade hulka. Kahjuks pole realistlik funktsionaalsuse laiendamise toe arendus, kuna selleks vajalik töö hulk on liiga suur.

Peatükis antakse mõned võimalikud lahendused, kuid suurimaks limiteerijaks on antud projekti puhul töö maht. Peatükis „7.2 Realiseeritud lahendus“ lk 56 selgitatakse, missugused valikud oleks tehtud ilma tööjõu piiranguta.

Funktsionaalsuse laiendamine jaguneb kaheks:

1) Sissekannete loomine muust allikast saadud andmete põhjal. Näiteks RSS voost, mõnest REST või WSDL veebiteenusest.

2) Sissekannete andmete manipuleerimine kuvamiseks. Näiteks salvestatud ajahetke põhjal kuvada aeg formaadis „aega selle hetkeni järgi“, matemaatilised tehted salvestatud numbritega, tinglik kuvamine või kujundus vastavalt mingitele andmete kriteeriumitele. Selle toetamiseks on salvestatud sissekande tüüpide külge iga välja tüüp – väljas olev info on tekst või number.

7.1 Võimalikud lahendused

7.1.1 Lahendus 1: lubada kasutajate enda skriptide käivitamine

Kasutajad saaksid süsteemi ise sisestada käivitatavaid skripte. Skriptitoe võiks teha Javascripti või muu keele jaoks. Suvalise skripti käivitamine kasutajate masinatel ilma ülevaateta on kohutavalt suur risk. Kahju tekkimist ning ulatust loeks suureks.

Ebarealistlik oleks luua keelatud käskude nimistut, mille abil filtreerida välja keelatud käsud kasutaja pakutud skriptist. Alternatiivselt saaks skriptide ülevaatamiseks luua administreeriva rolli. Probleemid oleksid sarnased neile, mida kirjeldati peatükis „6.2.4.1 Lahenduse probleemid ja turvariskid“ lk 48. Uue rolli loomine ning sellele vastutuse andmine nõuaks lisatööjõudu ning suureneb inimvea risk.

7.1.2 Lahendus 2: REST teenustel tuginemine

Kui rakenduse aluseks luuakse kättesaadavad REST teenused, on lõppkasutajatel võimalus luua täiesti oma kasutajaliides või mingit sorti oma tarkvara, näiteks skriptide abil luua ning sisestada andmeid infosüsteemi. Sel viisil võivad kasutajad oma kõiksugu andmeid talletada loodud süsteemis, kuid peavad ise olema võimelised vastavat tarkvara looma. Kahjuks ei saa eeldada, et kõik (või enamik) lõppkasutajatest oleksid vastavate teadmiste ning oskustega.

Laiendusena võib lubada kasutajaliidese abil JSON formaadis andmete sisestamist. See tähendab, et üks osa – otse REST teenustega suhtlemine mingi tarkvara abil – on kaetud lihtsustusega. Kasutajad peavad siiski olema võimelised oma andmeid muundama aktsepteeritavasse JSON formaati.

Sellise lahenduse puhul on andmete muutmine/uuendamine agregeeritult raskendatud. Iga sissekanne tuleb eraldi muuta või kustutada ning asendus luua.

7.1.3 Lahendus 3: teekidega tehnoloogiate toetamine

Võimaldada andmete muutmine mingit sorti teekide abil. Nende teekide seas oleks tugi tehnoloogiatele nagu näiteks RSS voo lugeja, XML ja JSON andmete sõelumine, ämblik

(inglise keeles „Web Crawler“) jt. Iga selline ala vajaks individuaalset lähenemist ning kasutajatele toe loomist (plus hooldust).

Arvukate teekide kaasamine süsteemi vajaks suurt tööjõuhulka. Leidub mitmeid tehnoloogiaid, mille tuge võivad lõppkasutajad tahta ning pärast implementeerimist tuleb neile rakendada järelvalvet. Samuti on kõik lisatavad teegid potentsiaalselt turvariskid.

7.1.4 Lahendus 4: turvalise skriptikeele loomine

Oleks võimalik luua uus skriptikeel, mille kasutamisega pole võimalik lubatud piiridest välja minna. Seda võiks vaadelda kui veebitehnoloogiates kasutatud Javascripti osahulka – uude keelde lubatakse ainult teatud funktsionaalsus, et vältida XSS tüüpi probleeme.

Antud lahendus on vastupidine lähenemine lahendusele 1 (vt „7.1.1 Lahendus 1: lubada kasutajate enda skriptide käivitamine“ lk 55) – saab välja jätta skriptist keelatud read või (nagu selles lahenduses pakutakse) luua lubatud käskude alamhulk. Kogu keele läbi käimine, et luua potentsiaalselt kahjulike käskude nimekirja, nõuab suurt tööhulka ning pole ilmtingimata lõpuni tehtav töö. Kui keel areneb või standard muutub, tuleb reegleid kohandada. Seetõttu on realistlikum luua heakskiidu saanud käskudest alamhulk või koostada täiesti uus skriptikeel.

Loodav skriptikeel peaks prioritiseerima kasutajate vajadusi ning arvestama nõudega, et kõik lõppkasutajad pole programmeerimise (või skriptide kirjutamise) oskustega. Julgustav oleks tugev visuaalsete elementide kasutamine, kuid visuaalne lihtsustamine ei tohi funktsionaalsust limiteerida. Süsteemidisainija ei oska ette näha, kuidas lõppkasutajad süsteemi utiliseerida soovivad.

