KARELIA-AMMATTIKORKEAKOULU
Viestinnän koulutusohjelma

Matti Mehtonen

SENCHA TOUCH 2 –PLUGINIT

Opinnäytetyö
Toukokuu 2013

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Toukokuu 2013
Viestinnän koulutusohjelma

Länsikatu 15
80110 JOENSUU
013 260 600

Tekijä
Matti Mehtonen

Nimeke
Sencha Touch 2 -pluginit

Tiivistelmä

Mobiilisovelluksia voidaan rakentaa HTML5:n avulla. Tämä mahdollistaa yhden sovelluksen rakentamisen, joka toimii sellaisenaan kaikissa mobiilikäyttöjärjestelmissä. Sencha Touch 2 on mobiilisovelluksien kehittämiseen tarkoitettu javaScript-kirjasto. Kirjasto perustuu vapaaseen lähdekoodiin, joten sitä voi käyttää maksutta.

Opinnäytetyö on tarkoitettu tietopaketiksi Sencha Touch 2 -pluginien tekoon. Se pyrkii tarjoamaan perusymmärryksen kirjaston ydintoiminnasta. Tarkoitus ei ole opettaa Sencha Touchin käyttöä vaan tarjota tietoa sen kehittämisestä.

Opinnäytetyö alkaa luokkajärjestelmän esittelyllä ja jatkaa siitä Sencha Touchin ominaispiirteisiin. Se antaa perustiedot plugin työympäristön rakentamisesta sekä Sencha Touchin matalan tason toiminnasta sen luokkajärjestelmään. Viimeisissä luvuissa se käy läpi vaihe vaiheelta muutaman pluginin rakentamisen.

Opinnäytetyö on suunnattu Sencha Touchin perustoiminnot jo tunteville. Jos aihe on uusi, on suositeltavaa perehtyä aiheeseen liittyvään materiaaliin esimerkiksi Senchan virallisilla verkkosivuilla.

Kieli
suomi

Sivuja 47

Asiasanat

Mobiili, Sencha Touch, HTML5

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
May 2013
Degree programme in communication

Länsikatu 15
80110 JOENSUU
013 260 600

Author
Matti Mehtonen

Title
Sencha Touch 2 -Plugins

Abstract

Mobile applications can be made with HTML5. This makes it possible to build one application to work on every mobile operating systems. Sencha Touch 2 is a javaScript library for mobile application development with HTML5. The library is based on open source, so anyone can use it without payment.

The purpose of this study is to offer an information package about making Sencha Touch 2 plugins. It aims at providing basic understanding about the core functions of the library. The purpose is not to teach how to use Sencha Touch, but give information about how to develop it even further.

The thesis begins with an introduction of class system and continues from there to characteristics about Sencha Touch. It gives basic knowledge about how to set up a plugin environment and from Sencha Touch low level functionality to its class system. In the lasts chapters it goes step-by-step through making a couple of plugins.

The thesis is intended for people who already know the basics of Sencha Touch. If the subject is new, it is recommended to study the materials available on the official Sencha Touch webpages.

Language
Finnish

Pages 47

Keywords

Mobile, Sencha Touch, HTML5

Sisältö

Sanasto

C	Matalan tason koodikieli, jota käytetään esimerkiksi käyttöjärjestelmien rakennuksessa (Kernighan & Ritchie 1988, 1).
CSS	Lyhenne sanoista Cascading Style Sheets. Verkkodokumentin tyyliohje (W3C 1999).
HTML	Lyhenne sanoista Hypertext Markup Language. Verkkodokumentin sisältö (W3C 1999).
HTML5	Käsite on tarkoitettu kattamaan nykyaikainen Internetin ydinteknologia. Tähän lukeutuu javaScript, CSS ja itse HTML. Teknologia on arvoitu valmistuvaksi vuonna 2022, mutta sen osia käytetään jo nykypäivänä. (HTML5rocks 2013.)
Java	Ohjelmointikieli, jota käytetään monissa eri ympäristöissä. Android sovellukset kirjoitetaan pääasiallisesti Javalla. (Android Developers 2013.)
JavaScript	Suosittu pääasiallisesti verkkoympäristössä käytettävä komentosarjakieli (Crockford 2008, 2).
MVC	Lyhenne sanoista Model, View ja Controller. Ohjelmistoarkkitehtuurityyli, (Kumar 2012, 10).
PHP	Lyhenne käsitteestä PHP: Hypertext Preprocessor. Palvelinpuolen komentosarjakieli, joka on erityisesti suunniteltu verkkokehittämiseen. (PHP 2013.)
WordPress	PHP kielellä kirjoitettu maailman suosituin julkaisujärjestelmä (Wordpress 2013.)

1 Johdanto

Yksi mobiilisovelluksien teon haasteista ovat lukuisat eri käyttöjärjestelmät, jotka vaativat yleensä omat sovellusversionsa. Esimerkiksi Android-sovelluksia kirjoitetaan Javalla, kun taas iOS-sovellukset pohjautuvat C-kielen laajennukseen nimeltään Objective-C. HTML5 tarjoaa mahdollisuuden kirjoittaa yhden sovelluksen, joka toimii kaikissa eri käyttöjärjestelmissä.

Sencha Touch 2 on mobiilisovelluksien tekoon tarkoitettu javaScript-kirjasto. Se on suunnattu pääasiassa iOS-, Android- ja BlackBerry-käyttöjärjestelmille. Sencha Touchin ohjelmointirajapinta vaatii jonkin verran opettelua, mutta tarjoaa mahdollisuuden tehokkaiden mobiilisovellusten tekoon HTML5:n avulla.

Sencha Touch mahdollistaa sovelluskehityksen tietokoneen verkkoselaimella, jossa näytön koskettamista sormella emuloidaan hiiren painalluksella. Tämä tarkoittaa sitä, että sovellusten kehittämiseen tarvitaan ainoastaan tietokone, verkkoselain ja koodi- tai tekstieditori. Verkkoselainta valitessa on hyvä muistaa, että Sencha Touch tukee ainoastaan Webkitillä varustettuja selaimia, joista mainittakoon esimerkiksi Google Chrome.

Kuten kaikessa verkkokehittämisessä myös Sencha Touchin kanssa työskennellessä selaimen kehitystyökalut ovat hyödyllinen apu. Google Chrome sisältää nämä työkalut oletuksena. Käsiksi niihin pääsee painamalla hiiren oikeaa painiketta jostain kohtaa selainikkunaa ja valitsemalla "tarkista elementti". Tämä avaa näkymän, joka sisältää lukuisia työkaluja. Näistä tarvitaan ensisijaisesti konsolia. Konsoliin tulostuu muun muassa javaScriptin aiheuttamia virheilmoituksia. Sencha Touch tulostaa konsoliin myös omia virhe- ja varoitusilmoituksiaan.

Vapaan lähdekoodin ohjelmistolla voi tehdä rahaa monin keinoin, joista mainittakoon tukipalvelut, lisäosat ja maksullinen dokumentaatio (Weiss, 2013). PHP-kieleen pohjautuva julkaisujärjestelmä WordPress on esimerkki menestyvästä

vapaan lähdekoodin ohjelmistosta. WordPress tarjoaa mahdollisuuden rakentaa lisäosia, joilla myös kolmannen osapuolen toimija voi tienata.

Sencha Touch perustuu vapaaseen lähdekoodiin, mikä tarkoittaa sitä, että koodi on ilman maksua vapaasti käytettävissä. Ohjeita sen käytöstä voi hakea verkon kautta Senchan virallisilta sivuilta tai muutamista aiheeseen liittyvistä kirjoista. Dokumentaatio on kuitenkin rajallinen.

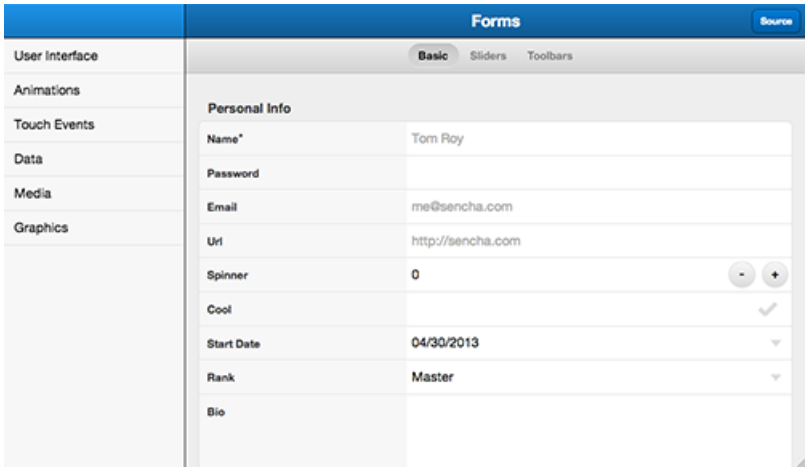
Sencha Touch tiettävästi järjestää maksullisia seminaareja ja kirjallisuutta aiheesta löytyy myytävänä. Maksullisia tukipalveluita on saatavilla verkkosivuilla. On mahdollista ja toivottavaa, että Sencha Touch aloittaa myös plugin- ja lisäosakaupan, avaten mahdollisesti käyttäjille mahdollisuuden myydä tuotoksiaan.

Tämä opinnäytetyö on tarkoitettu antamaan aineksia Sencha Touchin soveltavaan käyttöön ja pluginien tekoon. Se käy läpi aluksi teoreettisia asioita, kuten luokkajärjestelmän perusteet. Viimeisissä luvuissa siirrytään kohti konkreettisempaa pluginien tekoa. Opinnäytetyö pohjautuu Sencha Touch versioon 2.1.1.

2 Lyhyt Sencha Touchin esittely

Sencha Touch koostuu useista erilaisista komponenteista. Jotkin komponenteista käsittelevät sovelluksen dataa, jotkin ohjaavat sen toimintaa ja jotkin rakentavat sovellukselle visuaalisen ilmeen. Sencha Touch on rakennettu MVC-ohjelmistoarkkitehtuurin mukaan. MVC on lyhenne sanoista *model*, *view* ja *controller* (Kumar 2011, 10).

Jotkin Sencha Touchin komponenteista voivat sisältää toisia komponentteja. Esimerkiksi luokkaa *Ext.Panel* laajentava komponentti saattaa sisältää luokkaa *Ext.Button* laajentavan komponentin. Tämä vastaisi perusnäkymää, jossa on yksi painike. Sencha Touchin komponenttirakenne on havainnollistettuna seuraavassa kuvassa:



Kuva 1. Sencha Touch koostuu komponenteista

Kuvassa näkyy monia toisistaan riippuvia visuaalisia komponentteja. Esimerkiksi kukin vasemmalla laidalla oleva listan osa on komponentti. Listakomponentti itsessään ja sen yläpuolella oleva sininen yläpalkki ovat komponentteja. Tämän lisäksi tarvitaan komponentti pitämään näitä komponentteja paikoillaan.

