UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Goran Horvat

# IZDELAVA SPLETNEGA PORTALA V EXT JS 3.4

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE

RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Mira Trebar

Ljubljana, 2012

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00227/2012
Datum: 02.04.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **GORAN HORVAT**

Naslov: **IZDELAVA SPLETNEGA PORTALA V EXT JS 3.4**
**IMPLEMENTATION OF WEB PORTAL IN EXT JS 3.4**

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Za izdelavo bogatih spletnih aplikacij se pogosto uporablja Ext JS, ki predstavlja eno od tehnologij zapisanih v jeziku Java Script. Predstavite še druga potrebna orodja in postopke pri izdelavi spletnega portala. Kot rezultat diplomskega dela pa opišite razvoj in izdelavo portala za poslovno upravljanje v podjetju z uporabo orodja Ext JS 3.4.

Mentor:
doc. dr. Mira Trebar

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

# IZJAVA O AVTORSTVU

**diplomskega dela**

Spodaj podpisani **Goran Horvat**,

z vpisno številko 63070423,

sem avtor diplomskega dela z naslovom:

**IZDELAVA SPLETNEGA PORTALA V EXT JS 3.4**

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Mire Trebar,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ 2012 Podpis avtorja:

# Zahvala

Iskreno se zahvaljujem mentorici doc. dr. Miri Trebar za strokovno pomoč in usmerjanje pri izdelavi diplomske naloge.

Zahvaljujem se tudi podjetju Klika d.o.o., v okviru katerega sem imel možnost uspešno opraviti strokovni del diplomskega dela, ter sodelavcem in prijateljem za vso podporo in pomoč.

Posebna zahvala gre družini in Katarini za spodbudo in potrpežljivost v času študija.

Navsezadnje pa zahvalo posvečam tudi zdravniku, ki me je usmeril na študij računalništva.

# Povzetek

Diplomsko delo predstavlja uporabo ogrodja Ext JS, ki je za uporabnika relativno enostavno, istočasno pa omogoča podjetjem uspešno poslovanje. Z njim smo izdelali bogato spletno aplikacijo, portal. Ogrodje Ext JS spada v sam vrh informacijskih tehnologij napisanih v JavaScript jeziku in se lepo dopolnjuje s HTML in CSS kodo. Predvsem znanje JavaScript-a nam je precej olajšalo delo z ogrodjem in nas pripeljalo do zastavljenega cilja s strani naročnika. Naročnik nam je specifično podal zahteve za izgled in funkcionalnost portala, še posebej za njihove večinske uporabnike, tako imenovane klicne agente. V diplomskem delu na razumljiv način opišemo potek prenosa datotek ogrodja iz svetovnega spleta na razvojno enoto, ter izbiro primernih datotek glede na znanje razvijalca. Razvoja aplikacije se lotimo z uporabo komponent ogrodja Ext JS, njihovo razširitev in medsebojno odvisnost ustvarjenih komponent. Nato prikažemo pomen portala, ki smo ga ustvarili za njegove uporabnike, ki so lahko klicni agentje, nadzorniki ali administratorji. Namen ustvarjenega portala je poslovno upravljanje, od pisanja delovnih nalog, urejanja veščin uporabnikov ter ostalih klicnih nastavitev, pa vse do kreiranja kampanj, ki klicočim agentom pripomorejo k vizualizaciji govora. Na koncu podamo še naše sklepne ugotovitve in ideje do katerih smo prišli pri pisanju diplomske naloge, ter kako se izognemo določenim napakam ogrodja s katerim smo delali.

Ključne besede: spletna stran, spletna aplikacija portal, ogrodje Ext JS.

# Abstract

The thesis represents an user friendly Ext JS framework with great support for company, that can succesfully operate their business. It helped us to create a rich web application named portal. Ext JS framework is one of the top information technologies written in JavaScript and combines great with HTML and CSS. The knowledge of JavaScript have made our work with framework much easier and led us to our client's goal. Our client gave us a specific requests for portal appearance and functionality, especially for their main users, so-called marketing agents. In thesis we describe transfering files from the internet to the development unit and how to select the right ones based on developer's knowledge. We start developing with Ext JS components, creating our own extensions and interpendence of our components. Next we describe meaning of the created portal and his users, which can be agents, supervisors or administrator. Purpose of created portal is to manage, from creating work tasks, editing skills and other users call settings, to creating campaigns, which contributes to outbound call agents speech visualization. While working on the thesis, we came to our newly discovered decisions and how we could avoid some mistakes of the chosen framework.

Keywords: website, web application, portal, Ext JS framework.

# Seznam kratic

| Kratica | Angleški izraz | Slovenski izraz |
|---|---|---|
| AJAX | Asynchronous JavaScript and XML | Asinhroni JavaScript in XML |
| API | Application Programming Interface | Vmesnik za programiranje aplikacij |
| CSS | Cascading Style Sheet | Kaskadne slogovne predloge |
| DOM | Document Object Model | Objektni model dokumenta |
| Ext JS | Extended JavaScript | Razširjen JavaScript |
| GUI | Graphical user interface | Grafični uporabniški vmesnik |
| HTML | HyperText Markup Language | Označevalni jezik |
| IDE | Integrated development environment | Integrirano razvojno okolje |
| JSON | JavaScript Object Notation | JavaScript objektni zapis |
| MVC | Model View Controller | Model Pogled Kontroler |
| URL | Uniform Resource Locator | Enolični krajevnik vira |
| XHTML | Extensible HyperText Markup Language | HTML s sintakso XML |
| XML | Extensible Markup Language | Razširjen označevalni jezik |

# Kazalo vsebine

# 1 Uvod

Vse več podjetij si želi imeti spletno aplikacijo, ki je enostavna za uporabo in s katero lahko zaposlenim v podjetju na raznolik in zanimiv način izboljšajo poslovanje. Obstajajo številna ogrodja, ki nam to omogočajo: jQuery (JavaScript Query), MooTools (My Object Oriented Tools), YUI (Yahoo User Interface), SmartClient, ipd. Večina razvijalcev se nemudoma odloči za jQuery, ker je enostaven za uporabo, veliko ponuja in se že dolgo uporablja na spletnem tržišču. Vendar lahko najdemo boljšo rešitev, to je uporaba ogrodja Ext JS (Extended JavaScript).