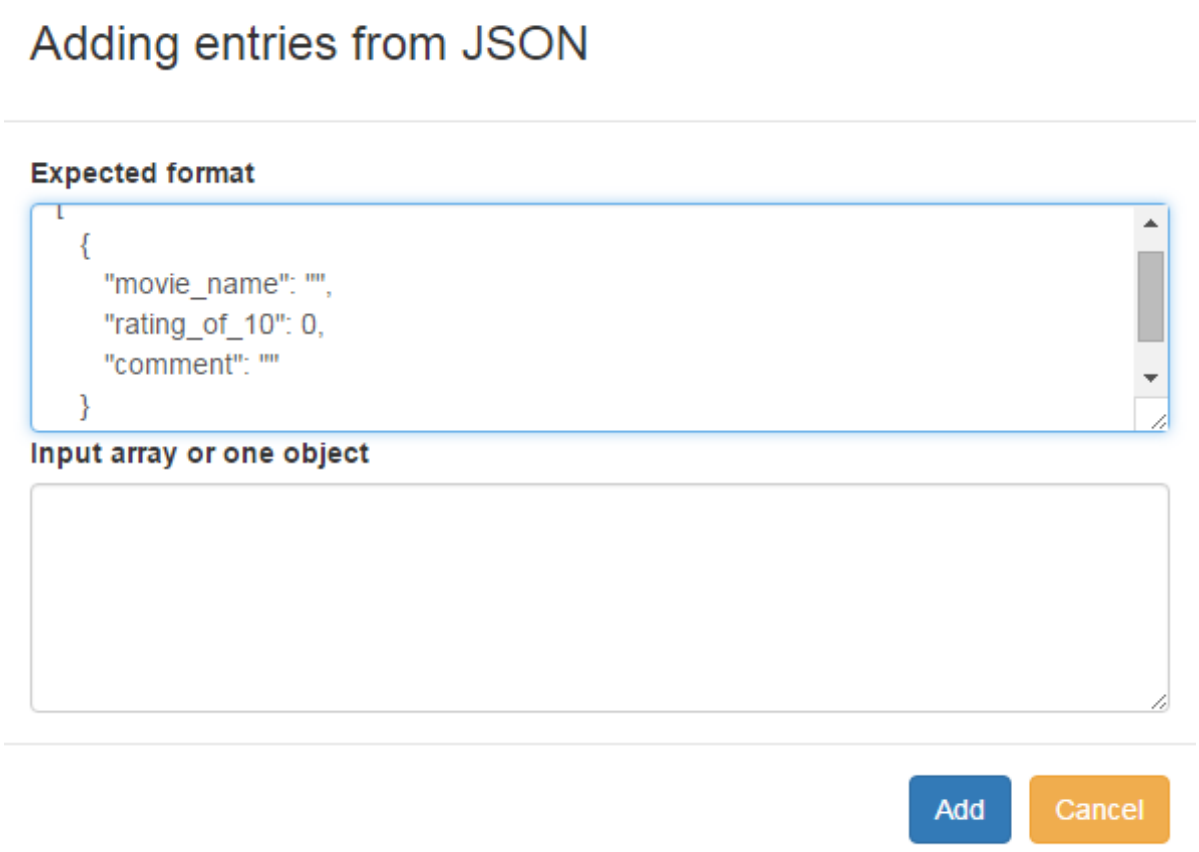
7.2 Realiseeritud lahendus

Antud projekti raames pole võimalik piisavalt sisuka skriptikeele loomine. Seetõttu ei saa kasutada peatükis „7.1.4 Lahendus 4: turvalise skriptikeele loomine“ lk 56 pakutut, kuid see oleks ideaalvariant – võimaldab lihtsust ja turvalisust.

Süsteem on ülesse ehitatud REST teenuseid kasutates. Sinna kuulub ka OAuth [17] kasutus, et andmete muudatused oleksid seotud õigustega kontoga. Ülevaade loodud APIst vt „13 Lisa 1: API dokumentatsioon“ lk 68.

Piirangutest lähtuvalt on kasutajaliidesesse realiseeritud vaid sissekannete sisestamine ning kausta sissekannete andmete väljastamine JSON formaadis. Sissekannete sisestamisel kuvatakse oodatav sisendi formaat (inglise keeles „Expected format“), kus sõnede väärtuseks tühi sõne, numbrite puhul null (vt „Joonis 14 Sissekannete lisamine JSON formaadi kaudu“ lk 57). Kasutaja saab seejärel sisestada kas ühe objekti või objektide massiivi. Kui sisendit ei suudetud sõeluda, kuvatakse kasutajale veateade.

Kasutaja saab kõik kaustas olevad sissekanded lasta väljastada JSON formaadis. Seda sama väljundit võib kasutada mõne muu kausta sisendina – võimaldab kogu kausta sisendite korraga kopeerimist ilma viideteta. Hetkel on sama sissekannet võimalik kuvada mitmes kaustas, kuid pole kasutajaliideses ette nähtud sissekandest identse koopia tegemine.



Joonis 14 Sissekannete lisamine JSON formaadi kaudu

8. Prototüübi testimine

8.1 Põhiprogrammi testid

Põhiprogrammi API on kaetud SoapUI [18] testidega. Osa testides kasutatud sisenditest genereeritakse testi käigus, osa fikseeritud. Järgnevad kuus peatükki selgitavad testide gruppe.

Testid käivitamisel läbitakse kõik edukalt. Vastuste testimiseks testitakse tagastatud olekukoode ning kontrollitakse, kas vastuses leidub veateade või mitte.

8.1.1 Kasutaja ning ligipääsuloa haldus

Esmalt luuakse uus kasutaja. Üritatakse sisestada sama kasutajainfot uue kasutaja loomisel kontrollimaks, et pole võimalik luua kattuvate andmetega kasutajat. Seejärel toimub päring kasutaja info muutmiseks. Järgnevalt proovitakse kasutajaga väljalogimist ning uuesti uute andmetega sisselogimist. Luuakse ka teine kasutaja, mida läheb vaja teiste testide gruppide läbimisel.

