Risto Auvinen

Verkkosovelluksen jatkokehitys

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

11.11.2016

Metropolia

Tekijä Otsikko Sivumäärä Aika	Risto Auvinen Verkkosovelluksen jatkokehitys 48 sivua 11.11.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Ilpo Kuivanen Lehtori Jussi Alhorinne

Tässä opinnäytetyössä tutustutaan erään yrityksen verkkosovellukseen ja esitellään sovellukseen tehtyjä muutoksia ja uusia ominaisuuksia. Sovellus on erityisesti vanhemmalle väestölle suunnattu muistojen tallennus- ja jakamispalvelu. Sovellus vaati jatkokehitystä, koska siitä löytyi suuri määrä ohjelmointivirheitä ja puuttuvaa toiminnallisuutta.

Opinnäytetyön alussa esitellään itse sovellus, sen rakenne sekä siinä käytetyt teknologiat. Tämän jälkeen siirrytään esittelemään ohjelman jatkokehityksen aikana tehtyjä muutoksia ja uusia ominaisuuksia. Jatkokehityksen ansiosta käyttöliittymästä löytyneet ongelmat ja niiden ratkaisuyritykset esitetään. Opinnäytetyö päättyy ohjelman vanhan ja uuden version tietokantojen yhdistämiseen.

Jatkokehityksen aikana ohjelmasta saatiin poistettua suurin osa ohjelmointivirheistä ja lisättyä vaadittuja ominaisuuksia. Ohjelman jatkokehitys tulee jatkumaan tämän opinnäytetyön kirjoittamisen jälkeen.

Avainsanat	Java, Spring, JavaScript, HTML5, Red5, Hibernate, MySQL

Author Title Number of Pages Date	Risto Auvinen Further Development of Web Application 48 pages 11 November 2016
Degree	Bachelor of Engineering
Degree Programme	Information technology
Specialisation option	Software engineering
Instructor(s)	Ilpo Kuivanen, Senior Lecturer Jussi Alhorinne, Senior Lecturer

In this thesis, a company's web application and the changes and new features developed for it are explored. The application is a service for the saving and sharing of memories especially aimed towards older people. The application requires further development, because it contained a sizable amount of software bugs and missing functionality.

The thesis begins with an explanation of the application itself, its structure and the technologies used in it. After this, the changes and new features made during the development are explained. User interface problems found during development and the attempts to fix them are shown. The thesis ends with the combination of the databases of the old and new versions of the program.

During the development, most software bugs were removed and requested features added. The software's development will continue after the writing of this thesis.

Japanese translation:
翻訳
この論文にはとある会社のウェブアプリケーションの説明と開発した変化や新しい特質が書かれます。アプリケーションはご老人に向かって思い出を保存と共用するサービスです。アプリケーションにいったソフトウェアバグと不足していた機能が多いので、なお一層の開発が必要になりました。

論文の開始にはアプリケーションの説明と構造と使用する技術が呈されます。その後、開発の時に作られた変化や新しい特質が説明されます。ユーザーインターフェースに見つかった問題と試した解決が見せられます。論文の終りには古いと新しいバージョンのデータベースの結合が説明されます。

開発の時に大分のソフトウェアバグが直されましたと必要な特質が加えられました。論文を書いた後、ソフトウェアの開発が続けています。

Keywords	Java, Spring, JavaScript, HTML5, Red5, Hibernate, MySQL

Sisällys

Lyhenteet

SaaS	Software as a Service. Tarkoittaa ohjelmiston hankkimista palveluna perinteisen lisenssipohjaisen tavan sijasta.
Spring MVC	Java-pohjainen sovelluskehys, joka on ohjelmassa käytössä.
Red5	Java-pohjainen avoimen lähdekoodin mediapalvelin.
BLOB	Binary Large Object. Kokoelma binääridataa tallennettuna yhtenä entiteettinä tietokantaan. Data voi sisältää kuvan, ääntä tai multimediaobjektin.
HTML5	HTML (Hypertext Markup Language) -merkintäkielen uusi versio.
Tagi	Erityisesti käyttöliittymäpuolen ohjelmistokielissä käytettävä merkintä, joka kuvaa koodielementin aloitusta sekä lopetusta. Toinen merkitys tagille on indeksoinnissa käytettävä avainsana.
Google Maps API	Mahdollistaa karttojen liittämisen verkkosivulle.
HDFVR	Mahdollistaa videon ja äänen tallentamisen nettisivun kautta.
Front-end	Front-en-kielillä (HTML, CSS, JavaScript) kehitetään käyttöliittymää.
Back-end	Back-end-kielillä (tässä sovelluksessa Java) luodaan yhteys tietokantaan.
SMTP	Simple Mail Transfer Protocol. Protokolla, jota käytetään sähköpostiviestien välittämiseen.
URI	Uniform Resource Identifier. Merkkijono, jolla pystytään osoittamaan johonkin informaatioon tai resurssiin.

FFmpeg	Kokoelma vapaita ohjelmia, joilla voidaan muuttaa tiedoston tyyppiä.
SSL	Secure Sockets Layer. Salausprotokolla, jolla voidaan suojata verkkosivun tietoliikenne.
C#	Microsoftin kehittämä ohjelmointikieli.
GET/POST	HTTP-protokollan määrittelemät kutsumetodit internetissä.
PDF	Portable Document Format. Ohjelmistoriippumaton dokumenttityyppi.
XSL	Microsoft Excelin tiedostotyyppi taulukkomuotoisille dokumenteille.
JSON	JavaScript Object Notation. Yksinkertainen tiedostomuoto tiedon varastointiin.

1 Johdanto

Tässä insinöörityössä tutustun erään yrityksen teettämään verkkosovellukseen sekä sen jatkokehitykseen. Yritys on pieni ja startup-vaiheessa, joten pääsin olemaan olennainen osa sovelluksen jatkokehitystä. Pääsin tutustumaan yritykseen ja sovellukseen sekä suoritin ohjelman jatkokehitystä viiden kuukauden harjoittelujakson ajan.

Sovellus on SaaS-pohjainen, verkkoselaimessa internetin välityksellä käytettävä ohjelma. Sovellus on luotu Spring MVC -sovelluskehystä apuna käyttäen, ja siinä käytetään palvelimena avoimen lähdekoodin Red5-palvelinta. Sovelluksesta oli jo vuosia sitten olemassa vanha versio, joka oli kuitenkin jo sen verran vanhanaikainen, että lähdettiin kehittämään uutta. Uusi versio luotiin alihankintana Intiassa. Tämän jälkeen sovellus tuotiin takaisin Suomeen jatkokehitykseen enimmäkseen työharjoittelijavoimin. Sekä yrityksen että sovelluksen nimet on pyydetty pitämään salassa, joten niitä ei mainita työssä. Ohjelman käyttöliittymää kuvaavat kuvat ovat tästä syystä luotu kuvankäsittelyohjelmassa.

Tulin alunperin yritykseen suorittamaan opintosuunnitelman mukaisia työharjoittelujaksoja, mutta sain myös samalla työstäni sovelluksen parissa insinöörityön aiheen. Tullessani yritykseen intialainen yhtiö, joka oli luonut pohjalla olevaa koodia, oli saanut sopimuksensa päätökseen. Tämä tarkoitti, että sovellus oli pääominaisuuksiltaan valmis. Sovellus vaati kuitenkin kehitystä, koska järjestelmästä löytyi virheitä, kieliasu oli monissa kohdin harhaanjohtava tai suorastaan väärä ja ohjelmaan haluttiin uusia ominaisuuksia. Startup-vaiheessa olevalla yrityksellä ei kuitenkaan ollut paljon varoja käytössä, joten työhön palkattiin työharjoittelijoita.

Pääsin tekemään muutoksia suoraan sovelluksen lähdekoodiin ja sain itse viedä suurimman osan muutoksistani kaupalliselle palvelimelle. Suurin osa muutoksistani koostui järjestelmävirheiden korjauksista ja kieliasun muutoksista, mutta olin myös kehittämässä uusia ominaisuuksia.

2 Sovellus

Yhtiön näkemys on, että nykypäivän internetpalveluissa ja sovelluksissa on aukko vanhempien sukupolvien kohdalla. Heille ei tarjota tällä hetkellä tarpeeksi mielekkäitä palveluita, ja kysyntä tällaisille palveluille tulee vain kasvamaan, koska tietokoneiden käyttöä ymmärtävät sukupolvet ovat tulossa eläkeikään. Sovellus onkin luotu erityisesti vanhempien kansalaisten käyttöön geriatrisen hyötynäkökulman vuoksi. Sovelluksesta on myös hyötyä vanhempien ihmisten muistojen tallettamisesta heidän jälkipolviensa käyttöön.

2.1 Esittely

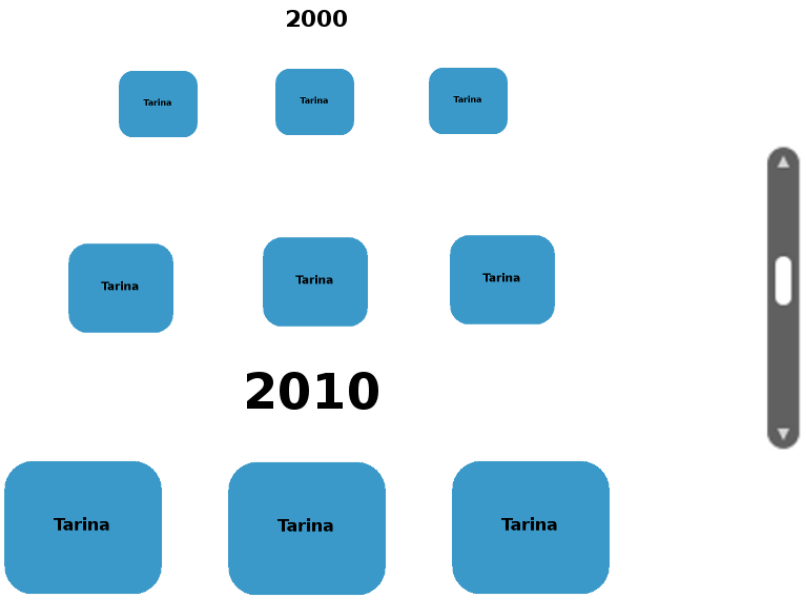
Ohjelma on muistojen tallennuspalvelu. Sen tärkein toiminto on pystyä tallentamaan henkilön muistoja teksti-, ääni-, kuva- sekä videomuodoissa. Tallennettaessa tarinaa se sijoittuu käyttäjän valitsemaan paikkaan sekä aikajanalla että kartalla vaihtoehtoisessa näkymässä.

Tarinaa tehtäessä käyttäjän tulee ensin valita tarinan tyyppi tekstin, äänen, kuvan tai videon väliltä. Tekstitarina on yksinkertaisin tarinatyyppi. Siinä tallennetaan vain käyttäjän syöttämä teksti tietokantaan. Kuvatarinassa kuva tallennetaan verkkosovelluksen hakemistoon tietokannan sijaan ja tietokantaan tallennetaan pelkästään viite sen sijainnista levyllä. Samaa periaatetta käytetään ääni- ja video-tarinatyyppien tiedostojen tallentamisessa. Suurin syy tälle BLOB-muotoisen tietokantatallennuksen sijaan on, että tietokannasta tulee huomattavasti kevyempi ja datasta helpompaa käsitellä.

Kuvatarinassa voi olla lisäksi vielä kuvateksti tai vaihtoehtoisesti äänitys. Kuvatarinan äänitykset, äänitarinat sekä videotarinat taltioidaan käyttäen sovelluksen sisältämää HDFVR-rajapintaa. Äänitarinan tapauksessa HDFVR:n luoma flv-muotoinen tiedosto muutetaan vielä mp3-muotoon käyttäen FFmpeg-työkalua. Samoin videotarina muutetaan mp4-muotoon ennen kuin se varsinaisesti tallennetaan. Videotarinalle on asetettu kahden minuutin rajoitus tilan säästämiseksi ja sovelluksessa muutenkin suositellaan ytimekkäiden tarinoiden tekemistä.

Sovelluksesta löytyy kolme eri näkyvyystasoa: julkiset tarinat, yksityiset tarinat ja yhteisötarinat. Julkisia tarinoita pystyvät tekemään rekisteröityneet käyttäjät rajoituksetta. Näitä tarinoita pystyy myös lukemaan kuka tahansa. Yksityiset tarinat ovat nimensä mukaisesti käyttäjän täysin omia tarinoita, joita voi nähdä vain käyttäjän omalta tililtä. Yksityisillä tarinoilla on aivan oma aikajanansa, joka on erillinen sekä julkisesta että yhteisöjen aikajanasta.

Uusi käyttäjä voi luoda yksityisiä tarinoita vain viisi. Kuitenkin ostamalla sovelluksen premium-palvelun, pääsee niitä tekemään käytännössä rajattomasti. Premium-palvelun ostaminen onkin yksi sovelluksen rahoitusstrategioista. Toinen niistä on yhteisöt. Yhteisöjä pystyy ostamaan, jonka jälkeen yhteisönvalvoja voi lisätä käyttäjiä haluamansa mukaan yhteisöön. Yhteisö on siis kuten julkinen aikajana, jossa vain tietyt henkilöt pystyvät kertomaan tarinoitaan. Yhteisöissä voi olla myös tiettyjä erikoistoimintoja, joita ei muissa näkyvyystasoissa ole. Esimerkiksi yhdessä yhteisössä oli ominaisuutena tehdä erikoistyyppisiä sotamuistotarinoita.



Kuva 1. Sovelluksen aikajananäkymä tullessani yritykseen

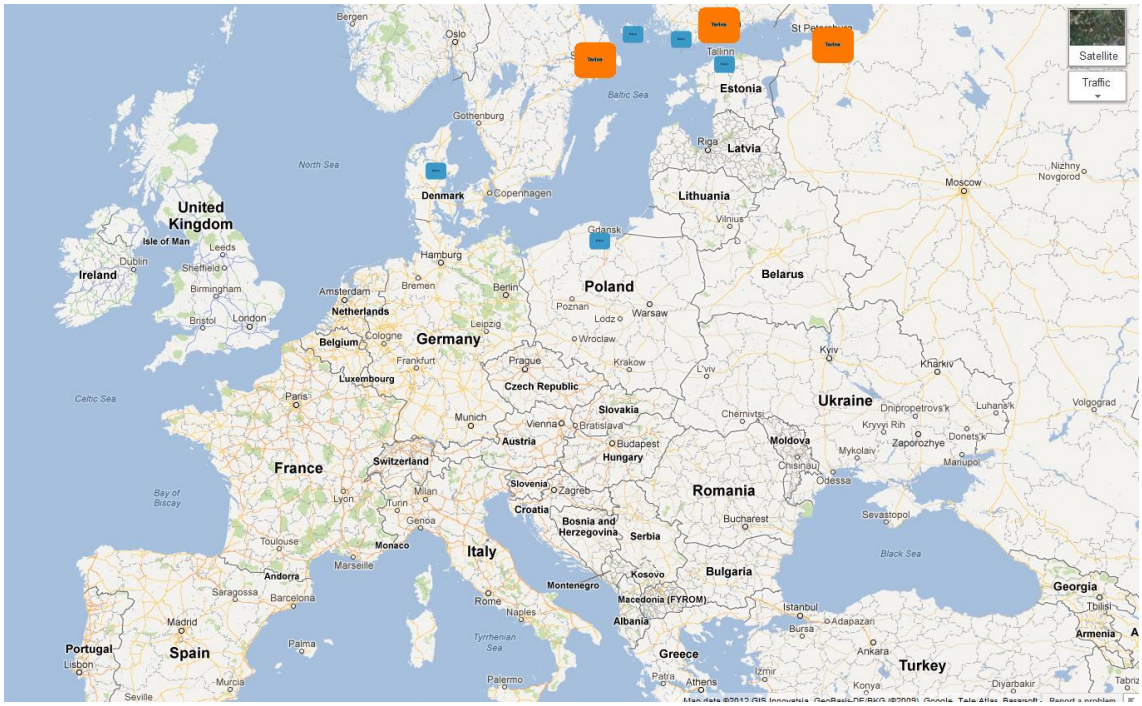
Kuvassa 1 on karkea versio ohjelman käyttöliittymän perustilasta, eli aikajananäkymästä. Aikajana kulkee z-akselilla käyttäjästä tietokonetta kohti, ja sitä voi selata käyttäen hiirtä tai oikealla näkyvää vierityspalkkia. Aikajanan selausnopeus riippuu vierityspalkin positiosta. Äärimmäisenä ylhäällä se liikkuu nopeimmin vuosia taaksepäin, alhaalla vuosia eteenpäin. Aikajanalla esiintyy vuosilukuja kymmenen vuoden välein, mikä tekee aikajanasta organisoidumman. Kuvassa siniset ”Tarina”-palkit ovat käyttäjien itse luomia tarinoita. Itse ohjelmassa tekstin ”Tarina” tilalla lukisi tarinan otsikko, ja tarinan tapahtumapäivämäärä olisi merkitty palkin yläosaan.

Tarinaa painettaessa hiirellä tulee esiin uusi näkymä, jossa näkyy tarinan sisältö jaettuna tarinasivuihin. Jokaisella tarinasivulla voi olla oma sisältönsä, eli esimerkiksi yhdellä sivulla voi olla kuvatarina, kun taas toisella sivulla voi olla videotarina. Video- ja äänitarinoiden esittämiseen sivulla käytetään HTML5:n sisäisiä <audio>- sekä <video>-tageja. Kuvatarinassa voi olla kuvan lisäksi myös kuvateksti, tai käyttäjän äänittämä ääniviesti. Ennen jatkokehityksen aloitusta tekstitarina koostui pelkästään käyttäjän kirjoittamasta tekstistä. Jatkokehityksen aikana asiakkaan pyynnöstä lisättiin kuitenkin ominaisuus, jossa tekstin sijasta voi tarinaan lisätä PDF-muotoisen tiedoston.

