Risto Salama

# Wicket

## Web-sovelluskehys

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tietotekniikka
8.5.2012

Metropolia

**Tiivistelmä**

| | |
|---|---|
| Tekijä(t)<br>Otsikko<br><br>Sivumäärä<br>Aika | Risto Salama<br>Wicket – Web-sovelluskehys<br><br>38 sivua + 2 liitettä<br>8.5.2012 |
| Tutkinto | Insinööri (AMK) |
| Koulutusohjelma | Tietotekniikka |
| Suuntautumisvaihtoehto | Ohjelmistosuunnittelu |
| Ohjaaja(t) | Lehtori Miikka Mäki-Uuro<br>Lehtori Jorma Räty |


Apache Wicket, josta käytetään yleisemmin lyhennettyä muotoa, on web-sovelluskehys Javalle. Wicket on komponenttiorientoitunut ja kevyt avoimen lähdekoodin kehys. Tämän opinnäytetyön pääasiallisena tarkoituksena on esitellä Wicket lukijalle. Wicket käydään työn puutteissa läpi kahdella tavalla: teoriaosuudella ja esittelemällä Wicketillä toteutettu websovellus.

Wicketistä käydään läpi taustaa, rakennetta, erilaisia komponentteja, malleja, Ajax-toiminnallisuutta sekä testaamista. Opinnäytetyössä käytetään tukena paljon havainnollistavia koodiesimerkkejä.

Tehty sovellus on web-pohjainen musiikkitoistin soittolistaominaisuudella, jonka toiminnallisuudet käydään tässä opinnäytetyössä läpi. Sovelluksessa on Wicketin lisäksi käytetty myös muita tekniikoita, kuten jQueryä, jQueyry Uita ja HTML5:tä. Nämä tekniikat esitellään myös tässä opinnäytetyössä.



| | |
|---|---|
| Avainsanat | Wicket, jQuery, HTML5, soitin, soittolista |

Metropolia

**Abstract**

| | |
|---|---|
| Author(s)<br>Title<br><br>Number of Pages<br>Date | Risto Salama<br>Wicket – web application framework<br><br>38 pages + 2 appendices<br>8 May 2012 |
| Degree | Bachelor of Engineering |
| Degree Programme | Information Technology |
| Specialisation option | Software Design |
| Instructor(s) | Miikka Mäki-Uuro, Senior Lecturer<br>Jorma Räty, Senior Lecturer |


Apache Wicket, often referred to as Wicket, is a web application framework for Java. Wicket is a component orientated and lightweight open source framework. The main purpose of this thesis was to give a useful introduction to the world of Wicket for the reader. That was accomplished by doing two things presented in the thesis: creating a theoretical framework and presenting a web application developed with Wicket.

The introduction first presents the background, structure, different components, models and Ajax. Especially the uses of the components are portrayed by using code examples with explanations.

The application is a web-based music player with playlist functionality. The study presents the functionality of the application rather thoroughly. The study also presents other technologies used in making the application, namely HTML5, jQuery and jQuery UI.



| | |
|---|---|
| Keywords | Wicket, jQuery, HTML5, audio player, playlist |

**Sisällys**

# 1 Johdanto

Apache Wicket (käytetään yleisemmin lyhennettyä muotoa Wicket) on web-sovelluskehys Javalle. Käytännössä Wicketin avulla pystytään eriyttämään kaikki toiminnallisuus HTML:n, tai jonkin muun merkintäkielen, tiedostoista varsinaisiin Java-luokkiin. Wickettiä käytettäessä ei tarvita servletti- tai JSP-tyylisiä ratkaisuja, eikä se pohjaudu XML-tiedostoihin kuten moni muu web-sovelluskehys.

Tässä opinnäytetyössä suunnitellaan ja toteutetaan web-pohjainen musiikkisoitin soittolistatoiminnallisuudella, jonka työnimi on Thunder List. Lisäksi päättötyön tarkoituksena on esitellä lyhyesti Wicket ja muita varsinaisessa kehitystyössä käytettyjä tekniikoita kuten HTML5:tä ja jQueryä. HTML5 tarjoaa audio-elementin, mitä käytetään soittimessa. jQuery tarjoaa ratkaisuja ja mekanismeja soittimen käsittelyyn.

Wicketistä kerrotaan sen taustaa, ominaisuuksia ja yleisiä Wicketin komponentteja. Sovelluksen tekemisessä käytetyt muut tekniikat käydään myös läpi, sekä kerrotaan niiden taustasta ja ominaisuuksista. Viimeisessä osiossa käydään läpi itse sovellus. Sovelluksesta käydään läpi ensiksi arkkitehtuuri, sitten soitinosa, ja lopuksi soittolistatoiminnallisuudet ja editointi. Tekniikoiden ja niiden toimintojen läpikäynnissä sekä sovellusosiossa käytetään paljon havainnollistavia koodiesimerkkejä.

Sovelluksen soittimessa vaaditut toiminnot ovat soitettavan kappaleen toisto, pysäytys, tauko ja kohdan siirto sekä kappaleiden vaihto. Soittimen pitää luontevasti pystyä toistamaan soittolistassa olevia kappaleita. Soittolistoja pitää pystyä vapaasti tekemään, muokkaamaan ja poistamaan. Kappaleita pitää myös pystyä lisäämään sovellukseen.

Sovellusta ei ole suunniteltu laajempaan käyttöön, joten siinä ei tueta useampaa eri selainversiota ja JavaScript pitää olla sallittu selaimessa. Sovellus on suunniteltu tekijän henkilökohtaiseen käyttöön, joten useamman eri käyttäjän mahdollisuutta ei tueta. Käyttöliittymästä halutaan kuitenkin mahdollisimman virtaviivainen, tyylikäs ja helppokäyttöinen.

# 2 Apache Wicket

## 2.1 Wicket lyhyesti

Wicket on web-sovelluskehys Javalle. Web-sovelluskehyksen pääasiallisena tarkoituksena on yhdistää kuilu tilattoman HTTP:n ja tilallisen palvelinpuolen ohjelmoinnin välille [1, s. 5]. Javalla ohjelmoitaessa ei tarvitse miettiä, kuinka Javan virtuaalikone hallitsee olioiden instansseja ja niiden jäsenmuuttujia. Sen sijaan nettisivustoja tehdessä HTTP:n päälle täytyy manuaalisesti hallita istunnon tila ja käyttöliittymä. Wicketissä palvelinpuolen tila hallitaan automaattisesti, ja se assosioidaan komponentteihin [2]. Jokainen palvelinpuolen sivu pitää sisällään hierarkian tilallisia komponentteja, Wicket ylläpitää karttaa näistä sivuista jokaisen käyttäjän istunnossa. Yksi tämän sivukartan tarkoituksista on mahdollistaa sovelluskehykselle tapa piilottaa yksityiskohdat, miten sovelluksen komponentteihin pääsee käsiksi. Kehittäjä voi keskittyä tuttuihin Java-olioihin ja Wicket hallitsee URL:t, istuntojen tunnisteet ja GET/POST-pyynnöt.

Wicketin kanssa tullaan toimeen pelkällä Java- ja HTML-koodilla. Toisin kuin monissa muissa web-sovelluskehyksissä, HTML-tiedostoissa ei tarvitse ollenkaan esiintyä palvelinpuolen koodia, ennen tai jälkeen niiden piirtämisen selaimeen. Ainoa xml-tiedosto, jota Wicketissä tarvitaan, on web.xml. Wicket rakentuu pitkälti POJO- (Plain Old Java Object) ja MVC (Model View Controller) –ajattelun varaan. Wicketissä on myös tuki Ajaxille (Asynchronous Javascript and XML).

Wicketin on alun perin kirjoittanut Jonathan Locke vuonna 2004 huhtikuussa. Wicketin ensimmäisen version (1.0) julkaisu tapahtui vuonna 2005 [3]. Wickettiä tehdessä päätavoitteina ovat olleet käytön helppous, uudelleenkäytettävyys, tungettelemattomuus (suomennettu englanninkielisestä termistä non-intrusive), turvallisuus, tehokkuus ja skaalautuvuus.

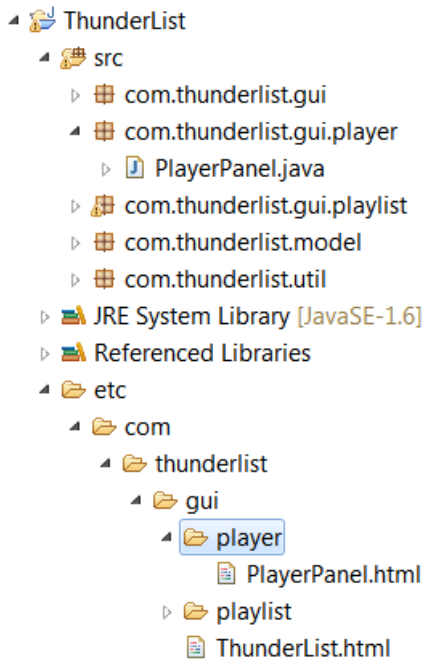
Käytön helppoudessa on otettu tavoitteeksi muun muassa konventioiden ja komponenttien johdonmukaisuus, minimaalinen riippuvuus erikoistyökaluihin, POJO-keskeisyys, kaiken ohjelmointikoodin kirjoittaminen Javalla sekä XML-konfiguraatioiden vähentäminen. Uudelleenkäytettävyydessä Wicketillä tehtyjen luokkien pitäisi olla täy-

sin uudelleenkäytettäviä ja niiden levitys pitäisi tapahtua helposti tavallisissa JAR-tiedostoissa.

Tungettelemattomuudella tarkoitetaan sitä, ettei HTML tai muu merkintäkieli täyttyisi ohjelmointisemantiikoilla, mikä on saavutettu sillä, että merkintäkieleen on upotettu vain yksi tarpeellinen merkintä (”wicket:id”). Tämä helpottaa grafiikkasuunnittelijoita vahingossa poistamasta sovelluskehyksen merkintöjä, mutta antaa myös helpon ja nopean tavan kehittäjälle lisätä merkinnät uudelleen.

## 2.2 Wicketin rakenne

Wicket-sovellus koostuu yleensä applikaatiosta, sivuista ja muista komponenteista. Jokaiselle komponentille voi olla oma HTML-pohjansa (”markup”), jonka polun etc-kansiossa pitää vastata komponentin pakettirakennetta. Komponentit voi karkeasti jakaa säiliöihin (sivut, paneelit), jotka sisältävät muita komponentteja, sekä toiminnallisiin, joita ovat esimerkiksi tekstikenttä tai linkki.




Kuvio 1. Hakemistopolku PlayerPanel.html-tiedostoon vastaa saman luokan pakettirakennetta.

Wicketissä on yllämainittujen lisäksi lukuisia muita luokkia kuten lisätoiminnallisuuksia (IBehavior-rajapinnan toteuttavia luokkia), kuuntelijoita (kuten IChangeListener-

rajapintaa toteuttavatt luokat) ja erilaisia validointi-luokkia (esim. CreditCardValidator-luokka), jotka voidaan liittää erinäisiin komponentteihin lisäämään niiden toiminnallisuutta.

### 2.2.1 Applikaatio

Applikaatio on Wicketissä sovelluksen perusluokka. Tehdessä uutta Wicket-sovellusta kannattaa kustomoitu applikaatio luokka periä abstraktista WebApplication-luokasta, joka vuorostaan on perinyt applikaatio-luokan.