Sencha Touch ei ollut ensimmäinen Senchan luoma verkkopohjaisiin tekniikoihin perustuva kirjasto. Sencha Touch syntyi idoista, jotka lähtivät verkkosovellusten tekemiseen tarkoitetun Ext JS –kirjaston kehittäjien keskuudessa (Garcia, Moss & Simoens 2012, 47). Vastoin isoveljeään Sencha Touch on tarkoitettu nimenomaan mobiilisovellusten tekoon.

Sencha Touch pyrkii parhaaseen mahdolliseen suorituskykyyn mobiililaittella. Se käyttää laajasti hyväkseen rajallista määrää verkkopohjaisille sovelluksille annetuista tekniikoista sovelluksen tekniseen keventämiseen. (Garcia ym. 2012, 4-5.)

3 Sencha Touch –luokkajärjestelmä

3.1 Prototyyppisestä perinnästä luokkajärjestelmään

Sencha Touch on kirjoitetty javaScriptillä. Toisin kuin valtaosa olio-ohjelmointikielistä, javaScript ei perustu luokkajärjestelmään vaan prototyyppei-

hin. Oleellista luokkajärjestelmään perustuvissa kielissä on, että kielen objektit ovat luokkien instansseja ja luokat perivät ominaisuuksia toisilta luokilta. Prototyyppisessä perinnässä taas kielen objektit perivät ominaisuutensa suoraan toisilta objekteilta (Crockford 2008, 46). Sencha Touch huolimatta isäntäkielestään perustuu javaScriptin avulla luotuun luokkajärjestelmään. Sencha Touch tarjoaa myös mahdollisuuden kirjoittaa omia luokkia.

Sencha Touch luokkien tekeminen ei ole vaikeaa. Kun käytetään Sencha Touchia niin omia luokkia kirjoitetaan lähes väistämättä. Esimerkiksi määritellessä uuden näkymän, joka laajentaa vaikkapa *Ext.tab.Panel*-luokkaa, kirjoittaa käytännössä uuden luokan. Sencha Touch tarjoaa sen lisäksi mahdollisuuden luoda valmiille luokille, kuten luokalle *Ext.tab.Panel*, alaluokkia, joita voi laajentaa näkymissä. Kun puhumme Sencha Touchin vakioluokan ja lopullisen komponentin, eli instanssin väliin itse rakennetusta luokasta, puhutaan pluginista.

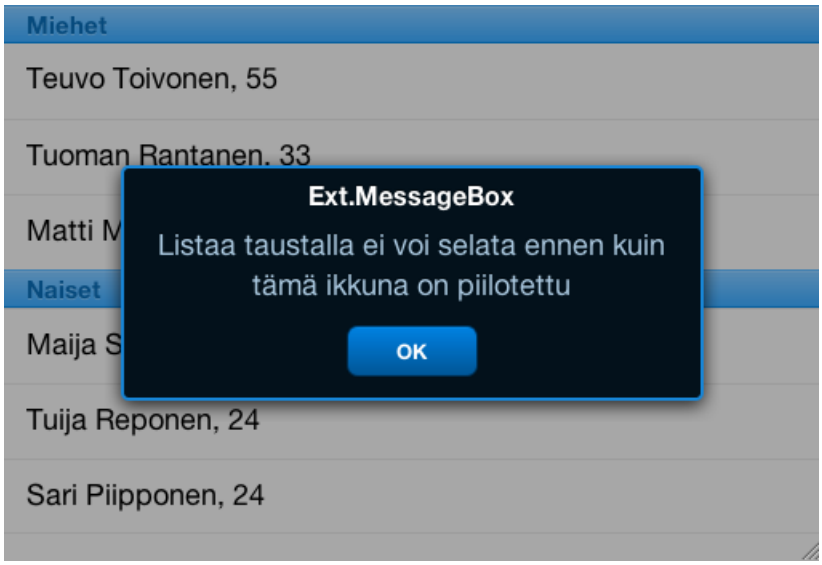
Miksi kirjoittaa omia plugineja? Vaikka Sencha Touch antaakin vapauden määritellä valmiiden komponenttiensa ominaisuuksia hyvin laajasti, jossakin vaiheessa tämä ei enää riitä. Jotakin osaa ei saa toimimaan halutulla tavalla, jokin osa sisältää piiloutuneen koodausvirheen tai jokin osa puuttuu kokonaan. Koska ongelman joutuu joka tapauksessa ratkaisemaan, miksei tekisi siitä pluginia. Plugin ratkaisee ongelman myös tulevista projekteista.

3.2 Luokat Sencha Touchin näkökulmasta

Sencha Touch sisältää luokan *Ext.AbstractComponent.* Luokkaa ei varsinaisesti voi käyttää sellaisenaan. Se on kuitenkin varsin merkittävä, sillä kaikki Sencha Touchin visuaaliset komponentit laajentavat tätä luokkaa (Garcia ym. 2012, 47). Tämä tarkoittaa sitä, että kaikki Sencha Touchin visuaaliset komponentit ovat perineet osan ominaisuuksistaan tältä luokalta.

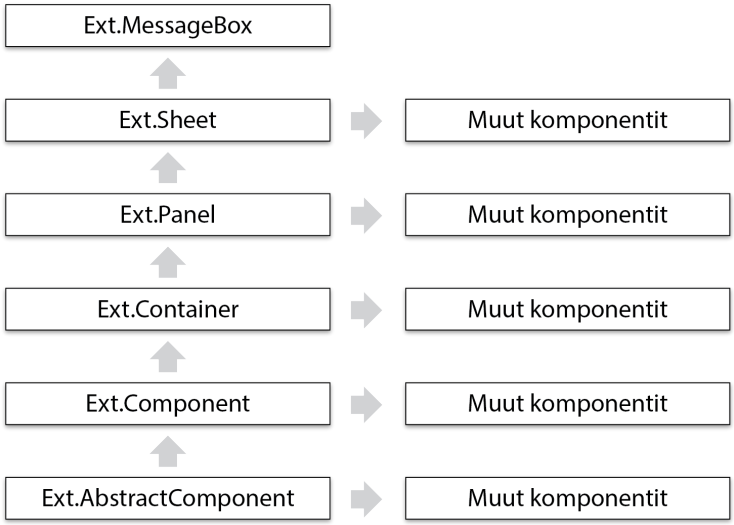
Sencha Touchin *Ext.MessageBox*-luokka, on koko ruudun alleen peittävä ilmoituslaatikko, johon on mahdollista upottaa esimerkiksi painikkeita. Komponentille on määritelty joitakin ominaisuuksia, jotka tekevät siitä yksilöllisen, mutta suu-

rimman osan ominaisuuksistaan se on perinyt luokalta *Ext.Sheet*. *Ext.Sheet*-luokka mahdollistaa muun muassa kaikkien toimintojen keskeytyksen siksi aikaa, kun tämä luokan instanssi on aktiivisena. *Ext.MessageBox* tarvitsee tätä ominaisuutta. Kun ilmoituslaatikko on näkyvissä, esimerkiksi taustalla olevaa listanäkymää ei voi selata, ennen kuin ilmoitusnäkymä on piilotettu. Katso kuva 2.



Kuva 2. Ext.MessageBox-komponentti.

Ext.Sheet-luokka laajentaa luokkaa *Ext.Panel*. Tämä tarkoittaa sitä, että kaikki tutut *Ext.Panel*-luokan toiminnot ja ominaisuudet pätevät myös *Ext.MessageBox*-luokkaan, ellei niitä ole poistettu tai yliajettu. *Ext.Panel*-luokka laajentaa luokkaa *Ext.Container*, joka laajentaa luokkaa *Ext.Component*, joka laajentaa luokkaa *Ext.AbstractComponent*. Edellinen havainnollistettuna seuraavalla sivulla kuvassa 3.



Kuva 3. Ext.MessageBox perintäketju havainnollistettuna.

Ymmärtääkseen luokan *Ext.MessageBox* toimintaa, on aloitettava tutkiminen luokasta *Ext.AbstractComponent*. Luokka kerrallaan on tutkittava luokkia ylöspäin, kunnes saapuu lopulta luokkaan *Ext.MessageBox*. Vasta kun ymmärtää jokaisen perintäketjun luokan toiminnan, ymmärtää täysin *Ext.MessageBox*-komponentin toiminnan, koska tietää kaikki sen perityt ja omat ominaisuudet.

Tavallisesti tälläista syvempää tietoa ei tarvita siitä, mitä pinnan alla tapahtuu, mutta jos aikoo kehittää kirjastoa pluginilla, on luonnollisesti tiedettävä mitä tekee. Kun kerran käy läpi esimerkiksi luokan *Ext.AbstractComponent*, on hallussa kaikkien visuaalisten komponenttien perusta. Luokkaa *Ext.Container* taas laajentavat kymmenet Sencha Touch –komponentit (Garcia ym. 2013, 49), joten sen ymmärrettyä hallitsee ison osan Sencha Touch –komponenttien toiminnasta. Luokkien tutkiminen ja niiden toiminnan opettelu eivät siis ole mahdoton tehtävä.

Kaikki Sencha Touch luokat ovat eriteltynä omiin tiedostoihinsa verkosta ladattavaan Sencha Touch –tiedostopakettiin.

3.3 Sencha Touchin ydin

Ymmärtääkseen Sencha Touchin toimintalogiikan on aluksi sukellettava aina sen syvimpään kerrokseen asti: tasolle, jossa enää juuri ja juuri ollaan edes tekemisissä Sencha Touchin kanssa. Seuraavassa luodaan luokka, joka ei laajenna mitään Sencha Touch –luokkaa:

```
Ext.define('PKAMK.human.Proto', {
});
```

Ext.define toiminto toimii luokkamäärittelyn kuorena. Toiminto tarvitsee argumenteikseen luokan nimiavaruuden tekstimuodossa sekä luokan ominaisuudet paketoituna objektin sisälle. Edellisessa on määritelty nimiavaruudeksi *PKAMK.human.Proto*. Ominaisuuksia luokalla ei vielä ole, joten seuraavaksi argumentiksi on määritelty tyhjä objekti.

Suositeltu nimiavaruuden rakenne perustuu kolmeen osaan, jotka on eroteltu pistein (Sencha Inc. 2013). Ensimmäisenä tulee yrityksen, organisaation tai tuotteen nimi. Tässä tapaksessa tähän tulee arvoksi *PKAMK*. Tämä ensimmäinen osa on tavanmukaisesti kirjoitettu isolla etukirjaimella, kuten Senchan *Ext*. Tässä tapauksessa kyse on lyhenteestä ja organisaation nimi kirjoitetaan kokonaan tikkukirjaimin.