Aikajananäkymästä löytyy myös tarinoiden suodatustoiminto. Tarinoita pystyy suodattamaan kahdella tavalla sekä hakusanan mukaan että tarinatyypin/aikavälin mukaan. Tarinatyyppihaussa annetaan haluttu tarinan tyyppi sekä aikaväli, jolloin ohjelma tekee tietokantahaun, ja hakee esille pelkästään annetulla aikavälillä esiintyneet käyttäjän valitseman tarinatyypin tarinat. Hakusanahaussa käyttäjä antaa tekstin, jonka haluaa löytää, ja ohjelma palauttaa kaikki tarinat, joiden otsikoista tai tageista löytyy kyseinen tekstinpätkä. Hakusanahaussa ei haeta tarinoita uudestaan tietokannasta, vaan pelkästään poistetaan näkyvistä kaikki tarinat, jotka eivät vastaa hakua. Tämä on hyvä ratkaisu, koska se ajaa periaatteessa saman asian kuin tietokantahaku mutta poistaa tietokantahausta aiheutuvat turhat latausajat.

Aikajanalla on tällä hetkellä merkittäviä ongelmia, joita on pyritty parantamaan jatkokehityksen aikana. Tarinat kulkevat kolmessa linjassa, eivätkä ne leviä ollenkaan sivu- tai pystysuunnassa. Tarinarivien välissä on vakioetäisyys, joka erottaa rivit toisistaan. Tämä soveltuu hyvin pienen tarinamäärän selaukseen, mutta mitä enemmän tarinoita on, sitä kaummin vuosien selaus kestää. Samoin ohjelman käynnistys hidastuu, koska tarinat ovat ohjelmoitu ladattaviksi aikajanalle kaikki yhdellä kertaa. Tämä on yksi merkittävimmistä ongelmista, kun sovelluksesta ollaan tekemässä kaupallisempaa.

Toinen ongelma on tarinoiden erottaminen toisistaan. Tällä hetkellä kaikilla normaaleilla tarinoilla on sama ikoni, eikä niitä pysty erottamaan toisistaan muuten kuin ikonin edessä olevan tarinan otsikon mukaan. Tarina pitää avata ennen kuin siitä saa selville edes tarinan tyypin. Mielestäni tarinamäärän kasvaessa on tärkeää saada tarinasta irti jotain jo pelkällä vilkaisulla. Jos aikajanalla on satoja tarinoita ja kaikki näyttävät samoilta, niin itseään kiinnostavien tarinoiden löytyminen vaikeutuu.



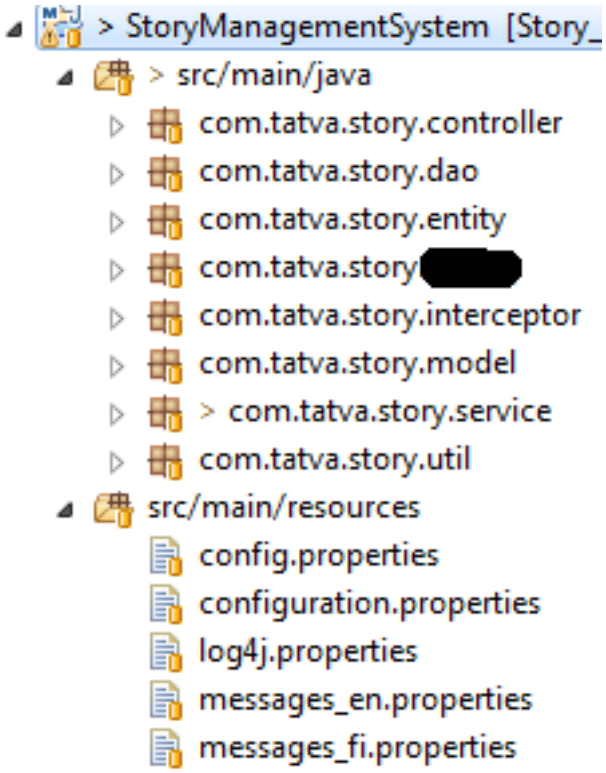
Kuva 2. Sovelluksen karttanäkymä

Kuvassa 2 näkyy vaihtoehtoinen karttanäkymä. Sovelluksessa käytetään kartan piirtämiseen Google Maps API -rajapintaa. Tarinaa tehtäessä annetaan samaisella Google Maps -kartalla tarinalle paikka, joka tallentuu tietokantaan leveys- ja pituussuuntaisina koordinaatteina. Karttanäkymään siirryttäessä ohjelma pudottaa google.maps.Marker-tyylisiä merkitsijöitä kartalle tietokannassa olevien koordinaattien mukaan. Jos tarinalle ei ole annettu koordinaattia, ei sitä myöskään pudoteta. Pidettäessä hiirtä tarinan yllä, tulee esiin tarinan otsikko ja painettaessa tarinaa tulee tarinan sisältö esiin samalla tavalla kuin aikajananäkymässä.

Kuvassa käyttämäni oranssit palkit kuvaavat kolmea käyttäjää lähinnä olevaa tarinaa mentäessä aikajanalta karttanäkymään. Itse sovelluksessa nämä esitetään hiukan eri tavoin, mutta kuitenkin selvästi erillisinä muista tarinoista. Muut tarinat esitetään pieninä sinisinä palkkeina. Ennen jatkokehityksen aloitusta tarinoita pystyi olemaan ruudulla vain kymmenen, ja ne putosivat kaikki samanaikaisesti.

2.2 Rakenne

Tässä luvussa selvitetään sovelluksen rakennetta ja perustoimintaperiaatetta.



Kuva 3. Sovelluksen back-end- sekä properties-tiedostoja

Kuvan alalaidan config.properties-tiedostosta löytyy ohjelman SMTP-protokollan tunnukset. Configuration.properties-tiedosto on alussa tyhjä, mutta ohjelma tallentaa siihen tietoja ohjelman käytön aikana järjestelmänvalvojan ohjelmaan tekemistä muutoksista. Muutoksia voivat olla esimerkiksi selausnopeuden kasvattaminen tai tarinoiden ikoneiden koon muuttaminen. log4j.properties on kolmannen osapuolen tiedosto ohjelmassa tapahtuvien virheiden ja muiden asioiden kirjaamiseen.

Kuvan kaksi viimeistä tiedostoa, messages_en.properties ja messages_fi.properties, sisältävät kaikki sovelluksesta löytyvät tekstinpätkät jaettuina tiedoston mukaan englanniksi ja suomeksi. Näiden tiedostojen ja sovelluksessa käytettävien Springin <spring:message> tagien ansiosta sovelluksesta pystytään tekemään monikielinen. Esimerkiksi koodinpätkä <spring:message code=”title”> pystytään samaan aikaan esittämään sovelluksessa sekä ”Title”, että ”Otsikko” -tekstinpätkinä, riippuen käyttäjän

kieliasetuksesta. Samalla periaatteella pystytään myöhemmin lisäämään muitakin kieliä sovellukseen, kun koetaan sille tarvetta.

Kuvan 3 com.tatva.story alkuiset paketit esittävät Java-luokkien kokonaisuuksia. Ensin näistä paketeista on tärkeä ymmärtää entity sekä model. Entity kuvaa tietokannassa olevaa taulua. Entity-luokan tietueet ovat Hibernate-annotaatioiden avulla kartoitettu osoittamaan tietokannassa olevaa vastaavaa tietuetta. Tämän ansiosta tietokantaan on helppo tehdä muutoksia muuttamalla entiteettiä ja tallentamalla sen. Entity-luokkia ei kuitenkaan ole helppo käsitellä tietokannan ja ”todellisuuden” erojen vuoksi, jonka vuoksi mallit ovat olemassa. Mallit ovat Spring MVC:n mukaisia Model-luokkia. Ne ovat kuin tavallisia Java-olioita, sisältävät suurin piirtein samat tiedot kuin Entity, mutta niissä ei ole sidoksia tietokantaan. Ne ovat käytössä sovelluksen View ja Controller osioissa, joissa niitä pystytään käsittelemään ja muokkaamaan. Muokkausten jälkeen Model pystytään kopioimaan sitä vastaavaksi Entityksi ja tekemään näin muutoksia tietokantaan.

Ensimmäinen paketti, nimeltään controller, sisältää Spring MVC:n mukaiset Controller-luokat eli käsittelijät. Käsittelijät ottavat vastaan käyttäjältä tulevat käskyt ja varmistavat että data on oikeanlaista ennen kuin lähettävät sen eteenpäin käsiteltäväksi Service-luokkiin. Jos data on vääränlaista tai puuttuvaa, käyttäjä joko uudelleenohjataan takaisin ohjelman perusnäkymään tai annetaan virheilmoitus. Jos data on oikeanlaista eikä datan käsittelyn aikana ole sattunut virheitä, palautetaan käyttäjälle uusi JSP-sivu, eli siirretään käyttäjää eteenpäin ohjelmassa. Tärkeimmille ohjelman kokonaisuuksille on omat käsittelijä-luokkansa, jotka käsittelevät vain niihin liittyviä asioita. Esimerkiksi UserController-luokka käsittelee kaikkea käyttäjään liittyvää rekisteröitymisestä salasanan vaihtoon.

Käsittelijä-luokista data siirretään Service- eli palveluluokkiin. Näissä luokissa käsitellään dataa, yleensä käyttäen apuna Util -paketin luokkia. Yleisin operaatio on kopioida Model-luokka Entityksi (ja toisinpäin) käyttäen Util-luokkaa ModelEntityMappingUtils.java. Tämä tehdään alla olevan kuvan mukaisesti käyttäen gettereitä ja settereitä.

```
/*
 * This method will convert Story model to entity
 */
public static Story getStoryEntity(com.tatva.story.model.Story model) {

    Story entity = new Story();

    entity.setId(model.getId());
    entity.setTitle(model.getTitle());
    entity.setCity(model.getCity());
    entity.setCountry(model.getCountry());
    entity.setStoryDate(model.getStoryDate());
    entity.setPublishing(model.getPublishing());
    entity.setReady(model.isReady());
    entity.setCreated(model.getCreated());
    entity.setLattitude(model.getLattitude());
    entity.setLongitude(model.getLongitude());
    entity.setYear(model.getYear());
    entity.setUser(getUserEntity(model.getUser()));
    entity.setStoryType(model.getStoryType());
```

Kuva 4. Modelin kopioiminen Entityksi

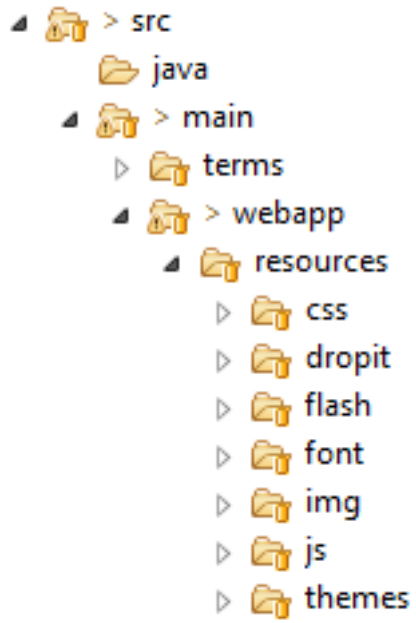
Kuvassa luodaan uusi Story-tyyppinen entiteetti asettamalla Model-luokan tietueet Entity-luokan vastaaviksi tietueiksi. Kopioitaessa mallin sisäinen kokonainen olio, kuten yllä olevan kuvan 4 Story-entiteetin sisältämä User, käydään samalla kopioimassa UserModel luokan tietuet UserEntity-luokan tietueiksi.

Data saapuu lopuksi Service-luokkien kautta DAO (Data Access Object) -luokkiin. Näissä luokissa suoritetaan itse tietokantaoperaatiot. Tietokannasta pystytään tekemään hakuja, lisäämään dataa, poistamaan dataa tai muokkaamaan sitä.

```
74      public List<Story> getStoriesByCommunity(Long communityId) {
75          List<Story> listStories = null;
76
77          try {
78              Session session = sessionFactory.getCurrentSession();
79              Criteria criteria = session.createCriteria(Story.class, "story");
80              criteria.createAlias("story.communities", "community");
81              criteria.addOrder(Order.desc("storyDate"));
82              criteria.add(Restrictions.eq("community.id", communityId));
83
84              listStories = criteria.list();
85          } catch (Exception e) {
86              e.printStackTrace();
87          }
88
89          return listStories;
90      }
```

Kuva 5. Tarinoiden haku tietokannasta

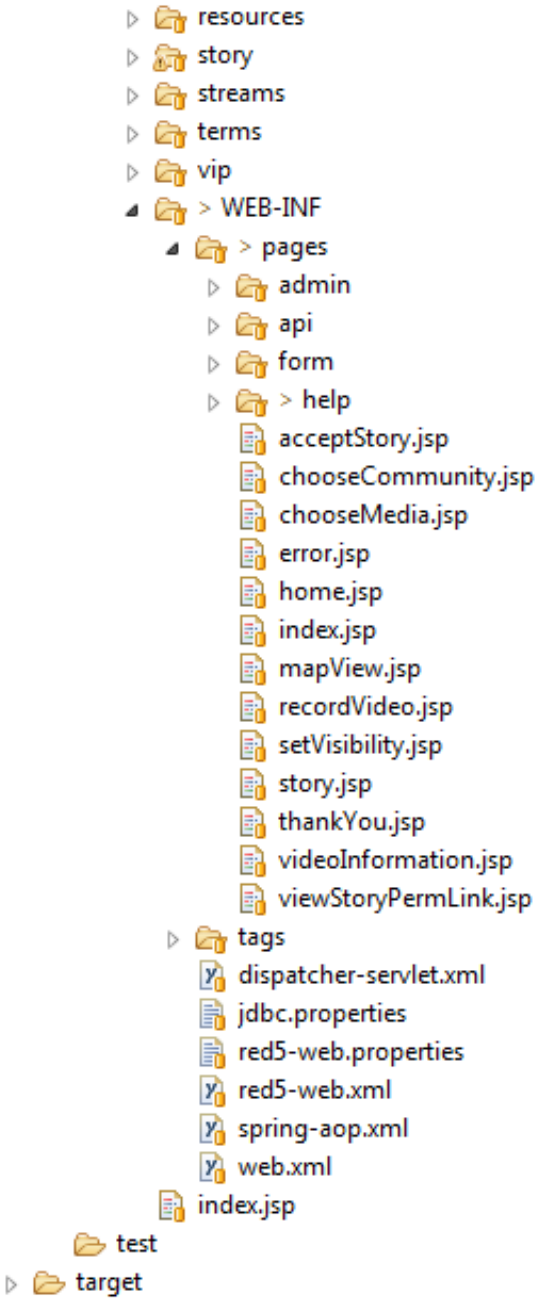
Kuvassa 5 on esimerkki StoryDao.java-tiedostosta löytyvästä funktiosta getStoriesByCommunity. Nimensä mukaisesti funktion tarkoitus on hakea kaikki tarinat yhden yhteisön ID:n mukaisesti. Rivillä 75 luodaan tyhjä lista, joka täytetään myöhemmin rivillä 84 kriterian täyttäneillä tarinoilla. Riveillä 79-82 asetetaan tietokannasta haetuille tarinoille kriteriat. Rivillä 79 ilmaistaan, että tarinan pitää olla tyyppiä ”Story”. Rivillä 80 tehdään assosiaatio Story-luokasta löytyvän communityId:n ja Community-luokasta löytyvän ID:n välillä. Rivillä 81 tarinat asetetaan päivämäärän mukaiseen järjestykseen uusimmasta vanhimpaan. Lopulta rivillä 82 tehdään rajoitus, että vain tarinat, joiden yhteisön ID on sama kuin funktiolle annettu communityId, lisätään listaan. Jos näiden operaatioiden aikana ei sattunut virheitä, palauttaa funktio kriterian mukaisen tarinalistan Service-luokalle. Tämä puolestaan palauttaa listan käsittelijälle, joka lopulta palauttaa listan näkymälle. Tämä muodostaa esimerkiksi aikajananäkymän tarinalistasta.



Kuva 6. Webapp/resources-kansio

Kansiosta resources löytyvät mm. erilliset CSS-tiedostot (kansiosta css) sekä JavaScript -tiedostot (kansiosta js). Suurin osa näistä tiedostoista ovat kolmannen osapuolen tekemiä, esim. JQueryyn tai Bootstrappiin kuuluvat tiedostot. Kansioista löytyy kuitenkin myös tärkeitä ohjelman omia tiedostoja, kuten css-kansion lomakkeiden muotoilutiedosto, sekä js-kansion tiedosto clouds.js, jossa käsitellään suurinta osaa käyttöliittymän graafisesta puolesta. Näiden lisäksi resources-kansiosta löytyvät kaikki ohjelmassa käytetyt kuvat (kansiosta img) sekä fontit (kansiosta font).

Resources-kansion toinen funktio on olla tallennuspaikka käyttäjän luomille videoille/änitiedostoille, sekä heidän lisäämilleen kuville ja pdf-tiedostoille. Luotaessa ensimmäinen tiedosto luodaan samalla myös tiedostotyyppiä vastaava kansio resources -kansioon. Esimerkiksi videoille luodaan Video-kansio ja kuville Image-kansio.