8.1.2 Kaustahaldus

Kaustahalduse testimine algab lihtsalt kaustade pärimisega ligipääsuloaga ning ilma. Tuleb tagada, et privaatsed kaustad ei kajastuks kolmandatele isikutele. Seda on põhjalikumalt uuritud testide grupis number 6 (vt „8.1.6 Kaustadega seotud õiguste testimine“).

Kausta loomist proovitakse ilma vanemat täpsustamata, ilma vanema omanikuõiguseta enne, kui edukas päring tehakse. Seejärel proovitakse seda kausta muuta ning kustutada: ilma õigusteta, ilma omaniku õigusteta.

8.1.3 Sissekandehaldus

Sissekandeid üritatakse luua ning muuta kolme päringuga: ilma ligipääsuloata, ilma omaniku ligipääsuloata ning alles siis edukalt.

8.1.4 Sissekande tüübi haldus

Sissekande tüüpe vaid päritakse ning luuakse. Loomiseks vajatakse kehtivat ligipääsuluba.

8.1.5 Visuaalse kujunduse haldus

Visuaalseid kujundusi vaid luuakse ning päritakse. Loomisel oodatakse kehtivat ligipääsuluba.

8.1.6 Kaustadega seotud õiguste testimine

Esmaselt luuakse kaust. Kaustale luuakse neli alakausta, millest kaks on peidetud teiste eest. Kaks loodud kaustadest antakse erinevatele teistele kasutajatele – üks nendest kaustadest oli peidetud. Eesmärk on kaustas olevate alamkaustade pärimisel kõigile kasutajatele kuvada erinev tulemus vastavalt nende rollidele kaustade suhtes. Selleks käivitatakse kolm päringut:

1) Päritakse alamkaustad ilma ligipääsuloata. Tagastatakse need kaks kasuta, mis polnud peidetud.

2) Päritakse alamkaustad kausta omanikuna. Kausta omanikule kuvatakse kõik neli.

3) Päritakse alamkaustad ühe peidetud alamkausta omanikuna. Sellisel puhul tagastatakse peitmata kaustad ning selle kasutaja enda peidetud kaust (kuid mitte teisele kasutajale kuuluv peidetud kaust).

8.2 Kasutajaliidese kontrollimine stsenaariumite põhjal

Kõige tähtsam on süsteemi puhul kontrollida, kas see vastab ootustele ning rahuldab lõppkasutajate vajadused. Selle kontrollimiseks käiakse antud peatükis läbi algselt defineeritud stsenaariumid (vt „3.2 Stsenaariumid“ lk 20) ning antakse kommentaare, kas tulenevad nõuded on täidetud.

Stsenaariumitest on hetkel toetatud ilma suuremate puudujääkideta kõik.

8.2.1 Stsenaarium 01

Süsteemis on võimalik luua järjehoidja objekti. Pealkirja, URLI ja märksõnad oleksid sissekande tüübi väljad. Kausta sisu saab filtreerida. Kausta otselinki saab määrata.

8.2.2 Stsenaarium 02

BBCode’e laiendati, et toetada YouTube videosid. Hetkel pole võimalik määrata, kui suur kuvatav video on, seega peab kujunduse loomisel sellega arvestama. Stsenaariumit saab toetada, kui kasutaja teeb vajaliku sissekande tüübi ning loob visuaalse kujunduse.

8.2.3 Stsenaarium 03

JSON formaadi tugi loodi. 3D objektide väärtuste salvestamiseks on vajalik sobiva sissekande tüübi loomine.

8.2.4 Stsenaarium 04

Antud stsenaariumi toetamiseks loodi kaustad ning nende peitmise tugi. Märkmed tuleb sissekannetena luua ning neid kuvada mitmes kaustas.

8.2.5 Stsenaarium 05

Materjalide jaoks tuleb luua sissekannete tüübid. Selle stsenaariumi lihtsustamiseks loodi võimalus ka sissekannete tüüpe luua teiste tüüpide toel. See tähendab, et uue tüübi loomisel saab kasutada eelmiste sisendeid – tuleb lisada või välja jätta välju vastavalt vajadustele.

Kaustast otsimiseks peab värv kajastuma eristatavalt sõnena sissekande ühes (või mitmes) väljas – otsingu puhul kontrollitakse, kas sissekande väljade liitsõne sisaldab otsingusõnu. Otsimise puhul ei saa määrata, et otsitaks vaid ühelt väljalt.

9. Loodud prototüübi retrospektiiv

9.1 Tulemuse hinnang

Loodud prototüüp vastab ootustele, mis seati stsenaariumite kirjeldamisega. Nende toetamiseks loodud kasutusjuhud said kõik realiseeritud. Põhiprogramm sai kaetud API testidega. Nende abil vigu ei tuvastatud.

Süsteemi kasutajaliides pole visuaalselt väga arenenud. Selle põhjustab vähene tööjõud – prioriteediks oli funktsioneeriv veebileht, mistõttu kasutati kättesaadavaid kasutajaliidese arendamisega seotud teeke (Bootstrap ja Angular näiteks).

9.2 Kavandatud süsteemi edasine areng

9.2.1 Põhjalikum turvaaukude uurimine

Antud projekti skoopi ei mahtunud põhjalik prototüübi testimine. Tuleks aga läbi proovida erinevate sisendite kombinatsioonid, et loodud filtritest läbi saada ning edukalt XSS rünnak sooritada. Kuna on teada, missuguses järjestuses toimub muutujate väärtuste sisestamine ning BBCode'ide sõelumine, saaks seda järjestust ära kasutada. Potentsiaalselt saab turvameetmetest läbi õige sisendi ning kujunduse kombinatsiooniga. Osa kaitsest annab asjaolu, et kasutati AngularJS teeki, mis pakub mitmesugust kaitset (näiteks teistelt domeenidelt ressursside mitte laadimine, sisendite filtreerimine jt).