Seuraavaksi nimiavaruuteen tulee toiminnallinen ryhmitys. Tässä tapauksessa tähän on lisätty arvo ”human”. Ryhmitys voi koostua useista osista, kuten vaikkapa ”animal.bird.dove”. Jokainen ryhmityksen osa on eroteltu pistein.

Viimeisenä tulee varsinainen luokan nimi. Tässä tapauksessa se on *Proto,* joka viittaa prototyyppiin. Sinänsä melko merkityksetön arvo saa kuitenkin merkityksen toiminnallisen osan ”human” mukaan. Kyseessä on siis ihmisen prototyyppi.

Alaviivan, väliviivan ja numeroiden käyttö nimiavaruudessa ei ole suositeltavaa (Kumar 2012, 78), mutta periaatteessa luokan nimiavaruuden voi rakentaa aivan kuten itse haluaa. Isoin kirjaimin, pienin kirjaimin, ryhmittelyin, yhdellä yksi-

kertaisella sanalla tai vaikka yhdellä kirjaimella, kunhan nimiavaruus ei ole varattu. Luokka toimii joka tapauksessa. Sotkuiset ja epäjohdonmukaiset nimirakenteet antavat kuitenkin huonon kuvan tekijästään ja itse luokasta. Jos nimiavaruus on sotkuinen, luultavasti loputkin luokasta on sotkuista sisältäen todennäköisesti sovellusvirheitä.

```
Ext.define('PKMAK.human.Proto', {

    constructor: function (config) {
        this.initConfig(config);
    }
});
```

Edellisessä luokalle on lisätty konstruktori. Konstruktori pyytää argumentiksi *config*-objektia jonka se asettaa argumentiksi toiminnolle *initConfig*. Tämä saa *config*-objektin toimimaan oikealla tavalla määritellessään alaluokkia tälle luokalle. Kyseessä on hyvin alkeellinen toiminto, josta yleensä ei tarvitse välittää, sillä se on valmiiksi määriteltynä jo hyvin perustason komponenteillakin.

```
Ext.define('PKMAK.human.Proto', {

    config: {
        alive: true,
        age: undefined,
        name: undefined
    },

    constructor: function (config) {
        this.initConfig(config);
    }
});
```

Config-objekti ja sen merkitys on mitä todennäköisemmin jo ennestään tuttua. Olet todennäköisesti tottunut muokkaamaan elementtejä lähinnä config-objektin avulla. Config-objektin luonne ei ole poikkeuksellinen tälle luokalle. Se pitää siis sisällään kaikki luokan muuttujat. Koska luokka ei laajenna mitään toista luokkaa, config-objektin sisälle määritellyt muuttujat ovat ainoita luokan muuttujia. Luokalle määritellään muuttujat *alive*, *age* ja *name*. Näistä muuttuja *alive* saa oletusarvokseen *true* ja muuttujien *age* ja *name* arvot määritellään toistaiseksi määrittelemättömiksi.

Sencha Touch luo automaattisesti luokan muuttujille erilaisia metodeja, kuten getterin, setterin, apply- ja update-metodin. Esimerkiksi *alive* muuttujan getteriksi määritellään automaattisesti *getAlive*. Toiminto palauttaa muuttujan *alive* arvon. Muuttujan *alive* setteriksi määritellään *setAlive*, joka asettaa kyseiselle muuttujalle uuden arvon.

```
getAlive();         // palauttaa true
setAlive(false);
getAlive();         // palauttaa false
```

Getterin ja setterin lisäksi muuttujat saavat apply- ja update-metodit. Apply-metodia kutsutaan aina, kun muuttujan arvoa ollaan vaihtamassa. Metodi antaa mahdollisuuden validoida arvoja tai muuten astua väliin muuttujan arvoa vaihdettaessa.

```
Ext.define('PKMAK.human.Proto', {

   config: {
      alive: true,
      age: undefined,
      name: undefined
   },

   constructor: function (config) {
      this.initConfig(config);
   },

   applyName: function (newValue, oldValue) {
      return newValue;
   }
});
```

Luokalle annetaan siis uusi metodi *applyName.* Sencha Touch kutsuu metodia aina, kun muuttujalle *name* annetaan uusi arvo. Sencha Touch kutsuu metodia kahdella argumentilla. Ensimmäiseksi eli *newValuen* arvoksi se lähettää arvon, joksi muuttujaa ollaan vaihtamassa. Seuraavaksi eli *oldValuen* arvoksi se lähettää arvon, joka muuttujalla oli aiemmin eli tässä tapauksessa arvon *undefined*. Lopulta muuttujalle annetaan arvoksi tämän apply-metodin palautusarvo.

Lopullinen toimintakaava on siis seuraavanlainen: muuttujalle *name* asetetaan uusi arvo käyttämällä metodia *setName.* Olkoon tuo uusi arvo vaikka *Matti.* Sencha Touch kutsuu toimintoa applyName lähettäen muuttujan *newValue* ar-

voksi arvon *Matti* ja muuttujan *oldValue* arvoksi muuttujan *name* edellisen arvon. *ApplyName* metodi palauttaa muuttujan *newValue* arvon eli arvon *Matti*. Arvo *Matti* asetetaan muuttujan *name* arvoksi.

Käytännössä *applyName* metodi ei siis tee mitään tavallisesta poikkeavaa. Vaikka se poistettaisiin, luokka toimisi edelleen samalla tavalla. Metodin *applyName* kuuluisi tehdä jotain muutakin kuin palauttaa muuttujan *newValue* arvo, jotta se olisi perusteltu.

```
Ext.define('PKMAK.human.Proto', {

    config: {
        alive: true,
        age: undefined,
        name: undefined
    },

    constructor: function (config) {
        this.initConfig(config);
    },

    applyName: function (newValue, oldValue) {
        return newValue;
    },

    applyAge: function (newValue, oldValue) {
        return newValue + 5;
    }
});
```

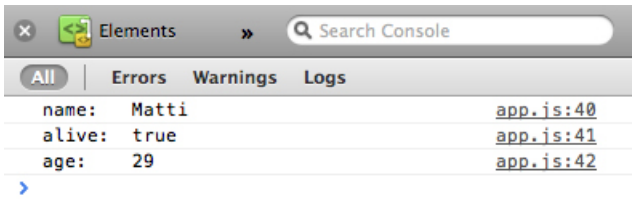
Lisätään apply-metodi muuttujalle *age*. Tämä apply-metodi ei palauta muuttujaa *newValue* vaan ennen palautusta lisää siihen luvun viisi. Tämä tarkoittaa, että aina kun luokalle yritetään asettaa arvoa muuttujalle *age*, arvoksi tuleekin suurempi luku. Jos esimerkiksi yritetään asettaa muuttujan *age* arvoksi lukua 15, muuttujan *age* arvoksi tuleekin 20.

```
Ext.define('PKAMK.human.Person', {
    extend: 'PKAMK.human.Proto',

    config: {
        name: 'Matti',
        age: 24
    }
});
```

Luodaan uusi luokka. Koska tämä luokka laajentaa edellistä luokkaa, on pidettävä huoli, että tämä luetaan vasta ensimmäisen jälkeen. Nimiavaruutena käytetään samaa alkua kuin edellisessä eli *PKAMK.human*, mutta luokkanimeksi annetaan *Person*. Määritellään luokka laajentamaan luokkaa *PKAMK.human.Proto* eli äskettäin luotua luokkaa. Tämä luokka perii siis kaikki edellisen luokan ominaisuudet ja muuttujat arvoineen. Config-objektin sisälle asetetaan muuttujat *name* ja *age*, joille annetaan arvoiksi *Matti* ja *24*.

```
var person = Ext.create('PKAMK.human.Person');

console.log('name:  ' + person.getName());
console.log('alive: ' + person.getAlive());
console.log('age:   ' + person.getAge());
```

Kuten luokkajärjestelmälle on tyypillistä, luokkaa ei voi varsinaisesti käyttää sellaisenaan vaan siitä on luotava instanssi. Tämä tapahtuu komennolla *Ext.create*. Luodaan siis uusi instanssi luokasta *PKAMK.human.Person*. Lopuksi havainnollisuuden vuoksi haetaan instanssin muuttujien arvoja gettereiden avulla ja tulostetaan niitä konsoliin. Lopullinen tuloste näyttää seuraavanlaiselta:



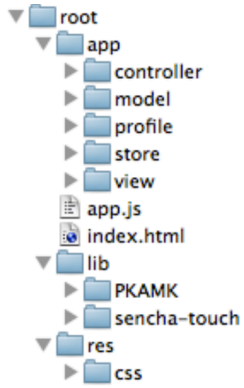
Kuva 4. Tuloste konsolissa.

Muuttujan *name* arvo annettiin luokan *PKAMK.human.Proto* mukaan määrittelemättömäksi, mutta luokka *PKMAK.human.Person* muutti arvon arvoksi *Matti*, joka periytyi myös instanssille. Muuttujan *alive* oletusarvoksi annettiin luokan *PKAMK.human.Proto* mukaan *tosi*. Muuttujaan ei sen jälkeen koskettu, joten se periytyi myös lopulliselle instanssille. Muuttuja *age* määriteltiin luokan *PKAMK.human.Proto* mukaan määrittelemättömäksi, mutta luokka *PKMAK.human.Person* muutti sen arvoksi 24. Luokka *PKAMK.human.Proto* otti arvon vastaan ja lisäsi siihen luvun 5. Lopullinen arvo 29 periytyi instanssille.

3.4 Virtuaalinen työympäristö pluginkokoelmalle

Ennen kuin aloitetaan rakentamaan plugineja, on virtuaalinen työympäristö laitettava kuntoon. Kun rakennetaan projektia MVC-ohjelmistoarkkitehtuurin mukaan, kansion app sisältö on melko lailla määrätty (Sencha Inc, 2013). Tätä vastoin varsinaisen Sencha Touch kirjaston ja tyylitiedoston voi sijoittaa projektiin vapaammin. Yksi siisti keino on lisätä projektin juurikansioon kansiot *res* ja *lib.* Sencha Touch –kirjasto tulee kansioon *lib* ja tyylitiedosto kansioon *res/css*.

Luonnollinen paikka pluginkansiolle on kansion *lib* sisällä. Katso kuva 5. Kansio on hyvä nimetä yrityksen, organisaation tai projektin mukaan. Tässä tapauksessa nimeksi tulkoon ”PKAMK”.



Kuva 5. Plugin kansio.