Ogrodje Ext JS nam ponuja mnogo več, le malo bolj se moramo poglobiti vanj. Omogoča nam namreč izdelavo bogatih aplikacij za katerikoli spletni brskalnik. Ponuja izjemen nabor gradnikov uporabniškega vmesnika, visoko zmogljivost prilagodljivih tabel, dreves, menijev in obvladljivih dogodkov. Najboljša lastnost ogrodja Ext JS je komponentni izgled, ki nam omogoča enostavno razširjanje komponent, za doseganje posameznikovih potreb. Še posebej pomembno je pri skupinskem razvoju velike aplikacije, da lahko vsak programer dela na svoji komponenti in mu ni potrebno poznati tuje kode. Zanimivost ogrodja je tudi v tem, da nam omogoča tako delo v skupini, kot tudi delo kot posameznik. Delo z ogrodjem se med seboj dopolnjuje in nadgrajuje, ni pa potek izdelave aplikacije odvisen od nekega določenega zaporedja.

V našem primeru je podjetje želelo imeti izdelan portal za telemarketinške namene. Prav zato mora aplikacija nuditi pregled in ureditev naslednjih storitev: delovnih nalog, uporabnikovih podatkov in privilegijev, seznama blokiranih številk, kampanj ter veščin uporabnikov portala.

# 2 Razvojno okolje in tehnologije

Za razvoj spletnih aplikacij so na voljo različna okolja in tehnologije. Med njimi smo za razvoj spletne aplikacije portal uporabili naslednje: Microsoft Visual Studio, v katerem smo razvijali aplikacijo; programski jezik C#; MVC arhitekturo; JavaScript za delo z ogrodjem Ext JS; itd. V nadaljevanju opišemo razvojno okolje in spletne tehnologije s katerimi smo se srečali.

## 2.1 Razvojno okolje Microsoft Visual Studio

Microsoft Visual Studio je integrirano razvojno okolje – IDE (Integrated Development Environment) [7]. Uporablja se za razvoj konzolnih aplikacij in aplikacij grafičnih uporabniških vmesnikov – GUI (Graphic User Interface). V okolju lahko izdelamo windows obrazce, spletne strani, spletne aplikacije in spletne storitve. Omogoča programiranje za vse naslednje podprte platforme: Microsoft Windows, Windows Mobile, Windows CE, ogrodje .NET Framework in Microsoft Silverlight.

Visual Studio vključuje urejevalnik kode in podpira IntelliSense kot tudi »refaktoriranje« kode. Integrirano orodje razhroščevalnik, deluje kot programski in strojni razhroščevalnik. Druga vgrajena orodja vključujejo oblikovalca obrazcev za izgradnjo grafičnih uporabniških vmesnikov, kot je spletni oblikovalec, razredni oblikovalec in oblikovalec bazne sheme. V Visual Studio je mogoče namestiti vtičnike (ang. plugin), ki povečujejo funkcionalnost na skoraj vseh ravneh. Ima tudi podporo za programsko kontrolne sisteme (kot je Subversion in Visual SourceSafe), ter dodajanje novih orodij kot so uredniki in vizualni oblikovalci za domensko specifične jezike ali orodja za druge vidike cikla razvoja programske opreme (Foundation Server Team klient: Team Explorer).

Visual Studio podpira različne programske jezike s pomočjo jezikovnih storitev, ki omogočajo uredniku kode in razhroščevalniku podporo skoraj vsakega programskega jezika.

Vgrajeni programski jeziki:

- C / C++ (z Visual C++),
- VB.NET (preko Visual Basic. NET),
- C# (z Visual C#) in
- F# (od Visual Studio 2010).

In še ostali jeziki:

- M,
- Python,
- Ruby
- XML / XSLT,

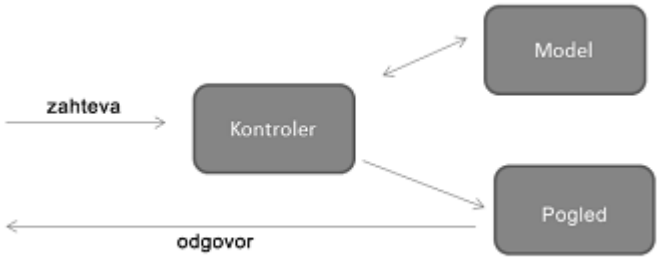
- HTML / XHTML,
- JavaScript in
- CSS.

### 2.1.1 Programski jezik C#

C sharp (v nadaljevanju C#) je razvila ekipa Microsoftovih strokovnjakov pod vodstvom Andersa Hejlsberga leta 2001. Programski jezik C# je razmeroma nov programski jezik, ki so ga razvili kot osnovni programski jezik za podporo funkcionalnosti ogrodja .NET. Zadnja verzija programskega jezika C# je 4.0. Za svoje delovanje potrebuje ogrodje .NET Framework 4, z njim pa lahko programiramo v razvojnem okolju Visual Studio 2010.