9.2.2 Kasutajaliidese areng

Praegune kasutajaliides on loodud kasutades sellekohaseid teeke ning neis pakutavaid malle. Seetõttu pole praegusel süsteemil oma unikaalset väljanägemist.

Kasutajaliidese parendamine aitab kaasa kasutatavusele kui ka lõppkasutaja arhitektuurist tulenevate probleemide leevendamisele. Selge kasutajaliides suunab kasutaja tööd ning lihtsustab keerulisemaid kohti.

Prototüüp toetab kontode kasutust, kuid ei avane võimalus oma konto infot kasutajaliidese vahendusel muuta või kustutada.

9.2.3 Otsing väljade kaupa

Praegu saab kasutaja otsida vaid ühe märksõna või fraasi korraga. Saaks fraasi lõigata märksõnadeks, et otsitaks iga märksõna eraldi. Lisaks on Stsenaarium 05 (vt jaotis 3.2.5 lk 21) alusel soovitav toetada otsimist, milles kasutaja määrab väljad, kust märksõnu otsitakse. Sedasi saaks filtreerimise teha täpsemaks vältides otsingusõnade kajastumist ebatähtsates väljades.

9.2.4 Tüüpide ja kujunduste piiramine ühtsest nimeruumist

Kõik kontoga kasutajad saavad luua sissekande tüüpe ning nende kujundusi. Praegu salvestatakse need kõik võrdsetena, mis tähendab kiire ühtse nimeruumi täitumist. Tuleks leida viis, kuidas piirata kasutajatele pakutavaid tüüpe. Samas ei saa eeldada, et kasutaja kõik tüübid ise looks või üks haaval enda nimistusse kaasaks. See teeks paljudele lõppkasutajatele süsteemi kasutamiseks vajaliku eelneva õppimisperioodi liiga pikaks või tüütuks (vt „2.3.3 Kõik kasutajad ei pea olema arendajad“ lk 18).

Mõned kasutajad oskavad süsteemi paremini ära kasutada. Tuleks nende lõppkasutajate loodu teha kättesaadavaks ka teistele. See tähendab, et peab väärtustama kogukondlikku süsteemi evolutsiooni. Kõik kasutajad ei pea olema võimelised keerukaid tüüpe ning kujundusi looma.

9.2.5 BBCode’ide laiendamine vastavalt vajadusele

Süsteemi kasutuse käigus võib tekkida vajadus uute märgendite toe jaoks. Praegu on raske ette näha, millised need olla võivad, kuid tuleb kuulata lõppkasutajate tagasisidet. Ebavajalikud märgendid võib kasutajaliidesest likvideerida, et visuaalset müra vähendada.

9.2.6 Sissekannete järjekorrastamine

Antud hetkel kuvatakse sissekanded alati viimati muutmise järjekorras – hiljuti muudetud eespool. Tegelikult oleks hea lasta nii teiste väljade järgi kui ka täiesti kasutaja poolt ise määratud järjestamist.

10. Kokkuvõte

Töö eesmärgiks oli uurida lõppkasutajate arenduse võimaldamist veebirakenduse keskkonnas. Selleks uuriti teoreetilisi aluseid; püstitati ning analüüsiti nõudeid; projekteeriti ning loodi nõudeid rahuldav prototüüp; analüüsiti loodud tulemust ning kirjeldati projekteeritud edasist arengut. Töö käigus toodi välja erinevate aspektide jaoks mitu lahendust.

Üks tähtsamaid järeldusi on, et puudub veebirakenduste kontekstis sobilik skriptikeel, mida lõppkasutajad saaks kasutada funktsionaalsuse laiendamiseks, kuid skripti käivitamine oleks siiski turvaline.

Lõppkasutajate arenduse laialdasemaks kasutamiseks võiks luua üldkasutatava skriptikeele. Töö käigus ei selgunud loodavale keelele mittefunktsionaalseid nõudeid peale soovi, et see oleks võimalikult kasutajasõbralik ja lihtne, kuid funktsionaalsete nõuete jaoks tuleks teha suuremat uuringut potentsiaalsete lõppkasutajate seas.

Aspektide toeks pakuti välja erinevad lahendused ning mingit sorti lähenemine realiseeriti prototüübi abil. Lahendusi oli mitmeid ning selgitati nende sobivust antud prototüübi kontekstis. Funktsioneeriv prototüüp tähendab, et vähemalt valitud lahendus oli valiidne ja realistlik.

Prototüüpi hinnati ning anti viise, kuidas olemasolevat infosüsteemi saaks parendada ja edasi arendada lähtudes teoreetilistest alustest.

11. Summary

The purpose of this thesis was to explore the validity of end user development in the context of a web application. For this, theoretical basis was formed; requirements were defined; a prototype was planned and created; the work done was analysed and future improvements were listed. Several possible solutions were proposed to save miscellaneous data, define the visual style and expand functionality. A solution was chosen and then implemented into a prototype to verify the validity.

One of the most important conclusions is that there is currently no suitable scripting language for use in such a context. Existing languages do not provide enough security if the creation was allowed by end users and execution were to be automated. There isn't a way to provide a high range of possibilities for the end-user to expand functionality without compromising safety.

Such a scripting language should be created. No non-functional requirements were arrived at within this thesis, apart from the wish for it to be simple and user friendly. For functional requirements, further research with potential end-users must be done.

The prototype was evaluated based on tests and accordance with the defined requirements. Further ideas for development – both improving and expanding – were provided based on the theoretical foundation.

12. Kasutatud kirjandus

[1] G. Fischer ja T. Herrmann, „Meta-design: Transforming and Enriching the Design and Use of Socio-technical Systems,“ 2014.