Seuraavaksi on kerrottava Sencha Touchille plugin kansiosta. Koska tiedosto *app.js* kuuluu jokaiseen projektiin ja se ladataan välittömästi Sencha Touch –kirjaston jälkeen, on se hyvä paikka lisätä seuraava koodi:

```
Ext.Loader.setConfig({
    enable: true,
    paths: {
        'PKAMK': 'lib/PKAMK'
    }
});
```

Varmistetaan, että Loader on päällä asettamalla muuttujan *enabled* arvoksi *tosi*. Asetaan poluksi *lib/PKAMK* käytettäessä nimiavaruudessa arvoa *PKAMK*. Tä-

mä asetus luokkaan *Ext.Loader* on tehtävä tiedoston alkuun, koska se on oltava määriteltynä ennen muita sovelluksen osia.

```
Ext.application({
    name: 'MyApp',
    requires: ['PKAMK.human.Person'],

    launch: function () {
        Ext.create('PKAMK.human.Person');
    }
});
```

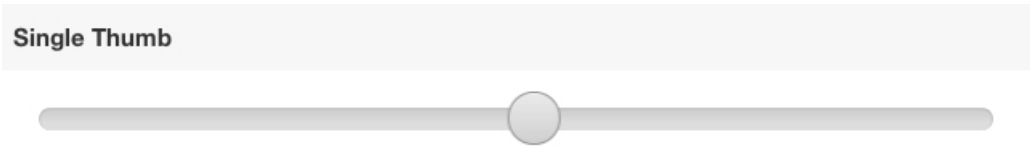
Kun nyt käytämme etuliitettä *PKAMK* viitatessa johonkin luokkaan, tiedostoa haettaessa käytetään juurikansiona kansiota *lib/PKAMK.* Esimerkiksi haettaessa luokkaa nimiavaruudella *PKAMK.human.Person*, Sencha Touch hakee tiedostoa nimeltään *Person.js* polusta */lib/PKAMK/human/*.

On huomattava, että pluginluokka on sisällytettävä muuttujan requires sisälle aina määritellessä uutta pluginia käyttävää komponenttia. Jos tämä jää tekemättä, ohjelma kaatuu kun komponentti yrittää käyttää osaa jota ei ole vielä ehditty ladata.

4 LabelSlider-plugin

4.1 Ratkaisua edeltää ongelma

Plugineja tehdään erilaisista syistä. Ehkä jokin virhe halutaan korjata, ehkä jokin elementti ei omaa tarvittavaa ominaisuutta tai jotain elementtiä ei yksikertaisesti ole olemassa. Ext.field.Slider komponentti on lähinnä lomakkeisiin tarkoitettu liukusäädin. Käyttäjä voi vaakasuunnassa vetää sormellaan liukusäädintä vasemmalle ja oikealle. Komponentti havainnollistettu seuraavassa kuvassa:



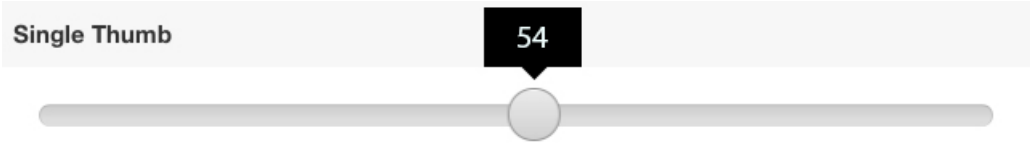
Kuva 6. Ext.field.Slider.

Ongelma komponentissa kuitenkin on, että se ei kommunikoi käyttäjän kanssa. Käyttäjä ei voi tietää, minkä arvon hän on liukusäätimen avulla valinnut, koska komponentti ei anna minkäänlaista ilmoitusta valinnasta.

Älypuhelimissa käyttäjällä ei ole käytössä hiirtä, vaan näyttöä kosketetaan sormella. Valittua arvoa ei siis voi lisätä esimerkiksi liukusäätimen palloon, sillä käyttäjä liikuttaa sitä sormellaan eikä voi nähdä sormensa läpi.

Kuva 7. Samsung keypad.

Älypuhelimien näppäimistöissä on yleisesti käytössä edellisen kuvan osoittama ratkaisu. Kun käyttäjä painaa jotain näppäimistön painikkeista, valinta näytetään hieman näppäimen yläpuolella. Tällä tavoin käyttäjä voi nähdä valintansa. Plugin voisi toimia samalla periaattella. Lisätään siis laatikko liukusäätimen pallon yläpuolelle. Laatikko sisältää tekstin, joka kertoo käyttäjälle valinnastaan.



Kuva 8. Suunnitelma pluginista.

4.2 Projektin aloittaminen

Aloitetaan rakentamalla testiprojekti. Idea on siis mahdollistaa pluginin testaus jokaisen muutoksen jälkeen. Luodaan kansioon *PKAMK* kansio nimeltään *field*. Kansio PKAMK on edellisen kappaleen mukaisesti samassa kansiossa kuin itse Sencha Touch –kirjasto. Kansion *field* sisään luodaan tiedosto nimeltään *LabelSlider.js*.

```
Ext.define('PKAMK.field.LabelSlider', {
    extend: 'Ext.field.Slider',
    xtype: 'labelslider'
});
```

Luodaan pluginille perusrunko käyttäen apuna toimintoa *Ext.define*. Huomaa nimiavaruuden muodon verrannollisuus tiedoston sijaintiin - ne täsmäävät toisiinsa. Määritellään plugin laajentamaan *Ext.field.Slider* luokkaa. Plugin perii näin kaikki Sencha Touchin oman *Ext.field.Slider* komponentin ominaisuudet. Määritellään xtypeksi *labelslider*. Huomaa, että xtypen arvo on kirjoitettu pienillä kirjaimilla. Jos katsoo Sencha Touchin omien komponenttien xtypejä, huomaat, että ne kaikki on kirjoitettu kokonaan pienin kirjaimin.

```
Ext.application({
    requires: [
        'Ext.Toolbar',
        'Ext.form.Panel',
        'PKAMK.field.LabelSlider'
    ],

    launch: function () {
        Ext.create('Ext.form.Panel', {
            fullscreen: true,
            items: [
                {
                    xtype: 'toolbar',
                    docked: 'top',
                    title: 'Label Slider'
                },
                {
                    xtype: 'labelslider',
                    label: 'Valitse luku',
                    value: 50,
                    minValue: 0,
                    maxValue: 100
                }
            ]
        });
```

```
        }
});
```

Seuraavaksi luodaan näkymä, joka sisältää plugin-komponentin. *PKAMK.field.LabelSlider* on muistettava lisätä *requires* muuttujan sisälle. Yksinkertainen sovellus on sellaisenaan toimiva applikaatio, joka koostuu yhdestä *Ext.form.Panel* näkymästä, joka pitää sisällään *Ext.Toolbar* komponentin ja *PKAMK.field.LabelSlider* pluginin. Katso kuva 9.

Kuva 9. Sovellus selaimessa.

Projekti on nyt valmiina aloitettavaksi. Voimme visuaalisesti nähdä kaikki tekemämme muutokset ja lukea konsolista mahdolliset virheilmoitukset jos jokin menee vikaan.

4.3 Ominaisuuksien perintä ja yliajo

Seuraavaksi on selvitettävä, kuinka *Ext.field.Slider* toimii niin sanotusti kulissien takana. Päätehtävä on jotenkin kaapata komennot, jotka mahdollistavat ajaa omia toimintoja alkuperäisten lisäksi käyttäjän raahatessa liukusäädintä.

Sencha Touch ladattavasta tiedostopaketista löytyy kansio nimeltään *src*. Avaa se. Etsi kansio nimeltään *field*. Avaa sekin ja etsi sen sisältä tiedosto nimeltään *Slider.js.* Avaa tämä tiedosto koodieditorissasi.

Kuten voi huomata, tiedosto on rakennettu täsmälleen samalla tavalla kuin mikä tahansa muu Sencha Touch –luokka. Luokka määritellään käyttämällä *Ext.define* komentoa. Se laajentaa luokkaa *Ext.field.Field* ja vaatii luokkaa *Ext.slider.Slider*. Luokka omaa config-objektin, jota seuraa kasa metodeita. Näiden metodeiden joukosta löytyy muutama kiinnostava. Nämä ovat nimeltään *onSliderDragStart, onSliderDrag ja onSliderDragEnd.* Metodeiden nimet paljastavat niiden liittyvän jotenkin liukusäätimen raahaamiseen. Nämä ovat siis toiminnot, jotka kaapataan pluginiin.

```
Ext.define('PKAMK.field.LabelSlider', {
    extend: 'Ext.field.Slider',
    xtype: 'labelslider',

    onSliderDragStart: function () {
        console.log('dragstart');
    },

    onSliderDrag: function () {
        console.log('drag');
    },

    onSliderDragEnd: function () {
        console.log('dragend');
    }
});
```

Kun pluginia testataan selaimessa, konsoliin tulostuu ilmoituksia liukusäädintä raahatessa. Aluksi konsoliin tulostuu ilmoitus raahaamisen aloituksesta, sitten itse raahauksesta ja lopulta raahauksen päättymisestä.

On tärkeää huomioida, että plugin käyttää samannimisiä metodeita sen yläluokan *Ext.field.Slider* kanssa. Tämä tarkoittaa sitä, että määritellyt metodit yliajavat isäntäluokan metodit. Plugin täten siis hajoittaa joitakin isäntäluokan ominaisuuksia. Vaikka liukusäädin näyttäisi toimivan moitteettomasti, jos sille lisätään kuuntelija kuuntelemaan esimerkiksi toimintoa *dragstart*, toiminto ei ikinä laukea. Tämä johtuu siitä, että toimintoa *fireEvent,* josta pluginin yläluokka on vastuussa, ei ikinä ajeta. Onneksi tähän on yksinkertainen ratkaisu.

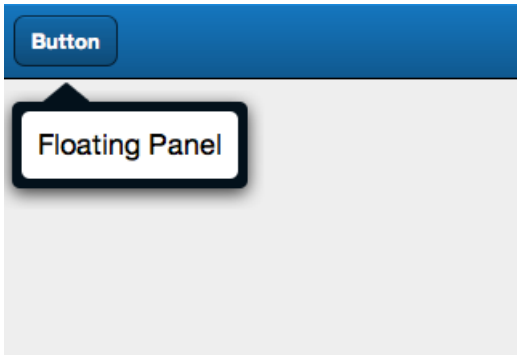
```
Ext.define('PKAMK.field.LabelSlider', {
    extend: 'Ext.field.Slider',
    xtype: 'labelslider',

    onSliderDragStart: function () {
```

```
            this.callParent(arguments);
        },

        onSliderDrag: function () {
            this.callParent(arguments);
        },

        onSliderDragEnd: function () {
            this.callParent(arguments);
        }
    });
```

Kun lisätään rivi *this.callParent(argument)* metodin sisään, se tarkoittaa että yläluokkien samannimiseen metodiin asettamat toiminnot ajetaan. Tämä mahdollistaa komponentin luonnollisen käyttäytymisen, johon on vain lisätty hieman extraa.