### 2.1.2 Arhitektura MVC

Model View Controller (MVC) je temeljni načrtovalski vzorec, ki ločuje logiko uporabniškega vmesnika od poslovne logike. Model je komponenta sistema, ki dejansko opravi delo (simulacija aplikacijske domene) in je popolnoma ločen od pogleda. Kontroler se uporablja za pošiljanje sporočil modelu. Zagotavlja tudi vmesnik med modelom ter pripadajočim pogledom in napravami (npr. tipkovnica, miška). Vsak pogled je tesno povezan s kontrolerjem in ima natančno en model, medtem ko ima lahko model več parov pogled/kontroler [10]. Za spletno aplikacijo smo uporabili arhitekturo MVC (Slika 1), v kateri smo imeli le dva pogleda.




Slika 1: Izgled MVC diagrama.

### 2.1.3 Ogrodje Microsoft .NET Framework

Microsoft .NET Framework je ogrodje za razvijanje programske opreme na Microsoft Windows operacijskih sistemih. Najnovejša različica je 4.0 in je naslednica prejšnjih verzij (.NET 1.0, 1.1, 2.0, 3.0 in 3.5). Različica .NET 3.5 je bila podana na trg novembra 2007. Aprila 2010 je bila izdana različica .NET 4.0 skupaj z razvijalskim okoljem Visual Studio 2010. Ogrodje .NET Framework 4 deluje skupaj s starejšimi različicami ogrodja .NET Framework.

## 2.2 Spletne tehnologije

### 2.2.1 HTML, XHTML

HyperText Markup Language (v nadaljevanju HTML), je označevalni jezik za izdelavo spletnih strani in predstavlja osnovo vsakega spletnega dokumenta. HTML kodo lahko pišemo v kateremkoli urejevalniku besedil (notepad, wordpad, ipd.), kot tudi v kateremkoli razvojnem okolju (Adobe DreamWeaver, Microsoft Visual Studio, ipd.). Označevalni jezik je enostaven za uporabo in ga lahko kombiniramo z ostalimi jeziki, kot so JavaScript, PHP, ASP.NET, XML, itd. Gradniki jezika HTML so elementi, ki jih sestavljata začetna in končna značka (angl. tag). Osnovna HTML stran je zgrajena iz minimalno treh značk: html, head in body. Pomembne so še ostale značke, kot so title, script, link, p, div, h, itd.. Vse značke imajo svoj blok kar pomeni, da začetek in konec bloka označimo z enakim imenom značke, na primer <body> in </body>. Obstajajo pa tudi značke, ki ne potrebujejo zaključka bloka (npr. nova vrstica, nov odstavek, slika, itd.) [4]. Zadnja verzija je HTML5.

Primer osnovne HTML strani:

```
<html>
   <head>
      <title>To je naslov strani</title>
   </head>
   <body>
      <p>To je odstavek v elementu body HTML dokumenta.</p>
   </body>
</html>
```

### 2.2.2 CSS

Cascading Style Sheets (CSS) so kaskadne slogovne predloge, predstavljene v obliki preprostega slogovnega jezika, ki skrbi za predstavitev spletne strani. Z njimi definiramo stil HTML oziroma XHTML elementov v smislu pravil, kako naj se ti prikažejo na strani. Določamo lahko barve, velikosti, odmike, poravnave, obrobe, postavitve in vrsto drugih atributov. Prav tako lahko nadziramo aktivnosti, ki jih uporabnik nad elementi strani izvaja (npr. prekritje povezave z miško).

Bistvo uporabe CSS je definicija pravil, ki omogoča ločitev vsebine strani, podane z označevalnik jezikom. S tem omogočimo lažje urejanje in dodajanje slogov ter tako poskrbimo za večjo preglednost dokumentov zasnovanih na HTML sintaksi. Prav tako zmanjšamo ponavljanja kode, saj omogočimo množici strani uporabo istih predlog, kar lahko bistveno zmanjša njihovo velikost.

Prva verzija CSS iz leta 1996 je CSS 1, ki je nudila podporo za določanje oblike črk, barve besedil, ozadij, poravnavo besedil, slik, razmike, obrobe in pozicije večine elementov, identifikatorje elementov in razrede.

Različica CSS 2 je bila objavljena leta 1998 in je nadgradila CSS 1 z novimi lastnostmi, kot so relativno, absolutno in fiksirano pozicioniranje elementov in večslojnost elementov z uporabo z-index. Omogoča tudi, da črkam določimo senco.

Specifikacija zadnje različice CSS 3 je razdeljena v več ločenih dokumentov, imenovanih moduli. Nekaj modulov: Color, Selectors, Namespaces. V modulih so nekatere lastnosti iz CSS 2 nadgrajene, dodane pa so tudi nove [6].

Primer CSS kode:

```
body { background-color: gray; }
h1 { color: red; text-align: left; font-size: 16px; }
p { font-family: "Arial"; font-size: 12px; }
.ena-klasa { width: 100px; text-align: right; float: left; padding-right: 5px; }
```

Učinek primera CSS kode:

- Element body ima sivo obarvano ozadje.
- Element h1 ima pisavo postavljeno na levo stran, rdečo barvo in velikost pisave 16 pikslov.
- Element p ima znake pisave družine Arial, ter velikost 12 pikslov.
- Element z razredom »ena-klasa« je širok 100 pikslov, pisavo ima poravnano na desno, element je poravnan na levo ter prazen prostor med desnim robom elementa in njegovo vsebino je 5 pikslov.

### 2.2.3 JavaScript

JavaScript je objektni skriptni programski jezik, ki ga je razvil Netscape, da bi spletnim programerjem pomagal pri ustvarjanju interaktivnih spletnih strani [5]. Jezik je bil razvit neodvisno od Jave, vendar si z njo deli številne lastnosti in strukture. JavaScript lahko vključimo v HTML kodo in s tem poživimo stran z dinamičnim izvajanjem. Podpirajo ga velika programska podjetja in kot odprt jezik ga lahko uporablja vsakdo, ne da bi pri tem potreboval licenco. Podpirajo ga vsi novejši spletni brskalniki.