Applikaatiossa voidaan määritellä siihen suoraan liittyvät sivut ja niiden polut. Applikaatio myös käsittelee kaikki istunnot ja istuntoihin liittyvät kutsut. Wicketissä voidaan kätevästi käyttää kustomoituja istuntoja, esimerkiksi pitämään tietoa käyttäjästä.

Sovelluksen web.xml–tiedostoon asetetaan yksi tai useampi applikaatio:

```
<web-app>
    <display-name>ThunderList</display-name>
    <filter>
        <filter-name>ThunderList</filter-name>
        <filter-class>org.apache.wicket.protocol.http.WicketFilter</filter-
class>
        <init-param>
          <param-name>applicationClassName</param-name>
          <param-value>com.thunderlist.gui.ThunderListApplication</param-
value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>ThunderList</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

Yksinkertaisimmillaan applikaation ei tarvitse sisältää mitään muuta kustomoitua koodia kuin kotisivun haun:

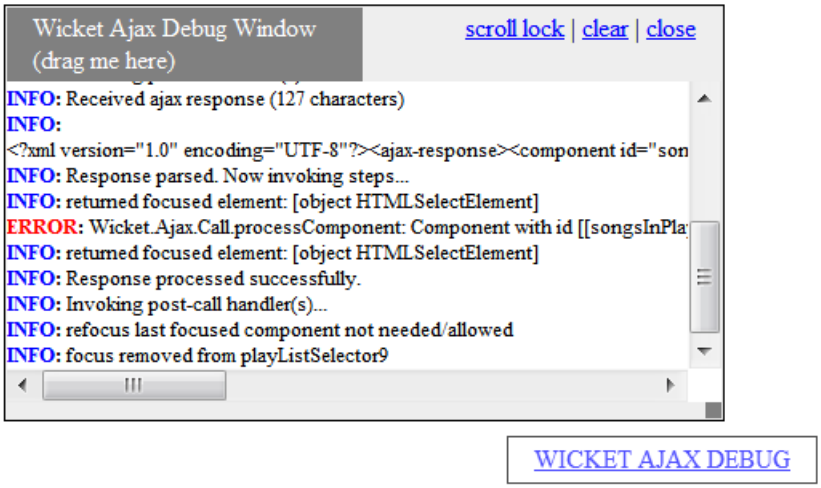
```
public class ThunderListApplication extends WebApplication {
            public ThunderListApplication() {
            }

            @Override
            public Class<ThunderList> getHomePage() {
```

```
                return ThunderList.class;
            }
}
```

Applikaatiolla on kaksi eri moodia, kehitys ja tuotanto, joista oletuksena päällä on kehitysmoodi. Kehitysmoodi tarjoaa kehittäjälle mahdollisimman paljon toimintoja ja tietoa sovelluksen kehityksessä, kun taas tuotantomoodissa suorituskyky ja turvallisuus ovat ensisijaisia.

Kehitysmoodissa tarkkaillaan muutoksia resursseihin ja ne ladataan uudelleen käynnissä olevaan sovellukseen tallennuksen yhteydessä. Tämä toimii esimerkiksi Eclipsen ja siitä ajettavan Jetty-palvelinohjelman kanssa (”Run Jetty Run” -lisätoiminnallisuus). Wicket tarkistaa, että piirretäänkö kaikki komponentit, mitkä on lisätty sivun hierarkiaan. Poikkeukset näytetään piirretyllä nettisivulla samoin kuten Wicketin käyttämät merkinnätkin. Wicket ei myöskään poista kommentteja JavaScript-tiedostoista, eikä Wicket kompressoi JavaScriptiä tai CSS-määritteitä. Jos sivulla on Ajax-komponentteja, näytetään selaimessa Wicketin tarjoama Ajax-debuggeri.




Kuvio 2. Wicketin Ajax-debuggeri toiminnassa.

Wicket myös muistuttaa käynnistettäessä sovellusta, että se on kehitysmoodissa. Tuotantomoodissa esitellyt kehittäjämoodin toiminnot on kytketty pois päältä tehokkuuden takia siksi, että sovelluksen yksityiskohdat eivät näkyisi loppukäyttäjälle.

Tuotantomoodin käynnistystä varten voidaan esimerkiksi asettaa järjestelmäominaisuus Java-virtuaalikoneelle (java -Dwicket.configuration = deployment). Asetus onnistuu myös web.xml tiedostossa servletti/suodattimen-initialisoinnissa parametrina. Muita mahdollisuuksia tuotantomoodin asettamiselle ovat kontekstin initialisoinnissa tai suoraan sovelluksen WebApplication-luokkaan asettamalla hakemalla tieto esim. sovelluksen tietokannasta tai erillisestä tiedostosta.

### 2.2.2 Sivut

Wicketin sivu perii MarkupContainer-luokan, mikä mahdollistaa sen, että sivu voi sisältää komponenttihierarkian ja merkkikielipohjan jollain merkkikielellä (esim. HTML). Kun sivu on rakennettu, se lisätään automaattisesti istunnossa olevaan sivukarttaan (PageMap-luokka), jolloin sivukartta lisää sivulle tunnisteen. Sivukartta vastaa suurin piirtein selainikkunaa, se kapseloi listan sivuja, joihin voi päästä kyseisestä ikkunasta. Kun ponnahdusikkuna luodaan, sille tehdään uusi sivukartta. Sivusta voi tehdä halutessaan kirjamerkittävän, johon riittää tyhjä alustusmetodi. On suositeltua, etteivät uudet sivut peri suoraan sivu-luokkaa vaan perisivät sivu-luokan perivän websivun.

Wicketin websivu perii sivu-luokan ja vastaa yksittäistä HTML-sivua. Sivua voidaan myös verrata Swingin kehykseen, sillä ne molemmat ovat ylemmän tason säiliöitä, joihin sisältyvät muut komponentit. Tässä on esitetty kustomoidun sivun konstruktori:

```
public ThunderList() {
            super();
            this.setRenderBodyOnly(true);
            this.add(new Label("thunderList", "Thunder List"));
            this.add(new PlayerPanel("player"));
            this.add(new PlayListPanel("playList", PlayListLoader
                        .getDefaultPlayList()));
}
```

Sivuun lisätään yksinkertainen tekstikomponentti (Label-olio) ja kaksi kustomoitua paneelia. Ensimmäinen komponenttien luonnissa annettu parametri (yllä esim. ”player”) on komponenttien id, jota käytetään HTML-pohjassa:

```
<body>
            <h1><span wicket:id="thunderList">Otsikko</span></h1>
            <div wicket:id="player" />
```

```
            <div wicket:id="playList" />
</body>
```

HTML-pohjaan on ylhäällä laitettu span-elementin sisään teksti ”Otsikko”. Kyseistä tekstiä ei varsinaisesti piirretä sivulle, mutta se toimii kätevänä merkintänä kehittäjälle ja kertoo mitä sivulla pitäisi lukea. Div-elementtien sisältöä ei annetussa esimerkissä näytetä, sillä niille on tehty oma tyylipohjansa. Piirretyssä sivussa, joka nähdään selaimessa, div-elementtien sisältö näkyy.

### 2.2.3 Paneelit

Paneelit ovat myös säiliöitä ja niitä voi lisätä muihin paneeleihin ja jopa rekursiivisesti paneeleihin itseensä. Paneelit ovat myös yleisimpiä kustomoituja komponentteja, ne vastaavat HTML:n div-elementtiä. Paneeleille olisi suositeltavaa tehdä oma tyylipohjasivunsa, jonka voi sitten sisällyttää komponenttihierarkian mukaisesti sivuihin ja muihin paneeleihin. Seuraavaksi näytetään soittolistasovelluksesta irrotettu kustomoitu paneeliluokka.

```
public PlayerPanel(String id) {
            super(id);
            this.add(new ImageButton("play", new DefaultButtonImageRe-
            source("Play")));
            this.add(new ImageButton("stop", new DefaultButtonImageRe-
            source("Stop")));
            this.add(new ImageButton("previous", new DefaultButtonImageRe-
            source("Previous")));
            this.add(new ImageButton("next", new DefaultButtonImageRe-
            source("Next")));
}
```

Kustomoitu paneeliluokka sisältää neljä painikekomponenttia (ImageButton). Sen HTML-pohja näyttää seuraavanlaiselta:

```
<wicket:panel>
            <input wicket:id="previous" class="previous" type="image" />
            <input wicket:id="stop" class="stop" type="image" />
            <input wicket:id="play" class="play" type="image" />
            <input wicket:id="next" class="next" type="image" />
</wicket:panel>
```

Tässä vaiheessa olisi hyvä huomioida, että HTML-pohjan elementtien asettelun täytyy vastata Javassa määriteltyä komponenttihierarkiaa. Eli kun painike on lisätty paneeliin, se ei saa esiintyä paneelin ulkopuolella html-pohjassa. Tämä siksi, että Wicket osaisi piirtää sivun komponenttihierarkian mukaisesti ja osaisi myöhemmin myös hakea komponentteja sivulta. Esimerkiksi tämä aiheuttaisi poikkeuksen Wicketissä (käyttäen edelleen yllä esiteltyä paneelia ja komponenttihierarkiaa):

```
<input wicket:id="previous" class="previous" type="image" />
<wicket:panel>
            <input wicket:id="stop" class="stop" type="image" />
            <input wicket:id="play" class="play" type="image" />
            <input wicket:id="next" class="next" type="image" />
</wicket:panel>
```

### 2.2.4 Lomake ja lomakekomponentit

Käyttäjän lähettäessä pyynnön palvelimelle käyttäen selaimessa olevaa lomake-elementtiä, mikä on sisällytetty Wicketin lomake-komponenttiin, niin Wicket prosessoi lomakkeen. Lomakkeen prosessointiin on useampi erilainen polku, mikä riippuu siitä miten lomake on luotu ja mitä komponentteja siihen on lisätty sekä menikö lomakkeen validointi läpi [4].

Lomakkeen validoinnin mennessä läpi lomakkeeseen sisällytettyjä komponentteja määrätään päivittämään mallinsa ja pitämään täten yllä niihin annettu tieto, mikä on tullut joko käyttäjän tai sovelluksen toimesta. Tämän jälkeen kutsutaan metodia delegateSubmit. Metodi käy ensimmäiseksi mahdollisen lomakkeelle lisätyn komponentin onSubmit-metodin läpi, josta pyyntö tuli lomakkeelle. Lopuksi käydään läpi lomakkeen oma onSubmit-metodi. Mikäli mitään erillistä lomakekomponenttia ei ole, mistä pyyntö palvelimelle olisi tapahtunut, kutsutaan pelkästään lomakkeen omaa onSubmit-metodia.

Jos lomakkeen validointi ei mene läpi, niin kaikki lomakkeeseen lisätyt lomakekomponentit asetetaan invalideiksi. Seuraavaksi niiden virhekäsittelyyn liittyvää onError-metodia kutsutaan joka, sovelluksen toteutuksesta riippuen, viestittää käyttäjälle mikä tai mitkä asiat menivät pieleen tarkistuksessa.

Komponentin pitää toteuttaa IFormSubmittingComponent-rajapinta, jotta lomake prosessoitaisiin. Vaikka lomakekomponentti toteuttaisi mainitun rajapinnan, se voidaan asettaa prosessoimasta koko lomaketta laittamalla epätoden arvon lomakekomponentin setDefaultFormProcessing-setterillä.

Yleisimpiä lomakekomponentteja ovat erilaiset painikkeet ja tekstisyötekomponentit kuten TextField ja TextArea. Lomakkeet voivat kuitenkin sisältää käytännössä mitä tahansa muitakin komponentteja kuten paneeleita ja linkkejä.

### 2.2.5 Mallit

Mallit Wicketissä sisältävät näytettäviä tai muokattavia arvoja, jotka asetetaan komponenteille [5]. Se miten näitä arvoja käsitellään, määritellään mallin IModel-rajapinnan implementaatiossa. IModel-rajapinta erottaa komponentin mallioliosta, joka muodostaa komponentin arvon. Komponentti määrittelee, minkälaisen olion siihen voi lisätä, esim. tekstikomponentille mallin arvon pitää olla jotain, mikä pystytään konvertoimaan merkkijonoksi, kun taas listanäkymälle sen pitää olla Javan lista-olio, johon on annettu näytettävät arvot.