4.4 Palikka paikoilleen

Komponentti tarvitsee laatikon liukusäätimen pallon yläpuolelle. Pluginilla ei luonnollisesti vielä ole tällaista laatikkoa tai muuten tälle pluginille ei luultavasti olisi tarvetta.



Kuva 10. Ext.Panel-komponentti ilman fullscreen-tilaa.

Sopivan oloinen palikka pluginille löytyy Sencha Touchin omista komponenteista. *Ext.Panel*-komponentista saa tarkoitukseen sopivan säätämällä leveyttä ja korkeutta sekä määrittelemällä fullscreen-asetuksen *epätodeksi.*

```
Ext.define('PKAMK.field.LabelSlider', {
    extend: 'Ext.field.Slider',
    xtype: 'labelslider',
    requires: ['Ext.Panel'],
```

```
    config: {
        floatingPanel: null
    },

    onSliderDragStart: function () {
        this.callParent(arguments);
    },

    onSliderDrag: function () {
        this.callParent(arguments);
    },

    onSliderDragEnd: function () {
        this.callParent(arguments);
    },

    initialize: function () {
        this.callParent(arguments);
        this.setFloatingPanel(Ext.create('Ext.Panel', {
            width: 60,
            height: 50,
            fullscreen: false,
            padding: '10px 5px',
            style: 'text-align: center',
            showAnimation: 'fadeIn',
            hideAnimation: 'fadeOut'
        }));
    }
});
```

Plugin vaatii nyt luokkaa *Ext.Panel*, koska se aikoo käyttää sitä. Määritellään config-objekti ja sen sisälle uusi muuttuja *floatingPanel*. Annamme toistaiseksi tälle muuttujalle arvoksi *null*, joka vastaa alustamatonta objektia, mutta lopullisen arvon annamme metodin *initialize* sisällä.

Initialize-metodi on lainattu luokalta *Ext.field.Slider*, joka on lainannut sen omalta yläluokaltaan ja lainaus jatkuu aina kaikkien komponenttien yläluokkaan nimeltään *Ext.Component*. Tämä tarkoittaa sitä, että kaikki komponentit omistavat toiminnon *initialize*. Kyseistä toimintoa ajetaan alustaessa komponenttia. Tämä tapahtuu, kun komponentti luodaan, eli tässä tapauksessa ohjelman käynnistyessä.

Initialize-metodin sisällä annetaan siis *floatingPanel*-muuttujalle uusi arvo. Arvoksi luodaan uusi komponentti, joka laajentaa luokkaa *Ext.Panel.* Komponentille annetaan muutamia arvoja parantaakseen käyttäjäkokemusta, mutta oleel-

liset arvot ovat *width* ja *height*. Näillä saadaan komponentti leijumaan muun sisällön päällä, kuten tarkoitus onkin. Lisätään myös muuttuja *fullscreen* ja annetaan sille arvoksi *epätosi*, jolla varmistetaan, että komponentti ei venytä itseään koko ruudun täyttäväksi näkymäksi.

4.5 Rattaat pyörimään

Tähän asti pluginin toimivuus ollaan voitu todeta ainoastaan tulostamalla ilmoituksia konsoliin ja toteamalla, että ohjelma ei ainakaan kaadu vaan lähtee päälle. Mitään visuaalista ei kuitenkaan ole vielä saatu aikaan.

```
onSliderDragStart: function () {
    var thump = this.getComponent().getThumb(),
        value = this.getValue()[0];

    this.callParent(arguments);
    this.getFloatingPanel().showBy(thump, 'bc-tc');
    this.getFloatingPanel().setHtml(value);
},
```

Lisätään toimintoja metodiin *onSliderDragStart*. Aluksi ajetaan muuttujan floatingPanel sisäinen komento *showBy*. FloatingPanel on siis *initialize*-metodin sisällä rakennettu komponentti. Annetaan argumenteiksi liukusäätimen säädinpallo ja teksti *bc-tc*. Tämä tarkoittaa sitä, että *floatingPanel*-komponentti ilmestyy ruutuun siten, että se osoittaa liukusäätimen palloa. Komponentti asettuu pallon päälle tekstillä *bc-tc*, joka on lyhenne ilmaisusta *bottomcenter-topcenter*. Kyseisen ilmaisun voi selittää siten, että komponentin kohta keskellä alhaalla asettuu samaan kohtaan kuin pallon kohta keskellä ylhäällä.

Tämän lisäksi päivitetään floatingPanel muuttujan *html*-arvo. Asetetaan arvoksi se, mikä liukusäätimen sen hetkinen muuttujan *value* arvo on. On huomattava, että liukusäätimen muuttujan *value* arvo on muodoltaan taulukko (array), joten joudumme käyttämään merkkausta [0] saadaksemme taulukon ensimmäisen arvon. Muutamme numeraalisen arvon tekstimuotoon komennolla *toString*.

Kun pluginia nyt kokeillaan selaimessa, näkyvää muutosta on havaittavissa. Aina kun liukusäätimeen tartutaan, ilmestyy Panel-komponentti näkyviin ja siihen ilmestyy liukusäätimen sen hetkinen arvo. Katso kuva 11.

Kuva 11. onSliderDragStart metodin toiminta selaimessa.

Plugin tuottaa siis jo tulosta, mutta on vielä keskeneräinen. Liukusäätimeen ilmestyy laatikko ilmaisemaan liukusäätimen arvoa liukusäätimeen koskiessa, mutta sitä raahatessa laatikko ei päivity. Laatikko jää paikoilleen ilmaisemaan arvoa, joka liukusäätimellä oli painalluksen aloittamisen aikaan, paikkaan jossa koskeminen aloitettiin.

```
onSliderDrag: function () {
    var thump = this.getComponent().getThumb(),
        value = this.getValue()[0];

    this.callParent(arguments);
    this.getFloatingPanel().alignTo(thump, 'bc-tc');
    this.getFloatingPanel().setHtml(value);
},
```

Lisätään toimintoja *onSliderDrag* metodiin. Toiminnot ovat muuten identtiset edellisen *onSliderDragStart* metodin toimintojen kanssa, mutta emme käytä toimintoa *showBy* vaan *alignTo*. Laatikon ei haluta ilmestyvän joka kerta, kun selain laukaisee *drag*-kuuntelijan eli käyttäjä raahaa liukusäädintä, koska laatikko on jo näkyvissä. Riittää kun sitä siirretään paikoilleen. Katso kuva 12 seuraavalla sivulla.

Kuva 12. onSliderDrag metodin toiminta selaimessa.

Nyt laatikko seuraa liukusäätimen palloa käyttäjän raahatessa sitä sormellaan ja sen osoittama luku päivittyy kulloisenkin arvon mukaan. Lopetettaessa raahaminen laatikko jää näkyviin niille sijoilleen.

```
onSliderDragEnd: function () {
    this.callParent(arguments);
    this.getFloatingPanel().hide();
},
```

Lisätään vielä yksi toiminto metodiin *onSliderDragEnd*. Toiminto on yksinkertainen. Se ajaa komponentille *floatingPanel* toiminnon *hide*, joka piilottaa laatikon näkyvistä kun käyttäjä ei raahaa liukusäädintä.

Kuva 13. Pluginin toiminta havainnollistettuna.

4.6 Hienosäätö

Plugin toimii. Pienellä testauksella kuitenkin huomataan, että se sisältää virheen tai pikemminkin puutteen. Kaikki toimii hyvin niin pitkään, kun käyttäjä raahaa

liukusäädintä, mutta jos käyttäjä napauttaa sormellaan jotain kohtaa säätimestä, pallo liukuu sinne. Valitettavasti arvoa ilmaiseva laatikko ei tule näkyviin. Tämä johtuu siitä, että napauttaessa sormellaan liukua jostain kohtaa, metodeja *onSliderDragStart, onSliderDrag* tai *onSliderDragEnd* ei koskaan kutsuta.

Jotta plugin todella olisi valmis, on tämä hoidettava kuntoon. Aluksi on jotenkin kaapattava komento, jota ajetaan liukusäätimen pallon liikkuessa käyttäjän napautettua säädintä. Tässä tapauksessa tehtävä on vaikeampi kuin viimeksi. Oikeantyyppistä metodia ei vaikuttaisi yläluokalla olevan. On siis turvauduttava muuttamaan alkuperäistä isäntäluokan koodia ja katsottava mitä selaimessa tapahtuu.

```
thumbConfig: {
    draggable: {
        translatable: {
            easingX: {
                duration: 300,
                type: 'ease-out'
            }
        }
    }
},
```

Luokan *Ext.slider.Slider config*-objektin sisältä löytyy mielenkiintoinen muuttuja *thumbConfig*. Jos muutat arvoa *duration* luvusta 300 esimerkiksi lukuun 2000, huomaat että animaatio napauttaessa liukusäädintä muttuu hyvin pitkäksi. Tästä kohtaa on siis lähdettävä etsimään. Kaikki alkuperäiseen Sencha Touch –koodiin tekemät muutokset on muistettava palauttaa ennalleen.

Etsitään siis *Ext.slider.Slider* luokasta muuttujan *thump* mukaisen komponentin laajentamaa luokkaa. Luokan toiminto *factoryThumb* paljastaa *thumb* muuttujan laajentavan luokkaa *Ext.slider.Thumb*. Etsitään tämän sisältä muuttujaa *draggabble* ja huomataan sille ainoastaan annettavan asetuksia tässä luokassa. Se on siis luultavasti määritelty tämän luokan isäntäluokassa, mikä on *Ext.Component*. Etsitään siis luokasta *Ext.Component* muuttujaa *draggable*. Huomataan, että *draggable* muuttuja määritellään muuttujan *draggableBehavior* mukaan metodilla *getDraggable*. Muuttuja *draggableBehavior* laajentaa luokkaa *Ext.behavior.Draggable*. Tuolta luokasta etsitään toimintoa *getDraggable*.