[2] G. Fischer ja E. Giaccardi, „Meta-Design: A Framework for the Future of End-User Development,“ 2006.

[3] J. Nielsen, „The Distribution of Users' Computer Skills: Worse Than You Think,“ 13 november 2016. [Võrgumaterjal]. Available: https://www.nngroup.com/articles/computer-skill-levels/.

[4] „Nodal - API Services Made Easy With Node.js,“ [Võrgumaterjal]. Available: http://www.nodaljs.com/.

[5] „ECMAScript 6 compatibility table,“ [Võrgumaterjal]. Available: https://kangax.github.io/compat-table/es6/.

[6] „ECMAScript 2015 (ES6) | Node.js,“ [Võrgumaterjal]. Available: https://nodejs.org/en/docs/es6/.

[7] „Learn Nodal.js,“ [Võrgumaterjal]. Available: https://github.com/keithwhor/nodal/blob/master/LEARN.md.

[8] „AngularJS - Superheroic JavaScript MVW Framework,“ [Võrgumaterjal]. Available: https://angularjs.org/.

[9] „AngularJS: Developers Guide: Data Binding,“ [Võrgumaterjal]. Available: https://docs.angularjs.org/guide/databinding.

[10] „Bootstrap,“ [Võrgumaterjal]. Available: http://getbootstrap.com/.

[11] „Angular directives for Bootstrap,“ [Võrgumaterjal]. Available: https://angular-

ui.github.io/bootstrap/.

[12] „Babel - The compiler for writing next generation JavaScript,“ [Võrgumaterjal]. Available: https://babeljs.io/.

[13] D. Bates, A. Barth ja C. Jackson, „Regular Expressions Considered Harmful in Client-side XSS Filters,“ *Proceedings of the 19th International Conference on World Wide Web*, 2010.

[14] „JSON Schema,“ [Võrgumaterjal]. Available: http://json-schema.org/.

[15] „XSS (Cross Site Scripting) Prevention Cheat Sheet,“ [Võrgumaterjal]. Available: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.

[16] „BBCode.org, bbcode users guide and tricks on implementing it,“ [Võrgumaterjal]. Available: https://www.bbcode.org/.

[17] „OAuth 2.0,“ [Võrgumaterjal]. Available: https://oauth.net.

[18] „SoapUI,“ [Võrgumaterjal]. Available: https://www.soapui.org/.

[19] V. Dwivedi, „End User Architecting,“ *End-User Development*, 2013.

13. Lisa 1: API dokumentatsioon

Põhiprogramm annab pärijale staatilisi faile (kasutajaliidese kuvamiseks) ning omab arvukaid REST teenuseid. Need teenused on dokumenteeritud antud peatükis. Antakse eeldatavad sisendid ning väljundid HTTP meetodite kaupa. Aadressid viitavad kõik spetsiifilisele klassile ning sellel aadressil on operatsioonid seotud selle klassiga. Näiteks: aadress „rest/folder“ annab võimaluse teha operatsioone kaustadega; „DELETE“ meetod sellel aadressil kustutab määratud kausta.

Tagastatakse JSON vastus, kus on alati „meta“ ja „data“ objektid. Kui viga pole, siis „error“ on väärtusega „null“ – vea puhul lisatakse veateade. „Data“ on alati massiivi tüüpi. Kui tagastatakse üks objekt, siis see on „data“ massiivi esimene väärtus.

```
{
  "meta": {
    "total": 0,
    "count": 0,
    "offset": 0,
    "error": {
      "message": "Not Implemented"
    }
  },
  "data": []
}
```

Ligipääsuloa (inglise keeles „access_token“) võib mitmele päringule kaasa anda nii päringu parameetrina kui ka edastatavate andmete seas.

13.1 User - /rest/user/{id}

MEETOD	PÄRING	NÄIDE

GET	Sisend: {id}; ligipääsuluba (peab kuuluma kasutajale, mida päritakse)
	
	Väljund: Kasutaja objekt, mille määras {id}
		Vastus:
		
		`{`
		`  "meta": {`
		`  "total": 1,`
		`  "count": 1,`
		`  "offset": 0,`
		`  "error": null`
		`  },`
		`  "data": [`
		`  {`
		`  "id": 1,`
		`  "email": "spunkyapple@gmail.com",`
		`  "username": "Zapple",`
		`  "created_at": "2016-10-31T13:04:44.120Z",`
		`  "updated_at": "2016-10-31T13:04:44.120Z",`
		`  "home_folder": 3`
		`  }`
		`  ]`
		`}`

POST	Sisend: loodava kasutaja objekt, kus kasutajanimi, email ja parool
	
	Väljund: loodud kasutaja, ligipääsuluba
		Päringu keha:
		
		`{`
		`  "username":"new_user_123",`
		`  "email":"emailaddress@mail.com",`
		`  "password":"Secr3tPa55word"`
		`}`
		Vastus:
		
		`{`
		`  "meta": {`
		`  "total": 1,`
		`  "count": 1,`
		`  "offset": 0,`
		`  "error": null`
		`  },`
		`  "data": [`
		`  {`
		`  "user": {`
		`  "id": 83,`
		`  "email": "emailaddress@mail.com",`
		`  "username": "new_user_123",`
		`  "created_at": "2016-12-19T13:37:05.350Z",`
		`  "updated_at": "2016-12-19T13:37:05.531Z",`

MEETOD	PÄRING	NÄIDE

		<pre>"home_folder": 82
		},
		"access_token": {
		"id": 85,
		"user_id": 83,
		"access_token":
		"60a2d1a1c84560593cf555554de663eb",
		"token_type": "bearer",
		"expires_at": "2017-01-
		18T13:37:05.611Z",
		"ip_address": null,
		"created_at": "2016-12-
		19T13:37:05.611Z",
		"updated_at": "2016-12-
		19T13:37:05.611Z"
		}
		}
		]
		}</pre>