```
nameField = new TextField<String>("name",
            new Model<String>(playList.getName()));
            songSelector = new ListChoice<Song>("songs");
```

Monet komponenteista, kuten ylhäällä, käyttävät Javan tarjoamaa geneerisyyttä. Ylhäällä esitetylle listavalintakomponentille se tarkoittaa, että komponenttiin asetettavan listan pitää sisältää vain Song-olioita. Tekstikentälle lisätään malli, jonka mallioliokSi (ModelObject eli mallin sisältämä olio) asetetaan soittolistan nimi.

Wicket tarjoaa useamman erilaisen valmiin mallitoteutuksen, joita ovat esimerkiksi PropertyModel, ResourceModel ja DetachableModel**.** PropertyModel on malli, joka käsittelee siihen liitetyn olion tiettyä jäsenmuuttujaa. Tämä jäsenmuuttuja annetaan PropertyModelin konstruktorissa merkkijonomuodossa, samoin kuten jäsenmuuttujan sisältävä oliokin. Seuraavana on versio aiemmin esitellystä tekstikentästä, joka käyttää PropertyModelia:

```
nameField = new TextField<String>("name",
            new PropertyModel<String>(playList, "name"));
```

Tyyppikonversioita varten konstruktorissa voidaan välittää luokka, jonka instanssiksi annettu jäsenmuuttuja muutetaan, kuten merkkijonomuuttuja desimaaliluvuksi antamalla konstruktorissa parametrina Double.class.

Resurssien lokalisointia Wicketissä tukee Localizer-luokka, mikä tarjoaa helpon tavan hakea ja formatoida resursseja käyttäjän lokaalin mukaan. Ohjelmoija voi hakea vaikka merkkijonon käyttämällä suoraan Localizeria komponenttien tarjoamalla getString()-metodilla, mutta monesti on kätevämpää käyttää Wicketin ResourceModel-luokkia. Komponentin konstruktorissa ei getString()-metodia tulisi käyttää, koska se voi joissain tapauksissa palauttaa väärän tai lokalisoimattoman resurssin Wicketin tarjoaman varoituksen mukaan.

DetachableModel auttaa sovelluksen muistinhallinnassa. Jokaisen sovellukselle lähetetyn pyynnön lopussa, kun HTML on täysin piirretty käyttäjän selaimeen, Wicket käynnistää detach-sekvenssin [1, s. 96]. Tällöin kaikki pyyntöön osallistuneet komponentit ja mallit saavat mahdollisuuden poistaa datansa.

### 2.2.6 Muita komponentteja

Wicketissä on useamman vuoden kehitystyön tuloksena iso määrä olemassa olevia komponentteja joista työn puitteissa, käydään läpi vain hyvin pieni osa. Monet niistä, kuten myös moni muu Wicketin luokka, ovat abstrakteista komponenteista perittyjä alaluokkia.

Kuvio 3. ImageButton-luokkaan johtava luokkahierarkia [6].

Linkeistä Wicketillä on tarjottavana useampi komponentti eri tarkoituksiin ja kaikki niistä on peritty AbstractLink-luokasta. Näitä ovat esimerkiksi DownloadLink, IndicatingAjaxLink ja ExternalLink. DownloadLink on linkki, jolla käyttäjä voi ladata nettisivulta tiedoston koneellensa. IndicatingAjaxLink on Ajaxia käyttävä linkki, mikä viestittää käyttäjälle olevansa varattu niin kauan kuin Ajax-kutsu on kesken. ExternalLink on yksinkertainen linkki, mitä tavallisesti käytetään linkkaamaan sovelluksen ulkopuolisiin kohteisiin. Esimerkki ExternalLink-luokan käytöstä:

```
this.add(new ExternalLink("bookLink", "http://www2.wsoy.fi/etusivu",
                          "Kirjakustantaja"));
```

Esimerkissä paneeliin lisätään uusi linkki, jolle annetaan parametreina konstruktorille avain, osoite ja sivulla näytettävä teksti. HTML-pohjaan linkki lisätään seuraavasti:

```
<a wicket:id="bookLink">Linkki kirjakustantajan sivuille.</a>
```

Listanäkymä (ListView) on yksi yleisimmistä komponenteista, kun halutaan näyttää useampi rivi tietoa. Usein listanäkymän rivit koostuvat useammasta komponentista.

```
List<Vegetable> veggies = getVegetables();
ListView<Vegetable> listview = new ListView<Vegetable>("veggies", veggies) {
    private static final long serialVersionUID = 1L;

    protected void populateItem(ListItem<Vegetable> item) {
            item.add(new Label("veggyName",item.getModelObject().getName()));
            item.add(new Label("price", item.getModelObject().getPrice()));
    }
};
```

Yllä on listanäkymä-komponentti, jolle annetaan alustaessa lista vihanneksia. Sen populateItem-metodissa lisätään jokaiselle listariville kaksi tekstikomponenttia.

```
<span wicket:id="veggies">
   <span wicket:id="veggyName"/> <span wicket:id=”price” /> <br />
</span>
```

Ylhäällä on esitellyn listanäkymä-komponentin HTML-pohja kokonaisuudessaan. Rivejä tulee piirron jälkeen olemaan niin monta kuin listassa oli vihannes-olioitakin.

DataTable mahdollistaa monipuolisen taulukon luomisen Wicketillä, jota on kohtuullisen helppo kustomoida [7]. DataTable tukee taulukon uudelleenjärjestämistä kolumnin mukaan.

```
BookProvider provider = new BookProvider();
List<IColumn<Book>> columns = new ArrayList<IColumn<Book>>();
columns.add(new PropertyColumn<Book>(Model.of("Title"),
            "title", "title"));
columns.add(new PropertyColumn<Book>(Model.of("Author"),
            "author", "author"));
DefaultDataTable<Book> dataTable = new DefaultDataTable<Book>(
            "bookTable", columns, provider, 4);
this.add(dataTable);
```

Edellä on esitelty DefaultDataTable-komponentin luonti ja lisäys paneeliin. DefaultDataTable saa konstruktorissaan avaimen, listan kolumneja, IDataProvider-rajapinnan toteuttavan luokan ja arvon, montako riviä näytetään per sivu. PropertyColumn-luokan konstruktorissa annetaan arvona kolumnin otsikko, jonka mukaan taulun voi uudelleen järjestää ja viimeisenä parametrina arvo, joka näytetään taulussa. PropertyColumn-luokalle on myös lyhyempi konstruktori, jossa jätetään pois mahdollisuus järjestää taulukko kolumnin mukaan.

BookProvider-luokka perii SortableDataProvider-luokan, joka tarjoaa tiedon järjestämiseen toiminnallisuutta. BookProvider-luokan pitää kuitenkin toteuttaa mm. iterator-metodi datan uudelleen järjestämiseksi:

```
public Iterator<? extends Book> iterator(int first, int count) {

    Collections.sort(books, new Comparator<Book>() {
        public int compare(Book o1, Book o2) {
            SortParam sort = getSort();
            int direction = sort.isAscending() ? 1 : -1;
            if ("title".equals(sort.getProperty())) {
                return direction * (o1.getTitle().compareTo(o2.getTitle()));
            } else {
                return direction *
                        (o1.getAuthor().compareTo(o2.getAuthor()));
            }
        }
    });
```

```
    return books.subList(first, first + count).iterator();
}
```

BookProvider-luokan iterator-metodissa järjestetään kirjalistaus joko kirjan nimen tai kirjoittajan mukaan, joko nousevassa tai laskevassa järjestyksessä. Seuraava kuva havainnollistaa, miltä taulu näyttää selaimessa:

Showing 1 to 4 of 5
<< < 1 2 > >>

| Title | Author |
|---|---|
| Hobitti | J.R.R Tolkien |
| Magian väri | Terry Pratchett |
| Silmarillion | J.R.R Tolkien |
| Taru sormusten herrasta | J.R.R Tolkien |

Kuvio 4. DefaultDataTable-komponentilla tehty kirjataulu.

Kirjapaneelin koodi löytyy kokonaisuudessaan liitteestä 1.

## 2.3 Wicket ja Ajax

Wicketin Ajax-koneen (”engine”) pääasiallinen tarkoitus on integroitua hyvin Wicketin komponenttien ja lisätoiminnallisuuksien kanssa [1, s. 242]. Ajax-kone Wicketissä ei ole aivan yhtä kattava toiminnallisuuksiltaan kuin jotkin muut Ajax-koneet, mutta sen pitäisi silti olla riittävä useimmille käyttötapauksille.

Wicketin Ajax-tuki on pääosin suunniteltu pääteohjelman ja palvelinohjelman väliseen viestinkäsittelyyn. Kaikki Wicket-komponentit pystytään piilottamaan ja uudelleen piirtämään, riippumatta siitä ovatko ne Ajax-komponentteja.

Wicketin Ajax-tuki mahdollistaa kustomoidun JavaScriptin suorittamisen lähettäessä se Ajax-vastauksessa takaisin selaimeen. Samoin sivulle voidaan lisätä dynaamisesti JavaScriptiä tai CSS-määritteitä, joko yksittäisinä määrittelyinä komponentteihin tai sivulle liitettävinä kokonaisina resurssitiedostoina. Myös tuki debuggaamiselle ja aikakatkaisu toiminnallisuudelle löytyy.

Komponenttien uudelleenpiirtoa varten täytyy niissä oleva lippu, outputMarkupId, olla asetettu. Tällöin Wicket piirtää wicket:id-kentän sivupohjaan, joka mahdollistaa komponentin löytämisen sivupohjasta Ajaxille. Jos halutaan tuoda näkyville piilotettu komponentti ja sen sisältämät mahdolliset muut komponentit, täytyy komponentille asettaa outputMarkupPlaceHolderTag-lippu päälle. Kun outputMarkupPlaceHolderTag-lippu on päällä, Wicket piirtää komponentille sivuun paikan pitäjän (”placeholder”), johon komponentti voidaan myöhemmin piirtää.

### 2.3.1 Ajax-komponentteja

Wicketistä löytyy valmiina useampi erilainen Ajax-komponentti. AjaxSubmitLink ja AjaxSubmitButton ovat lomakekomponentteja, joita painettaessa lähetetään lomake, joihin lomakekomponentit on liitetty. AjaxLink on geneerinen linkki, joka tekee vain osittaisen pyynnön sen sijaan, että koko sivu päivitettäisiin. AjaxFallbackLink on taas linkki, joka tekee normaalin pyynnön, jos Ajax-mahdollisuutta ei ole tai JavaScriptin käyttö on poistettu.