Huomataan, että se palauttaa tämän luokan muuttujan *draggable,* joten etsimme sitä. Huomataan, että muuttuja laajentaa luokkaa *Ext.util.Draggable*. Huomataan tuolla luokalla olevan muuttuja *translatable* ja metodin *applyTranslatable* perusteella voidaan päätellä sen laajentavan luokkaa *Ext.util.Translatable*. Ext.util.Translatable paljastuu välitysluokaksi, joka palauttaa joko luokan *Ext.util.translatable.CssTransform* tai luokan *Ext.util.translatable.ScrollPosition.* Näistä kummatkin laajentavat luokkaa *Ext.util.translatable.Abstract*. Vihdoin ollaan luultavasti löydetty ratkaisu ongelmaan, sillä tähän luokkaan voi kiinnittää kuuntelijoita kuuntelemaan tapahtumia *animationframe, animationstart* ja *animationend.*

```
initialize: function () {
    var thumb = this.getComponent().getThumb(),
        draggable = thumb.getDraggable(),
        translatable = draggable.getTranslatable();

    this.callParent();

    translatable.on({
        animationframe: function () {
            console.log('animationframe');
        }
    });

    this.setFloatingPanel(Ext.create('Ext.Panel', {
        width: 60,
        height: 50,
        padding: '10px 5px',
        style: 'text-align: center',
        showAnimation: 'fadeIn',
        hideAnimation: 'fadeOut'
    }));
}
```

Yhteys luokkaa *Ext.util.translatable.Abstract* käyttävään instanssiin on saatava. Kuten aiemmin huomattiin, on kaivettava melko syvältä sen löytääkseen. Aluksi haetaan muuttuja *component*. Kuten aiemmin, se palautta *Ext.slider.Slider* luokan mukaisen komponentin. Tämän sisältä haetaan muuttujaa *thumb,* joka vastaa luokan *Ext.slider.Thumb* mukaista komponenttia. Käytännössä tämä tarkoittaa elementtiä, joka on liukusäätimen liukupallo. Tämän sisältä haetaan muuttujaa *draggable*, joka on siis luokan *Ext.util.Draggable* mukainen. Tämän sisältä haetaan muuttujaa *translatable,* joka on luokan *Ext.util.Translatable* mukainen, joka laajentaa joko luokkaa *Ext.util.translatable.CssTransform* tai luokkaa

Ext.util.translatable.Scrollposition. Nämä kummatkin perii arvonsa luokalta *Ext.util.translatable.Abstract*, joka hyväksyy muun muassa kuuntelijan *animationframe*, joka nimenomaan siihen lisätään.

Kun tätä testataan selaimessa, konsoliin tulostuu teksti *animationframe* aina napauttaessa liukusäädintä jostain kohtaa ja liukusäätimen pallon liukuessa kohti painalluskohtaa. Plugin voidaan viimeistellä.

```
onSliderMoveStart: function () {
    var thump = this.getComponent().getThumb(),
        value = this.getValue()[0];

    this.getFloatingPanel().showBy(thump, 'bc-tc');
    this.getFloatingPanel().setHtml(value);
},


onSliderMove: function () {
    var thump = this.getComponent().getThumb(),
        value = this.getValue()[0];

    this.getFloatingPanel().alignTo(thump, 'bc-tc');
    this.getFloatingPanel().setHtml(value);
},


onSliderMoveEnd: function () {
    this.getFloatingPanel().hide();
},
```

Lisätään kolme metodia, jotka ovat samankaltaiset aiempien toimintojen *onSliderDragStart, onSliderDrag* ja *onSliderDragEnd* kanssa. Ainoa ero nimen lisäksi on se, että kustakin puuttuu rivi *this.callParent(arguments)*. Tämä johtuu siitä, että pluginin isäntäluokilla ei ole tämän nimisiä toimintoja. Kyse ei ole siis ominaisuuksien ylikirjoittamisesta vaan niiden lisäämisestä.

```
initialize: function () {
    var thumb = this.getComponent().getThumb(),
        draggable = thumb.getDraggable(),
        translatable = draggable.getTranslatable();

    this.callParent();

    translatable.on({
        scope: this,
        animationstart: 'onSliderMoveStart',
        animationframe: 'onSliderMove',
```

```
            animationend:   'onSliderMoveEnd'
        });

        this.setFloatingPanel(Ext.create('Ext.Panel', {
            width: 60,
            height: 50,
            padding: '10px 5px',
            style: 'text-align: center',
            showAnimation: 'fadeIn',
            hideAnimation: 'fadeOut'
        }));
    }
```

Lisätään kuuntelija *animationstart*, *animationframe* ja *animationend* tapahtumiin ja linkitetään kuutelijoiden kutsu vasta luotuihin metodeihin. Plugin toimii nyt kuten kuuluukin niin raahatessa kuin myös käyttäjän napauttaessa liukusäädintä.

5 Flipper-plugin

5.1 Kirjaston rajaton laajentaminen

Pluginin ei tarvitse rajoittua vain näkymiin ja komponentteihin, vaikka sellaiset pluginit ovatkin suhteellisen yksinkertaisia tehdä. Voit tehdä tarvittaessa kustomoidun kontrollerin, mallin tai minkä tahansa Sencha Touch –kirjaston osan. Tässä kappaleessa luodaan uusi animaatio.

Sencha Touch –animaatiot perustuvat CSS-transitioihin. Näille on ominaista, että ainoastaan alku- ja loppupiste on annettu. Selain hoitaa animoinnin näiden pisteiden välillä. Jos selaimelta löytyy tuki, Sencha Touch pyrkii käyttämään 3D-animointeja. 3D-transitiot ovat raskaita animaatioita, mutta koska ne pystyvät useilla mobiililaitteilla hyödyntämään laitteistokiihdytystä (Gasston 2011, 179), ne lopulta toimivat muita animointikeinoja jouhevammin.

5.2 Raaka-animaatio

Lisätään kansioon *PKAMK* kansio nimeltään *fx.* Tämän kansion sisälle sijoitetaan kaksi kansiota: *animation* ja *layout.* Kansion *layout* sisällä lisätään vielä kansio *card.* Luodaan kansioon *animation* tiedoston nimeltään *Flipper.js*.

Koodista tulee melko pitkä, sillä meidän pitää sijoittaa useita eri animaatiolajeja. Näitä ovat animaatio sisään oikealta vasemmalle, animaatio sisään vasemmalta oikealle, animaatio ulos oikealta vasemmalle ja animaatio ulos vasemmalta oikealle.

```
Ext.define('PKAMK.fx.animation.Flipper', {
    extend: 'Ext.fx.animation.Abstract',
    alias: 'animation.flipper',

    config: {
        easing: 'ease-in',
        out: false,
        direction: 'right'
    },

    applyDirection: function (newValue, oldValue) {

    },

    getData: function () {

    }
});
```

Aloitetaan laajentamalla luokkaa *Ext.fx.animation.Abstract*. Näin saadaan käyttöömme kaikki animaation perusasetukset ja toiminnot. Annetaan animaatiolle alias *animation.flipper.* Tämä helpottaa animaation kutsumista jatkossa. Sitten annamme animaatiolle oletusarvoina kiihtyvyyden hitaasta kiihtyväksi antamalla muuttujalle *easing* arvon ”ease-in” ja tyypiksi sisääntulon antamalla muuttujalle *out* arvon *epätosi.* Animaation oletussuunnaksi annamme suunnan vasemmalta oikealle.

```
applyDirection: function (newValue, oldValue) {

    if (newValue === this.DIRECTION_UP) {
        newValue = this.DIRECTION_RIGHT;
    }

    if (newValue === this.DIRECTION_DOWN) {
        newValue = this.DIRECTION_LEFT;
    }

    return newValue;
},
```

Tätä metodia kutsutaan Sencha Touchin toimesta silloin, kun animaation suuntaa ollaan asettamassa. Lisätään rivit varokeinoksi siltä varalta, että animaatio jostain syystä yritetään asettaa pystysuuntaiseksi. Toiminto nappaa kaikki käskyt, jotka pyrkivät asettamaan animaatiota pystysuuntaiseksi ja muuttaa käskyn siten, että animaatio onkin vaakasuuntainen.

```
getData: function () {
    var from            = this.getFrom(),
        to              = this.getTo(),
        direction       = this.getDirection(),
        reverse         = this.getReverse(),
        out             = this.getOut(),
        origin          = '',
        fromOpacity     = 0,
        toOpacity       = 0,
        fromRotationZ   = 0,
        toRotationZ     = 0,
        fromTranslateX  = 0,
        toTranslateX    = 0,
        fromTranslateY  = 0,
        toTranslateY    = 0;
```

Aloitetaan metodi *getData* lisäämällä siihen kaikki tarvittavat muuttujat. Tämä metodi muodostaa varsinaisen animaation. Muuttujat *from* ja *to* vastaavat

eräänlaisia animaatio-objekteja. Sencha Touchin käyttämät CSS-transitiot toimivat siten, että animaatiolle annetaan alkupisteen sekä loppupisteen tyyliarvot ja selain hoitaa animoinnin näiden pisteiden välillä. Muuttuja *from* pitää sisällään kaikki animoitavan komponentin alkupisteen tyyliarvot ja *to* pitää sisällään kaikki animoitavan komponentin tyyliarvot animaation lopussa. Muuttujat *direction, reverse* ja *out* pitävät sisällään animaation suunnan, käännetyn animaation totuusarvon ja sisääntulon tai ulosmenon kääntimen. Muuttuja *origin* pitää sisällään animaation painopisteen.

Kullekin tyyliarvon muuttujalle annetaan ainoastaan oletusarvo. Lopulliset arvot määräytyvät myöhemmin animaation lajin mukaan, eli esimerkiksi sen, onko animaation suunta vasemmalta oikealle vai oikealta vasemmalle.

```
if (direction === this.DIRECTION_LEFT) {
    reverse = !reverse;
}
```

Tällä yksikertaisella komennolla säästetään monta koodiriviä. Jos animaation suunta on oikealta vasemmalle, muutetaan animaation kääntöarvo vastakkaiseksi itsensä kanssa. Näin muuttujasta *direction* ei tarvitse enää huolehtia, vaan ainoastaan muuttujasta *reversed.* Jos muuttuja *reversed* on *tosi*, animaatio on suunnaltaan oikealta vasemmalle. Jos muuttuja *reversed* on *epätosi*, animaatio on suunnaltaan vasemmalta oikealle.

```
if (reverse) {
    if (out) {
        origin          = '0% 10%';
        fromOpacity     =  1;
        toOpacity       =  0;
        fromRotationZ   =  0;
        toRotationZ     =  10;
        fromTranslateX  =  0;
        toTranslateX    = -100;
        fromTranslateY  =  0;
        toTranslateY    =  50;
```

```
        } else {
            origin          = '100% 10%';
            fromOpacity     =  0;
            toOpacity       =  1;
            fromRotationZ   = -10;
            toRotationZ     =  0;
            fromTranslateX  =  100;
            toTranslateX    =  0;
            fromTranslateY  =  20;
            toTranslateY    =  0;
        }

    } else {
        if (out) {
            origin          = '100% 10%';
            fromOpacity     =  1;
            toOpacity       =  0;
            fromRotationZ   =  0;
            toRotationZ     = -10;
            fromTranslateX  =  0;
            toTranslateX    =  100;
            fromTranslateY  =  0;
            toTranslateY    =  50;

        } else {
            origin          = '0% 10%';
            fromOpacity     =  0;
            toOpacity       =  1;
            fromRotationZ   =  10;
            toRotationZ     =  0;
            fromTranslateX  = -100;
            toTranslateX    =  0;
            fromTranslateY  =  20;
            toTranslateY    =  0;
        }
    }
```

Lisätään erilaisia tyylipaketteja eri tilanteisiin. Ensin tarkastetaan, onko animaatio käännetty animaatio. Tämä karsii toiminnoista puolet pois. Sen jälkeen tar-

kastetaan, onko animaatio sisäänpäin tuleva vai ulospäin menevä. Tämä karsii taas toiminnoista puolet pois ja lopulta jäljelle jää oikeanlainen tyylipaketti kullekin tilanteelle.