PUT	Sisend: {id} konto kohta, mille andmeid uuendatakse, ligipääsuluba (käib muudetava konto kohta)
	
	Väljund: uuendatud konto objekt
		Päringu keha:
		
		<pre>{
		"username":"new_user_123",
		"email":"emailchanged@mail.com",
		"password":"Secr3tPa55wordChanged",
		"access_token":"60a2d1a1c84560593cf5555
		54de663eb"
		}</pre>
		Vastus:
		
		<pre>{
		"meta": {
		"total": 1,
		"count": 1,
		"offset": 0,
		"error": null
		},
		"data": [
		{
		"id": 83,
		"email": "emailchanged@mail.com",
		"username": "new_user_123",
		"created_at": "2016-12-
		19T13:37:05.350Z",
		"updated_at": "2016-12-
		19T13:41:21.040Z",
		"home_folder": 82
		}
		]
		}</pre>

MEETOD	PÄRING	NÄIDE

DELETE	Sisend: {id} konto kohta, mis soovitakse kustutada;
	ligipääsuluba (käib kustutava konto kohta)
	
	Väljund: kustutatud konto
		Vastus:
		
		{
		"meta": {
		"total": 1,
		"count": 1,
		"offset": 0,
		"error": null
		},
		"data": [
		{
		"id": 83,
		"email": "emailchanged@mail.com",
		"username": "new_user_123",
		"created_at": "2016-12-19T13:37:05.350Z",
		"updated_at": "2016-12-19T13:41:21.040Z",
		"home_folder": 82
		}
		]
		}

13.2 Access Token – /rest/access_token

MEETOD	PÄRING	NÄIDE

POST	Sisend: sisselogimisinfo (kasutajanimi ja parool); OAuth jaoks vajalik „grant type“
	
	Väljund: loodud ligipääsuluba
		Päringu keha:
		
		{
		"username":"new_user_123",
		"password":"Secr3tPa55wordChanged",
		"grant_type":" password",
		
		}
		Vastus:
		
		{
		"meta": {
		"total": 1,
		"count": 1,
		"offset": 0,
		"error": null
		},
		"data": [
		{
		"id": 87,
		"user_id": 84,
		"access_token": "6c05f77d4d7c4d04dce7808cd34b387c",
		"token_type": "bearer",
		"expires_at": "2017-01-18T13:48:52.660Z",
		"ip_address": null,
		"created_at": "2016-12-19T13:48:52.660Z",
		"updated_at": "2016-12-19T13:48:52.660Z"
		}
		]
		}

DELETE	Sisend: ligipääsuluba, mis soovitakse tühistada
	
	Väljund: kustutatud ligipääsuluba
		Päringu keha:
		
		{
		"access_token":"6c05f77d4d7c4d04dce7808cd34b387c"
		}
		
		Vastus:
		
		{
		"meta": {
		"total": 1,
		"count": 1,
		"offset": 0,
		"error": null

MEETOD	PÄRING	NÄIDE

		},
		"data": [
		{
		"id": 87,
		"user_id": 84,
		"access_token":
		"6c05f77d4d7c4d04dce7808cd34b387c",
		"token_type": "bearer",
		"expires_at": "2017-01-
		18T13:48:52.660Z",
		"ip_address": null,
		"created_at": "2016-12-
		19T13:48:52.660Z",
		"updated_at": "2016-12-
		19T13:48:52.660Z"
		}
		]
		}

13.3 Folder – /rest/folder/{id}

MEETOD	PÄRING	NÄIDE

GET	Sisend: võib parameetrites määrata „unique_id“, mis puhul kuvatakse kaust selle unikaalse ID-ga, või {id}, et pärida spetsiifilise kausta info. Teine võimalus on pärida parameetriga „parent“ (väärtuseks kausta ID number), mis puhul kuvatakse selles vanemas asuvad alamkaustad
	
	Väljund: Kui määrati „unique_id“ või {id}, tagastatakse vaid see kaust. Muul puhul tagastatakse kaustade massiiv, mida pärija (vastavalt ligipääsu loa olemasolule) võib näha. Peidetud kaustu ei kuvata, kui pärija pole omanik.
		Vastus:
		
		{
		"meta": {
		"total": 3,
		"count": 3,
		"offset": 0,
		"error": null
		},
		"data": [
		{
		"id": 85,
		"name": "subfolder1",
		"listed": true,
		"unique_id": null,
		"user_id": 1,
		"default_styles": null,
		"user": {
		"id": 1,
		"username": "Zapple"
		}
		},
		{
		"id": 86,
		"name": "subfolder2",
		"listed": true,
		"unique_id": null,
		"user_id": 1,
		"default_styles": null,
		"user": {
		"id": 1,
		"username": "Zapple"
		}
		}
		]
		}

POST	Sisend: loodava kausta andmed (vanem ehk „parent“, nimi ehk „name“, võib sisaldada: „listed“, „default_styles“, „unique_id“); ligipääsuluba (peab käima vanemkausta	Päringu keha:
		
		{
		"access_token":"162b6f3e945d0d96092a7e7a21c0e7d2",
		"parent":84,
		"name":"New folder"
		}
		Vastus:
		
		{
		"meta": {
		"total": 1,

MEETOD	PÄRING	NÄIDE

	omaniku kohta)
	
	Väljund: loodud kaust
		<pre>"count": 1,
		"offset": 0,
		"error": null
		},
		"data": [
		{
		"id": 88,
		"name": "New folder",
		"listed": true,
		"unique_id": null,
		"user_id": 84,
		"parent": 83,
		"default_styles": null
		}
		]
		}</pre>