```
this.setOutputMarkupId(true);
this.setOutputMarkupPlaceholderTag(true);
AjaxButton useButton = new AjaxButton("Use", new Model<String>("Use"),
            form) {

            @Override
            protected void onSubmit(AjaxRequestTarget target, Form<?> form) {
                        HideablePanel panel = this.getParent();
                        panel.setVisible(panel.isVisible());
                        target.add(panel);
            }
```

Ylhäällä on esitelty kustomoitu AjaxButton-komponentti, joka piilottaa tai tuo näkyviin paneelin painiketta painettaessa. AjaxButton saa onSubmit-metodissa parametrina AjaxRequestTarget-olion, johon lisätyt komponentit piirretään uudelleen. Kaikkien uudelleen piirrettävien komponenttien ”wicket:id”-merkinnät täytyy näkyä lopullisessa piirretyssä sivupohjassa, jota varten asetetaan outputMarkupId. Ylhäällä esitetylle paneelille on asetettu outputMarkupPlaceHolderTag, jotta paneelin voi tehdä uudelleen näkyväksi.

AjaxRequestTargetille voi myös määritellä uudelleenpiirrettävälsi kaikki komponentin lapsikomponentit. Seuraavassa koodiesimerkissä AjaxRequestTargetille on lisätty kaikki sivuun lisätyt paneelit uudelleen piirrettäväksi:

```
target.addChildren(this.getPage(), Panel.class);
```

AjaxRequestTargetiin voi myös lisätä JavaScript-koodia, joka suoritetaan käyttäjän selaimessa pyynnön lopuksi:

```
target.appendJavaScript("alert('Panel is hidden')");
```

### 2.3.2 Liitettävät Ajax-toiminnallisuudet

Wicketissä voidaan kätevästi lisätä Ajax-toiminnallisuutta komponentteihin käyttäen IBehaviorListener-rajapinnan toteuttavia luokkia. Nämä tarjoavat joustavan tavan lisätä toiminnallisuutta komponentteihin kompositiolla eikä perintää tarvita (kuten valmiiden Ajaxkomponenttien tapauksissa). Koska Javassa luokka voi periä vain yhden yläluokan, ei perinnällä saa toiminnallisuutta kahdesta erillisestä luokasta (ellei jompikumpi luokista ole toisen perillinen).

```
bookSelector = new DropDownChoice<Book>("bookSelector",
            new Model<Book>(), new ArrayList<Book>());
bookSelector.add(new OnChangeAjaxBehavior() {

            @Override
            protected void onUpdate(AjaxRequestTarget target) {
                        Book book = bookSeletor.getModelObject();
                        authorField.setModelObject(book.getAuthor());
                        target.add(authorField);
            }

});
authorField.setOutputMarkupId(true);
```

Edellä on esitelty pudotusvalikko, johon on lisätty lisätoiminnallisuus OnChangeAjaxBehavior. Pudotusvalikon arvoa muuttaessa kysytään valikosta kirjaoliota. Kirjaoliosta kysytään vuorostaan tekijäkenttää, mikä asetetaan tekstikentän arvoksi. AjaxRequestTargetille lisätään tekstikenttä uudelleen piirtoa varten. Huomataan jälleen, että outputMarkupId-lippu on asetettu tekstikentälle uudelleen piirron mahdollistamiseksi. Myöskään listaa kirjaolioita ei ole vielä annettu pudotusvalikolle (esitelty koodi on otet-

tu paneelin konstruktorin kutsumasta init-metodista), vaan ne voidaan antaa esimerkiksi paneelin onBeforeRender-metodissa, jota kutsutaan aina ennen uudelleenpiirtoa.

### 2.4 Wicket ja testaus

Wicketillä on testausta varten olemassa WicketTester-luokka, jonka avulla kehittäjä voi testata sovelluksensa käyttäen yksikkötestejä. WicketTester tarjoaa esimerkiksi mahdollisuuden tarkistaa, tuliko sivu ja sen komponentit piirretyksi, ovatko komponentit näkyviä tai näkymättömiä, sekä mahdollisuuden suorittaa käyttöliittymätoimintoja kuten linkin klikkaus. Myös Ajax-tapahtumien testaaminen onnistuu.

```
@Test
public void shouldRender() {
            WicketTester tester = new WicketTester();
            tester.startPage(ThunderList.class);
            tester.assertLabel("thunderList", "Thunder List");
            tester.assertComponent("player", PlayerPanel.class);
}
```

Edellä on koodiesimerkki WicketTesterin käytöstä. Siinä tarkistetaan kentän sisältö ja se, piirtyykö paneeli. Wicket tarjoaa myös FormTester-apuluokan lomakekomponenttien tarkistamisen ja lähettämisen testaamiseksi.

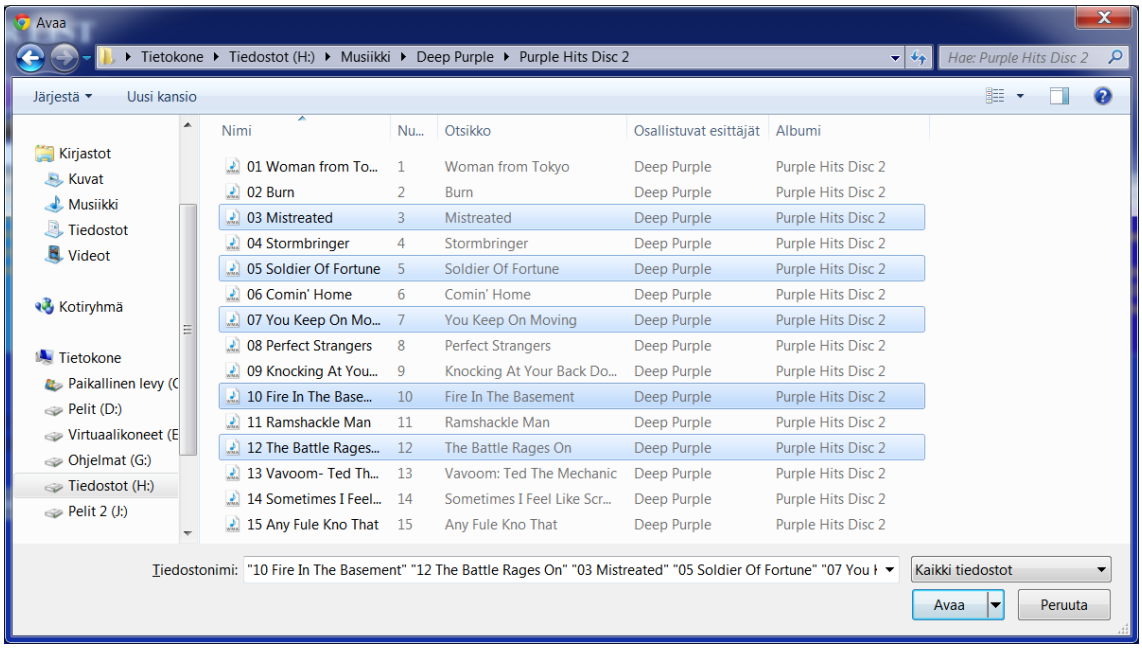
## 3 Työssä käytettäviä muita teknologioita

### 3.1 HTML5

HTML5 mahdollisti sovelluksen musiikkisoitinosion toteutuksen audio-elementillään. Vaikka kontrollit luotiin käyttäen Wickettiä ja niiden toiminnallisuus tehtiin käyttäen jQueryä, niin HTML5 tarjosi kuitenkin JavaScriptillä helposti käsiteltävän ja muokattavan audio-elementin.

Sovelluksen editointiominaisuuksiin sisältyi toiminto useamman kappaleen lataamiselle sovellukseen käyttäjän koneelta. HTML5 tarjosi tähän liittyvälle lomake-elementille mahdollisuuden valita useampi kappale kerralla. Ilman lomake-elementin monivalinta-

ominaisuutta käyttäjän olisi pitänyt käyttää lomake-elementtiä kerran per tiedosto tai ladata tiedostot sovellukseen pakattuna.




Kuvio 5. HTML5:n tarjoama monivalinta-toiminnallisuus.

### 3.1.1 Taustaa

HTML5 (”Hyper Text Markup Language”) on viides kehityksen alla oleva versio HTML-standardista [8]. Viidennen version standardisoinnin oletetaan tapahtuvan vuoteen 2014 mennessä (edellinen HTML-versio standardisoitiin vuonna 1997) [9].

HTML5:sta on tehty yhteistyössä W3C:n (Word Wide Web Consortium) ja WHATWG:n (Web Hypertext Application Technology Working Group) kesken [10]. HTML5:n kehittämiselle on laadittu seuraavat säännöt:

- uusien ominaisuuksien pitäisi perustua seuraaviin teknologioihin: HTML, CSS, DOM ja JavaScript
- vähennetään tarvetta ulkopuolisille ohjelmalisäkkeille (esim. Flash)
- HTML5 tulee sisältää paremman virheiden käsittelyn
- vähennetään skriptaamisen tarvetta tuomalla uusia termejä kuvauskieleen
- HTML5:n tulee olla alustariippumaton.

- Kehitysprosessin pitää olla julkisesti näkyvää.

### 3.1.2 Uudistukset

Uudistuksina HTML5:ssä ovat uudet maalauspohjat (”canvas”), video ja audio-elementit sekä SVG-muotoisen (”Scalable Vector Graphics” eli kaksiulotteisten vektorikuvien kuvauskieli) sisällön näyttäminen ilman objekti-elementin käyttöä. Mainittujen ohella muita lisäyksiä ovat dokumentin semanttisuutta rikastuttavat otsikko-, sektio-, artikkeli- ja navigointi-elementit sekä uudet attribuutit elementtien määrittelyyn ja selkeyttämiseen.

HTML5:ssa on myös poistettu seuraavia aikaisemman version elementtejä jotka on koettu vanhentuneiksi, käyttämättömäksi tai väärinkäytetyiksi [11]. Näitä poistettuja elementtejä ovat esim. kehykset, keskitys (” ”) ja akronyymi.

### 3.1.3 Multimediaominaisuudet

Toistaiseksi HTML5 tukee seuraavien audiotiedostojen toistoa selaimessa: MP3, Wav ja Ogg. Kaikki selaimet eivät tue kuitenkaan kaikkia tiedostomuotoja, esim. Internet Explorer 9 tukee vain MP3-Tiedostoja [12]. Kun soitetaan äänitiedosto selaimessa, periaatteessa seuraava koodi sisältää kaiken mitä tarvitaan:

```
<audio controls="controls">
            <source src="music.ogg" type="audio/ogg" />
            <source src="music.mp3" type="audio/mpeg" />
            Selaimesi ei tue audio-elementtiä.
</audio>
```

Tässä on esitetty kaksi lähdetiedostoa kappaleen toistamiseksi. Selain käyttää niistä ensimmäistä tunnistamaansa versiota. ”Selaimesi ei tue audio-elementtiä” -teksti tulee näkyviin selaimilla, jotka eivät ole HTML5-yhteensopivia kuten Internet Explorer 8 ja vanhemmat selainversiot. Audio-elementti tarjoaa myös kontrollit äänitiedoston toistamiselle:


Kuvio 6. HTML5:n tarjoama kontrolli äänitiedoston toistamiseen.

Kontrolli tarjoaa mahdollisuuden toistaa ja pysäyttää toiston audiotiedostolle, ajastimen, liukusäätimen, jolla voidaan vaihtaa äänitiedoston kohtaa, sekä volyymikontrollin.

Videotiedostojen toistoon on käytössä hyvin samanlainen merkintäkieli ja kontrollit kuin äänitiedostollekin. Video-elementtiä määrittäessä on myös hyvä määritellä sille leveys- ja korkeusominaisuudet, koska silloin selain osaa varata tarvittavan tilan videolle ladattaessa nettisivua. Tällä hetkellä tuetaan, selaimesta riippuen, kolmea eri videotiedostoa: MP4:ää, WebM:ää ja Oggia. Myös tekstitys on mahdollista liittää videoon track-merkinnän avulla, mutta vielä mikään selainversio ei tue sitä.

## 3.2 jQuery

Tehdyssä sovelluksessa jQueryä käytetään musiikkisoittimen kontrolleissa. jQuery tarjosi kätevän tavan lisätä kontrolleihin toiminnallisuutta ilman, että olisi tarvinnut tehdä JavaScript-funktiokutsuja suoraan HTML-elementtien sisältä.

jQueryn tarjoamalla toiminnallisuudella mahdollistettiin kappaleen vaihto seuraavaan kappaleeseen soittolistassa kun kappale on soitettu loppuun. Samalla tekniikalla onnistui kappaleen tämänhetkisen kohdan näyttäminen liukusäätimessä. Myös kappaleen toistamiseen kuluneen ajan päivitys käyttäjälle onnistui.

### 3.2.1 Taustaa

jQuery on selaimille tarkoitettu ilmainen, avoimen lähdekoodin JavaScript-kirjasto. jQuery sopii toimintojen käsittelyyn, animaatioiden tekemiseen, DOM-elementtien valitsemiseen ja Ajax-sovelluksien toteutukseen. jQuery julkaistiin vuonna 2006, ja se on nykyään maailman suosituin JavaScript-kirjasto [13].

jQuery-kirjasto on erillinen JavaScript-tiedosto, mikä voidaan sisällyttää nettisivuun linkittämällä joko paikalliseen palvelimen kopioon tai johonkin julkisten palvelimien ko-

pioihin, esim. Googlen tarjoamaan kopioon. jQuery funktioiden esittäminen tapahtuu yleensä käyttäen ready()-funktiota tai sen lyhennettä:

```
//$(document).ready()-funktion lyhenne
$(function() {

            $(".play").click(function(e) {
                        //funktion sisältö
            });


            $(".stop").click(function(e) {
                        //funktion sisältö
            });
});
```

### 3.2.2 Toiminnot

Sizzle, joka on JavaScriptillä tehty CSS-valitsin (Cascading Style Sheets), on lisätty jQueryyn [14]. jQuery mahdollistaa myös DOM-puun (Document Object Model) läpikäymisen ja modifikaation, mukaan lukien CSS-elementtien. Tapahtumien käsittelyn liittäminen jQuerylla onnistuu lukuisiin JavaScript-tapahtumiin kuten *change, click* ja *blur*. JavaScript tapahtuman laukaisu onnistuu myös jQuerylla. Seuraava esimerkki havainnollistaa jQueryn tapahtumien käsittelyn liittämisen:

Yksinkertainen HTML-valinta elementti:

```
<select class="choice">
            <option value="option1" selected="selected">Valinta 1</option>
            <option value="option2">Valinta 2</option>
 </select>
```

Sidotaan jQuerylla valinnan change-tapahtuma funktioon:

```
$('.choice').change(function() {
            alert('Kutsuttu käsittelijää change-tapahtumalle.');
});
```

Edellä on esitelty, miten change-tapahtumalle on lisätty toiminnallisuutta jQuerylla. Muita jQueryn tarjoamia ominaisuuksia ovat:

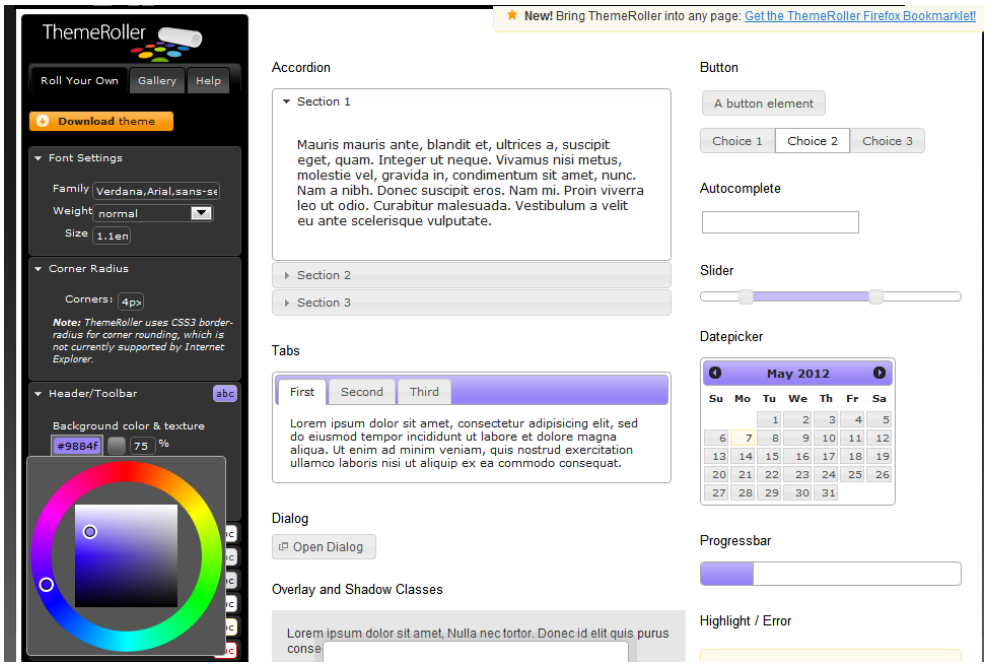
- erilaisia efektejä animaation kera tai ilman animaatiota kuten toggle-toiminnallisuus, joka tuo näkyviin tai piilottaa määrättyjä elementtejä
- kattavat Ajax-toiminnallisuudet
- työkalufunktioita kuten tämän hetkisen ajan kysely now()-funktiolla tai välimerkkien poisto merkkisarjasta trim()-funktiolla
- erilaisia ohjelmalisäkkeitä
- yhteensopivuus eri selainten ja vanhojen selainversioiden kanssa.

## 3.3 jQuery UI

jQuery UI on JavaScript-kirjasto, mikä tarjoaa toimintoja animaation, käyttöliittymän vuorovaikutteisuuden ja erilaisten tehosteiden käyttämiseen [15]. jQuery UI tarjoaa myös erilaisia teemallisia verkkosivulla käytettäviä pienohjelmia kuten kalenterin, liukusäätimen ja tehtäväpalkin. Kirjaston tarjoamia käyttöliittymätoimintoja on mm. elementtien vetäminen ja pudottamisen toisiin elementteihin (”drag and drop”), elementin koon muuttaminen ja elementin valitsemisen.

Kuten perus jQuery, jQuery UI voidaan myös liittää verkkosivulle joko suoraan tai viitaten julkisen palvelimen kopioon. jQuery UI on nimensä mukaisesti rakennettu jQuery–kirjaston päälle. jQuery UI kirjasto julkaistiin 17.9.2007 [16].

jQuery UI:n verkkosivuilla on palvelu nimeltään ThemeRoller, jolla voi kustomoida jQuery UI:n käyttämien HTML-elementtien ulkoasua. Palvelusta löytyy myös useampi valmis CSS-pohja.




Kuvio 7. ThemeRoller-palvelu jQuery UI:n CSS-määritysten kustomointiin.

Musiikintoistosovelluksessa käytettiin jQuery UI:n liukusäädintä, jotta käyttäjä voisi manuaalisesti vaihtaa soitettavan kappaleen kohtaa. Sovelluksen CSS-määrityksistä monet tulevat jQuery UI:n mukana tulleessa kustomoidusta CSS-tiedostosta.

# 4 Musiikintoistosovellus

Musiikintoistosovellus, työnimeltään Thunder List, voidaan jakaa karkeasti kahteen osioon: soittimeen ja soittolistatoimintoihin. Soitin mahdollistaa kappaleiden toistamisen, toiston pysäytyksen sekä soitettavan kohdan vaihtamisen kappaleelta. Soittimella voi myös vaihtaa manuaalisesti kappaletta edelliseen tai seuraavaan, ja se automaattisesti alkaa toistaa soittolistassa olevaa seuraavaa kappaletta, kun kappale on soitettu loppuun.


Kuvio 8. Musiikintoistosovellus.

Soittolistatoimintoja on soittolistan luonti, poisto ja muokkaus. Soittolistaan voi siirtää kappaleita toisista soittolistoista ja kappaleita voi poistaa soittolistasta. Soittolistan voi myös uudelleen nimetä. Soittolistatoimintoihin lasketaan myös uusien kappaleiden lisäys sovellukseen.

Näitä kahta sovelluksen osaa yhdistää käsite aktiivinen soittolista. Aktiivinen soittolista on se, mitä voidaan muokata soittolistatoiminnoissa ja sen kappaleita soitetaan soittimessa.

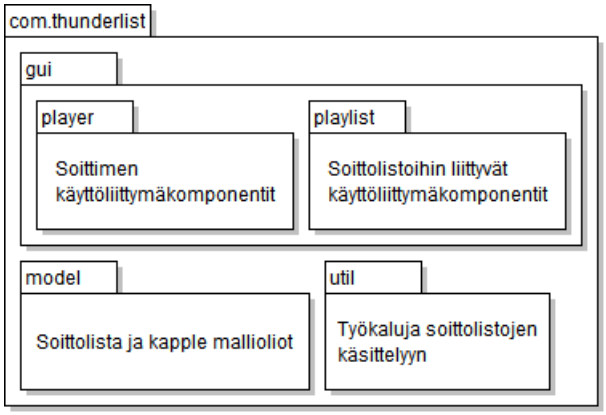
Työ on tehty käyttäen Wicketin versiota 1.5.4, jQueryn versiota 1.7.1 ja jQuery Ui:n versiota 1.8.18. Integroituna ohjelmointiympäristönä on käytetty Eclipse Indigoa, johon on asennettu ohjelmalisäke Run Jetty Run. Tämän ohjelmalisäkkeen avulla sovelluksen testaaminen ja debuggaus oli nopeaa ja kätevää, koska sovelluksen käynnistäminen onnistui suoraan Eclipsestä.

## 4.1 Arkkitehtuuri

Sovellus koostuu pääosin Wicket-komponenteista ja niiden merkkikielipohjista, sekä komponentteihin liitetyistä malliolioista. Sovellukseen kuuluu myös kolme JavaScript kirjastoa: jQuery, jQuery Ui ja sovelluksen oma JavaScript-kirjasto soittimelle, mikä käyttää edellisten tarjoamaa toiminnallisuuksia.

### 4.1.1 Pakettirakenne

Sovelluksen Java-osuuden kolme keskeisintä pakettia ovat malli-, käyttöliittymä- sekä työkalupaketti. Mallipaketti sisältää sovelluksen keskeiset mallioliot: soittolistan ja kappaleen. Käyttöliittymäpaketti sisältää sovelluksen tarvitsemat käyttöliittymäkomponentit. Työkalupaketti sisältää työkaluja soittolistaolioiden lukemiseen, poistamiseen ja tallentamiseen.

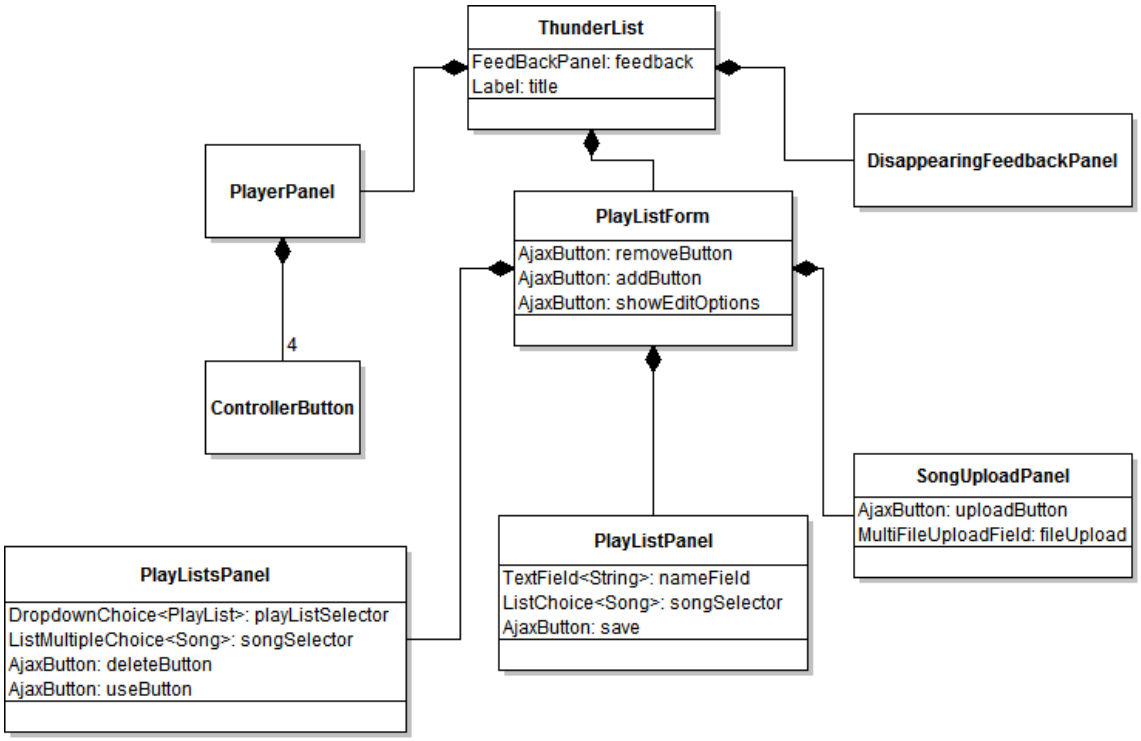



Kuvio 9. Sovelluksen pakettirakenne

Käyttöliittymäpaketti sisältää kaksi pienempää pakettia soittimen ja soittolistan, jossa on niihin sovelluksen osiin liittyvät käyttöliittymäkomponentit. Sovelluksen kustomoitu WebPage-luokka ja palautepaneeli löytyvät suoraan käyttöliittymäpaketista.

### 4.1.2 Käyttöliittymäkomponentit

Sovelluksen käyttöliittymäkomponentteina on käytetty sekä valmiita Wicket-komponentteja että niistä perittyjä kustomoituja komponentteja. Seuraavassa luokkakaaviossa näytetään sovelluksen käyttöliittymäkomponentit ja niiden koostumus.




Kuvio 10. Luokkakaavio käyttöliittymäkomponenteista.

ThunderList-luokka perii Wicketin WebPage-luokan, ja se koostuu palautepaneelista, otsikosta, soitinpaneelista ja soittolistalomakkeesta. Soitinpaneeli sisältää neljä kustomoitua painikekomponenttia, jotka toimivat soittimen kontrolleina.

Soittolistalomake koostuu soittolistatpaneelista, soittolistapaneelista ja kappaleidenlisäyspaneelista. Soittolistalomake sisältää myös painikkeet kappeleiden lisäykselle soittolistaan ja poistamiselle soittolistasta sekä painikkeen, millä näytetään tai piilotetaan editointikomponentit.

### 4.1.3 Malleissa käytettävät luokat

Sovelluksessa on kaksi keskeistä POJO-luokkaa (”Plain Old Java Object”), soittolista ja kappale. Soittolista sisältää soittolistan nimen ja listan kappaleista. Kappale taas sisältää kappaleen polun ja nimen. Näistä tehtyjä olioita liitetään sovelluksessa useampaan eri Wicket -komponenttiin, kuten pudotusvalikkoon (”DropdownChoice”). Molemmat myös toteuttavat Serializable-rajapinnan, jotta niiden lukeminen levyltä ja tallentaminen levylle onnistuu. Seuraavaksi esitellään soittolistaluokka.