Kun asetetaan muuttujan *origin* arvoksi esimerkiksi ’0 10%’, tarkoittaa se että kun animaatio pyörittää objektia, pyörityksen painopiste on objektin vasemmassa laidassa 10 prosenttia ylälaidasta alaspäin. Läpinäkyvyyden arvoksi voidaan antaa lukuja nollasta yhteen, jossa nolla on läpinäkyvä ja yksi on täysin näkyvä. Pyöritykselle annetaan arvot asteina ja liikkeelle annetaan arvot pikseleinä.

```
from.setTransform({
    rotateZ: fromRotationZ,
    translateX: fromTranslateX,
    translateY: fromTranslateY
});

to.setTransform({
    rotateZ: toRotationZ,
    translateX: toTranslateX,
    translateY: toTranslateY
});

from.set('transform-origin', origin);
to.set('transform-origin', origin);

from.set('opacity', fromOpacity);
to.set('opacity', toOpacity);

return this.callParent(arguments);
```

Muuttujat ovat tässä vaiheessa valmiina käytettäviksi ja viimeiseksi jää animaation käynnistys. Annetaan animaatiolle lähtöarvot toiminnolla *from.setTransform.* Annetaan arvot pyöritykselle ja liikkeelle. Tehdään sama asettamalla arvot animaation loppuarvoille toiminnolla *to.setTransform.* CSS-tyylit *transform-origin* ja *opacity* pitää asettaa animaatioille erikseen. Lopuksi palautamme toiminnon kutsua isäntäluokan vastaavaa metodia. Raaka-

animaatio on valmis. Seuraavassa vielä kaikki tähän asti tehty koodi sellaisenaan.

```
Ext.define('PKAMK.fx.animation.Flipper', {
    extend: 'Ext.fx.animation.Abstract',
    alias: 'animation.flipper',

    config: {
        easing: 'ease-in',
        out: false,
        direction: 'right'
    },

    applyDirection: function (newValue, oldValue) {

        if (newValue === this.DIRECTION_UP) {
            newValue = this.DIRECTION_RIGHT;
        }

        if (newValue === this.DIRECTION_DOWN) {
            newValue = this.DIRECTION_LEFT;
        }

        return newValue;
    },

    getData: function () {
        var from            = this.getFrom(),
            to              = this.getTo(),
            direction       = this.getDirection(),
            reverse         = this.getReverse(),
            out             = this.getOut(),
            origin          = '',
            fromOpacity     = 0,
            toOpacity       = 0,
            fromRotationZ   = 0,
            toRotationZ     = 0,
            fromTranslateX  = 0,
            toTranslateX    = 0,
```

```
    fromTranslateY  = 0,
    toTranslateY    = 0;

if (direction === this.DIRECTION_LEFT) {
    reverse = !reverse;
}

if (reverse) {
    if (out) {
        origin          = '0% 10%';
        fromOpacity     =  1;
        toOpacity       =  0;
        fromRotationZ   =  0;
        toRotationZ     =  10;
        fromTranslateX  =  0;
        toTranslateX    = -100;
        fromTranslateY  =  0;
        toTranslateY    =  50;

    } else {
        origin          = '100% 10%';
        fromOpacity     =  0;
        toOpacity       =  1;
        fromRotationZ   = -10;
        toRotationZ     =  0;
        fromTranslateX  =  100;
        toTranslateX    =  0;
        fromTranslateY  =  20;
        toTranslateY    =  0;
    }

} else {
    if (out) {
        origin          = '100% 10%';
        fromOpacity     =  1;
        toOpacity       =  0;
        fromRotationZ   =  0;
        toRotationZ     = -10;
        fromTranslateX  =  0;
```

```
                toTranslateX    =  100;
                fromTranslateY  =  0;
                toTranslateY    =  50;

            } else {
                origin          = '0% 10%';
                fromOpacity     =  0;
                toOpacity       =  1;
                fromRotationZ   =  10;
                toRotationZ     =  0;
                fromTranslateX  = -100;
                toTranslateX    =  0;
                fromTranslateY  =  20;
                toTranslateY    =  0;
            }
        }

        from.setTransform({
            rotateZ: fromRotationZ,
            translateX: fromTranslateX,
            translateY: fromTranslateY
        });

        to.setTransform({
            rotateZ: toRotationZ,
            translateX: toTranslateX,
            translateY: toTranslateY
        });

        from.set('transform-origin', origin);
        to.set('transform-origin', origin);

        from.set('opacity', fromOpacity);
        to.set('opacity', toOpacity);

        return this.callParent(arguments);
    }
});
```

5.3 Animaation isäntä

Luodaan uusi tiedoston kansioon *card*, joka löytyy siis aiemmin tehdystä kansiosta *layout*. Nimetään tiedosto samalla nimellä kuin edellinen eli *Flipper.js*. Tämän tiedoston on tarkoitus hallinnoida edellistä.

```
Ext.define('PKAMK.fx.layout.card.Flipper', {
    extend: 'Ext.fx.layout.card.Style',
    alias: 'fx.layout.card.flipper',

    requires: ['PKAMK.fx.animation.Flipper'],

    config: {
        duration: 500,

        inAnimation: {
            type: 'flipper',
            easing: 'ease-out',

            before: {
                'backface-visibility': 'hidden',
                'z-index': 0
            },

            after: {
                'backface-visibility': null,
                'z-index': 100
            }
        },

        outAnimation: {
            type: 'flipper',
            easing: 'ease-in',

            before: {
                'backface-visibility': 'hidden',
                'z-index': 100
```

```
            },

            after: {
                'backface-visibility': null,
                'z-index': 0
            },

            out: true
        }
    },

    updateDuration: function(duration) {
        var thirdDuration = duration / 3,
            inAnimation = this.getInAnimation(),
            outAnimation = this.getOutAnimation();

        inAnimation.setDelay(thirdDuration);
        inAnimation.setDuration(thirdDuration * 2);
        outAnimation.setDuration(thirdDuration * 2);
    }
});
```

Aloitetaan laajentamalla luokkaa *Ext.fx.layout.card.Style.* Annetaan tälle luokalle alias *fx.layout.card.flipper.* Tämä tarkoittaa, että animaatiota voidaan kutsua mistä tahansa näkymästä asettaen sen *animation* arvoksi teksti *flipper*. Seuraavaksi vaaditaan aiemmin tehtyä *Ext.plugins.fx.animation.Flipper* luokkaa. Aloitetaan *config*-objekti määrittelemällä animaatiolle oletuskesto. Arvo annetaan millisekuntteina. Määritellään kaksi objektia *inAnimation* ja *outAnimation.* Animaation tyypiksi kummassakin asetetaan arvo ”flipper”. Arvo käyttää edellisen tiedoston aliasta hyväkseen. Asetetaan sisääntulo jarruttavaksi ja ulosmeno kiihdyttäväksi.

Lisätään kummassakin muuttujat *before* ja *after,* joiden sisään asetetaan tyyliasetuksia ennen animaatiota ja animaation jälkeen. Tyyli *backface-visibility* tarkoittaa sitä, onko 3D-tason kääntöpuoli näkyvä vai piilotettu. Animaation tasojen kääntöpuoli ei ikinä tule näkyviin, mutta asetetaan silti animaation ajaksi arvo piilotetuksi. Tyyli estää bugin android-älypuhelimissa, joka aiheuttaa val-

koisen välkehdinnän animaation aikana. Toinen tyyliarvo on *z-index*, joka tarkoittaa sitä millä tasolla elementti on. Suuremman *z-indexin* omaava elementti on piirretään ruutuun pienemmän *z-indexin* omaavan elementin päälle.

Tämän tyyppisessä animaatiossa on yleensä kyse kahden elementin yhtäaikaisesta animoinnista. Esimerkiksi Ext.tab.Panel-elementin vaihtaessa näkymää, toinen näkymä lähtee pois näkyvistä, kun toinen taas tulee samaan aikaan tilalle. Tässä tilanteessa ei kuitenkaan haluta animaatioiden tapahtuvat täysin yhtäaikaisesti. Muutetaan siis animaation kestoasetuksia heti kun Sencha Touch on ne aluksi itse asettanut. Lisätään metodi *updateDuration*. Otetaan asetettu aika, jaetaan se kolmeen osaan, lisätään sisääntulevalle animaatiolle pieni viive ja asetataan kummankin animaation kestoksi kaksi kolmasosaa animoitavasta ajasta.

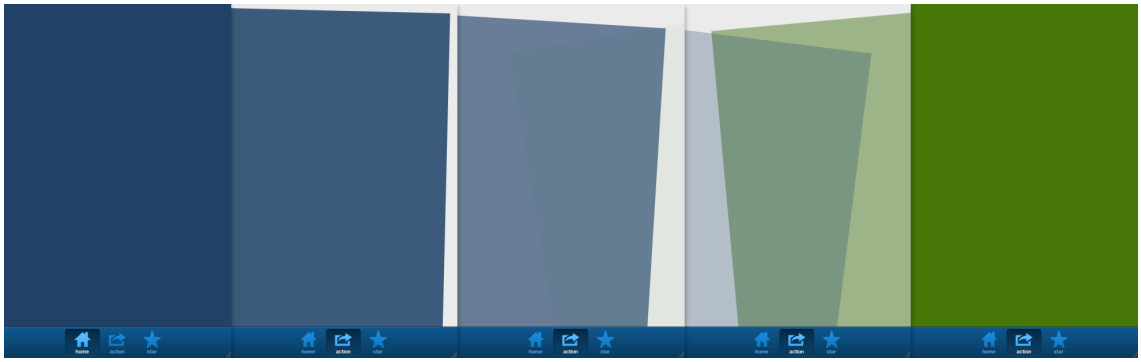
5.4 Animaation käyttö näkymässä

Animaatio on valmis käytettäväksi. Luodaan yksinkertainen näkymä, jotta voidaan nähdä animaatio käytännössä.

```
Ext.Loader.setConfig({
    enabled: true,
    paths: {
        'PKAMK': 'lib/PKAMK'
    }
});

Ext.application({
    requires: [
        'Ext.tab.Panel',
        'Ext.Panel',
        'PKAMK.fx.layout.card.Flipper'
    ],

    launch: function() {
        Ext.create("Ext.tab.Panel", {
```

```
                fullscreen: true,
                tabBarPosition: 'bottom',

                layout: {
                    animation: 'flipper'
                },

                items: [
                    {
                        xtype: 'panel',
                        iconCls: 'home',
                        title: 'home',
                        style: 'background-color: #489590'
                    },
                    {
                        xtype: 'panel',
                        iconCls: 'action',
                        title: 'action',
                        style: 'background-color: #959048'
                    },
                    {
                        xtype: 'panel',
                        iconCls: 'star',
                        title: 'star',
                        style: 'background-color: #904895'
                    }
                ]
            });
        }
    });
```

Kyseessä on siis tavallinen Ext.tab.Panel-näkymä, joka sisältää kolme eri taustavärein varustettua paneelia. Huomaa, että näkymä vaatii animaatioluokkaamme *PKAMK.fx.layout.card.Flipper.* Huomaa myös, että animaatioarvoksi on asetettu ”flipper”. Animaatio toimii kuin itsestään. Voimme antaa sille mitä tahansa tavallisen Sencha Touch –animaation arvoja, kuten animaation keston. Voimme käyttää animaatiota jatkossa missä tahansa projektissa kopioimalla tiedostot projektista toiseen ja kutsumalla animaatiota nimeltään *flipper.*



Kuva 12. Flipper-animaatio toiminnassa.