Kuva 7. WEB-INF/pages-kansio

WEB-INF-kansiosta löytyy loppu ohjelman front-end-puolen tiedostoista. Tähän kansioon on sijoitettu ohjelman käyttöliittymän yksittäiset sivut, eli JSP-tiedostot. JSP-tiedostot koostuvat HTML:n lisäksi tiedostossa tarvittavista CSS-tyyleistä sekä JavaScriptistä. Nämä tiedostot toimivat Spring MVC:n mukaisena View-komponenttina.

Kuvan kansiosta admin löytyy ohjelman järjestelmänvalvojien näkymiä, joita vain järjestelmänvalvoja voi käyttää suoraan ohjelmasta. Näitä ovat esimerkiksi yhteisöjen ja ankkuri- tai muistiavaintarinoiden luonti. Api-kansiosta kuuluisi löytyä ohjelman API, eli

tietokantakutsujen selitykset, mutta tätä ei ole vielä viety kovin pitkälle. Kansio form on kaikista tärkein, koska se sisältää kaikki ohjelman julkiset lomakkeet. Esimerkiksi tarinoiden luontiin välttämättömät lomakkeet löytyvät tästä kansiosta. Help-kansiosta löytyy erinäisiä apulomakkeita, joiden avulla käyttäjä pystyy avaamaan esimerkiksi ohjelman käyttöohjeet sekä introvideon. Kuvan 7 viimeinen kansio target on ohjelmaa käännettäessä Mavenin luoma kansio, johon se tallentaa käännetyn ohjelman.

Kuvassa 7 näkyvä dispatcher-servlet.xml on ehkä Spring-kehyksen tärkein tiedosto. Tiedoston tehtävä on vastaanottaa ohjelman lähettämä URI-merkkijono, ja kutsua oikeaa käsittelijämetodia. Käsittelijän palauttaessa JSP-sivun ja muut tiedot takaisin dispatcher-servletille, osaa se luoda tästä tiedosta html-dokumentin ja palauttaa sen käyttäjälle. Tämä tiedosto on siis ikään kuin Spring MVC:n osien View ja Controller yhdistävä tekijä. [6.]

2.3 Teknologiat

Seuraavaksi esitetään ohjelmassa käytetyt tärkeimmät kolmannen osapuolen teknologiat.

Taulukko 1. Tärkeimmät teknologiat

Teknologia	**Kuvaus**
Eclipse Java EE IDE	Eclipse Foundationin luoma ohjelmointiympäristö (IDE eli integrated development environment), jota käytetään ohjelman kehityksessä. Eclipsen Java EE-versio sisältää ohjelmalle tärkeitä työkaluja, kuten Maven.
Java	Oraclen luoma oliopohjainen ohjelmointikieli, jota käytetään ohjelman back-end -puolella tietokantaoperaatioissa, sekä mallejen luonnissa.
HTML5, CSS, JavaScript	Ohjelman front-end-puolen ohjelmistokieliä. HTML (Hypertext Markup Language) on internetsivuilla standardoitu kieli. CSS:ää (Cascading Style Sheets) käytetään HTML-sivujen visuualisen tyylin muotoiluun. JavaScriptin avulla pystytään lisäämään sivulle dynaamista toiminnallisuutta.
jQuery	jQuery on JavaScript-kirjasto, joka esimerkiksi tekee HTML -dokumentin käsittelystä vaivattomampaa ja helpottaa tapahtumienkäsittelyä (kuten hiirellä klikkausta). [1.]
three.js	three.js on WebGL-rajapintaa toteuttava JavaScript-kirjasto, jolla pystytään luomaan 3D -grafiikkaa internet selaimessa. Tätä kirjastoa

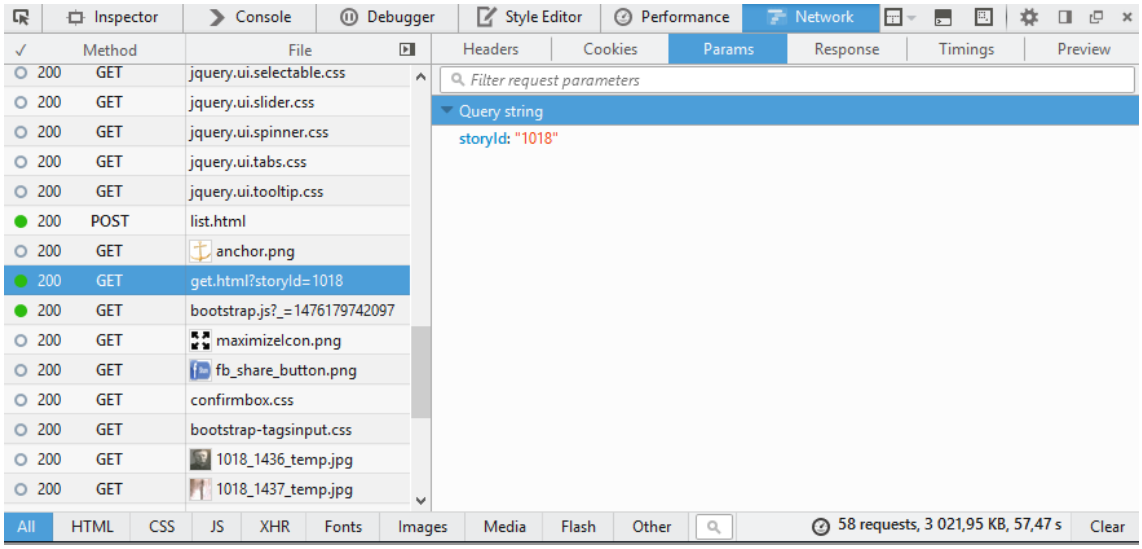
	käytetään varsinkin ohjelman aikajananäkymässä tarinoiden ja muiden objektien mallintamiseen. [2.]
Bootstrap	Bootstrap on front-end sovelluskehys verkkosovellusten luontiin. Se sisältää valmiita malleja esimerkiksi napeille ja siitä on erityisesti hyötyä verkkosovelluksen ulkoisen presentaation kasassa pitämisessä päätelaitteesta huolimatta.
Ajax	Ajax (Asynchronous JavaScript and XML) on kokoelma tekniikoita, joiden avulla pystyy lähettämään ja vastaanottamaan palvelimelta dataa asynkronisesti.
Spring Web MVC	Spring Web MVC on vapaan lähdekoodin sovelluskehys verkkosovellusten luomiseen. Spring -sovelluskehyksessä käytetään MVC-arkkitehtuuria, eli ohjelman jakoa Malliin (Model), Näkymään (View) sekä Käsittelijään (Controller). [3.] Sovelluksessa malli kuvaa tietokantarakenteita, joita käsittelijää muokkaa, poistaa tai lisää näkymästä tulevien ohjeiden mukaisesti. Tässä toteutuksessa mallit sekä käsittelijät sijaitsevat back-end-puolella käyttäjän ulottumattomissa. Näkymä sijaitsee front-end -puolella ja vastaanottaa käyttäjän antamia käskyjä, jotka sitten välitetään käsittelijälle.
Hibernate ORM	Hibernate ORM (Object/Relational Mapping) on Java-pohjainen työkalu, joka mahdollistaa käyttäjän antaman datan tallentamisen tietokantaan ohjelmallisesti.
JSP	JSP (JavaServer Pages) on Javaan perustuva menetelmä luoda HTML -muotoisia verkkosivuja. Käytössä Spring sovelluskehyksessä.
JSTL	JSTL (JavaServer Pages Standard Tag Library) on kokoelma JSP-tageja, jotka ovat yleisessä käytössä JSP-sovelluksissa. [4.]
Red5	Red5 on sovelluksessa käytössä oleva mediapalvelin. Red5 -palvelimesta on olemassa sekä ilmaisversio, että maksullinen versio, josta löytyy enemmän ominaisuuksia. Sovelluksessa on käytössä palvelimen ilmaisversio. Red5-palvelin on käytössä tuhansissa yhtiöissä, mukaanlukien Amazon sekä Facebook. [5.] Tästä huolimatta Red5:n ilmaisversion dokumentaatio on jostain syystä hyvin rajallista, suurin osa tiedosta on viime vuosikymmeneltä itseoppineiden käyttäjien toimesta. Tämä aiheutti ongelmia palvelimen käyttöönotossa sekä SSL-sertifikaatin lisäämisessä verkkosivulle.
Maven	Mavenia käytetään sovelluksen kääntämiseen lopulliseksi verkkosovellukseksi, jonka voi ajaa käyttäen mediapalvelinta kuten Red5.
HDFVR	HDFVR on sovelluksessa käytetty videon tai äänen taltiointityökalu. Videota pystyy kuvaamaan omalla webkameralla ja ääntä tallettamaan mikrofonilla.
FFmpeg, FFplay, FFprobe	Näitä sovelluksia käytetään HDFVR-sovelluksen taltioiman video- tai äänivirran (.flv) muuntamiseen sovelluksessa käytettäviin muotoihin (.mp3, .mp4).
Apache POI	POI on Java-pohjainen rajapinta, jota käytetään Microsoft dokumenttejen dynaamiseen luontiin. Sovelluksessa tätä rajapintaa käytetään varsinkin XLS-dokumenttien luontiin.

Apache Log4j	Apachen Log4j on Java-pohjainen kirjaamistyökalu. Tätä työkalua käytetään ohjelman back-end-puolen virheiden ja muiden ilmoitusten kirjaamiseen Red5-ikkunaan tai erilliseen tiedostoon, josta on apua ohjelman virheiden löytämisessä ja korjauksessa.

3 Jatkokehitys

Aloittaessani sovelluksen jatkokehitystä, en tiennyt Spring-kehyksestä käytännössä muuta kuin nimen. Olin käyttänyt aikaisemmin koulun innovaatioprojektissa samanlaista C#-pohjaista kehystä nimeltään ASP.NET webapi2. Tämä antoikin hyvää pohjaa ohjelman koodin lukemista varten. Ohjelman back-end puolen rakenne osoittautui hyvin samanlaiseksi innovaatioprojektimme kanssa.

Kokemattomuuteni front-end-ohjelmoinnin kanssa aiheutti kuitenkin ongelmia ja kesti suhteellisen kauan ennen kuin pääsin siitä perille. Spring-kehyksen automaattisuus sekä dispatcher-servletin toiminta hämmensivät minua suuresti. Yritin lukea Springin virallisia tutoriaaleja, mutta kuten yleensä, niitä ei ole kirjoitettu kovin aloittelijalähtöisesti. Tämän vuoksi lähdin itsenäisesti testaamaan ohjelmaa ja lukemaan sen antamaa palautetta käyttäen konsolin viestejä, Eclipsen Debug-asetusta (jolla voi ajaa koodia askel askeleelta, jolloin saa tietää missä kohtaa koodia tietyllä hetkellä ollaan) sekä varsinkin Firefoxin ja Chromen kehittäjätyökaluja.



Kuva 8. Firefoxin kehittäjätyökalut

Kuvassa 8 on esimerkki kehittäjätyökalujen network-osion näyttämästä lisäinformaatiosta avattaessa tarinaa sovelluksessa. Kaikki selaimen tekemät kutsut näkyvät vasemmalla. Tämän näkymän avulla saatiin selville, mitä kutsuja ohjelman front-end tekee painettaessa mitäkin käyttöliittymän nappia. Avattaessa tarinaa kutsuu sovellus sijaintia [ohjelman osoite]/get.html?[parametri]. Tämän jälkeen oli helppo etsiä back-end-puolen käsittelijästä tämä metodi ja vastaavasti front-end-puolen koodista linkki tai lomake (form), jossa mainitaan tämä metodi. Network-ikkunan params-osioon on merkitty kaikki kutsun lähettämät parametrit, josta oli erityisen paljon hyötyä varsinkin silloin, kun parametrit oli piilotettu POST-kutsussa kutsun body eli vartaloosaan. Tämän avulla pystyttiin testaamaan, että uudet itse luodut kutsut sisälsivät varmasti oikeat tiedot oikeassa muodossa.

Kehittäjätyökalut mahdollistivat myös käyttöliittymäpuolen debuggauksen käyttäen työkalujen Web Console sekä Debugger -toimintoja. Ilman näitä toimintoja käyttöliittymän kehitys olisi ollut huomattavasti vaikeampaa ellei mahdotonta. Consoleen pystyy kirjaamaan omia lokejaan, jonka ansiosta on helppo tietää, missä kohtaa koodia mennään ja missä virheet sijaitsevat. Console ilmoittaa myös syntaksivirheet, mikä tekee niiden korjaamisesta nopeaa ja helppoa. Debuggerin ansiosta löydettiin vähän vaikeammin ymmärrettäviä virheitä, kuten käyttäjän syötteenä antaman tekstin virheitä. Näissä tilanteissa esimerkiksi käyttäjän syöttämä heittomerkki ” rikkoi koodin, koska se on myös koodin lopetusmerkki. Näiden vuoksi piti käyttää yleistä XML escape -funktiota, joka muuttaa erikoismerkit tekstistringeiksi (esimerkiksi ” -merkki muuttuu muotoon ").

3.1 Uudet ominaisuudet

Työharjoitteluaikana koodiin tehtiin lukuisia muutoksia ja uusia ominaisuuksia. Seuraavassa esitellään niistä tärkeimmät muutosten tekemisjärjestyksessä.

3.1.1 PDF-tarina

Erityisesti PDF-tyyppisen tiedoston lisääminen tarinaan tuli vaatimuksena asiakkaalta. PDF-tarinasta ei kuitenkaan haluttu tehdä erillistä tarinatyyppiä neljän muun tarinatyypin joukkoon, koska siitä voisi aiheutua turhaa hämmennystä. PDF-tarinasta päädyttiin lopulta tekemään alatyyppi tekstitarinalle, koska se muistutti neljästä tarinatyypistä eniten sitä. Koska tämä oli ensimmäinen tehtäväni yrityksessä, tuli minun luoda vain pohja, josta intialainen yritys tekisi lopullisen version. Seuraava koodi on siis tuotettu oman koodini pohjalta pienin muutoksin. Tämä on ainoa muutoksista, joissa valmista koodiani ei viety suoraan julkiselle palvelimelle.

```
1139 <c:if test="${fn:contains(storyPage.storyContent.type,'TEXT')}">
1140     <div class="storypage" style="background: #ededed">
1141     <input type="hidden" class="storyPageType" value="text"/>
1142         <p style="padding-top: 15px; font-size: 18px; color: gray;"><spring:message code="storyReview.addText" /></p>
1143         <p style="padding: 10px 25px 25px;">
1144             <form:textarea
1145                 path="storyPages[${pages.index}].storyContent.content"
1146                 form="acceptStory" rows="12" cols="70" class="textstory"
1147                 style="font-size:20px;"/>
1148         </p>
1149         <a href="#" onclick="finish();" class="linkbtn-blue btnsavestory "
1150             style="margin-left: 45%;"><spring:message
1151                 code="storyReview.finish" /></a>
1152
1153         <!-- Add pdf code -->
1154         <c:set var="userSession" value="${sessionScope.userSession}" />
1155         <%--<c:forEach var="communityUsers" items="${userSession.communityUsers }" >
1156             <c:if test="${communityUsers.isAdmin eq 'Y'.charAt(0) }" >--%>
1157             <c:if test = "${newStory.storyType ne '3dObject'}">
1158                 <a href="#" onclick="document.getElementById('storyPageIndex').value=${pages.index}; document.getElementById('pdfUpload').click(); return false;"
1159                 class="linkbtn-blue btnsavestory " style="margin-left: 2%;">
1160                     <spring:message code="storyReview.addPdf" />
1161                 </a>
1162             </c:if>
1163             <%--</c:if>
1164         </c:forEach>--%>
1165
1166     </div>
1167
1168 </c:if>
```

Kuva 9. Muutos tekstitarina-näkymään

Kuvassa näkyy tapa, jolla ohjelma erittelee tekstitarinan muista tarinatyypeistä. Rivillä 1139 JSTL:n <c:if>-tagi määrää, että vain, jos käyttäjä on aikaisemmassa näkymässä valinnut tarinan tyypiksi tekstin, tulee ohjelma tähän kohtaan koodia. Video-, kuva- sekä äänitarinoille on olemassa samanlaiset koodinpätkät. Riviltä 1153 alkaa varsinainen PDF -tarinaan liittyvä koodi. Rivillä 1157 sanotaan, että uuden tarinan tyyppi ei saa olla muotoa 3dObject. Tällä hetkellä 3D-objekti tarinat saavat olla vain tekstimuodossa. 3D -objekti tarinoihin palataan myöhemmässä kappaleessa. Riviltä 1158 alkaen <a>-tagilla merkitty painonappi aloittaa PDF-tarinan luomisen. Painettaessa tätä nappia

tapahtumankäsittelijä onclick painaa alempana dokumentissa lomakkeen sisälle piilotettua pdfUpload-nappia.

```
1195            <form:form
1196                id="uploadPDF"
1197                modelAttribute="newStory" method="post"
1198                enctype="multipart/form-data">
1199            <input type="file" id="pdfUpload" name="file" accept="application/pdf" onchange="addPdf()" style="display: none;" />
1200            <input type="text" id="description" name="description" style="display: none;">
1201            <input type="text" id="storyPageIndex" name="storyPageIndex" style="display: none;"/>
1202            </form:form>
```