```
public class PlayList implements Serializable {

            private static final long serialVersionUID =
                        3253485477301452594L;

            private List<Song> songs;
            private String name;
```

Soittolistaluokka sisältää listan kappaleita ja nimen soittolistalle. Muuttujan serialVersionUID täytyy olla ainutlaatuinen, jotta soittolistan tallennus ja luku ovat mahdollisia. Kappale luokka taas sisältää tiedon kappaleen nimestä ja hakemistopolusta kappaleeseen:

```
public class Song implements Serializable {
            private static final long serialVersionUID =
                        -1304779653182692335L;
            private String name;
            private String songPath;
```

## 4.2 Soitin

Sovelluksen soitinosaan kuuluu kappaleen soittamiseen liittyvät kontrollit, liukusäädin ja tieto soitettavasta kappaleesta. Soitettavasta kappaleesta näytetään nimi ja tieto siitä, missä kohdassa soitin on toistettavaa kappaletta. Soitin jakaa aktiivisen soittolistan soittolistatoimintojen kanssa.


Kuvio 11. Soitinosa sovelluksesta.

Soitin on toteutettu JavaScriptin ja HTML 5:n avulla. JavaScriptissä on käytetty hyödyksi JavaScript-kirjastoja jQuery ja jQuery UI. Soittimessa soitettavat kappaleet tulevat Wicketin avulla luodusta soittolistasta.

### 4.2.1 Musiikin toisto

Musiikin toisto tapahtuu valitsemalla soittolistasta kappale ja sitten joko painamalla toistopainiketta tai uudelleen painamalla samaa kappaletta soittolistasta (myös tuplaklikkaus toimii). Seuraavaa JavaScript-metodia kutsutaan, kun valitaan soittolistasta kappale.

```
function selectSong(option){
            selectedSongId = option.selectedIndex;
            var comboValue = option.options[selectedSongId].text;
            if(selectedSong == comboValue) {
                        curPlaying = null;
                        $(".play").click();
            } else {
                        selectedSong = comboValue;
            }
}
```

Siinä otetaan globaaleihin muuttujiin tieto listakomponentin valitusta indeksistä ja sen valinnan sisältämä teksti. Tuplaklikkauksen yhteydessä kutsutaan toistometodia. Seuraavaksi on toistometodista leikeltyä koodia (katso liite 2).

```
song = $(this).next('audio')[0];
song.play();

$(song).bind('timeupdate', function() {
            if(song.duration <= song.currentTime) {
                        $(".next").click();
            }
});
```

Ensimmäiseksi metodissa haetaan HTML:ssä oleva audio-elementti, jolle on annettu arvoksi valittu kappale. Sen jälkeen laitetaan kappale soimaan ja käyttämällä jQueryn tarjoamaa bind-toiminnallisuutta sidotaan kappale metodiin, joka vaihtaa kappaleen soittolistassa olevaan seuraavaan kappaleeseen, kun kappale on soitettu. Sovellus siirtyy soittamaan soittolistan ensimmäistä kappaletta, jos juuri soitettu kappale on soittolistan viimeinen.

Kappaleen vaihtamiselle on myös painikkeet seuraava ja edellinen. Sovellus siirtyy soittamaan soittolistan viimeistä kappaletta painettaessa edellinen-painiketta, jos valittu kappale on ensimmäinen. Samoin sovellus siirtyy toistamaan ensimmäistä kappaletta painettaessa seuraava-painiketta, jos valittu kappale on soittolistan viimeinen.

### 4.2.2 Musiikin pysäytys

Musiikin pysäytys tapahtuu painamalla pysäytys-painiketta, johon on liitetty seuraava funktiokutsu:

```
$(".stop").click(function(e) {
            if(curPlaying == null) {
                        return;
            }
            e.preventDefault();
            curPlaying.pause();
            curPlaying.currentTime = 0.
            curPlaying = null;
});
```

Metodi pysäyttää soitetun kappaleen ja laittaa sen soittoajan kappaleen alkuun. Taukopainiketta painettaessa kappale vain pysäytetään. Taukopainike korvaa toistopainikkeen kappaleen toiston ajaksi ja päinvastoin toistopainike korvaa taukopainikkeen, kun kappaletta ei toisteta. Tämä tapahtuu muuttamalla kuvapainikkeen lähdeattribuuttia.

```
if (curPlaying.paused) {
            curPlaying.play();
            $(this).attr("src", "res/Pause.png");
} else {
            curPlaying.pause();
            $(this).attr("src", "res/Play.png");
```

### 4.2.3 Aikajana

Aikajana on myös sidottu soitettavaan kappaleeseen. Sen toteutuksessa hyödynnetään jQuery UI:n tarjoamaa liukusäädin-toimintoa. Seuraavassa JavaScript-koodissa alustetaan liukusäädin HTML:ssa olevalle div-elementille.