PUT	Sisend: {id}, uus muudetava kausta info (sama, mis loomisel POST meetodiga); ligipääsuluba (peab kuuluma kausta omanikule)
	
	Väljund: uuendatud konto objekt
		Päringu keha:
		
		<pre>{
		"access_token":"162b6f3e945d0d96092a7e7a21c0e7d2",
		"name":"Changed folder name",
		"listed":false
		}</pre>
		Vastus:
		
		<pre>{
		"meta": {
		"total": 1,
		"count": 1,
		"offset": 0,
		"error": null
		},
		"data": [
		{
		"id": 88,
		"name": "Changed folder name",
		"listed": false,
		"unique_id": null,
		"user_id": 84,
		"parent": 83,
		"default_styles": null
		}
		]
		}</pre>

DELETE	Sisend: {id} kausta kohta, mida soovitakse kustutada;	Vastus:
		
		<pre>{
		"meta": {</pre>

MEETOD	PÄRING	NÄIDE

	ligipääsuluba (kuulub kustutatava kausta omanikule)
	
	Väljund: kustutatud kaust
		<pre>"total": 1,
		"count": 1,
		"offset": 0,
		"error": null
		},
		"data": [
		{
		"id": 88,
		"name": "Changed folder name",
		"listed": false,
		"unique_id": null,
		"user_id": 84,
		"parent": 83,
		"default_styles": null
		}
		]
		}</pre>

13.4 Entry – /rest/entry/{id}

MEETOD	PÄRING	NÄIDE

POST	Sisend: loodava sissekande objekti info („folder_id“, „type_id“, „data“); ligipääsuluba, mis kuulub kasutaja omanikule
	
	Väljund: loodud sissekanne („entry“) ja sissekannet ning kausta siduv objekt („entry_in_folder“)
		Päringu keha:
		
		`{ "access_token":"162b6f3e945d0d96092a7e7a21c0e7d2", "folder_id":89, "type_id":2, "data":{"text":"this text will be saved as the entry data"} }`
		
		Vastus:
		
		`{ "meta": { "total": 1, "count": 1, "offset": 0, "error": null }, "data": [{ "entry": { "id": 33, "data": { "text": "this text will be saved as the entry data" }, "user_id": 84, "type_id": 2, "created_at": "2016-12-19T14:34:38.790Z", "updated_at": "2016-12-19T14:34:38.790Z" }, "entry_in_folder": { "id": 63, "folder_id": 89, "entry_id": 33, "created at": "2016-12-19T14:34:38.794Z", "updated_at": "2016-12-19T14:34:38.794Z" } }] }`

PUT	Sisend: muudetava sissekande {id};	Päringu keha: `{`

MEETOD	PÄRING	NÄIDE

	ligipääsuluba (kuulub sissekande omanikule)
	
	Väljund: uuendatud sissekande objekt
		"access_token":"162b6f3e945d0d96092a7e7a21c0e7d2",
		"data":{"text":"this text will be saved as the entry data"}
		}
		
		Vastus:
		
		{
		"meta": {
		"total": 1,
		"count": 1,
		"offset": 0,
		"error": null
		},
		"data": [
		{
		"id": 33,
		"data": {
		"text": "this text has been changed"
		},
		"created_at": "2016-12-19T14:34:38.790Z",
		"updated_at": "2016-12-19T14:36:59.789Z"
		}
		]
		}

13.5 Entry Type – /rest/entry_type

MEETOD	PÄRING	NÄIDE

GET	Sisend: -
	
	Väljund: masiiv olemasolevatest sissekande tüüpidest
		Vastus:
		
		`{`
		`  "meta": {`
		`  "total": 14,`
		`  "count": 14,`
		`  "offset": 0,`
		`  "error": null`
		`  },`
		`  "data": [`
		`  {`
		`  "id": 1,`
		`  "name": "Simple link",`
		`  "structure": [`
		`  {`
		`  "name": "url",`
		`  "type": "string"`
		`  },`
		`  {`
		`  "name": "url_name",`
		`  "type": "string"`
		`  }`
		`  ]`
		`  },`
		`  {`
		`  "id": 2,`
		`  "name": "Text",`
		`  "structure": [`
		`  {`
		`  "name": "text",`
		`  "type": "string"`
		`  }`
		`  ]`
		`  },`
		`  ...`
		`  ]`
		`}`

POST	Sisend: ligipääsuluba; loodava tüübi andmed: nimi ning struktuur
	
	Väljund: loodud sissekande tüüp
		Päringu keha:
		
		`{`
		`  "access_token":"162b6f3e945d0d96092a7e7a21c0e7d2",`
		`  "name":"A new text type",`
		`  "structure":[{"name":"title",`
		`  "type":"string"},`
		`  {"name":"text","type":"string"}]`
		`}`
		
		Vastus:

MEETOD	PÄRING	NÄIDE

		{
		"meta": {
		"total": 1,
		"count": 1,
		"offset": 0,
		"error": null
		},
		"data": [
		{
		"id": 15,
		"name": "A new text type",
		"structure": [
		{
		"name": "title",
		"type": "string"
		},
		{
		"name": "text",
		"type": "string"
		}
		],
		"created_at": "2016-12-19T14:44:18.335Z",
		"updated_at": "2016-12-19T14:44:18.335Z"
		}
		]
		}

13.6 Visual Style – /rest/visual_style

MEETOD	PÄRING	NÄIDE

GET	Sisend: parameetrites võib määrata „entry_type_id“, et määrata, missuguse tüübi visuaalseid stiile pärida
	
	Väljund: masiiv olemasolevatest sissekande tüüpidest
		Vastus:
		