V programu JavaScript lahko napišemo funkcije, katerim pravimo JavaScript skripte. Če si želimo ogledati preprosto HTML spletno stran, opremljeno z JavaScript funkcijami, v osnovi ne potrebujemo strežnika. Potrebujemo le spletni brskalnik, ki bo dokument odprl. JavaScript funkcije se izvedejo ob sprožitvi dogodka. To je npr. ob kliku na gumb, ob premiku miške, ob naložitvi strani, ipd.

JavaScript je dinamični jezik s prvo-razrednimi funkcijami. To pomeni, da so funkcije v jeziku obravnavane kot prvo-razredni objekti.

To pa pomeni, da jezik omogoča naslednje:

- funkcija pri klicu za vhodne vrednosti lahko sprejme drugo funkcijo,
- funkcija lahko kot rezultat vrne rezultat druge funkcije,
- vrednost spremenljivke je lahko funkcija,
- funkcijo lahko shranimo v podatkovno strukturo.

Ko brskalnik, ki podpira jezik JavaScript, naloži spletno stran hkrati ustvari vrsto objektov, dva sta opisana spodaj. Ti objekti omogočijo programerju dostop do spletne strani in elementov HTML, ki jih stran vsebuje. Trenutna različica je JavaScript 1.8.5.

**Objekt Window**

Preko objekta window imamo dostop do brskalnikovega okna. Ta objekt vsebuje med drugim naslednje lastnosti in funkcije.

Lastnosti:

- document – trenutni dokument naložen v oknu brskalnika,
- history - objekt vsebuje naslove strani, ki jih je obiskal uporabnik znotraj tega okna,
- location - vsebuje informacijo o trenutnem naslovu strani.

Funkcije:

- alert() - prikaže opozorilno okno in gumb za potrditev,
- confirm() - prikaže pogovorno okno z enim gumbom za preklic in drugim za potrditev,
- prompt() - prikaže pogovorno okno z urejevalnim poljem,
- setTimeout() - izvrši v parametru podano funkcijo po določenem času izraženem v milisekundah,
- clearTimeout() - počisti časovnik narejen s funkcijo setTimeout().

**Objekt Document**

Programerju omogoča dinamično spreminjanje dokumenta, naloženega v oknu brskalnika in omogoča dostop do vseh elementov XHTML.

Lastnosti:

- forms - vrne zbirko obrazcev, ki se nahajajo v dokumentu trenutnega okna,
- images - vrne zbirko slik, ki se nahajajo v dokumentu trenutnega okna,
- title - vrne ali nastavi naslov dokumenta,
- URL - vrne polni spletni naslov dokumenta.

Funkcije:

- getElementById() - vrne HTML element s podanim identifikatorjem,
- write() - zapiše HTML izraz ali JavaScript kodo na dokument.

AJAX (asinhroni JavaScript in XML) je skupina medsebojno povezanih spletnih razvojnih tehnik uporabljenih za ustvarjanje interaktivnih spletnih aplikacij. Z uporabo AJAX-a si lahko spletne aplikacije izmenjujejo podatke s strežnikom asinhrono v ozadju, brez potrebe po ponovnem nalaganju strani.

### 2.2.4 Firebug

Firebug je dodatek za Mozillin brskalnik, imenovan Firefox. Razvijalcem spletnih aplikacij omogoča spremljanje, urejanje in razhroščevanje HTML-ja, CSS-a, JavaScript-a direktno iz brskalnika na katerikoli spletni strani. Poleg ostalih lastnosti kot so prikazovanje CSS odmikov, robov, velikosti elementov, omogoča tudi spremljanje internetne aktivnosti, kar dostikrat pripomore k hitrejšemu nalaganju strani, kot tudi prikazovanju CSS in JavaScript napak. Ima tudi nameščen iskalnik, ki nam pomaga pri hitrejšem odkrivanju HTML elementov. Trenutna verzija je Firebug 1.10a9.

## 2.3 Pregled Ext JS vmesnika API

Podjetje Sencha je ustanovitelj Ext JS ogrodja in ponuja na enem izmed svojim strežnikov celoten pregled Ext JS vmesnika za programiranje aplikacij (v nadaljevanju API) [9]. Omogoča tudi podroben vpogled v izvorno kodo Ext JS ogrodja, kar nam pripomore k boljši razumljivosti celotnega ogrodja. V nadaljevanju je opisanih nekaj razredov, ki so bili uporabljeni za izdelavo spletne aplikacije.

### 2.3.1 Vsebovalnik

Vsebovalnik (ang. container) je osnova za vsak Ext.BoxComponent razred, ki lahko vsebuje druge komponente. Vsebovalnik omogoča dodajanje, vstavljanje in brisanje komponent.

Najpogosteje uporabljeni razredi so: Ext.Panel, Ext.Window in Ext.TabPanel. Če imamo primer, kjer ne potrebujemo vseh zmogljivosti ki jih ponuja dotični razred, lahko ustvarimo oskubljen vsebovalnik, ki ga enkapsulira HTML element s pomočjo autoEl konfiguracije. Ta tehnika je na primer zelo uporabna pri ustvarjanju vgrajenih stolpičnih postavitvah znotraj nekega obrazca.

### 2.3.2 Postavitev

Postavitev (ang. layout) je pomembna lastnost vsakega vsebovalnika. Zagotavlja, da se vsi otroci vsebovalnika pravilno razširijo in pozicionirajo.

Za pravilno delovanje postavitve moramo nastaviti dve konfiguraciji:

- postavitev – ki verižno zagotavlja velikost otrok in
- dimenzija – ki brez postavitve ne more pravilno funkcionirati in obratno (npr. višino v pikslih s »height« konfiguracijo, ali »autoHeight« pri staršu).