```
$("#slider").slider({
            step: 0.01,
            range: "min",
```

```
            max: song.duration,
            animate: true,
            slide: function() {
                        manualSeek = true;
            },
            stop:function(e,ui) {
                        manualSeek = false;
                        song.currentTime = ui.value;
            }
});
```

Liukusäätimelle on määritelty useampi eri muuttuja ja funktio. Askel määrää, kuinka pienissä askelissa liukusäädintä voidaan siirtää. Range määrittelee, mistä käyttöliittymäelementti, joka näyttää kuluneen matkan liukusäätimessä, lähtee säätimen kahvaan. Minimiarvolla se lähtee vasemmalta ja maksimiarvolla oikealta. Arvo tosi on mahdollinen jos liukusäätimessä on kaksi kahvaa (epätosi on oletusarvo). Seuraava kuva havainnollistaa:

Kuvio 12. Kulunut matka (oranssi alue) lähtee vasemmalta liukusäätimen kahvaan.

Maksimiarvo kuvaa viimeistä arvoa liukusäätimessä, joka soittimen tapauksessa on kappaleen kesto. Animaatioarvolla määritellään mallinnetaanko liukusäätimen kahvan liike animaatiolla. Liukumetodiin, jota kutsutaan, kun käyttäjä siirtää kahvaa, määritellään manuaalinen etsintä (”manualSeek”) päälle, jotta käyttäjä voi siirtää kahvaa liukusäätimessä ilman, että soitin yrittää samalla siirtää kahvaa kappaleen mukaan. Pysäytysmetodissa, jota kutsutaan kun käyttäjä vapauttaa kahvan, laitetaan manuaalinen etsintä pois päälle ja samalla laitetaan kappale soimaan kohdasta, johon liukusäätimen kahva on siirretty. Seuraavaksi ovat loput aikajanaan liittyvästä toiminnallisuudesta:

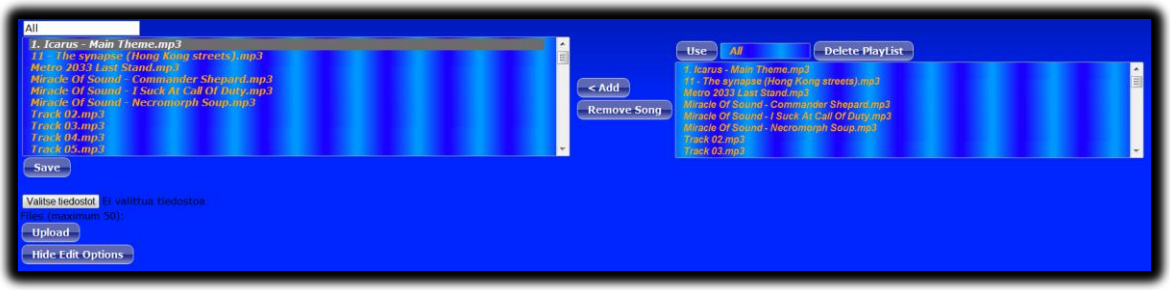
```
var currentTime = parseInt(song.currentTime, 10);
var mins = Math.floor(currentTime/60,10);
var secs = currentTime - mins*60;
timePassed.innerHTML = mins + ':' + (secs > 9 ? secs : "0" + secs);
if(!manualSeek) {
            $( "#slider" ).slider( "option", "value", song.currentTime );
}
```

Siinä asetetaan HTML-elementin, johon ”timePassed” viittaa, arvoksi kappaleessa kulunut aika minuutteina ja sekunteina. Sen lisäksi siirretään liukusäätimen kahva osoitta-

maan nykyistä kappaleen kohtaa, mikäli käyttäjä ei ole siirtämässä kahvaa manuaalisesti.

## 4.3 Soittolistatoiminnot

Soittolistatoimintoihin kuuluu soittolistan luonti, poisto ja muokkaus. Soittolistaa voi muokata uudelleen nimeämällä soittolistan sekä lisäämällä tai poistamalla siitä kappaleita. Myös uusien kappaleiden lisäys lasketaan soittolistatoimintoihin.




Kuvio 13. Soittolistat, kun editointiominaisuudet ovat näkyvillä.

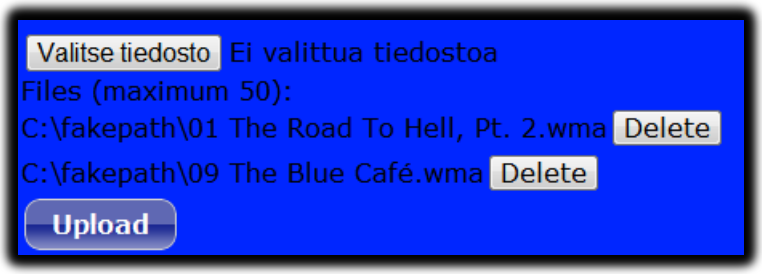
Soittolistatoiminnot on suurimmaksi osaksi toteutettu Wicketillä. Toiminnoissa on käytetty paljon Wicketin Ajax-toiminnallisuutta, jotta sivun eri osia on mahdollista päivittää dynaamisesti. Monet toiminnoista on oletusarvoisesti piilotettuina ja ne saadaan näkyviin ja piilotettua Ajax-painikkeella, mikä on ylhäällä esitetyssä kuvassa ”Hide Edit Options” –painike. Ajax-painikkeen teksti päivitetään vastaavasti.

### 4.3.1 Kappaleiden lisääminen sovellukseen

Kappaleiden lisääminen sovellukseen on toteutettu käyttämällä MultiFileUploadField-komponenttia. Se on lomakekomponentti, joka sallii käyttäjän ladata verkkoon useamman tiedoston kerrallaan.

```
MultiFileUploadField fileUpload = new MultiFileUploadField(
            "songUpload", new PropertyModel<Collection<FileUpload>>(this,
            "uploads"), 50);
WebComponent component = (WebComponent) fileUpload.get("upload");
component.add(new AttributeModifier("multiple", "multiple"));
```

Komponentti saa konstruktorissaan avaimen, PropertyModel-olion sekä maksimimäärän tiedostoja, jotka voidaan ladata. PropertyModel-olioon asetetaan tyhjä kokoelma FileUpload-olioita, joihin lisätään tiedostoja sitä mukaan, kun käyttäjä lataa niitä lisää verkkoon. MultiFileUploadFieldin sisältämään komponenttiin, joka vastaa HTML:n input-elementtiä, lisätään AttributeModifier-olio HTML5:n tarjoamalla monivalintamääritteellä. Näin siksi, jotta useamman kuin yhden tiedoston valinta onnistuisi kerralla.




Kuvio 14. Käyttäjä on valinnut kaksi tiedostoa ladattavaksi verkkolevylle.

Tiedostot tallennetaan kustomoidun AjaxButton-luokan avulla verkkolevylle. Tallennettaessa tarkistetaan, jokaisen tiedoston kohdalla, ettei samannimistä tiedostoa ole jo olemassa verkkolevyllä. Lopuksi käyttäjälle ilmoitetaan palautepaneelin kautta onnistuiko tiedostojen lataaminen verkkolevylle.

### 4.3.2 Soittolistan valinta ja uuden soittolistan luonti

Soittolistan voi valita pudotusvalikosta ja klikkaamalla käytä-painiketta siitä tulee toistimessa toistettava aktiivinen soittolista. Seuraavaksi esitellään käytä-painikkeen metodi, josta aktiivisen soittolistan vaihto tapahtuu.

```
protected void onSubmit(AjaxRequestTarget target, Form<?> form) {
            PlayListPanel panel = ((PlayListForm) form).getPlayListPanel();
            panel.setPlayList(playListSelector.getModelObject());
            target.add(panel);
}
```

Painike on soittolistat-paneeliin lisätty Ajax-painike, ja se hakee lomakkeelta (johon soittolistat-paneeli on lisätty) soittolistapaneelin. Soittolistapaneeliin asetetaan pudotusvalikko-olioon (”DropDownChoice”) valittu soittolista hakemalla pudotusvalikon mal-

liolio. AjaxRequestTarget-olioon lisätään soittolistapaneeli uudelleen piirtoa varten. Seuraavaksi esitellään soittolistapaneelin ylikirjoitettu metodi jota kutsutaan ennen kuin komponentti piirretään.

```
protected void onBeforeRender() {
            List<Song> songs = playList.getSongs();
            songSelector.setChoices(songs);
            if (songs.size() > 0 && !keepSelection) {
                        songSelector.setModel(new Model<Song>(songs.get(0)));
            }
            keepSelection = false;
            nameField.setModelObject(playList.getName());
            boolean isEditOptionsShown = ((PlayListForm) this.getParent())
                                                .isEditOptionsShown();
            save.setVisible(isEditOptionsShown);
            nameField.setEnabled(isEditOptionsShown);
            super.onBeforeRender();
}
```

Siinä asetetaan listavalinnassa (”ListChoice”) näytettävät kappaleet ja asetetaan valittu kappale listan ensimmäiseksi, mikäli soittolista ei ole tyhjä ja ”säilytä valinta” –muuttuja (”keepSelection”) ei ole tosi. Tekstikentän arvoksi tulee soittolistan nimi ja sen käyttö sallitaan sen mukaan ovatko muokkaustoiminnot näkyvissä vai eivät. Myös tallennus-painike näytetään sen mukaan. Lopuksi kutsutaan, normista poiketen, yliluokan meto-dia. Yleensä yliluokan metodia kutsutaan ylikirjoitettaessa heti metodin alussa, mutta koska tämä kyseinen metodi on vastuussa tapahtumasarjoittamisesta (”cascading”) on suositeltavaa kutsua ylikirjoitettua metodia vasta lopussa.


Kuvio 15. Soittolistan luominen.

Uuden soittolistan luonti sovelluksessa tapahtuu valitsemalla pudotusvalikosta uuden soittolistan ja painamalla käytä-painiketta. ”New” on soittolistoja haettaessa luotu tyhjä

soittolistaolio. Lopuksi nimetään soittolista ja painetaan tallenna-painiketta, jolloin pudotusvalikkoon ilmestyy juuri tehty soittolista.

### 4.3.3 Kappaleiden lisääminen soittolistaan

Kappaleiden lisääminen soittolistaan tapahtuu valitsemalla yksi tai useampi kappale oikeanpuoleisesta listasta ja painamalla lisää-painiketta. Lisää-painike painamisen yhteydessä lähetetään Ajax-pyyntö, jonka seurauksena päivitetään Soittolista-paneelin soittolistaoliota ja piirretään soittolistapaneelin sisältö uudelleen. Seuraavaksi esitetään lomakkeeseen lisätyn Ajax-painikkeen metodi, mitä kutsutaan kyseistä painiketta painettaessa.

```
protected void onSubmit(AjaxRequestTarget target, Form<?> form) {
            PlayList playList = playListPanel.getPlayList();
            for (Song song : playListsPanel.getSelectedSongs()) {
                        playList.addSong(song);
            }
            playListPanel.setKeepSelection();
            target.add(playListPanel);
}
```

Ensimmäiseksi pyydetään soittolistapaneelista aktiivinen soittolista ja siihen soittolistaan lisätään toisessa paneelissa olevasta listanäkymässä olevat valitut kappaleet. Lopuksi lisätään soittolistapaneeli uudelleen piirrettäväksi.

### 4.3.4 Kappaleiden poisto soittolistasta

Kappale voidaan poistaa soittolistasta valitsemalla se ja sitten painamalla poista-painiketta. Lomake-luokassa pyydetään soittolistapaneelista soittolistaa ja sen listavalinta-komponentissa (”ListChoice”) valittua kappaletta. Tämän jälkeen soittolistasta poistetaan kyseinen kappale ja soittolistapaneeli piirretään uudelleen.

### 4.3.5 Soittolistan tallennus

Soittolistan tallennuksessa käyttäjä painaa tallenna-painiketta tekemiensä muutoksien jälkeen. Mahdollisia muutoksia ovat soittolistan uudelleennimeäminen, kappaleiden lisäys ja kappaleiden poisto.