Kuva 10. Lomake PDF:n lataukseen

Kyseinen nappi näkyy kuvan 10 rivillä 1199. Painettaessa pdfUpload-nappia pyytää ohjelma käyttäjää antamaan PDF-tyyppisen tiedoston, koska napin tyypiksi on ilmoitettu file mutta se on konfiguroitu hyväksymään vain application/pdf-muotoisia tiedostoja. Tähän nappiin on koodattu toinen tapahtumankäsittelijä nimeltä onchange. Tämä tarkoittaa, että kun käyttäjä valitsee tiedoston, kutsuu onchange sille konfiguroitua JavaScript-metodia addPdf(). Tämä mahdollistaa myös sen, että jos käyttäjä ei valitse mitään tiedostoa, ei mitään myöskään tapahdu. PDF-tiedoston lisäksi lomake sisältää kaksi muuta tietuetta. Rivin 1200 ”description” on valinnainen PDF:n kuvaus. Rivin 1201 ”storyPageIndex” on tarinasivu, jolle palataan, jos PDF:n latauksessa esiintyi ongelmia.

```
812     function addPdf() {
813         document.getElementById("uploadPDF").action = "/story/addPdf.html";
814         document.getElementById("uploadPDF").submit();
815     }
```

Kuva 11. addPdf()-funktio

Onchange-tapahtumankäsittelijän kutsuessa addPdf()-funktiota, tullaan kuvan 11 riville 812. Metodi asettaa kuvan 10 rivin 1195 lomakkeen osoitteeksi ”/story/addPdf.html”. Tämän jälkeen rivillä 814 metodi lähettää DispatcherServlet:n kautta lomakkeen osoitetta vastaavalle käsittelijälle hoitamaan itse PDF:n latauksen. Lomakkeen metodiksi asetetaan POST (kuva 10, rivi 1197), koska dataa lähetetään prosessoitavaksi, ja se enkoodataan multipart/form-data -menetelmällä (kuva 10, rivi 1198), jota on käytettävä kutsun sisältäessä tiedoston.

```
3576  // Add pdf
3577      @RequestMapping(value = "/story/addPdf", method = RequestMethod.POST)
3578      public ModelAndView addPdfStoryPage(@ModelAttribute("newStory") Story newStory,
3579              @RequestParam("file") MultipartFile file,
3580              @RequestParam("description") String description,
3581              @RequestParam("storyPageIndex") int storyPageListIndex) throws IOException
3582      {
3583          log.debug("Add pdf");
3584          if(newStory==null)
3585          {
3586              // redirect to chooseMedia
3587              ModelAndView mav = new ModelAndView("chooseMedia");
3588              mav.addObject("newStory", new Story());
3589
3590              return mav;
3591          }
3592          else
3593          {
3594              // check if uploaded file is of type pdf
3595              String contentType = file.getContentType();
3596              if(!contentType.equals("application/pdf") && !contentType.equals("application/x-download") && !contentType.equals("application/x-dowload"))
3597              {
3598                  // send error message
3599                  ModelAndView mav = new ModelAndView("story");
3600                  mav.addObject("formUrl", "form/getStoryReviewForm");
3601                  mav.addObject("error", message.getMessage("error.invalidPdf", null, LocaleContextHolder.getLocale()));
3602                  mav.addObject("viewPageNo", new String(storyPageListIndex+1+""));
3603                  return mav;
3604              }
3605
3606              // add image storypage to newStory
3607              try
3608              {
3609                  newStory = storyService.addMediaStoryPage(newStory, file, description, contentType);
3610              }
3611              catch(Exception e)
3612              {
3613                  e.printStackTrace();
3614              }
3615
3616              ModelAndView mav = new ModelAndView("redirect:/story/form/storyReviewForm.html");
3617              mav.addObject("newStory", newStory);
3618
3619              return mav;
3620          }
3621      }
```

Kuva 12. Käsittelijän metodi addPdfStoryPage

Tämän jälkeen lomake saapuu kuvan 12 käsittelijän metodiin addPdfStoryPage, jonka osoitteeksi on rivillä 3577 asetettu ”/story/addPdf”. Lomakkeeseen sisällytetty PDF-tiedosto otetaan talteen muuttujaan file rivillä 3579 ja muut tiedot riveillä 3580 ja 3581. Rivillä 3584 käsittelijä tarkastaa, että tarina (Story) -olio on ylipäätään olemassa ja tarinan tekeminen on aloitettu normaalilla tavalla. Jos tähän tilanteeseen ollaan jostain syystä päästy ilman sovelluksen oikeanlaista käyttöä, ohjelma palauttaa käyttäjän takaisin tarinan aloituspisteeseen riveillä 3587-3590. Jos tarina on aloitettu oikein, pääsee funktiossa eteenpäin. Rivillä 3595 otetaan talteen ohjelman sisältötyyppi. Sisältötyypin tulee olla joko ”application/pdf”, ”application/x-download” tai ”application/x-dowload”. Tiedoston sisältötyypin kuuluisi käytännössä olla aina ”application/pdf”, mutta joissain selaimissa olevien bugien vuoksi voi se esiintyä myös kahtena jälkimmäisenä. Jos sisältötyyppi ei jostain syystä ole joku näistä, lähettää käsittelijä näkymälle virheen (rivit 3599-3603), jonka se esittää käyttäjälle.

Jos näistä kahdesta ehdosta päästään eteenpäin, päästään itse PDF-tiedoston lataukseen try-catch-lauseeseen riveille 3607-3614. Kutsun addMediaStoryPage tarkoitus on luoda kansio PDF-tiedostoille, jos sellaista ei ole jo olemassa, ja tallentaa sinne uusi PDF-tiedosto. Kaiken mennessä hyvin, palauttaa ohjelma rivillä 3619

käyttäjälle esikatselun tekemästään PDF-tarinasta. Tässä näkymässä käyttäjä voi vielä päättää, haluaako viedä tarinansa loppuun tai halutessaan vaihtaa PDF-tiedoston.

3.1.2 XLS-raportin generointi

Seuraava asiakasvaatimus oli saada mahdollisuus pystyä saamaan lisää tietoa hallinnoimastaan yhteisöstä. Asiakkalle oli tärkeää saada tietää yhteisössään olevien tarinoiden määrä, sekä kuka oli tehnyt minkäkin. Raportin olisi voinut tehdä monella tavalla, mutta päädyimme työnantajan ehdotuksesta XLS-tyyliseen taulukkorakenteeseen. Koska Javaan ei ole sisäänrakennettu Microsoft Office -dokumenttejen generointia, oli pakko ottaa käyttöön kolmannen osapuolen Apache POI -kirjasto. POI:n avulla pystyy ohjelmallisesti luomaan monenlaisia Microsoftin omistamien tiedostotyyppejen tiedostoja, yksi näistä XLS.

```
369    @RequestMapping(value="/community/getInfo", method = RequestMethod.GET)
370    public void getInfo(@RequestParam("communityId") long communityId,
371                                                    HttpServletResponse httpResponse) throws IOException
372    {
373        ArrayList<Story> stories = (ArrayList<Story>) storyService.getStoriesByCommunity(communityId);
374        ExcelRaport e = new ExcelRaport();
375        HSSFSheet sheet = e.buildDocument(stories);
376        try{
377            httpResponse.setContentType("application/vnd.ms-excel");
378            httpResponse.setHeader("Content-Disposition", "attachment; filename="+communityService.getCommunityById(
379                    communityId).getName().replaceAll("\\s+","")+"_raportti.xls");
380
381            ServletOutputStream output = httpResponse.getOutputStream();
382            sheet.getWorkbook().write(output);
383            output.flush();
384        }catch(Exception ex){
385            log.info("Unable to write report to output stream.");
386        }
387    }
```

Kuva 13. Käsittelijän metodi getInfo

Yhteisönäkymään lisättiin nappi, jonka tarkoitus on kutsua kuvassa 13 näkyvää getInfo metodia, ja lähettää samalla yhteisön id, josta halutaan tietää enemmän. GetInfo-metodi hakee alussa kaikki yhteisön tarinat listaan, jonka jälkeen se antaa tarinalistan buildDocument funktiolle. BuildDocument funktio palauttaa valmiin raportin, joka asetetaan POI:sta löytyvään HSSFSheet -objektiin. Tämän jälkeen rivillä 377 asetetaan käyttäjälle lähetetyn tiedoston sisältötyypiksi ”application/vnd.ms-excel”, eli periaattessa kerrotaan selaimelle että kyseessä on XLS-tiedosto. Seuraavalla rivillä tiedoston nimeksi asetetaan [yhteisön nimi]_raportti.xls ja riveillä 381-383 lähetetään valmis tiedosto käyttäjälle.

Seuraavassa kerrotaan lisää itse raportin luonnista.

```
22      public HSSFSheet buildDocument(ArrayList<Story> stories) throws IOException{
23          int amount = 0, textStory = 0, imageStory = 0, audioStory = 0, videoStory = 0, pdfStory = 0;
24
25          Collections.sort(stories, new Comparator<Story>() {
26                public int compare(Story o1, Story o2) {
27                    Long a = (Long)o1.getCreated().getTime();
28                    Long b = (Long)o2.getCreated().getTime();
29                    if (a == null || b == null)
30                      return 0;
31                    return ((Long)a).compareTo((Long)b);
32                }});
33
34          HSSFWorkbook wb = new HSSFWorkbook();
35          HSSFSheet sheet = wb.createSheet("Raportti");
```

Kuva 14. BuildDocument-funktio

Kuvan 13 rivillä 375 kutstuaan kuvan 14 buildDocument-funktiota. Funktion tarkoitus on solu solulta rakentaa ja tyylitellä XLS-taulukko yhteisön tietojen mukaisesti käyttäen POI -rajapintaa. Ensimmäinen tehtävä on laittaa tarinalistan tarinat järjestykseen tarinan luontiajan mukaisesti, koska tarinalista ei ole valmiiksi tässä järjestyksessä. Javassa listan uudelleenjärjestämiselle ei ole helppoa, minkä takia joudutaan käyttämään rivien 25-32 Collections-rajapinnan sort-metodia. Metodin lopputuloksena on lista halutussa järjestyksessä. Listan uudelleenjärjestyksen jälkeen voidaan siirtyä itse raportin tekemiseen. Rivillä 35 luodaan tyhjä HSSFSheet -objekti nimeltään ”Raportti”. Taulukon soluille asetetaan koot, fontti ja reunojen paksuus.

```
87          for(Story story : stories){
88              for(StoryPage storypage : stories.get(rows).getStoryPages()){
89                  amount++;
90
91                  Row row1 = sheet.createRow((short)rows+1);
92                  HSSFCell cellId = (HSSFCell) row1.createCell(0);
93                  cellId.setCellValue(rows+1);
94                  cellId.setCellStyle(bodyCellStyle);
95                  HSSFCell cell5 = (HSSFCell) row1.createCell(1);
96                  cell5.setCellValue(story.getUser().getFirstName()+" "+story.getUser().getFamilyName());
97                  cell5.setCellStyle(bodyCellStyle);
98                  HSSFCell cell6 = (HSSFCell) row1.createCell(2);
99                  cell6.setCellValue(story.getUser().getEmail());
100                 cell6.setCellStyle(bodyCellStyle);
101                 HSSFCell cell7 = (HSSFCell) row1.createCell(3);
102                 cell7.setCellValue(story.getTitle());
103                 cell7.setCellStyle(bodyCellStyle);
104                 HSSFCell celltype2 = (HSSFCell) row1.createCell(4);
105
106                 if(storypage.getStoryContent().getType().toLowerCase().trim().contains("text")){
107                     if(storypage.getStoryContent().getContent().toLowerCase().trim().contains("pdf")){
108                         pdfStory++;
109                         celltype2.setCellValue("PDF");
110                     }
111                     else{
112                         textStory++;
113                         celltype2.setCellValue("Teksti");
114                     }
115                 }
116                 else if(storypage.getStoryContent().getType().toLowerCase().trim().contains("image")){
117                     imageStory++;
118                     celltype2.setCellValue("Kuva");
119                 }
120                 else if(storypage.getStoryContent().getType().toLowerCase().trim().contains("audio")){
121                     audioStory++;
122                     celltype2.setCellValue("Ääni");
123                 }
124                 else if(storypage.getStoryContent().getType().toLowerCase().trim().contains("video")){
125                     videoStory++;
126                     celltype2.setCellValue("Video");
127                 }
```

Kuva 15. Taulukon täyttäminen

Seuraavaksi raportti täytetään yhteisön tiedoilla. Tarinan tiedoiksi tulevat käyttäjän etu ja sukunimi, käyttäjänimi, joka on sama kuin käyttäjän sähköpostiosoite, ja tarinan otsikko (rivit 91-104). Riveillä 106-127 määritetään tarinan tyyppi ja kirjoitetaan se raporttiin. Samalla pidetään laskuria kustakin tarinatyypistä myöhempää käyttöä varten. Kuvan 15 viimeisen rivin jälkeen raporttiin kirjoitetaan myös tarinan luontiaika minuutilleen.

Laskurit (kuvassa 15 näkyvät amount, pdfStory, textStory, imageStory, audioStory, videoStory) kertovat, kuinka monta kutakin tarinasivua on yhteisössä. Ohjelman poistuttua kuvan 15 silmukasta lisätään nämä vielä dokumenttiin. Valmis raportti palautetaan tämän jälkeen kuvan 13 getInfo-metodille.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	ID	Tekijän nimi	Käyttäjänimi	Tarinan otsikko	Tarinan tyyppi	Luontiaika		Tarinasivujen määrä	Tekstitarinoiden määrä	Kuvatarinoiden määrä	Äänitarinoiden määrä	Videotarinoiden määrä	PDF-tarinoiden määrä
2	1	[redacted]		test	Teksti	25.5.2016 14:58		6	3	2	1	0	0
3	2			Testi	Kuva	4.10.2016 10:25							
4	3			Testi2	Teksti	14.10.2016 12:54							
5	4			Kuvatesti	Kuva	14.10.2016 12:55							

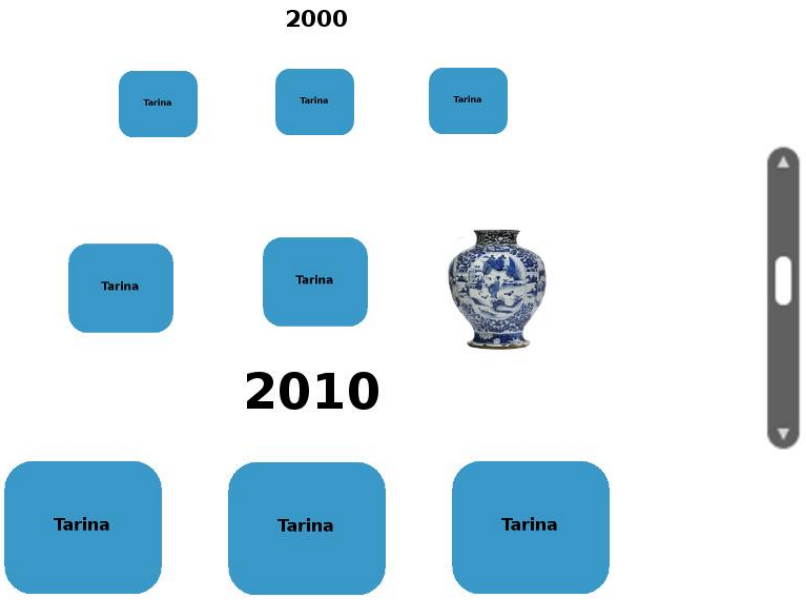
Kuva 16. Valmis raportti

Käyttäjälle saapuva tiedosto on kuvan 16 mukainen. Raportti sisältää kaikki asiakasvaatimuksen mukaiset tiedot. Yksi parannuksen aihe dokumentissa olisi yhteisön tarinoiden kokonaismäärän lisääminen, jonka kyllä näkee dokumentista ID:n kohdalta, mutta sillä ei ole omaa selkeätä kohtaa dokumentissa. Tarinan tyyppi -osio voi olla hiukan turha, koska se kertoo vain tarinan ensimmäisen sivun tyypin, kun taas kaikki tarinasivujen tyypit on jo lueteltu oikealla. Tulevaisuudessa dokumenttiin on mahdollista lisätä kaikenlaista muutakin tietoa tarpeen mukaan suhteellisen helposti. Järjestelmänvalvojan tasolla yksi kehitysmahdollisuus olisikin tehdä yleiskatsaus koko ohjelman kaikkiin tarinoihin/käyttäjiin/yhteisöihin ilman tarvetta käydä penkomassa tietokantaa.

3.1.3 3D-objekti

Ohjelman yksi suurimmista haasteista oli tarinoiden erottaminen toisistaan visuaalisesti. Aikajanan tarinat esiintyvät käyttäjälle suurena massana samoja ikoneita, eikä niistä otsikkoa lukuunottamatta saa juurikaan mitään irti ennen niiden avaamista. Tämä oli yksi syistä luoda ”3D-objekti” -nimellä kulkeva tarina.

Uuden tarinatyypin tarkoitus on luoda aikajanalle kiintokohteita, joihin käyttäjän huomio kohdistuu. Ne esiintyvät aikajanalla uniikkeina kuvina. Kuvat voivat olla esimerkiksi historiallisista tai ylipäätään mielenkiintoa herättävistä esineistä. Kuvien tarkoitus on luoda keskustelua ja tarinaan liittyviä uusia tarinoita. Tätä varten luotiin myös toinen uusi ominaisuus, josta kerrotaan enemmän luvussa 3.1.4.