Privzeto je postavitev nastavljena na 'auto', kar pomeni, da je uporabljen privzeti layout manager, ki izriše vse svoje otroke sekvenčno v svoj vsebovalnik, brez pozicioniranja in spreminjanja njegove velikosti. Postavitev mora biti podana kot niz ali kot objekt kateremu obvezno podamo tip in druge lastnosti kot so poravnava, odmik, itd [1].

Vrste postavitve:

- absolute – omogoča postavljanje komponente preko standardnih x in y koordinat.
- accordion – postavitev zagotavlja, da imajo otroci komponente prikazane naslove, medtem ko je vedno v celoti prikazan samo en otrok.
- anchor – otrokom nastavimo širino in višino kar v anchor konfiguraciji nastavimo z besedo »left«, »right«, »top« in »down«, ali procentualno koliko želimo da otrok zasede prostora svojega starša ali pa kar fiksno v pikslih, ki se odraža z odmiki od roba.
- auto – privzeta postavitev, v primeru da vsebovalniku ne podamo nobene postavitve.
- border – postavitev razdeli svoje otroke na strani vsebovalnika. Vsakemu otroku moramo podati stran s konfiguracijo imenovano »region«, katera je lahko »west«, »east«, »north«, »south« ali »center«.
- card – podobna accordion postavitvi, le da ta ne prikazuje naslove svojih otrok in lahko prikaže največ enega otroka, lahko pa tudi nobenega.
- column – razdeli svoje otroke na stolpce, katerim podamo konfiguracijo »columnWidth« v decimalnem številu v vrednosti od 0 do 1.
- fit – ima lahko samo enega otroka katerega raztegne po celi svoji širini in višini.
- form – uporablja se za omogočanje dodatne konfiguracije svojim otrokom, za oznake, ki jih ti uporabljajo.
- hbox/vbox – otroke razporedi horizontalno oziroma vertikalno skozi vsebovalnik.
- menu – uporablja se za prikazovanje menija.
- table – svoje otroke prikazuje v obliki tabele. V konfiguracijo mu moramo podati še število stolpcev. Otroke lahko prerazporedimo s konfiguracijo »rowspan«, ki zajame več vrstic in »colspan«, ki zajame več stolpcev.

### 2.3.3 Plošča

Plošča (ang. panel) je podrazred vsebovalnika, ki ima posebno funkcionalnost in strukturirane komponente zaradi katerih postane popolni gradnik za aplikacijsko usmerjen uporabniški vmesnik.

Posebnost plošč je v njeni postavitvi, katera z uporabo svojih komponent, lepo in na željen način prikaže svoje otroke. Za lepši izgled ji lahko spodaj in zgoraj dodamo orodno vrstico in naslov.

Nekaj podrazredov, ki smo jih uporabili v diplomski nalogi:

- Ext.form.FormPanel – plošča za izgradnjo obrazcev.
- Ext.grid.GridPanel – plošča tabele za prikaz podatkov iz podatkovnih skladišč.
- Ext.TabPanel – plošča, ki deluje kot vsebovalnik za otroke, kateri se prikazujejo s pomočjo zavihkov.
- Ext.Tip – plošča za prikazovanje namigov.
- Ext.tree.TreePanel – plošča, ki prikazuje svoje otroke v obliki drevesne strukture.
- Ext.Window – plošča kot pojavno okno, ki se pojavi nad vsemi HTML elementi.

### 2.3.4 Obrazec

Ext.form.FormPanel je standarden obrazec, vsebovalnik, ki upravlja s komponentami otrok. Privzeto je obrazec nastavljen s postavitvijo »form«, zato da uporabi FormLayout manager, kateri ustrezno izriše in dodeli sloge za polja in oznake. Pri gnezdenju dodatnih vsebovalnikov znotraj obrazca je nujno potrebno zagotoviti, da ima starš katerega koli polja (TextField, ComboBox, DateField, ipd.), nastavljen 'form' layout.

Obrazec vsebuje BasicForm (enkapsulira DOM <form> element), katerega zgradi s svojo prvotno konfiguracijo. DOM (Document Object Model) definira strukturo dokumenta in način kako je ta dokument dosegljiv in manipuliran. BasicForm se uporablja za nalaganje, validiranje in potrjevanje obrazca.

Vnosni elementi so potrebni za pravilen izgled obrazca. Vsak element ima svojo opcijsko konfiguracijo, kot tudi nekaj skupnih:

- allowBlank – true/false za preverjanje, če je dolžina vnesenega niza polja večja od 0.
- emptyText – tekst, ki se prikaže za prazen element.
- enableKeyEvents – true/false za proženje dogodkov iz vhodnih naprav kot je tipkovnica.
- fieldLabel – tekst, ki se prikaže ob elementu (primer »Name:«).
- regex – JavaScript RegExp objekt, ki preverja vrednost elementa (value) skozi validacijo.
- value – vrednost s katero inicializiramo element.
- getRawValue() – funkcija, ki vrne veljavno ali neveljavno vrednost.
- getValue() – funkcija, ki vrne normalizirano vrednost elementa v obliki niza.

- reset() – ponastavi vrednost elementa.
- setDisabled(true/false) – funkcija, ki omogoči ali onemogoči editiranje elementa.

Elementi imajo tudi nekaj pomembnih poslušalcev, kot so:

- blur – sproži se takoj, ko element izgubi fokus.
- keyup – sproži se, ko spustimo tipko iz tipkovnice, ki je bila pritisnjena.
- valid – sproži se ob validiranem elementu brez napak.

### 2.3.5 Polje

Osnovni razred za vsa polja obrazca, ki omogočajo spreminjanje dimenzij, urejanja vrednosti elementov, obravnavo dogodkov in ostalo funkcionalnost. V nadaljevanju sledi opis nekaj podrazredov razreda Ext.form.Field, s katerimi smo se srečali v diplomski nalogi.