		`{`
		`  "meta": {`
		`  "total": 6,`
		`  "count": 6,`
		`  "offset": 0,`
		`  "error": null`
		`  },`
		`  "data": [`
		`  {`
		`  "id": 2,`
		`  "name": "Simple url",`
		`  "template": "[url=$url$]$url_name$[/url]",`
		`  "user_id": 1,`
		`  "entry_type_id": 1,`
		`  "created_at": "2016-10-31T13:10:57.763Z",`
		`  "updated_at": "2016-10-31T13:10:57.763Z"`
		`  },`
		`  ...`
		`  ]`
		`}`

POST	Sisend: ligipääsuluba; loodava visuaalse stiili andmed: nimi, mall („template“), sissekande tüübi id
	
	Väljund: loodud visuaalne stiil
		Päringu keha:
		
		`{`
		`  "access_token":"162b6f3e945d0d96092a7e7a21c0e7d2",`
		`  "name":"A new visual style",`
		`  "entry_type_id":15`
		`  "structure":"[div][b]$title$[/b][/div][div]$text$[/div]"`
		`}`
		
		Vastus:
		
		`{`
		`  "meta": {`
		`  "total": 1,`
		`  "count": 1,`
		`  "offset": 0,`
		`  "error": null`
		`  },`
		`  "data": [`
		`  {`
		`  "id": 7,`
		`  "name": "A new visual style",`
		`  "template": "[div][b]$title$[/b][/div]`

MEETOD	PÄRING	NÄIDE

		<pre>[div]$text$[/div]",
		      "user_id": 84,
		      "entry_type_id": 15,
		      "created_at": "2016-12-
		19T14:54:47.316Z",
		      "updated_at": "2016-12-
		19T14:54:47.318Z"
		    }
		  ]
		}</pre>

13.7 Entry in Folder – /rest/entry_in_folder

MEETOD	PÄRING	NÄIDE

GET	Sisend: ligipääsuluba (peab olema kasutaja oma, mis omab kausta, kui kaust on peidetud); „folder_id“ määrab kausta, kust sissekandeid kuvatakse
	
	Väljund: massiiv sissekannetest („entry_in_folder“ tüüpi objektid, kuid sisaldavad „entry“ objekte)
		Vastus:
		
		{
		"meta": {
		"total": 1,
		"count": 1,
		"offset": 0,
		"error": null
		},
		"data": [
		{
		"id": 63,
		"folder_id": 89,
		"created_at": "2016-12-19T14:34:38.794Z",
		"updated_at": "2016-12-19T14:34:38.794Z",
		"entry": {
		"id": 33,
		"data": {
		"text": "this text has been changed"
		},
		"created_at": "2016-12-19T14:34:38.790Z",
		"updated_at": "2016-12-19T14:36:59.789Z",
		"type_id": 2,
		"entryType": {
		"id": 2,
		"name": "Text"
		},
		"user": {
		"id": 84,
		"username": "new_user_123"
		}
		}
		}
		]
		}

POST	Eesmärk on kopeerida sissekannet teise kausta ehk luua uus seos sissekande ja kausta vahel.
	
	Sisend: „folder_id“, mis viitab sihtkaustale;
		Päringu keha:
		
		{
		"access_token":"162b6f3e945d0d96092a7e7a21c0e7d2",
		"folder_id":90,
		"entry_id":33
		}
		Vastus:

MEETOD	PÄRING	NÄIDE

	„entry_id“ sissekandelt, mille viide uude kausta tuleb; ligipääsuluba, mis kuulub sihtkausta omanikule
	
	Väljund: loodud kasutaja, ligipääsuluba
		<pre>{
		  "meta": {
		  "total": 1,
		  "count": 1,
		  "offset": 0,
		  "error": null
		  },
		  "data": [
		  {
		  "id": 64,
		  "folder_id": 90,
		  "entry_id": 33,
		  "created_at": "2016-12-19T15:17:08.344Z",
		  "updated_at": "2016-12-19T15:17:08.344Z"
		  }
		  ]
		}</pre>

PUT	Eesmärk on tõsta sissekanne ühest kaustast teise. See tähendab olemasoleva kausta ja sissekande vahelise seose muutmist.
	
	Sisend: {id} sissekanne-kaustas objekti kohta, mida muudetakse; „folder_id“; ligipääsuluba (kuulub kasutajale, kes on lähte- ning sihtkausta omanik)
	
	Väljund: uuendatud konto objekt
		Päringu keha:
		
		<pre>{
		  "access_token":"162b6f3e945d0d96092a7e7a21c0e7d2",
		  "folder_id":89
		}</pre>Vastus:
		
		<pre>{
		  "meta": {
		  "total": 1,
		  "count": 1,
		  "offset": 0,
		  "error": null
		  },
		  "data": [
		  {
		  "id": 64,
		  "folder_id": 89,
		  "entry_id": 33,
		  "created_at": "2016-12-19T15:17:08.344Z",
		  "updated_at": "2016-12-19T15:33:17.738Z"
		  }
		  ]
		}</pre>

MEETOD	PÄRING	NÄIDE

DELETE	Sisend: {id}
	sissekanne-kaustas objekti oma, mida kustutatakse; ligipääsuluba (kuulub kasutajale, kes omab kausta, kus paikneb kustutatav objekt)
	
	Väljund: kustutatud sissekanne-kaustas objekt
		Vastus:
		
		{
		"meta": {
		"total": 1,
		"count": 1,
		"offset": 0,
		"error": null
		},
		"data": [
		{
		"id": 64,
		"folder_id": 89,
		"entry_id": 33,
		"created_at": "2016-12-19T15:17:08.344Z",
		"updated_at": "2016-12-19T15:33:17.738Z"
		}
		]
		}

14. Lisa 2: Prototüübi veebimaterjalid

Prototüübiga tutvumise aadress:

http://dinkory.com/

Prototüübi Git keskkond (ligipääs lähtekoodile)

https://gitgud.io/zapple/dinkory