Kuva 17. 3D-objekti aikajanalla

Tällä hetkellä 3D-objektit on asetettu olemaan pelkästään tekstimuotoisia tarinoita. Objektia painettaessa tulee esiin samanlainen tarinasivu kuin normaaleissa tarinoissa, mutta tarinaan on lisäksi lisätty vielä kuva 3D-objektista. Tarina itse koostuu kuvan objektiin liittyvästä tiedosta sekä kuvan lähteestä. Lisäksi tarinasta pystyy kertomaan omia tarinoita, ja tuomaan esiin jo tarinasta kerrottuja tarinoita. 3D-objektin pystyy luomaan sekä julkiselle aikajanalle, yhteisöihin että yksityiselle aikajanalle, vaikkakin tähän asti on käytetty pelkästään julkista aikajanaa. 3D-objekteja pystyy luomaan vain järjestelmänvalvoja.

Koska sovellus on muistojen tallennuspalvelu, on kuvien lisäämisestä aikajanalle erityisesti hyötyä. Esimerkiksi vanha kaupan tuote, joka on aikaa sitten poistunut jo hyllyiltä, voi tuoda aikajanalla käyttäjälle mieleen vanhoja muistoja. Toisaalta siitä voi olla myös hyötyä tiedon lisäämisessä. Aikajanalla yhdistyy sekä tuotteen ilmestymisvuosi, esine itse sekä sen ulkomuoto, luontipaikka ja tarinaa itse tuotteesta. Lisäämällä lukuisia 3D-objekteja omalle aikajanalleen, pystytään tekemään esimerkiksi yhden yhtiön tuotteiden kirjasto, joka voi olla mielenkiinnon kohde itse yritykselle tai normaaleille käyttäjille. Toisaalta sovelluksen tullessa tunnettummaksi, voidaan miettiä 3D-objektin

käyttöä jopa mainonnassa. Koska objekti on osa sovellusta, se ei olisi tungetteleva kuten perinteinen mainos. Riippuen niiden käyttötavasta, 3D-objekteilla voi olla siis myös myönteinen vaikutus sovelluksen myyntiin.

```
@Entity
@Table(name="3dobject")
public class ThreeDObject {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "name")
    private String name;

    @Column(name = "filePath")
    private String filePath;

    @Column(name = "dimensionX")
    private Integer dimensionX;

    @Column(name = "dimensionY")
    private Integer dimensionY;
```

Kuva 18. 3D-objektin entiteetti sovelluksessa

3D-objekti kuvataan sovelluksessa kuvan 18 vastaavalla tavalla. Objektilla on nimi, polku kuvatiedostoon ja pikselikoot sekä x- että y-suunnassa. Sovelluksen järjestelmänvalvojalla on käytössään kaksi metodia 3D-objekteihin liittyen. Sovelluksen tietokantaan voi lisätä 3D-objektin tai sovellukseen jo lisätyistä 3D-objekteista voi valita yhden ja kertoa siitä tarinan. Lisättäessä tarinaa tietokantaan otetaan siitä samalla ylös sovelluksen leveyden ja korkeuden pikselikoot ja tallennetaan ne dimensionX- ja dimensionY-muuttujiin. Objektin kuva tallennetaan /resources/image/admin/3dobject-kansioon, ja annetaan sille uniikki nimi sen id:n ja nimen mukaisesti. Tämän jälkeen tämä osoite tallennetaan vielä kuvan 18 filePath-muuttujaan.

Järjestelmänvalvojan metodi kertoa tarina 3D-objektista toimii lähes samoin kuin normaali tarinan kertominen sovelluksessa. Ainoa ero on, että järjestelmänvalvojan on tehtävä tarina käyttäen eri polkua, jolloin tarina pystytään merkitsemään tietokantaan 3D -objektina. Tietokannan Story-tauluun lisätään tarinan tyypiksi 3dObject normaalin story -tyypin sijaan, ja asetetaan yksi yhteen suhde taulujen Story- sekä 3dObject-väleillä.

Uudenlaisen tarinan näyttämiseen aikajanalla vaaditaan hiukan erilaista käsittelyä. Tässä tietokantaan tallennetut filePath, dimensionX- ja dimensionY-muuttujat tulevat hyödyllisiksi.

```
780 //loads 3dObjects based on their filePath attribute
781 else if (storiesList[j].storyType == "3dObject"){
782     storyImageTexture = THREE.ImageUtils.loadTexture(
783             storiesList[j].threeDObject.filePath, new THREE.UVMapping());
784     storyImageTexture.id = year;
785     storyImageTexture.needsUpdate = true;
786     var storyImageMaterial = new THREE.MeshBasicMaterial({
787         map: storyImageTexture,
788         transparent: true,
789         opacity: 0.8,
790         depthText: false
791     });
792 }
```

Kuva 19. Tarinan tekstuurin ja materiaalin asettaminen aikajanalla

Tarinoille annetaan tekstuuri, joka määritellään silmukassa tarinan tyypin mukaisesti. Tarinan tekstuuri ladataan riveillä 782 ja 783. Lataukseen käytetään aikaisemmin asetettua filePath-muuttujaa, joka osoittaa kuvan sijaintiin levyllä. Tekstuurin lataamisen jälkeen asetetaan rivillä 785 sen muuttuja needsUpdate todeksi. needsUpdate kertoo ohjelmalle, että tekstuuri on vaihtunut ja se pitää päivittää. Riveillä 786-791 luodaan tarinan kuvan materiaali ja asetetaan tekstuurin lisäksi objektille hienoinen läpinäkyvyys. Lopullisen kuvan piirto ruudulle suoritetaan käyttäen 3D-objektin dimensionX- ja dimensionY-muuttujien sisältämiä pikselikokoja sekä kuvan 19 storyImageMaterial-muuttujaa. Näin varmistetaan, että kuva piirretään ruudulle samassa koossa, jossa se tallennettiin ohjelmaan.

Ennen 3D -objektin lisäystä silmukassa olivat tekstuurin luonnit ankkureille, muistiavaimille sekä normaaleille tarinoille. Koska kaikilla näillä tarinoilla on samat tekstuurit (ankkureilla omansa, muistiavaimilla omansa ja tarinoilla omansa), voidaan tekstuuri jo ladata aikaisemmin ohjelmassa, jolloin kaikki tarinoiden tekstuurit ovat jo valmiiksi ladattu, kun ohjelma käynnistyy. Näin pystytään välttämään mustaa ruutua tekstuurin kohdalla, joka esiintyy muutaman sekunnin ajan ohjelman käynnistyksen alussa. Samaa konseptia ei pystytty käyttämään 3D-objektien kanssa, koska jokainen kuva on ainutlaatuinen. Tämän takia musta ruutu esiintyy ohjelman käynnistyessä muutaman sekunnin 3D-objektin kohdalla. Tätä ongelmaa ei ehditty ratkaisemaan muiden kiireellisempien asioiden takia.

3D-objekti on tällä hetkellä vielä hiukan harhaanjohtava ilmaus, koska 3D-objektit eivät sinänsä ole vielä kolmeulotteisia, vaan pelkkiä taustattomia kuvia. Tarkoitus on myöhemmin kehittää 3D -objektia lisää niin, että kuvaan lisätään mahdollisuus pyörittää

sitä, joka mahdollistaa objektin näkemisen sekä sivusta että takaa. Kuvien aikajanalle lisäämiselle luotiin pohja, mutta itse kolmiulotteisuuteen ei ehditty puuttua.

3D-objekti helpottaa mutta ei ratkaise kappaleen alussa mainittua tarinoiden samankaltaisuuden ongelmaa. 3D-objektit lisäävät aikajanalle mielenkiinnon kohteita, mutta tavalliset tarinat pysyvät vieläkin samanvärisenä massana. Ongelma tulee olemaan yksi ohjelman merkittävimpiä kehityskohtia. Tulevaisuudessa tarina-ikonit on määrä erotella neljään pääryhmään tarinatyyppien mukaan. Ruutua lähinnä olevista tarinoista avataan pienet esikatselukuvat, jolloin tarinoista tulee saamaan paljon enemmän irti ilman, että niitä tarvitsee kokonaan avata. Tämä tulee tekemään tarinamassan selaamisesta käyttäjälle paljon mielekkäämpää.

3.1.4 Tarinoiden väliset suhteet

3D-objekteja luotaessa tuli eteen ominaisuus, joka löytyi ohjelman vanhasta versiosta, mutta jota ei oltu syystä tai toisesta tuotu uuteen ohjelmaversioon. Ohjelman vanhassa versiossa oli erikoistarinatyyppejä, joista käyttäjä pystyi kertomaan oman tarinansa. Näitä tarinoita kutsutaan nimillä ”ankkuri” ja ”muistiavain”.

Ankkuri-tyyppisten tarinoiden tarkoitus oli kertoa historiallisista tapahtumista. Yksi ankkuri kertoi esimerkiksi talvisodasta, ja toinen vaikka Viron itsenäistymisestä. Muistiavain-tarinat taas sisälsivät erinäisiä kysymyksiä. Molempien tarkoitus oli luoda keskustelua sekä tuoda mieleen uusia tarinoita kyseisistä tapahtumista/kysymyksistä. Tämän vuoksi ohjelmaan oli kehitetty mahdollisuus kertoa oma tarinansa ankkurin tapahtumasta tai muistiavaimen kysymyksestä. Erikoistarinoihin oli myös lisätty nappi, josta pystyi tuomaan esille kaikki erikoistarinasta jo kerrotut tavalliset tarinat. Tällä tavalla saatiin aikaan hyvä tapa lajitella tarinoita aikajananäkymän sisällä.

Uudesta ohjelmaversiosta löytyivät sekä ankkurit että muistiavaimet. Niitä pystyy lisäämään vain järjestelmänvalvoja ja niiden aikajanalla esiintyvät ikonit eroavat selkeästi tavallisista tarinoista. Näistä tarinoista ei kuitenkaan löytynyt yllä mainittua toiminnallisutta tarinoiden linkittämisestä toisiinsa, minkä vuoksi sitä lähdettiin lisäämään ohjelmaan. Sama ominaisuus päätettiin myös lisätä 3D-objekteihin ankkurien ja muistiavaimien lisäksi.

```
@ManyToOne
@JoinColumn(name="PARENT_ID")
private Story parent;

@OneToMany(mappedBy="parent")
private Set<Story> childStories = new HashSet<Story>();
```

Kuva 20. Story-olioon lisätyt uudet tietueet

Tarinoiden välisten suhteiden luonti oli Hibernatessa suhteellisen helppoa. Tämä on kuvattu kuvassa 20. Story-olioon lisättiin parent- ja childStories-tietueet. Kuvan mukaisesti parent on yksittäinen Story-olio ja childStories on kokoelma Story-olioita. Hibernaten sisäinen logiikka on toteutettu niin, että yhdellä parent-muuttujalla voi olla monta child-muuttujaa, joka on merkitty koodissa Hibernate-annotaatioilla @ManyToOne ja @OneToMany. Näiden ansiosta yksi Story-olio voi omistaa monta ”lapsi” Story -oliota, jolloin saadaan aikaan aikaisemmin kappaleessa mainittu suhde tarinoiden välillä. Kuvassa on vielä esitetty, miten tämä kuvataan tietokannassa. Tietokannan Story-tauluun lisättiin PARENT_ID-muuttuja. Tarinan ollessa lapsitarina lisätään PARENT_ID-muuttujan kohdalle lapsen omistavan tarinan ID. Tallennettaessa olioita tietokantaan, tallentuu olioiden välinen looginen yhteys automaattisesti Hibernaten ansiosta.

Yhteys oli helppo toteuttaa ja siinä olikin enemmän kyse hyvien koodaustapojen noudattamisesta, jotta koodi olisi mahdollisimman helppolukuista ja siitä aiheutuisi mahdollisimman vähän virheitä. Ohjelman alkuperäiset tekijät eivät olleet tehneet ankkuri- tai muistiavain-tarinatyypeistä erillisiä tauluja tietokantaan. Erikoistarinat eroavat tietokannassa pelkästään yhdellä tietueella, jonka arvoina ovat joko ”story” (tavallinen tarina), ”anchor” (ankkuritarina) tai trigger (muistiavain). Mielestäni tämä on huono ratkaisu ja tekee tietokannasta sekavan tarinamäärän kasvaessa. Ankkurit ja muistiavaimet olisi hyvä erottaa omiin tauluihinsa, jolloin ne erottuisivat erillisinä kokonaisuuksina tavallisista tarinoista. Tällöin niiden käsittely myös helpottuisi ja ne pystyisi löytämään tietokannasta helpommin. Tämä tuotti vielä lisää ongelmia sen takia, että ohjelmaan oltiin lisäämässä koodia, jossa on viittaus yhdestä saman tyypin oliosta toiseen. Tämä lisää koodin epäselkeyttä ja voisi pahimmassa tapauksessa aiheuttaa ongelmia tietokantahauissa. Tietokantaa ei tässä vaiheessa haluttu kuitenkaan lähteä täysin uudistamaan, koska todettiin, että se olisi vienyt turhan paljon työaikaa sen hyötyihin verrattuna. 3D-objektit toteutettiin lopulta samalla tavalla kuin muut

erikoistarinat, kuitenkin sillä erotuksella että 3D-objektin tiedostosijainti sekä muut tiedot sijaitsevat erillisessä taulussa OneToOne-yhteydessä Story-tauluun selkeyden vuoksi.

```
490     //get child stories for parent
491     @RequestMapping(value = "/story/listChildStories", method = RequestMethod.GET)
492     public ResponseEntity<String> listChildStories(HttpServletRequest request){
493         String json = "false";
494         Story parent = storyService.getStoryById(Long.parseLong(request.getParameter("storyId")));
495         List<Story> listChildren = storyService.getChildStories(parent);
496         if(listChildren != null && listChildren.size() > 0){
497             ObjectMapper mapper = new ObjectMapper();
498             try{
499                 json = mapper.writeValueAsString(listChildren);
500             }catch(Exception e){
501                 e.printStackTrace();
502             }
503         }
504         HttpHeaders responseHeaders = new HttpHeaders();
505         responseHeaders.add("Content-Type", "text/html; charset=utf-8");
506         return new ResponseEntity<String>(json, responseHeaders, HttpStatus.OK);
507     }
```

Kuva 21. Tarinan lapsitarinoiden hakufunktio käsittelijässä

Kuvassa 21 on käsittelijän funktio tarinan lapsitarinoiden haulle tietokannasta sekä niiden palauttamisesta näkymälle. Ensin funktio hakee rivillä 494 tietokannasta lapsia sisältävän tarinan sen ID:n perusteella. Tämän jälkeen haetaan kaikki tämän tarinan lapset ja asetetaan ne listaan. Suodattimina käytetään alkuperäistarinan ID:tä sekä näkyvyystasoa, jonka ansiosta julkisesta tarinasta kirjoitettua yksityistä tarinaa ei lisätä listaan. Lapsitarinaoliolista muutetaan rivillä 499 JSON-muotoiseksi, koska tietoa on helppo käsitellä tässä muodossa näkymän puolella. JSON-objekti palautetaan lopulta rivillä 506 näkymälle, jossa siitä muodostetaan kuvan 1 mukainen pelkistä lapsitarinoista koostuva aikajananäkymä.

3.1.5 Karttanäkymän muutokset

Karttanäkymästä löytyi monia erilaisia bugeja ja vääränlaista toiminnallisuutta ennen jatkokehityksen aloitusta. Tarinoita ilmestyi kartalle yhteensä vain kymmenen, vaikka haluttiin, että kaikki aikajanalla esiintyvät tarinat tulisivat myös karttanäkymään. Tämän lisäksi tarinat tippuivat kartalle samanaikaisesti ja väärässä järjestyksessä, eikä kartan lataamiselle oltu asetettu mitään prioriteettia. Haluttu toiminnallisuus oli tiputtaa aikajanan kymmenen käyttäjän näkökulmaa lähintä tarinaa kauimmaisesta alkaen tietyin aikavälein. Syy tälle oli luoda käyttäjälle ikään kuin polku elämänsä varrella tapahtuneista tarinoista ja niiden tapahtumapaikoista. Loput tarinat tulisivat tippua samanaikaisesti kartan lataamisen jälkeen ennen kymmenen lähimmän tarinan tiputusta.

Tarinoiden yksittäinen tiputtaminen kartalle ratkaistiin käyttäen setTimeout -funktiota, jonka tarkoitus on odottaa funktiolle annettu määrä millisekunteja, ennen kuin se kutsuu tarinoiden tiputusfunktiota. Kartalle asetettiin lisäksi tapahtumankäsittelijä tilesloaded, joka tarkastaa, että kartta on täysin ladattu, ennen kuin tiputtaa sen päälle yhtään tarinaa. Tarinoiden järjestys oli alunperin vääränlainen, koska tarinat haetaan tietokannasta niiden ID:iden mukaisessa järjestyksessä. Tämän vuoksi tarinat tippuivat niiden tapahtumisajankohdan sijaan niiden tekemisajankohdan mukaisesti. Ongelma ratkaistiin järjestämällä tarinalista käänteisesti päivämäärän mukaiseen järjestykseen.