**Tekstovno polje (TextField)**

Je osnovno polje za pisanje teksta. Lahko se ga uporablja za tradicionalna tekstovna polja ali kot osnoven razred za bolj zapletena vnosna polja kot so Ext.formTextArea in Ext.form.ComboBox.

**Izbirno polje (ComboBox)**

Izbirno polje je vnosno polje, podobno tekstovnemu polju, le da ima ob strani gumb za prikaz izbirnih elementov. Posebnost izbirnega polja je tudi v podpori za samodejno zapolnjevanje teksta glede na vnesene črke (ang. autocomplete), nalaganje podatkov iz strežnika (ang. remote-loading) ter oštevilčenje strani izbirnega polja in filtriranje (ang. pagination). Ta deluje samo, če ima nastavljen v konfiguraciji »mode« na »remote«.

Potrebna je dodatna konfiguracija:

- store – podatkovno skladišče za prikaz zapisov v izbirnem polju,
- valueField – ime vrednosti s katero se izbirno polje veže na podatkovno skladišče in nastavlja zapis in
- displayField – ime vrednosti s katero se izbirno polje veže na podatkovno skladišče in prikaže zapis.

Ima tudi poseben dogodek:

- select(combo, record, index) – se sproži, ko je bil izbran en element na seznamu. Kot parametre poda izbirno polje, zapis iz podatkovnega skladišča in indeks mesta kjer se zapis nahaja.

**Sestavljeno polje (CompositeField)**

Sestavljeno polje je sestavljeno iz večih polj in omogoča postavitev polj enega zraven drugega. Polja so izrisana s pomočjo hbox postavitve, tako da je na voljo vsa konfiguracija omenjene postavitve za otroke sestavljenega polja.

**Datumsko polje (DateField)**

Zagotavlja vnos datuma s pomočjo Ext.DatePicker komponente. Ob izbiri datuma zna avtomatsko validirati polje.

Dodatna konfiguracija za zagotavljanje natančnejšega vnosa:

- minValue/maxValue – minimalna/maksimalna vrednost dovoljenega datuma. Lahko je JavaScript Date format ali niz v veljavnem formatu.
- minText/maxText – tekst, ki se prikaže, če je vnesen datum manjši/večji od podanega minValue/maxValue.
- showToday – true/false za prikaz gumba za nastavljanje današnjega datuma. Privzeta vrednost je true.
- startDay – indeks dneva s katerim se začne teden. Privzeta vrednost je 0, kar predstavlja nedeljo.

**Potrditveno polje (CheckBox)**

Ext JS potrditveno polje (CheckBox) se uporablja kot tradicionalno HTML potrditveno polje.

Dodatna konfiguracija:

- checked – true/false, če želimo da se potrditveno polje izriše označeno/neoznačeno. Privzeta vrednost je false.
- setValue(Boolean/String checked) – nastavi zapis potrditvenemu polju in sproži »check« dogodek. Podan parameter je lahko true, »true«, »1«, ali »on«. Katerikoli drugi podan parameter bo označil potrditveno polje kot neizbrano.

### 2.3.6 Tabela

Ext.grid.GridPanel razred se uporablja za prikaz podatkov v obliki tabele (vrstic in stolpcev).

Za njihov prikaz je potrebno v konstruktorju obvezno podati:

- store – podatkovno skladišče, ki drži zapise podatkov (vrstic).
- colModel – uporablja se za branje zapisov iz podatkovnega skladišča in njihovo prikazovanje v stolpcih.
- selModel – omogoča izbiro zapisov v tabeli.
- gridView – uporabniški vmesnik tabele.

Tipične napake pri uporabi tabele so:

- tabela ne prikazuje podatkov,
- podatkovno skladišče se ni naložilo,
- tabela se ni izrisala,
- tabela ne posluša na evente od podatkovnega skladišča, ipd.

### 2.3.7 Podatkovno skladišče

Ext.data.Store je razred, ki hrani podatke (zapise) na odjemalčevi strani in zagotavlja vnos podatkov ostalim komponentam, kot so: GridPanel, ComboBox, DataView, itd.

Skladiščni tipi:

- ArrayStore – podatke ustvari iz JavaScript polj, za branje polj pa uporabi razred imenovan ArrayReader, kateri zna brati polje večih polj.
- JsonStore – podobno kot ArrayStore, le da ta dobi podatke v formatu JSON, pri tem mu pomaga razred JsonReader, ki podatke prebere iz polj objektov zakodiranih v niz.
- GroupingStore – ponavadi se uporablja pri prikazovanju grupiranih podatkov v tabeli. Je preprosto podatkovno skladišče, le da ima omogočeno razvrščanje po večih elementih (multiSort). Grupirano polje in smer sta vedno vstavljena kot prvi razvrščen par.

GroupingStore smo na portalu uporabili za prikazovanje delovnih nalog v tabeli. Prikaz seznama smo grupirali po dnevih in razvrstili po padajočem vrstnem redu. Znotraj grupe smo sortirali naraščajoče po skupini nalog.

# 3 Zahteve naročnika

Naročnik želi imeti zgrajen spletni portal za zaposlene. Portal mora biti zgrajen v Ext JS ogrodju in zaposlenim (uporabnikom) nuditi hitro in enostavno uporabo le tega. To omogoča klicnim agentom, da tudi uspešneje opravljajo svoje delo, saj oni namreč predstavljajo večinski delež uporabnikov.