6 Pohdintaa Sencha Touchista

6.1 Mihin Sencha Touch on tarkoitettu?

Sencha Touch tukee *Webkitillä* varustettuja selaimia. Webkit on vapaaseen lähdekoodiin perustuva selainmoottori, jota käyttää esimerkiksi Apple, Google ja Nokia (Webkit 2013). Täten esimerkiksi Applen *Safari* ja Googlen *Chrome* tukevat Sencha Touchia. Kaikki selaimet eivät perustu Webkitiin. Esimerkiksi Opera perustuu selainmoottoriin nimeltään *Presto* (Opera 2013) ja Firefox selainmoottoriin nimeltään *Gecko* (Mozilla 2013). Näillä selaimilla Sencha Touch ei toimi.

Jos tarkoituksena on tehdä sovellus verkkoon, osa käyttäjistä ei pysty käyttämään Sencha Touch –sovellusta. IPhonen ja Androidin käyttäjiä voi ohjeistaa käyttämään laitteensa vakioselainta, mutta tämä voi ajaa osan käyttäjistä pois. Tälläisessa tilanteessa voi olla aihetta harkita jonkin muun kirjaston käyttöä. Esimerkiksi *jQuery Mobile* lupaa selaimesta riippumattoman tuen (jQuery 2013).

IPhonen, Androidin ja BlackBerryn vakioselaimet perustuvat Webkittiin. Jos tarkoituksena on luoda natiivi sovellus *webviewin* kautta, Sencha Touch tukee näitä kaikkia laitteita. Jos tarkoituksena on tehdä natiivi sovellus HTML5:n avulla, Sencha Touch on hyvä valinta.

6.2 Tehokkuus

Sovelluksien teko mobiililaitteille tuo mukanaan haasteita, koska nämä laitteet eivät ole yhtä tehokkaita kuin tietokoneet. Sovelluksien teko mobiiliin HTML5:n avulla on vielä vaikeampaa, koska tämä tekniikka vaatii huomattavasti enemmän laitteistolta kuin natiivit sovellukset. Mobiililaitteille on olemassa visuaalisesti todella näyttäviä applikaatioita, jotka sisältävät jopa kolmiuloitteista grafiikkaa. Nämä sovellukset on kuitenkin rakennettu natiivein koodikielin. Esimerkiksi Android-sovellukset rakennetaan tavanmukaisesti käyttäen Javaa (Android Developers 2013). Näillä sovelluksilla on täydet oikeudet laitteistokiihdytettyihin grafiikoihin, mikä mahdollistaa huomattavasti monimutkaisemman visuaalisuuden. HTML5:n avulla tehtyjen sovellusten oikeudet laitteistokiihdytettyyn grafiikkaan on rajalliset ja joissain tapauksissa oikeuksia ei ole. Esimerkiksi Adroid käyttöjärjestelmän mobiililaitteet mahdollistavat laittestokiihdytyksen vasta alkaen käyttöjärjestelmäversiosta 3.0, jolloin tuki 3D-transitiolle aloitettiin.

Sencha Touch käyttää kaiken mahdollisen tuen laitteistolta saadakseen aikaan sulavat animaatiot ja hyvän käyttökokemuksen. Uusimmilla mobiililaitteilla Sencha Touch –sovellus ei enää näkyvästi juuri eroa natiiveihin koodeihin perustuvista sovelluksista.

6.3 Sencha Touch vastaan puhdas javaScript

Puhtaan javaScriptin avulla kirjoitettu sovellus toimii luonnollisesti paremmin kuin kirjaston avulla luotu sovellus. Tämä johtuu siitä, että kirjastot joutuvat ottamaan huomioon kaikki mahdolliset tilanteet ja mahdollisuudet. Puhtaan javaScriptin avulla taas on mahdollista kirjoittaa täysin optimoitu sovellus, jossa ei ole mitään ylimääräistä.

Mobiilioppaiden kaltaiset sovellukset eivät ole kovin monimutkaisia, joten kustannustehokkaasti puhtaan javaScriptin käyttö ei ole mahdoton valinta. Sencha

Touch vaatii paljon opettelua, joten voi olla jopa nopeampaa käyttää puhdasta javaScriptiä, jos se on jo ennestään tuttu.

Miksi siis käyttää Sencha Touchia, jos puhdas javaScript on siihen nähden yliverto? Mobiililaitteet ovat vielä suhteellisen uusi asia. Käyttöjärjestelmät ja niiden selaimet ovat jatkuvasti muuttuvia ja niissä on myös suunnitteluvirheitä. Jos nykyään verkkokehittäjät valittavat vanhoista Internet Explorer –selaimista, joilla mikään ei tunnu toimivan kunnolla, mobiililaitteiden parissa tilanne on moninverroin pahempi. Hyvä keino todeta asia, on luoda yksikertainen sovellus, jossa on alaosassa navigointivalikko ja sisältöalueessa listanäkymä. Navigaatiovalikon tulee pysyä paikoillaan ja listanäkymää voi vierittää korkeussuunnassa.

Mahdollisesti ensimmäinen mieleen tuleva keino on pitää navigaatiovalikko paikoillaan suhteessa näyttöön, on asettaa sille tyylittely:

```
position: fixed;
bottom: 0;
```

Tämän pitäisi toimia siten, että valikko on nyt kiinnitetty tiettyyn kohtaan suhteessa näyttöön. Valitettavasti läheskään kaikki nykyään käytettävät mobiililaitteet eivät tue näitä tyylejä (Can I use, 2013). Joissakin mobiililaitteissa tuki saattaa löytyä, mutta se on vajanainen, jolloin navigaatiovalikko pomppii eikä pysy paikoillaan.

Seuraava vaihtoehto on rakentaa kaksi aluetta: yksi navigaatiovalikolle ja toinen listanäkymälle. Listanäkymälle määritellään oma sisäinen vierityksensä ja kummatkin kiinnitetään paikoilleen. Käytetään siis seuraavanlaista tyylittelyä:

```
#list {
    position: fixed;
    height: 80%;
    top: 0;
    overflow-y: scroll;
}

#navigation {
    position: fixed;
    height: 20%;
    bottom: 0;
}
```

Tämä toimii hyvin jopa vanhoilla Andoid laitteilla. Palaset pysyvät paikoillaan ja listan vieritys sujuu hyvin. Kun sovellusta testataan vanhemmilla iPhoneilla, huomataan, että listanäkymää ei voi vierittää. Tämä johtuu siitä, että vanhojen iOS käyttöjärjestelmien (ennen versiota iOS5) mukaan alueen sisäistä vieritystä kuuluisi käyttää kahdella sormella, joten tämäkään vaihtoehto ei ole toimiva. Jäljelle jää siis ratkaista ongelma javaScriptin avulla. On kirjoitettava koodi tarkailemaan ruudun painallusta ja vierityksen tunnistamista. On kirjoitettava koodi itse vieritykselle, animoinnille ja hidastukselle. Vierytystä on rajoitettava, ettei se mene listan tai näkymän yli ja kaikki tämä on saatava toimimaan tehokkaasti. Tämä vaatii vähitään satojen rivien verran javaScriptiä, joka on testattava kaikilla laitteilla. Sencha Touch tarjoaa nämä ominaisuudet valmiina ja edellä kuvattua huomattavasti kehittyneempänä.

Esimerkki osoittaa vain jäävuoren huipun. Tästä syystä sovelluksen tekoa puhtaan javaScriptin avulla on hyvä miettiä tarkkaan. Koska Sencha Touch on avoimeen lähdekoodiin perustuva, sitä testaavat ja kehittävät lukemattomat määrät ihmisiä. Sitä on testattu lukemattomilla eri laitteilla ja sudenkuoppia korjaillaan koko ajan.

Lähteet

Android Developers. 2013. App Components. http://developer.android.com/guide/components. 24.2.2013.

Can I use. 2013. Can I use CSS position:fixed?. http://caniuse.com/css-fixed. 24.2.2013.

Crockford, D. 2008. JavaScript: The Good Parts. Sebastopol: O'Reilly Media, inc.

Garcia, J., De Moss, A. & Simoens, M. 2013. Sencha Touch in Action version 10. Shelter Island: Manning Publications Co.

Gasston, P. 2011. The Book of CSS3. San Francisco: No Starch Press Inc.

HTML5rocks. 2012. Why HTML5 rocks. http://www.html5rocks.com/en/why. 30.4.2013.

Kernighan, B. & Ritchie, D. 1988. The C programming language. New Jersey: AT&T Bell Laboratories.

Kumar, A. 2012. Sencha MVC Architecture. Birmingham: Packt Publishing Ltd.

Mozilla developer network. 2013. Gecko. https://developer.mozilla.org/en-US/docs/Mozilla/Gecko. 24.2.2013.

Opera Software ASA. 2013. Web specifications support in Opera Presto 2.12. http://www.opera.com/docs/specs/presto2.12. 24.2.2013.

Sencha Inc. 2012. Sencha Touch 2.1.1 Docs. http://docs.sencha.com/touch/2-0. 28.3.2013.

The jQuery Foundation. 2013. jQuery Mobile. http://jquerymobile.com. 24.2.2013.

PHP. 2013. What is PHP? http://www.php.net/manual/en/intro-whatis.php. 30.4.2013.

W3C. 1999. HTML 4.01 Specification. http://www.w3.org/TR/html4/cover.html. 30.4.2013.

The Webkit Open Source Project. 2013. http://www.webkit.org. 24.2.2013.

Weiss, A. 2013. 5 Ways to Make Money With Open Source Software. http://opensource.about.com/od/basics/tp/5-Ways-To-Make-Money-With-Open-Source-Software.htm. 24.2.2013.

Wordpress. 2013. About Wordpress. http://wordpress.org/about/. 30.4.2013.