Aikajanan yläosassa esiintyvä ylätunnistepalkki, jossa kerrotaan, mikä aikajana on kyseessä, katosi karttanäkymään siirryttäessä. Tämän takia käyttäjä ei pystynyt tietämään, mitä tarinoita kartalla on menemättä takaisin aikajanalle.

```
45          String publishing = request.getParameter("publishing").trim();
46
47          String headerContent = null;
48          String premium = null;
49          if(publishing.equalsIgnoreCase("mystories")){
50              headerContent = request.getParameter("privateStories").trim();
51          }
52          else if(publishing.equalsIgnoreCase("community")){
53              headerContent = request.getParameter("communityStories").trim();
54          }
55          else if(publishing.equalsIgnoreCase("childStories")){
56              headerContent = request.getParameter("parentStories").trim();
57              premium = request.getParameter("premium").trim();
58          }
59          else{
60              headerContent = "public";
61          }
```

Kuva 22. Ylätunnisteen korjaus

Ongelma ratkaistiin lisäämällä ohjelmaan muuttuja publishing, joka pitää kirjaa siitä, missä näkyvyystasossa liikutaan. Näkyvyystaso asetetaan muuttujaan headerContent publishing-muuttujasta löytyvän tiedon perusteella. Tämän jälkeen headerContent palautetaan karttanäkymälle, joka osaa tiedon perusteella muodostaa oikeanlaisen ylätunnisteen. Tämän ansiosta näkyvyystasoa merkitsevä ylätunniste saatiin toimimaan myös karttanäkymässä. Samalla tehtiin lisäksi muita pieniä muutoksia ylätunnisteisiin, kuten käyttäjän nimen lisääminen yksityisten tarinoiden ylätunnisteeseen ja ylätunnisteen lisääminen myös lapsitarinoihin.

Lisää ongelmia oli kuitenkin luvassa. Karttanäkymästä palattaessa aikajananäkymään, palattiin näkymän alkuun, vuosihaarukkaan 2010-2020. Tämä oli epäjohdonmukaista ja häiritsevää sovelluksen selaamisen kannalta. Tarinamäärän kasvaessa kestäisi aina enemmän ja enemmän aikaa selata aikajanaa takaisin siihen kohtaan, mihin jäi. Käyttäjä haluaisi todennäköisemmin jatkaa aikajanan selaamista kohdasta, josta siirtyi karttanäkymään. Tämä korjattiin lisäämällä kuvassa 22 rivin 45 näkyvän publishing-muuttujan kaltainen arvo ohjelmaan. Uusi muuttuja on kameran Z-positio, joka otetaan talteen ohjelman siirtyessä aikajanalta karttanäkymään. Kun on taas aika siirtyä karttanäkymästä takaisin aikajanalle, siirretään kamera takaisin vanhaan positioonsa. Lopputuloksena on, että aikajana näyttää käyttäjälle täysin samalta kuin silloin, kun käyttäjä alunperin siirtyi aikajananäkymältä karttanäkymään.

Viimeinen ongelma, joka löytyi, tuli esiin selattaessa karttanäkymää päätelaitteella, jossa oli android-käyttöliittymä. Tätä ei oltu todennäköisesti aiemmin testattu, koska vaikka tarinat ja kartta latautuivatkin, niin tarinoita ei pystynyt avaamaan. Tarinoiden otsikot oli oudosti kirjoitettu niiden viereen kartalla, minkä vuoksi kartasta tuli hyvin sekava. Ongelmat korjattiin yksinkertaisesti poistamalla otsikot ja asettamalla tarinoille oikeat tapahtumienkäsittelijät tarinoiden avaamiseen. Tästä huolimatta varsinkin android -kännykällä karttanäkymän selaaminen käytännössä on vieläkin hieman vaikeata, koska puhelimen selain kaatui ainakin yhdellä kännykällä ohjelman raskauden vuoksi usein ja kartan objektit ovat aivan liian pieniä sormella painettaviksi. Näihin ongelmiin ei ehditty puuttumaan muiden kiireellisempien ongelmien vuoksi.

3.1.6 Tarina-ikonien kokojen muutokset

Kuten kuvassa 2 näkyy, karttanäkymään siirryttäessä käyttäjää lähinnä olevat kolme tarinaa esiintyvät erilaisina muista tarinoista. Aikajananäkymässä kaikki normaalit tarinat olivat kuitenkin samankokoisia ja -näköisiä. Tähän haluttiin muutos tuomaan jatkumoa aikajanan ja karttanäkymän välillä. Päädyttiin suurentaman käyttäjää lähinnä olevat kolme tarinaa, ja pienentämään kaikki muut.

```
2064 function enlargeThreeClosest(indexArray){
2065     var chestCounter = 0;
2066     if(typeof indexArray == "undefined" || !(indexArray instanceof Array)){
2067         var indexArray = [];
2068     }
2069     if(indexArray.length == 0){
2070     for(var i = 0; i<storiesList.length;i++){
2071         if(camera.position.z >= 0){
2072             chestCounter = 0;
2073         }
2074         else if(camera.position.z < 0){
2075             for(var k = 0;k<storiesList.length;k++){
2076                 if(storyImageMesh[k].material.visible){
2077                     if(Math.abs(Math.abs(storyImageMesh[k].position.z+parseInt(
2078                             $("#threeChestCameraBuffer").val()))<Math.abs(camera.position.z))){
2079                         chestCounter++;
2080                     }
2081                     else{
2082                         break;
2083 
2084                     }
2085                 }
2086             }
2087 
2088         }
```

Kuva 23. enlargeThreeClosest-funktio

Ensin tuli selvittää, mitkä tarinat ovat käyttäjää lähinnä. Kuvassa 23 näkyy, kuinka funktio enlargeThreeClosest saa selville käyttäjän näkökulmaa lähinnä olevat kolme tarinaa. Tähän käytetään chestCounter-muuttujaa. Rivejen 2075-2088 silmukassa käydään läpi aikajanalla esiintyvää tarinoiden listaa. Kun tarinan positio + mielivaltaisen puskurin arvo on pienempi kuin kameran position arvo, kasvatetaan chestCounter-muuttujan arvoa yhdellä. Tämä arvo ilmaisee sitä, että sitä pienemmän arvon omaavat tarinat eivät ole näkyvissä ruudulla. Taas chestCounter, chestCounter+1 ja chestCounter+2 ovat kolme käyttäjää lähinnä olevaa tarinaa. Esimerkiksi, jos chestCounterin arvo olisi 215, niin tarinalistan tarinat 215, 216 ja 217 suurennettaisiin, ja kaikki muut tarinat pienennettäisiin.

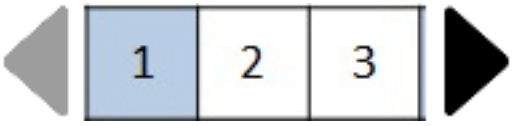
Mielivaltaisen puskurin threeChestCameraBuffer arvon avulla pystytään muuttamaan kohtaa ruudulla, jossa tarinan ikoni muutetaan suureksi. Mitä pienempi arvo on, sitä lähempänä ruutua tarinaikonit muuttuvat suuriksi ja toisin päin. Arvolle on tehty itse ohjelmaan järjestelmänvalvojan konfiguraatiofunktio, jossa sitä voi vaihtaa ohjelman ajon aikaisesti. Samaiseen funktioon lisättiin myös mahdollisuus muuttaa tarinaikoneiden kokoja.

3.1.7 Tarinoiden muokkaus

Seuraavat asiakasvaatimukset kohdistuivat tarinoiden muokkaustoimintoon. Tarinasivuja oli jo mahdollista muokkata: lisäämään/poistamaan tekstiä, vaihtamaan kuvan, uudelleenäänittämään äänitarinan tai uudelleenkuvaamaan videon. Myös tarinan paikan kartalla pystyi siirtämään, muuttamaan sen otsikon ja vaihtamaan tapahtumisajan. Tarinasivuja pystyi myös poistamaan, mutta niiden lisäämiselle ei löytynyt vaihtoehtoa. Työnantajan mukaan tällainen ominaisuus ohjelmassa oli joskus ollut, mutta se on siis jossain kohtaa hävinnyt ohjelman kehityksen aikana.

Koodista löytyikin melko valmis funktio tarinasivujen lisäämiselle, jota kutsuva nappi oli kommentoitu pois näkymästä. Joitakin muutoksia koodin piti tehdä, että se saatiin täysin toimivaksi, minkä vuoksi nappi oli varmaankin poistettu näkymästä. En perehdy kuitenkaan itse muokkausfunktioihin, koska suurin osa koodista on jonkun toisen kirjoittamaa.

Uusien tarinasivujen luonti muokkausfunktiossa toi kuitenkin uusia mutkia matkaan. Uusi tarinasivu luotaisiin aina tarinan viimeiseksi sivuksi. Ohjelmasta ei löytynyt mitään tapaa siirrellä tarinasivuja vapaasti haluamaansa järjestykseen. Tarinan väliin ei siis pystynyt tekemään uusia sivuja, eikä esimerkiksi luomaan otsikkosivua tarinalle jälkikäteen, jos niin haluaisi. Tästä tuli paljon huomautuksia asiakkailta, minkä takia lähdettiin työstämään tapaa muuttaa tarinasivujen paikkoja mahdollisimman luonteenomaisella tavalla.



Kuva 24. Tarinasivujen selauspainikkeet

Kuvassa 24 näkyy luonnos tarinasivu-ikoneista ja niiden vierellä olevista nuolista, joilla niitä voi selata. Sinisellä merkitty sivu on aktiivinen, muut passiivisia. Harmaalla merkitty nuoli merkitsee sitä, että sitä ei pysty painamaan, koska laitimmainen tarina on aktiivinen. Mustaa nuolta voi painaa, jolloin siirrytään tarinasivulle 2. Passiivisia tarinasivuja voi myös painaa, jolloin siirrytään painetulle sivulle, ja ikoni muuttuu aktiiviseksi.

Luontevin tapa tarinasivujen paikkojen vaihdolle oli tarinasivu-ikonien paikkojen vaihto vetämällä niitä hiirellä toistensa yli. Kun otetaan kuva 23 esimerkiksi, toimisi tämä siis painaen nappia 1 ja vetäen se napin 2 yli, jonka jälkeen päästetään irti. Tarinasivu 1 putoaisi paikkaan 2 ja tarinasivu 2 siirtyisi paikalle 1. Tämän jälkeen tarinasivut olisivat järjestyksessä 2, 1, 3.

HTML-elementtien siirtely on suhteellisen helppoa suorittaa käyttäen dragStart-, dragOver- ja drop-tapahtumankäsittelijöitä. Päänvaivaa kuitenkin lisäsivät tarinasivu-ikonien ja nuolien aktivointi/deaktivointi sekä oikeiden tarinasivujen avaaminen painettaessa niitä vastaavia ikoneita.

```
250             var pageNumberArray2 = pageNumberArray1.slice();
251             pageNumberArray1[i] = temp2;
252
253             if(diff > 0){
254                 for(var x = 0; x < (Math.abs(diff)); x++){
255                     pageNumberArray1[i-1-x] = pageNumberArray2[i-x];
256
257                     lists[i-x].getElementsByTagName("a")[0].innerHTML = pageNumberArray2[i-x];
258
259                     if(x == 0)
260                         handleCancelEvent(lists[i+1].getElementsByTagName("span")[0].getElementsByTagName("a")[0]);
261
262                     handleCancelEvent(lists[i-x].getElementsByTagName("span")[0].getElementsByTagName("a")[0]);
263
264
265                     //sets moved pages to be clickable at their new position
266                     lists[i-x].getElementsByTagName("a")[0].onclick = function(){
267                         viewPage(this.innerHTML, pageNumberArray1.indexOf(parseInt(this.innerHTML))+1);
268                     };
269
270
271                 }
272             }
```

Kuva 25. Pätkä handleDrop-funktiota

Kuvassa 25 on osa funktiosta handleDrop, jota tapahtumankäsittelijä drop kutsuu, kun käyttäjä päästää irti tarinasivuikonista. Tärkein muuttuja kuvassa on pageNumberArray1, johon päivitetään koko ajan tarinasivujen tilaa. Rivillä 250 tämä muuttuja kopioidaan toiseksi taulukoksi, nimeltä pageNumberArray2, jonka jälkeen vasta pageNumberArray1 -taulukkoon asetetaan tarinasivun numero paikalle, johon käyttäjä sen pudotti. Jos käytämme aikaisempaa esimerkkiä apuna, niin rivillä 251 muuttuja pageNumberArray2 sisältäisi arvot 1, 2, 3 ja pageNumberArray1 1, 1, 3.

Rivillä 253 näkyvä muuttuja diff kuvaa askelien määrää, jonka tarinasivu on liikkunut. Jos tarinasivu on vaihtanut paikkaa ja arvo on suurempi kuin 0, tiedetään, että tarinaa on siirretty eteenpäin vasemmalta oikealle ja siirrytään seuraavan rivin silmukkaan. Silmukassa pysytään niin kauan, että muuttuja x saa saman arvon kuin tarinasivun liikkumien askelten määrä. Rivillä 255 tehdään tärkeä operaatio, jossa aikaisemman

esimerkin mukaisesti pageNumberArray1 muutetaan oikeaan muotoonsa 2, 1, 3. Tällä rivillä saamme talteen siis tarinasivujen oikean järjestyksen, joka on välttämätön muiden ohjelman tarinasivuihin liittyvien funktioiden kannalta. Tämän jälkeen paikkaa vaihtavien elementtien poistamiseen ja avaamiseen liittyvät tapahtumankäsittelijät vielä päivitetään niin, että ne toimivat uusissa sijainneissaan.

Tämän jälkeen koodattiin samanlainen silmukka tarinasivun taaksepäin siirtämiselle. Silmukka on lähes identtinen sillä muutoksella, että ylitetyt tarinasivuikonit siirtyvät vasemmalta oikealle eivätkä oikealta vasemmalle. Sen jälkeen käydään läpi rekursiivisesti tarinasivulista ja deaktivoidaan kaikki jo aktiiviset tarinasivu-ikonit. Aktiiviseksi asetetaan aikaisemmin aktiivinen ikoni uudessa sijainnissaan. Esimerkiksi jos tarinasivujen uusi järjestys on 2, 1, 3 asetetaan aktiiviseksi toinen ikoni, ja deaktivoidaan ensimmäinen. Itse tarinasivun avaamiseen ei tässä kohtaa puututa, koska tarinasivu-ikonin siirtäminen ei vielä vaikuta siihen. Lopuksi vielä tarkistetaan aktiivisen sivun sijainti, ja deaktivoidaan/aktivoidaan nuolet sen perusteella, missä kohtaa pageNumberArray1-taulukkoa aktiivinen sivu sijaitsee. Esimerkin mukaisesti siirrettäessä aktiivinen sivu 1 väliin 2, 1, 3 mukaisesti, aktivoidaan molemmat nuolet. Jos taas 2 siirretään uudestaan vanhalle paikalleen sivun 1 päälle (lopputulos 1, 2, 3), deaktivoituu nuoli taaksepäin, koska tarinasivuikoni 1 pysyy vieläkin aktiivisena.

Tarinasivujen avaaminen/sulkeminen oli tähän asti toiminut tarinasivu-elementtien HTML:n sisältämien numeroiden avulla. Tähän oli tehtävä muutos, koska tarinasivujen järjestystä muutettaessa muodosta 1, 2, 3 muotoon 2, 1, 3 ja painettaessa keskimmäistä nappia avaisi se silti vanhan järjestyksen mukaisesti sivun 2. Tarinasivujen siirtely rikkoi samalla nuolinappien painamisfunktiot, koska ne toimivat pelkästään lineaarisesti yhdestä kahteen, kahdesta kolmeen jne. pageNumberArray1-taulukko tuli tässä hyödyksi, koska siinä ylläpidetään jatkuvasti tarinasivujen oikeata uutta järjestystä. Annettaessa tämän taulukon järjestys muuttujalle, joka aikaisemmin käsitteli tarinasivujen järjestystä, pystyttiin korjaamaan molemmat ongelmat. Taulukon ansiosta pystytään avaamaan oikea sivu ja aktiviomaan oikea ikoni sivujen silloisesta järjestyksestä huolimatta.

```
2681    //rearrange storypages
2682    if(pageArray != "" && !pageArray.isEmpty()){
2683        String pageNumberArr[] = pageArray.split(",");
2684        List<StoryPage> storyPages = story.getStoryPages();
2685
2686        for(int i = 0; i < storyPages.size(); i++){
2687            storyPages.get(i).setIndex(Arrays.asList(pageNumberArr).indexOf(String.valueOf(i+1))+1);
2688        }
2689
2690        Collections.sort(storyPages, new Comparator<StoryPage>() {
2691        public int compare(StoryPage o1, StoryPage o2) {
2692            Integer a = o1.getIndex();
2693            Integer b = o2.getIndex();
2694            return (a).compareTo(b);
2695        }});
2696        story.setStoryPages(storyPages);
2697    }
```

Kuva 26. Tarinasivujen uudelleenjärjestäminen käsittelijässä

Lopuksi tarinasivut tallennetaan kuvan 26 mukaisella tavalla. Tarinasivujen järjestys lähetetään käsittelijälle tekstinpätkänä, joka muutetaan taulukoksi. Riveillä 2686-2688 tehdään tarinasivujen päivitys. Ohjelma hakee indeksin mukaisesti tarinasivun sijainnin taulukossa ja asettaa sen tarinasivun uudeksi numeroksi. Tällöin, jos taulukon järjestys oli 2, 1, 3, niin tarinasivun 1 numeroksi tulee 2, koska tarinasivu 1 on taulukossa kohdassa 2. Riveillä 2690-2695 päivitetyt tarinasivut järjestetään uudestaan tarinasivun numeron mukaisesti muotoon 1, 2, 3. Tarina päivitetään vielä tarinasivujen uudella järjestyksellä ja lähetetään takaisin näkymälle. Näin tarinasivut ollaan saatu uudelleenjärjestettyä.