Uporabniki portala morajo biti razdeljeni v tri skupine:

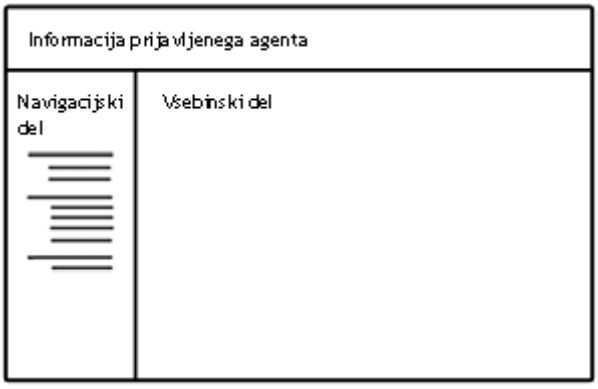
- Agenti – imajo privilegij za vnašanje delovnih ur.
- Nadzorniki – imajo privilegij za vnašanje delovnih ur in upravljanjem z blokado številk.
- Administratorji – imajo vse privilegije, ki zajemajo: Upravljanje z vsemi uporabniki, blokado številk, kampanjami in vsemi veščinami.

## 3.1 Izgled portala

Naročnik zahteva, da je portal razdeljen na tri dele. Zgornja stran naj prikazuje podatke o uporabniku prijavljenemu v sistem. Leva stran služi za navigacijo uporabnika. Centralni del pa naj bo namenjen za prikazovanje vsebine.

Drevesno strukturo navigacijskega dela želi imeti omejeno za skupine uporabnikov, ki bi bili vidni uporabnikom z različnimi privilegiji (ang. privileges). Privilegije administrator dodeli vlogam (ang. roles), te vloge pa uporabnikom. Uporabnik ima lahko dodeljenih več vlog.

Centralni del portala želi imeti odvisen od klika na navigacijski element. Zato smo za rešitev tega razdelili navigacijsko ploščo na pet delov (menijev). Na sliki 2 je prikazana skica zahtevanega izgleda portala.




Slika 2: Skica naročnikove zahteve za izgled spletne strani.

Navigacijsko ploščo želi imeti razdeljeno na naslednje menije:

- Upravljanje z agenti (ang. Agent Management) - Meni mora omogočati agentom vpisovanje delovnih ur. Izdelati je potrebno dva različna pogleda za delovne ure in stanje delovnih nalog. Delovne ure morajo biti vidne vsem uporabnikom, seznam stanj delovnih nalog pa samo administratorjem. Uporabnik lahko dodaja ali briše svoje delovne naloge. Administrator mora imeti tudi možnost spremeniti status delovnim nalogam (na »approved«, »rejected« ali »pending«) in možnost urejanja ur posameznega uporabnika.
- Upravljanje z uporabniki (ang. User Management) - Meni mora biti viden samo administratorjem, kateri morajo imeti na voljo vse podrobnosti o uporabnikih in možnosti urejanja veščin ter klicnih nastavitev. Druga zadeva je urejanje vlog. Vsaki vlogi lahko administrator dodeli različne privilegije.
- Upravljanje z blokado številk (ang. Blacklist Management) - Naročnik želi, da se mu omogoči pregled in vnos blokiranih klicnih številk, kot tudi obrazec za vnos pravil za avtomatsko blokado številk. Dostop do menija morajo imeti nadzorniki in administratorji.
- Upravljanje s kampanjami (ang. Campaign Management) – Kampanja je vizualno predstavljen nekakšen klicni vzorec, ki se ga mora agent držati v času odhodnega klica. Naročnik želi, da se mu omogoči tudi upravljanje s kampanjami za administratorje. Pod upravljanje mora spadati:
  - pregled kampanj in podkampanj,
  - urejanje podkampanj in
  - dodajanje gradnikov podkampanjam.
- Upravljanje z veščinami in sklopi veščin (Skill and Skillset Management) - Veščine predstavljajo znanje oziroma usposobljenost agenta. Naročnik zahteva, da se samo za administratorja naredi upravljanje z veščinami, sklopi veščin, pregled njhove odvisnosti ter masovno dodajanje in odstranjevanje veščin uporabnikom.

## 3.2 Funkcionalnost portala

Vsak portal mora imeti svojo funkcionalnost. Naročnik je še posebej zahteval enostaven in enoten pregled komponent, ter upravljanje z vsebino.

Nekaj specifičnih zahtev naročnika:

- Iskalnik – v portalu želi imeti nad tabelo iskalnik, ki filtrira rezultate. Ob vsaki aktivnosti na obrazcu se mora po določenem številu milisekund, sprožiti zahteva na strežnik, kar prepreči preveliko število zahtev.
- Poenotenje obrazcev in tabel – tabela mora v zgornji orodni vrstici vsebovati naslov in desno zgoraj gumbe za akcije nad tabelo in obrazec (če le ta obstaja). Obrazec naj ima elemente na levi strani, enega zraven drugega ter oznako elementa dovolj široko da ob lokalizaciji ne pride do zamikov ali odsekanih črk.
- Validacija vnešenih podatkov na strani odjemalca – vnešeni podatki se morajo validirati sproti. Sprotna validacija omogoča uporabnikom boljši in hitrejši pregled napak. Napake na poljih se označijo z rdečo obrobo in podrčtavo. Ob poljih pa se mora pojaviti tudi rdeče

obarvan klicaj, ki agentom prikaže namig napake. Novo vnesena imena ne smejo nastopiti v podatkovnem skladišču, kar pomeni, da se ne smejo ponavljati.

- Prikaz podatkov v tabeli – tabela mora vsebovati:
  - Funkcijo urejanja podatkov, s katero lahko urejamo le trenutno izbrano vrstico.
  - Model selekcije, ki omogoča označevanje ali odznačevanje vrstic.
  - Prilagodljivost širine stolpcev, katere lahko vsak uporabnik prilagaja svojim potrebam.
- Enakost komponent – vsa okna morajo imeti minimalno dva gumba (enega za potrditev in drugega za preklic vnešenih podatkov). Vedno se mora najprej prikazati obrazec (v primeru da je za upravljanje potrebno tudi iskanje) in nato tabela, ki prikazuje podatke za urejanje.
- Prilagajanje zunanjim podatkom - podatki za delovne naloge prihajajo tudi iz drugega sistema, zato je potrebno uskladiti morebitne časovne luknje pri vnosu dnevnih nalog agentov.