```
protected void onSubmit(AjaxRequestTarget target, Form<?> form) {
            String playListName = nameField.getInput();
            if (playListName.isEmpty()
                        ||  AbstractPlaylistName
                            .valueEqualsAbstractPlaylistName(playListName)) {
                        error(playListName.isEmpty() ? "Name of the playlist
                        cannot be empty" : playListName + " is invalid name
                        for playlist.");
            } else {
                        PlayListLoader.writePlayList(new
                        PlayList(playListName, (List<Song>) songSelec-
                        tor.getChoices()));
                        info("Playlist " + playListName + " saved.");
            }
            target.addChildren(form, Panel.class);
            target.addChildren(getPage(), FeedbackPanel.class);
}
```

Edellä esitellyssä koodilistauksessa on esitetty tallenna-painikkeen metodi. Ensimmäiseksi kysytään soittolistan nimeä, eli syötettä tekstikentästä. Nimi ei saa olla tyhjä merkkijono, eikä se myöskään saa vastata listassa (”AbstractPlaylistName”) esitettyjä soittolistan nimiä. Väärästä syötteestä annetaan asiakkaalle virheviesti, joka näytetään sivulle lisätyssä palautepaneelissa (”FeedbackPanel”).

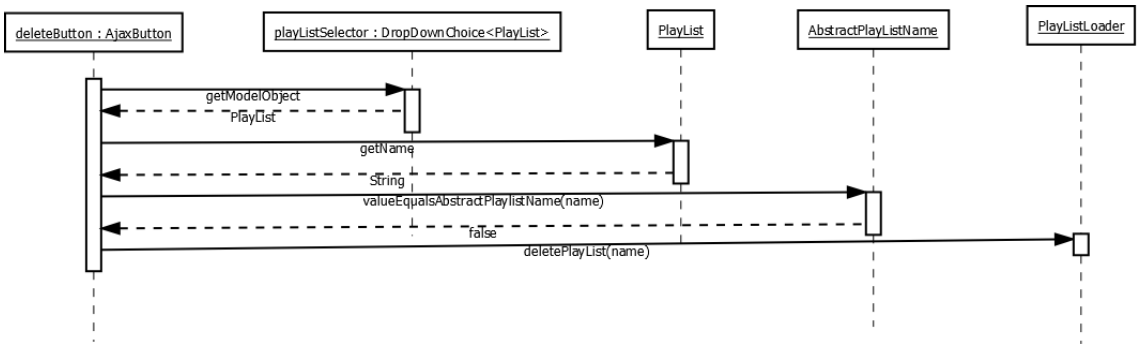
Soittolistan nimen ollessa toimiva, tallennetaan soittolista olio ja sen sisältämät kappaleet kutsumalla PlayListLoader-luokan staattista metodia. Soittolista tallennetaan käyttämällä Javan olioulostulovirtaa (”ObjectOutputStream”). Tämän jälkeen näytetään käyttäjälle viesti onnistuneesta tallentamisesta ja piirretään uudelleen palautepaneeli ja lomakkeen sisältämät paneeliluokat.

### 4.3.6 Soittolistan poisto

Soittolistan poistossa poistetaan pudotusvalikosta valittu soittolista, mikä ei välttämättä ole soittolista, jota käytetään sillä hetkellä toistimessa. Soittolistan poisto on muuten hyvin samanlainen toimenpide kuin soittolistan tallennuskin.

Tarkistetaan, että valittu soittolista on poistettavissa, poistetaan se käyttämällä staattista PlayListLoader-luokan metodia sekä päivitetään pudotusvalikko piirtämällä soittolistatpaneeli. Näytetään myös käyttäjälle viesti onnistuiko soittolistan poistaminen aset-

tamalla info- tai virheviesti piirrettyyn palautepaneeliin. Seuraavassa sekvenssikaaviossa havainnollistetaan soittolistan poistoa.




Kuvio 16. Soittolistan poisto.

# 5 Yhteenveto

Työssä on käyty läpi Wicketin taustaa, toimintaa, komponentteja, malleja sekä Ajax-tukea. Komponenteista ja malleista käytiin läpi yleisimmät. Myös lisätoimintoja sekä Wicketillä testaustakin sivuttiin.

Muut teknologiat käytiin läpi pintapuolisemmin. Kerrottiin hieman niiden taustasta ja ominaisuuksista. Muiden teknologioiden läpikäynnissä pääpaino oli toiminnoissa, joita käytettiin varsinaisessa sovelluksessa.

Sovelluksen kehittämisessä tuli melko vähän mitään todellisia ongelmia. Siinä edesauttoi se, että oli sisäistänyt Wicketin komponenttiajattelun ja osasi välttää muutamia pieniä kompastuskiviä liittyen malleihin ja Ajaxiin. Netistä löytyi tarvittaessa hyvin erilaisia koodiesimerkkejä erilaisille Wicketin komponenteille ja myös audioelementin käsittelyyn JavaScriptillä.

Periaatteessa soittimessa olisi voinut käyttää HTML5:n audio-elementin tarjoamaa kontrolliominaisuutta, mutta käytännössä se ei tukenut kovin hyvin kappaleen kohdan vaihtamista ainakaan ensisijaisesti tekohetkellä käytetyssä uusimmassa Chrome-selaimessa. Musiikintoisto usein pysähtyi kesken, kun vaihdettiin kappaleen kohtaa. Kustomoidut kontrollit sallivat myös paremman muokattavuuden.

Mikäli sovellusta haluttaisiin laajentaa laajempaan käyttöön, mahdollisia lisättäviä toimintoja ja teknologioita olisivat: relaatiotietokantaohjelmisto (esim. MySQL), olio-relaatio-kartoituskehys (esim. Hibernate) sekä käyttäjien tunnistautuminen palveluun. Tämä tietysti vaatisi palvelinraudalta paljon enemmän suorituskykyä ja talletustilaa.

## Lähteet


1. Wicket in Action – Martijn Dashorst & Eelco Hillenius.

2. Meet Apache Wicket [verkkodokumentti, viitattu 26.4.2012]. Saatavissa: http://wicket.apache.org/meet/introduction.html.

3. Apache Wicket [verkkodokumentti, viitattu 26.4.2012]. Saatavissa: http://en.wikipedia.org/wiki/Apache_Wicket.

4. Class Form<T> [verkkodokumentti, viitattu 26.4.2012]. Saatavissa: http://wicket.apache.org/apidocs/1.4/org/apache/wicket/markup/html/form/Form.html.

5. Working with Wicket models [verkkodokumentti, viitattu 26.4.2012]. Saatavissa: https://cwiki.apache.org/WICKET/working-with-wicket-models.html.

6. Class ImageButton [verkkodokumentti, viitattu 26.4.2012]. Saatavissa: http://wicket.apache.org/apidocs/1.5/org/apache/wicket/markup/html/form/ImageButton.html.

7. Displaying Data Using DataTable in Apache Wicket. [verkkodokumentti, viitattu 26.4.2012]. Saatavissa: http://www.javabeat.net/articles/310-displaying-data-using-datatable-in-apache-wicket-1.html.

8. HTML5: A vocabulary and associated APIs for HTML and XHTML [verkkodokumentti, viitattu 26.4.2012]. Saatavissa: http://dev.w3.org/html5/spec/introduction.html#history-1.

9. W3C Confirms May 2011 for HTML5 Last Call, Targets 2014 for HTML5 Standard [verkkodokumentti, viitattu 26.4.2012]. Saatavissa: http://www.w3.org/2011/02/htmlwg-pr.html.

10. HTML5 Introduction [verkkodokumentti, viitattu 26.4.2012]. Saatavissa: http://www.w3schools.com/html5/html5_intro.asp.

11. HTML5 New Elements [verkkodokumentti, viitattu 26.4.2012]. Saatavissa: http://www.w3schools.com/html5/html5_new_elements.asp.

12. HTML5 Audio [verkkodokumentti, viitattu 26.4.2012]. Saatavissa: http://www.w3schools.com/html5/html5_audio.asp.

13. Usage of JavaScript libraries for websites [verkkodokumentti, viitattu 26.4.2012]. Saatavissa: http://w3techs.com/technologies/overview/javascript_library/all.

14. jQuery 1.3 and the jQuery foundation [verkkodokumentti, viitattu 26.4.2012]. Saatavissa: http://blog.jquery.com/2009/01/14/jquery-13-and-the-jquery-foundation/.

15. jQuery UI [verkkodokumentti, viitattu 26.4.2012]. Saatavissa: http://jqueryui.com/.

16. jQuery UI: Interactions and Widgets [verkkodokumentti, viitattu 26.4.2012]. Saatavissa: http://blog.jquery.com/2007/09/17/jquery-ui-interactions-and-widgets/.

## DataTable-esimerkki

```
public class Book implements Serializable {
            private static final long serialVersionUID = 1L;
            private String author;
            private String title;

            public Book(String title, String author) {
                        this.author = author;
                        this.title = title;
            }

            public String getAuthor() {
                        return author;
            }

            public String getTitle() {
                        return title;
            }

            public void setAuthor(String author) {
                        this.author = author;
            }

            public void setTitle(String title) {
                        this.title = title;
            }
}



public class BookPanel extends Panel {

            private static final long serialVersionUID = 1L;

            public BookPanel(String id) {
                        super(id);
                        init();
            }

            private void init() {
                        BookProvider provider = new BookProvider();
                        List<IColumn<Book>> columns =
                                    new ArrayList<IColumn<Book>>();
                        columns.add(new
                                    PropertyColumn<Book>(Model.of("Title"),
                                    "title", "title"));
                        columns.add(new
                                    PropertyColumn<Book>(Model.of("Author"),
                                    "author", "author"));
                        DefaultDataTable<Book> dataTable = new
                                    DefaultDataTable<Book>(
                                    "bookTable", columns, provider, 4);
                        this.add(dataTable);
```

```
            }

}


public class BookProvider extends SortableDataProvider<Book> {

    private static final long serialVersionUID = 1L;
    List<Book> books = new ArrayList<Book>();

    public BookProvider() {
        setSort("author", SortOrder.ASCENDING);
        books.add(new Book("Taru sormusten herrasta", "J.R.R Tolkien"));
        books.add(new Book("Silmarillion", "J.R.R Tolkien"));
        books.add(new Book("Hobitti", "J.R.R Tolkien"));
        books.add(new Book("Magian väri", "Terry Pratchett"));
        books.add(new Book("Valon tanssi", "Terry Pratchett"));
    }

    @Override
    public Iterator<? extends Book> iterator(int first, int count) {

        Collections.sort(books, new Comparator<Book>() {
            public int compare(Book o1, Book o2) {
                SortParam sort = getSort();
                int direction = sort.isAscending() ? 1 : -1;
                if ("title".equals(sort.getProperty())) {
                    return direction *
                        (o1.getTitle().compareTo(o2.getTitle()));
                } else {
                    return direction
                        *(o1.getAuthor().compareTo(o2.getAuthor()));
                }
            }
        });

        return books.subList(first, first + count).iterator();
     }

     @Override
     public int size() {
         return books.size();
     }

     @Override
     public IModel<Book> model(final Book object) {
         return new AbstractReadOnlyModel<Book>() {
            private static final long serialVersionUID = 1L;

            @Override
            public Book getObject() {
                return (Book) object;
            }
         };
     }

}
```

## Soittimen toistometodi

```
$(".play").click(function(e) {
            if (timePassed == null) {
                        timePassed = document.getElementById('timePassed');
            }
            song = $(this).next('audio')[0];
            if (curPlaying == null) {
                        song.src = "track/" + getSelectedSong();
                        song.play();
                        curPlaying = song;
                        updatePlayedSongName();
                        $(song).bind('timeupdate', function() {

                                    $("#slider").slider({
                                        step: 0.01,
                                        range: "min",
                                        max: song.duration,
                                        animate: true,
                                        slide: function() {
                                                manualSeek = true;
                                        },
                                        stop:function(e,ui) {
                                                manualSeek = false;
                                                song.currentTime = ui.value;
                                        }
                                    });
                                    if(song.duration <= song.currentTime) {

                                        $(".next").click();
                                    } else {
                                        var currentTime =
                                           parseInt(song.currentTime, 10);
                                        var mins =
                                           Math.floor(currentTime/60,10);
                                        var secs = currentTime - mins*60;

                                        timePassed.innerHTML = mins + ':' +
                                          (secs > 9 ? secs : "0" + secs);
                                        if(!manualSeek) {
                                                $( "#slider" ).slider(
                                                   "option", "value",
                                                   song.currentTime );
                                                }
                                        }
                        });
                        $(this).attr("src", "res/Pause.png");
            } else if (curPlaying.paused) {
                        curPlaying.play();
                        $(this).attr("src", "res/Pause.png");
            } else {
                        curPlaying.pause();
                        $(this).attr("src", "res/Play.png");
            }
});
```