3.1.8 Muutokset ”kultaiseen avaimeen”

Ohjelman näkyvyystasoja (julkinen/yksityinen/yhteisöt) ohjaavaa käyttöliittymän elementtiä kutsutaan nimellä ”kultainen avain”. Nimi tulee siitä, että sen ikoni on kultainen avain. Pidettäessä hiirtä kultaisen avaimen yllä tulee esiin lista näkyvyystasoista, joita painamalla pääsee painetun näkyvyystason aikajanalle. Tähän haluttiin muutos, koska ratkaisu oli ulkonäöltään muusta käyttöliittymästä poikkeava, ja se sisälsi bugeja. Joskus painettaessa listan painikkeita hävisi lista vain näkyvistä eikä tapahtunut mitään. Jos käyttäjä oli yli noin kymmenessä yhteisössä, listan yhteisöt -alalista meni yli ruudun alaosasta, minkä takia käyttäjä ei pystynyt avaamaan kaikkia yhteisöjään. Lisäksi ohjelmaan haluttiin lisätä painikkeita, jotka eivät oikein mahtuneet ruudulle, koska aikajana vei niin paljon tilaa.

Kuva 27. Näkymä kultaisen avaimen painamisen jälkeen

Päädyttiin kuvan 27 mukaiseen ratkaisuun. Kultaisesta avaimesta tehtiin painonappi, jota painettaessa ohjelman peittäisi läpinäkyvä tumma näkymä, jonka päälle asetetaan kuvan 27 mukaiset painonapit. ”Yksityiset tarinat” tekstin tilalle tulisi käyttäjän nimi ja yhteisö-nappien tekstejen tilalle tulisi sekä yhteisön nimi että yhteisön ikoni. Jos käyttäjä on enemässä kuin kolmessa yhteisössä lisätään näkymään nuoli oikealle, jotta yhteisöjä pystyisi selaamaan. Luonnollisesti nuoli vasemmalle lisätään myös, kun käyttäjä on painanut kerran nappia oikealle. Julkiset tarinat -painiketta ei lisätty näkymään, koska sille on oma nappinsa käyttöliittymässä.

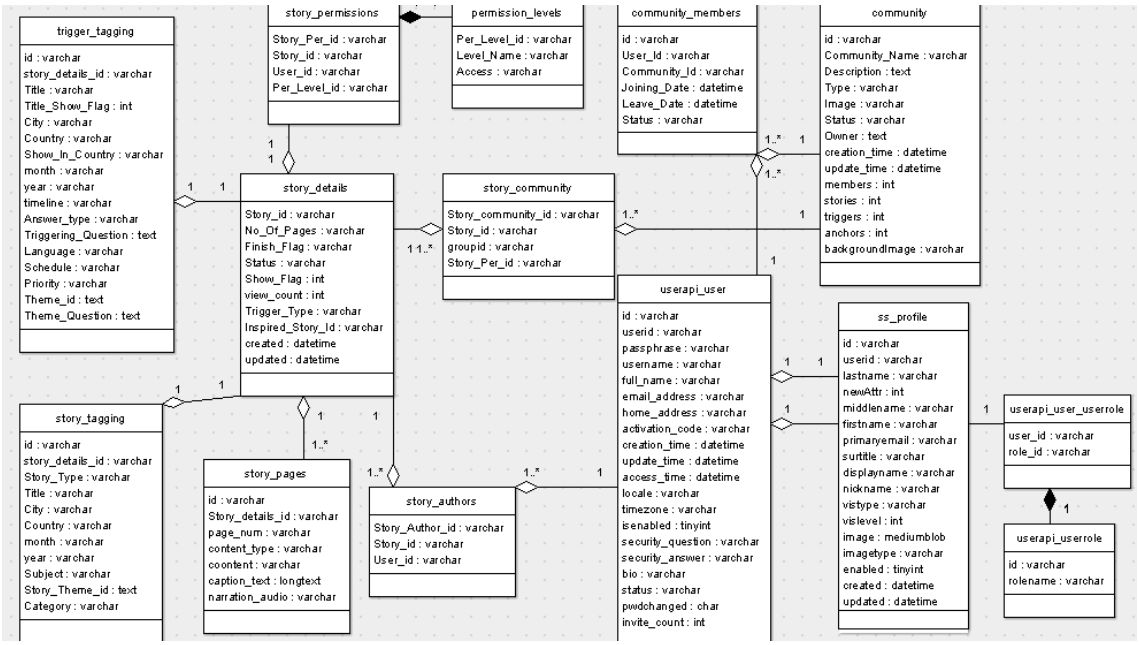
Tämä oli yksi viimeisistä töistäni ohjelman parissa, eikä sitä saatu ulkoasultaan täysin valmiiksi, mutta suuntaa antavan pohjatyö saatiin valmiiksi. Painonapit ovat täysin toiminnallisia. Uusi näkymä mahdollistaa monien uusien painonappien lisäämisen ohjelmaan. Yksi näistä on ohjelman sisäinen verkkokauppa esimerkiksi ohjelman premium-version ostamiseen, mikä on välttämätön lisä ohjelman tulevaisuuden kannalta. Tähän asti ei ole ollut mitään tapaa ostaa ohjelmaan liittyviä asioita ohjelman sisältä, mikä heikentää ohjelman kannattavuutta. Näkymään saatiin lisättyä myös verkkokaupan painonappi sekä tyhjän sivu tätä painettaessa, mutta muuta toiminnallisuutta ei ehditty luomaan. Seuraavien jatkokehittäjien on määrä lähteä luomaan toiminnallisutta verkkokauppaan sekä kehittämään uuden näkymän ulkoasua.

3.2 Käyttöliittymän kehitys ja tietokannat

Jatkokehityksen aikana tuli ilmi merkittävä ongelma käyttöliittymän käytettävyyden kannalta. Kuten luvussa 2.1 mainittiin, suuren tarinamäärän selaaminen osoittautui hyvin vaivalloiseksi ja hitaaksi. Aikajanalle tulisi tehdä merkittäviä muutoksia ongelman ratkaisemiseksi.

Ongelma paljastui yhdistettäessä ohjelman vanhan version tietokantaa uuteen. Ohjelman vanha versio on vieläkin tämän tekstin kirjoitushetkellä internetissä. Ohjelmaa ei kuitenkaan enää kehitetä, ja se haluttaisiin poistaa palvelinmaksujen sekä uuden ohjelmaversion edistämisen kannalta. Vanha versio sisältää kuitenkin tuhansia tarinoita, satoja käyttäjiä ja kymmeniä yhteisöjä, joita ei voi niin vain poistaa, koska osasta sisällöstä on maksettu. Toisaalta sisältöjen tuominen uuteen versioon tekisi ohjelmasta käytetymmän näköisen ja moninkertaistaisi käyttäjäkannan. Vanhan ohjelman käyttäjille pystyttäisiin lähettämään sähköposti uuteen ohjelmaan siirtymisestä, mikä toisi ohjelmaan lisää aktiivisia käyttäjiä.

Ohjelmien tietokannat olivat kuitenkin täysin erilaisia, ja varsinkin vanhasta tietokannasta löytyi paljon outoja piirteitä. Tämän vuoksi tietokantojen yhdistäminen ei sujunut vaikeuksitta ja osoittautui yhdeksi suurimmista tehtävistä harjoitusjaksoni aikana.



Kuva 28. Vanhan ohjelmaversion tietokanta

Ymmärtääksemme tietokantojen yhdistämisen vaikeuden syyn katsotaan molempien tietokantojen rakennetta. Kuvassa 28 on ohjelman vanhan version tietokannan työlleni olennainen osa. Kuten kuvasta näkee jo pelkällä vilkaisulla, on tietokanta aivan turhan monimutkainen. Näiden taulujen lisäksi tietokannasta löytyi vielä ainakin 20 muuta taulua. Suurin osa näistä tauluista oli tyhjiä ja osa nimetty erikoisella tavalla. Nämä asiat tekivät tietokannan ymmärtämisestä hyvin hankalaa.

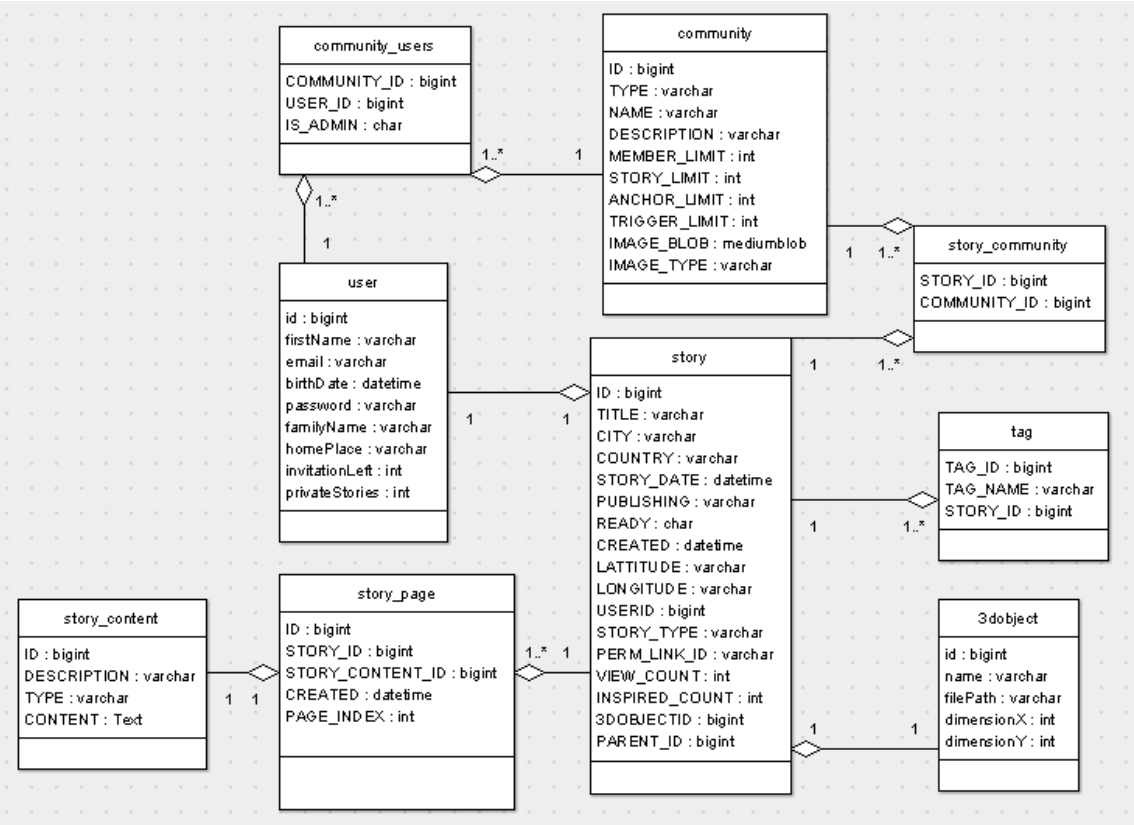
En saanut käyttööni ohjelman lähdekoodia tai tietokannan rakennekuvaa, minkä vuoksi tietokantaa oli vieläkin vaikeampi lähestyä. Tietokannassa ei ole yhtään keskittäistä taulua, ellei sitten story_details -taulua voisi kutsua sellaiseksi. Toisaalta story_details taulu on OneToOne -yhteydessä kolmeen muuhun tauluun, joiden sisällöt voisivat aivan yhtä hyvin sijaita story_details taulussa. Suurin osa sekä story_tagging että trigger_tagging taulujen tiedoista kuuluisi sijaita story_details taulussa. story_permissions ja permission_levels -taulujen olemassaololle ei ole mitään syytä, ja ne tekevät tietokannasta vain monimutkaisemman. ss_profile, userapi_user_userrole ja userapi_userrole tulisi kaikkien sijaita taulussa userapi_user.

Tauluista löytyy paljon aivan turhia tietueita. Suurin osa kuvan tietuieista ei ole edes käytössä. Jostain syystä niitä ei kuitenkaan koskaan poistettu, vaan jätettiin tietokantaan. Tauluissa on myös paljon päällekkäisiä tietueita, kuten userapi_user taulusta löytyvät email_address, username ja ss_profile -taulusta löytyvä primary_email. Jostain syystä tarinan tapahtumispäivämäärä on asetettu tauluun kahtena varchar -muuttujana month ja year. Tietokannan tekijällä on kuitenkin ollut ymmärrys datetime -muuttujista, koska niitä on käytetty joissain tauluissa. Joissain tarinoissa kohdan month paikalla löytyi November, ja toisesta taas 11. Tämä vaikeutti entisestään tietojen tuomista uuteen tietokantaan, jossa oli käytetty datetime -muuttujaa. Yleisestikin vanhassa tietokannassa on käytetty aivan liikaa varchar -muuttujia.

Tarinoiden sisältö on tallennettu story_pages-taulun content-muuttujaan. Kuten uudessa ohjelmaversiossa, tämä muuttuja on pelkkä linkki tarinan tiedostosijaintiin. Tekstitarinoiden tallennuksessa oli kuitenkin ero. Uudessa versiossa tekstitarina tallennetaan suoraan tietokantaan tekstinä, kun taas vanhassa versiossa tekstitarina sijaitsee tekstitiedostossa. Tämä ei sinänsä ollut ongelma, mutta joissain näistä tekstitiedostoista oli pelkästään linkki johonkin toiseen tekstitiedostoon (jossa mahdollisesti linkki vielä toiseen), minkä vuoksi tällaisia tarinoita piti käydä manuaalisesti korjaamassa.

Tietokantojen yhdistämisen jälkeen paljastui vielä yksi mielenkiintoinen tietokannan ominaisuus. Ohjelman käytön aikana poistetut tarinat eivät itse asiassa ikinä poistuneetkaan tietokannasta. Poistetut tarinat sekä yhteisöt esiintyivät tietokannassa pelkästään status-nimisen tietueen arvona. Kaikki poistetut tarinat/yhteisöt/käyttäjät olivatkin jätetty jostain syystä vielä kuormittamaan tietokantaa. Tämä paljastui vasta

verrattaessa vanhan ohjelmaversion aikajanaa uuteen, josta löytyi tarinoita, joita ei vanhassa ollut.



Kuva 29. Uuden ohjelmanversion tietokanta

Kuvassa 29 on ohjelman uuden version tietokannan tärkeimmät osat. Kuvasta on poistettu työhön liittymättömät taulut. Tietokanta on luotu story-taulun ympärille, mikä tekee vanhaan verrattuna tietokannasta selkeämmän ja helposti luettavamman. Ohjelmasta löytyvät konseptit ovat selkeästi jaoteltu omiin osioihinsa. Tietokannan ydinosassa ei ole turhia/käyttämättömiä tietueita eikä tauluja. Tietueet on nimetty suhteellisen selkeästi. Kuvan 29 tietokanta on siis periaatteessa sama kuin kuvan 28, kun siitä poistetaan kaikki turha.

Ainoa huono puoli, mikä uudesta tietokannasta löytyy vanhaan verrattuna, on ID-muuttujien tietotyyppi. Tyyppi on bigint-arvoinen, kun taas vanhassa tietokannassa se on varchar-tyyppinen GUID. GUID:in tarkoitus on olla lähes täysin ainutkertainen 32 heksaluvun sarja. GUID:ia käytettäessä voidaan olla käytännössä varmoja, että vaikka uutta tarinaa luotaessa ID:t eivät koskaan mene päällekkäin jonkun jo olemassa olevan tarinan kanssa.[7.] Samaa ominaisuutta ei ole bigint-muuttujissa, vaan niiden arvot ovat

pelkkiä numeroita nollasta lukuun 2^63-1 (9,223,372,036,854,775,807) asti.[8.] Tämä on varmastikin tarpeeksi suuri luku tallentamaan kaikki mahdolliset tulevaisuuden tarinat maailman loppuun saakka, mutta tietokantoja yhdistettäessä tästä syntyi ongelma. Jos tietotyyppi olisi ollut GUID, olisi tietokantojen yhdistäminen ollut hyvin helppoa. Koska GUID on ainutlaatuinen, ei päällekkäisyyksiä olisi syntynyt. Taulujen suhteet olisivat säilyneet rikkoutumattomina, koska tiedon järjestyksellä ei olisi ollut väliä. Näin ei kuitenkaan ollut, vaan tarinaa lisättäessä suoraan story-tauluun, tulisi sen ID:ksi taulun viimeistä seuraava ID:n luku. Samalla lisättäessä kyseisen tarinan tarinasivu story_page-tauluun, tulisi taas tämän tarinan ID:ksi joku toinen luku. Koska lukuja ei pysty ennalta arvaamaan, ei ole mahdollista säilyttää taulujen välisiä suhteita, minkä takia tarinan ja sen tarinasivun välinen suhde menisi rikki.

ID:iden arvojen muuttaminen tässä kohtaa GUID-tyyppisiksi olisi tuottanut niin merkittäviä muutoksia koodiin, että sitä ei lähdetty tekemään. Pitkän pähkäilyn jälkeen keksittiin kuitenkin tapa yhdistää taulut käyttäen välitauluja, joihin on lisätty uuden numeerisen id:n lisäksi vanha GUID, jolla suhteet pystyttiin pitämään kasassa. Tällöin luodussa story_page2-taulussa olisi vanhasta tietokannan taulusta löytyvän tarinan GUID, jota käyttäen tarina pystytään liittämään omaan tarinasivuunsa muutaman lisäoperaation avulla.

```
1  INSERT INTO epooq.story_page(STORY_ID, STORY_CONTENT_ID, CREATED, PAGE_INDEX)
2  SELECT a.id, b.id, NOW(), c.page_num
3  FROM epooq.story2 a
4  JOIN epooq.story_content2 b
5  JOIN epooqdb.story_pages c
6  JOIN epooq.story d
7  ON a.STORYDETAILSID = b.STORYDESCRIPTIONID AND c.id = b.OLDID
8  AND d.id = a.id
```

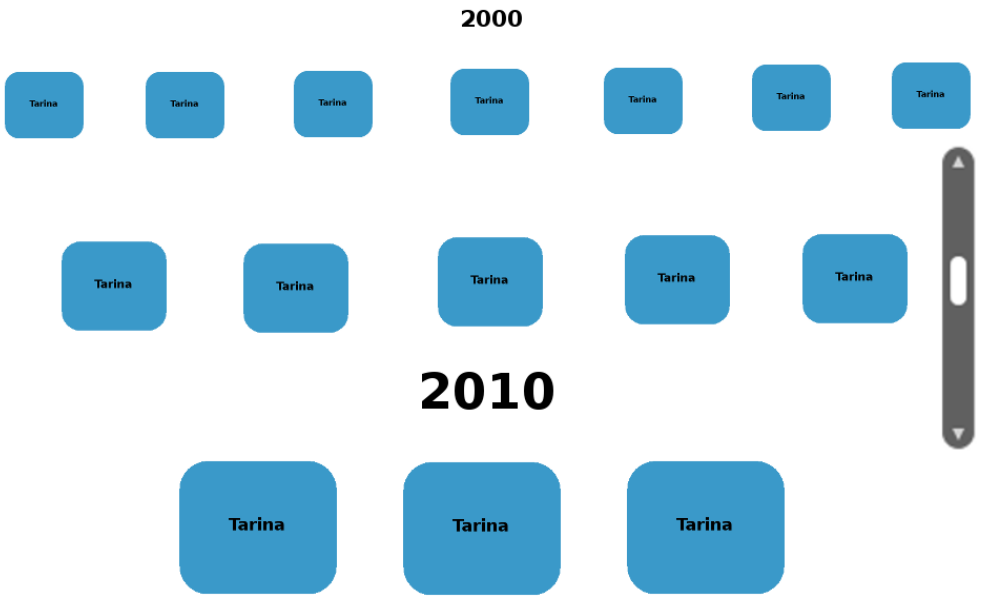
Kuva 30. Vanhan tietokannan tarinasivun tuominen uuteen

Kuvassa 30 on SQL-skripti, jonka avulla voidaan tuoda vanhasta tietokannasta kaikki tarinasivut uuteen. Samalla säilytetään taulun story_page suhteet tauluihin story ja story_content. Operaatiota ennen tietokantaan on jo tuotu kaikki vanhan kannan story- ja story_content-elementit. Näillä ei ole kuitenkaan vielä yhteyttä toisiinsa, minkä vuoksi ohjelman aikajanalla näkyvät tarinat eivät vielä sisällä mitään. Kuvan 30 skriptissä tarinat yhdistetään sisältöihinsä käyttäen kahta välitaulua story2 ja story_content2, uuden tietokannan story-taulua sekä vanhan tietokannan story_pages-taulua. Välitaulujen sisältämät STORYDETAILSID-, STORYDESCRIPTIONID- ja OLDID-muuttujat ovat

vanhasta tietokannasta haetut GUID:it. Näitä vertaamalla saadaan yhdistettyä tietokannan sisäinen logiikka toisiinsa, ja vietyä se rivillä 1 uuden tietokannan tauluun story_page. Lopputuloksena tarinat on saatu yhdistettyä sisältöihinsä sekä tarinasivuihinsa ja kaikki vanhan tietokannan tarinat tuotua uuteen. Samalla periaatteella käytiin läpi myös yhteisöt sekä käyttäjät.

Uudesta tietokannasta luotiin vielä kaiken kokoava skripti, jonka avulla saataisiin yhdellä kertaa tuotua vanhat tarinat/käyttäjät/yhteisöt myös julkisen palvelimen tietokantaan. Suuren tarinamäärän aiheuttamien ongelmien vuoksi tietokantojen yhdistämistä julkisella palvelimella ei kuitenkaan saatu suoritettua. Ennen tätä operaatiota tulisi selvittää ohjelman aikajanan ongelmat.

Aikajananäkymän ongelmana oli siis suuren tarinamäärän selaamisen hitaus, koska tarinat kulkivat vain kolmessa rivissä. Käyttäjä ei myöskään saanut edes likimääräistä kuvaa tarinoiden määrästä aikajanalla, koska kaikki tarinat kulkivat samalla linjalla korkeusakselilla.



Kuva 31. Alustava muutos aikajanalle

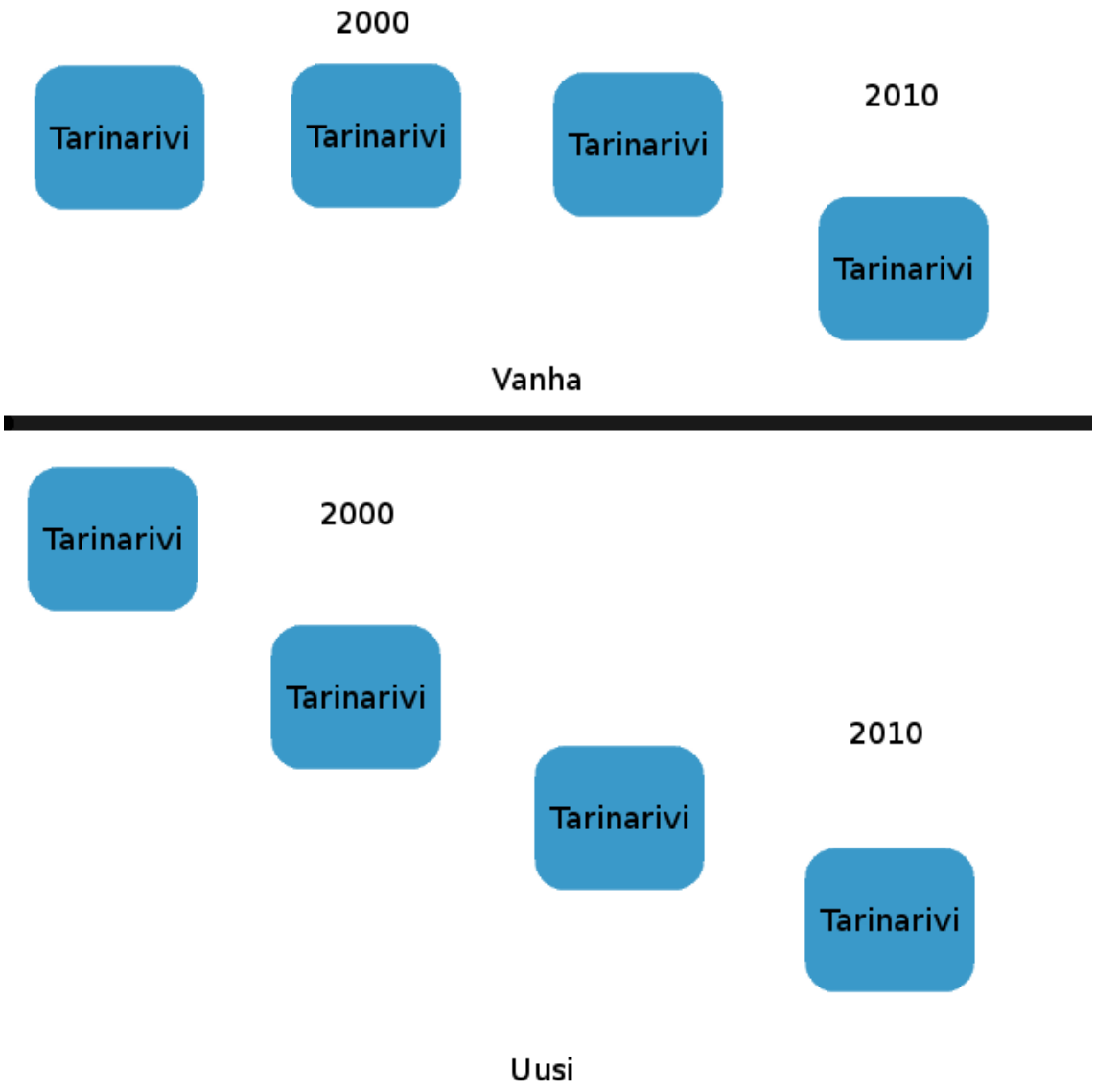
Aikajanalle testattiin kuvan 31 mukaista muutosta. Tarkoituksena oli levittää tarinoita leveyssuunnassa sen mukaan, miten paljon tarinoita on kerrottu kahden vuosikymmenen välisenä aikana. Jos vuosikymmenten 2010 ja 2000 välillä on kerrottu 10 tarinaa, niin tarinat voitaisiin asettaa kulkemaan kolmessa rivissä kuten ennen. Jos taas 1940 ja 1950 välisenä aikana tarinoita olisi kerrottu 100, tarinarivien määräksi voitaisiin asettaa vaikka 10. Tällöin aikajanan selaaminen selkeästi nopeutuisi.

Ratkaisu lievittää aikajanan ongelmaa, mutta ei poista sitä. Jos tarinoita olisi vuosikymmenten välillä vaikka 1000, ja vaikka tarinarivejä olisikin 10, niin vaikka tarinamassan selaaminen olisikin nopeampaa kuin jos rívejä olisi 3, olisi se silti hidasta. Toinen ongelma oli, että lähestyttäessä yli kolmerivistä tarinariviä, reunalla olevat tarinat hävisivät näytöstä tarina-ikonin suuruuden takia. Tähän pyrittiin luomaan ratkaisu siirtämällä tarinoita näytön eteen käyttäjän selaamistavan mukaisesti. Jos käyttäjä liikkuu eteenpäin janalla, siirrettäisiin lähimmän tarinarivin kolme käyttäjän kuvakulmaa lähinnä olevaa tarinaa kameran eteen. Käyttäjän selatessa eteenpäin tai taaksepäin kolme kameran edessä olevaa tarinaa muuttuisivat sen mukaan, mihin suuntaan käyttäjä janaa selaa. Jos käyttäjä selaisi taaksepäin, tuotaisiin kameran ”takaa” uusia tarinoita kolmen

aikaisemman tarinan tilalle. Vanhat kolme tarinaa taas vietäisiin takaisin omille paikoilleen tarinariville. Aikajana näyttäisi siis vähän kuin kuvassa 31 näkyy.

Tämä ei varsinaisesti ratkaissut muuta kuin tarinarivien levityksestä syntyneen ongelman. Vaikka näiden muutosten jälkeen aikajanaa olikin nopeampi selata, mielestäni ne vaikuttivat aikajanan selaamismukavuuteen. Pitkät tarinarivit olivat ulkonäöllisesti rumia, koska ne vieläkin liikkuivat yli kameran reunusten. Ratkaisu vaikeutti myös jo olemassa olevaa ongelmaa tarinoiden erottamisesta, koska pitkät tarinarivit aiheuttivat tunteen siitä, että tarinat olivat pelkkää massaa.

Samalla yritettiin ratkaista ongelmaa tarinoiden määrän näkymisestä käyttäjälle.



Kuva 32. Vanha aikajananäkymä sekä luonnos uudesta kuvattuna sivusuunnasta

Kuvassa 32 on kuvattu vanha sekä uusi aikajananäkymä x-akselia pitkin kuvattuna. Siniset palkit kuvaavat tarinoiden rivejä, jotka on kuvattu kuvassa 31 z-akselia eli ohjelman näkymää pitkin. Vanhassa versiossa kolme ensimmäistä tarinaa on kameran alaosassa, ja kaikki tämän jälkeiset tarinarivit tulevat tästä hiukan koholla samassa rivissä. Kuvakulman takia toisen tarinarivin takana olevat tarinat ovat vaikeasti näkyvissä, eikä käyttäjä saa kuvaa tarinoiden kokonaismäärästä.

Uudessa versiossa tarinarivit kulkevat yläviistoon tietyin välimatkoin. Siirryttäessä eteenpäin aikajanalla, kameraa lähin tarinarivi tippuu kuvan 32 mukaiseen kohtaan. Liikuttaessa aikajanaa taaksepäin nousevat kamerasta loittonevat tarinarivit taas ylöspäin. Tämä aiheuttaa sen, että käyttäjä saa selvemmän kuvan aikajanan tarinoiden

määrästä. Toisin kuin tarinoiden levitys leveyssuunnassa, tämä ominaisuus osoittautui käytännölliseksi, eikä aiheuttanut ongelmia käyttöliittymässä.

Tekemäni työ aikajanan parissa ei päätynyt julkiselle palvelimelle, koska aikani sen kehityksen parissa ehti loppua, ennen kuin yrityksen teettämä lopullisen käyttöliittymän suunnitelma saatiin valmiiksi. Työtä on kuitenkin käytetty testaamaan uusia aikajanaan liittyviä ominaisuuksia ja ominaisuuksia tullaan kehittämään tulevaisuuden jatkokehittäjen voimin.

4 Yhteenveto

Tekemäni työ sovelluksen parissa oli tuottoisaa. Suurin osa tekemistäni muutoksista päätyi suoraan julkiselle palvelimelle, ja esimerkiksi 3D-objekteja on käytetty jopa sovelluksen markkinoinnissa. Tässä työssä mainittujen lisäksi ohjelmassa oli lukuisia muitakin bugeja, jotka satiin korjattua. Esimerkkeinä näistä on kuvatarinoiden kuvien esikatselussa esiintyneet virheet, sekä kuvatarinaa tehtäessä äänityksen mahdollisuuden puute. Näistä en kuitenkaan kertonut sen tarkemmin tässä työssä, koska korjauksissa suurin osa ajasta meni virheen löytämiseen koodin tuottamisen sijasta. Yhden lisäämäni ominaisuuden tarkempi selitys jäi myös työstä pois, vaikka ominaisuus onkin pintapuolisesti mainittu työssä muutamaan kertaan. Ominaisuus on uusi järjestelmänvalvojan näkymä, josta tämä voi vaihtaa aikajananäkymän selausnopeutta, tarinaikoneiden kokoja ja taustakuvaa ynnä muuta ajon aikaisesti. Ominaisuus oli suhteellisen mekaaninen, joten päätin jättää sen tarkemman selityksen työstä pois.

Opin verkkosovelluksista paljon uutta. Vaikka ohjelmien back-end-puoli oli tullessani yritykseen jo suhteellisen selkeää, oli front-end taas kaikkea muuta kuin sitä. Jatkokehityksen aikana opin paljon front-end kielistä, ja näkymän yhdistämisestä palvelinpuoleen. Myös ongelmanratkaisutaitoni sekä muiden koodin lukeminen kehittyivät, varsinkin koska koodista ei ollut mitään dokumentaatiota ja sain selvittää sen toiminnan itse. Jatkokehityksen aikana sain perehtyä myös muihin koodista hiukan erossa oleviin asioihin, kuten verkkopalvelimiin, niiden luontiin ja ylläpitoon, sekä SSL-avaimen hankkimiseen ja käyttämiseen. Siirsimme vanhan maksullisen testipalvelimen uuteen maksuttomaan pilveen, jonka ansiosta saimme vähennettyä paljon palvelimista koituneita kuluja. Ohjelman sivusto tuli suojata käyttäen SSL-avainta, koska Chrome-

selain ei hyväksy tietokoneen mikrofonin eikä web-kameran käyttöä ilman, että sivusto on suojattu. Sain luotua itsekirjatun avaimen testipalvelimelle, joka mahdollisti HTTPS -protokollan käytön tällä palvelimella. Työnantaja ei vielä kuitenkaan päättänyt virallisen SSL-avaimen ostamisesta/käyttämisestä julkisella palvelimella, minkä takia julkiselle palvelimelle ei omana työaikanani saatu SSL -suojausta. Tämän vuoksi videoiden kuvaamisessa sekä äänityksessä esiintyy vieläki ongelmia Chrome-selainta käytettäessä.

Näillä näkymin tulevaisuudessa pyritään kehittämään työssä mainittua aikajanan käyttöliittymää. Siinä esiintyvien ongelmien korjaaminen on välttämätöntä ohjelman kaupallistamisen vuoksi.

Kiitän harjoittelupaikkani esimiestä saamastani antoisasta tehtävästä ja tuesta. Toivon, että tästä työstä on hyötyä tulevaisuudessa yritykselle ja ohjelman kanssa työskenteleville henkilöille, koska siihen on kirjattu asioita, joita on lähdetty työstämään mutta ei saatu valmiiksi. Työ toimii myös dokumentaationa ohjelman jatkokehityksen aikana tehdyistä merkittävimmistä muutoksista.

Lähteet

1 What is jQuery? Verkkodokumentti. https://jquery.com/. Luettu 29.9.2016.

2 JavaScript 3D library. Verkkodokumentti. https://github.com/mrdoob/three.js/. Luettu 29.9.2016.

3 Introduction to Spring Web MVC framework. Verkkodokumentti. http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html. Luettu 29.9.2016.

4 JSP – Standard Tag Library (JSTL) Tutorial. Verkkodokumentti. http://www.tutorialspoint.com/jsp/jsp_standard_tag_library.htm. Luettu 3.10.2016.

5 Verkkodokumentti. http://red5.org/. Luettu 29.9.2016

6 The DispatcherServlet. Verkkodokumentti. https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm. Luettu 10.10.2016.

7 GUID Structure. Verkkodokumentti. https://msdn.microsoft.com/en-us/library/aa373931(VS.85).aspx. Luettu 1.11.2016.

8 int, bigint, smallint, and tinyint (Transact-SQL). Verkkodokumentti. https://msdn.microsoft.com/fi-fi/library/ms187745.aspx. Luettu 1.11.2016.