# 4 Izdelava spletnega portala v Ext JS ogrodju

## 4.1 Osnove portala

Izraz »portal« se je začel uporabljati za spletne strani oziroma spletne iskalnike večjih podjetij, kot so Yahoo, MSN, Netscape Netcenter in še veliko ostalih podjetij, ki so se pojavila v preteklosti. Večina uporabnikov je svoje brskanje po internetu pričela prav tukaj, od koder so jih hiperpovezave vodile na naslednje podstrani ali druge spletne strani. Tako se je veliko portalov razvilo prav iz spletnih iskalnikov ali imenikov. S širitvijo storitev, ki so jih ponujali ponudniki spletnih iskalnikov, so si zagotovili stalen obisk uporabnikov in zadržali svojo popularnost.

Spletni portal je obsežno spletno mesto ali servis. Za razliko od običajnih spletnih iskalnikov, ponujajo spletni portali še veliko več storitev, kot so podatki, informacije, znanja in ostale storitve.

Razlika med definicijo spletnega portala in spletne strani je v orientiranosti. Spletna stran je orientirana na institucijo, njeno storitev ali temo, medtem ko orientacijo portala označuje uporabnik. Portal namreč ponuja uporabnikov pogled na svet.

**Vrste spletnih portalov**

Portale lahko delimo glede na veliko kriterijev. Prva smiselna delitev je glede na zaposlene, poslovne partnerje in stranke. Večina takšnih portalov se je razvilo iz intraneta. Intranet je odličen vir informacij za vsako podjetje, sredstvo internega komuniciranja in orodje, ki zaposlenim pomaga pri delovnih procesih, izmenjavi internih informacij in omogoča elektronsko sodelovanje.

Glede na vsebino portale razdelimo na:

- Poslovne – namenjeni so strankam in partnerjem.
- Podjetniške – namenjeni so zaposlenim v podjetju.

Oboji so namenjeni kvalitetnejšemu poslovanju in nudijo dokaj ozko usmerjene vsebine. Mi se bomo osredotočili na izdelavo podjetniškega portala, ki bo imel možnost objavljanja vsebine in urejanja osebnih podatkov zaposlenih.

## 4.2 Uvoz Ext JS ogrodja

Odprtokodno verzijo Ext JS ogrodja si naložimo na disk in uvozimo z značko <link> za CSS datoteke ter <script> za JavaScript datoteke. Slika 3 prikazuje primer uvoza Ext JS ogrodja znotraj enega HTML dokumenta:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <!-- ** CSS ** -->
      <!-- base library -->
      <link rel="stylesheet" type="text/css" href="../../resources/css/ext-all.css" />

      <!-- ** Javascript ** -->
      <!-- ExtJS library: base/adapter -->
      <script type="text/javascript" src="../../adapter/ext/ext-base.js"></script>
      <!-- ExtJS library: all widgets -->
      <script type="text/javascript" src="../../ext-all-debug.js"></script>

      <!-- overrides to library -->

      <!-- extensions -->

      <!-- page specific -->

      <script type="text/javascript">
         Ext.onReady(function(){
           // all code goes here
         });
      </script>
   </head>
   <body>
   </body>
</html>
```

Slika 3: Osnovna HTML stran z uvozom Ext JS ogrodja.

Pri razpakiranju Ext JS ogrodja dobimo naslednje pomembne datoteke:

- ext-all.css - vsebuje vse stilske informacije, ki jih potrebuje ogrodje za prikaz.
- ext-base.js - vsebuje minimalne funkcije Ext JS ogrodja. Velikokrat je neuporaben za večje aplikacije.
- ext-debug.js - uporablja se samo v razvoju aplikacije. Ponuja minimalno število pomembnih razredov za delovanje. Za vse ostale dodatne potrebe, se lahko naknadno nalaga preko dodatnih razširitvenih Ext datotek.
- ext-all-debug.js - datoteka vsebuje celotno Ext JS knjižnico. Priporočena za hitrejše začetno učenje razvoja spletnih aplikacij z Ext JS ogrodjem. Vseeno pa večina za razvoj raje uporablja ext-debug.js.
- ext-all.js - je minimirana verzija od ext-all-debug.js, ki jo lahko uporabimo v produkcijskem okolju. Vseeno pa ta verzija ni priporočena, saj ne bomo uporabili vseh razredov, ki jih Ext JS ponuja. Tako se priporoča, da se za posamezno aplikacijo izdela svojo verzijo s pomočjo programa Ext JS Builder, katero lahko prilagodimo svojim zahtevam in se tako znebimo odvečnega zasedenega prostora.

Poleg JavaScript datotek, dobimo tudi CSS datoteke in slike, ki se uporabljajo v komponentah Ext JS ogrodja. Po izbiri potrebnih datotek lahko začnemo uporabljati Ext JS ogrodje. Znotraj značke <script> pokličemo funkcijo Ext.onReady. Kot parameter funkcije podamo novo funkcijo, ki se bo izvedla takoj, ko se bo DOM do konca naložil. Tu ponavadi pišemo vso nadaljno kodo in gradnike aplikacije narejene z Ext JS ogrodjem.

## 4.3 Izdelava spletnega portala

Izdelava spletne aplikacije, ki jo imenujemo portal, je temeljila na programski arhitekturi MVC, pri čemer pa je bil za celoten portal izdelan le en sam pogled. Portal se osvežuje s pomočjo proženja zahtevkov AJAX, ki obenem naredijo stran bolj dinamično. Zahtevke AJAX kličemo s funkcijo Ext.Ajax.request. To je klic funkcije za pošiljanje in sprejemanje podatkov iz strežnika preko AJAX-a.

V klic funkcije podamo objekt s parametri:

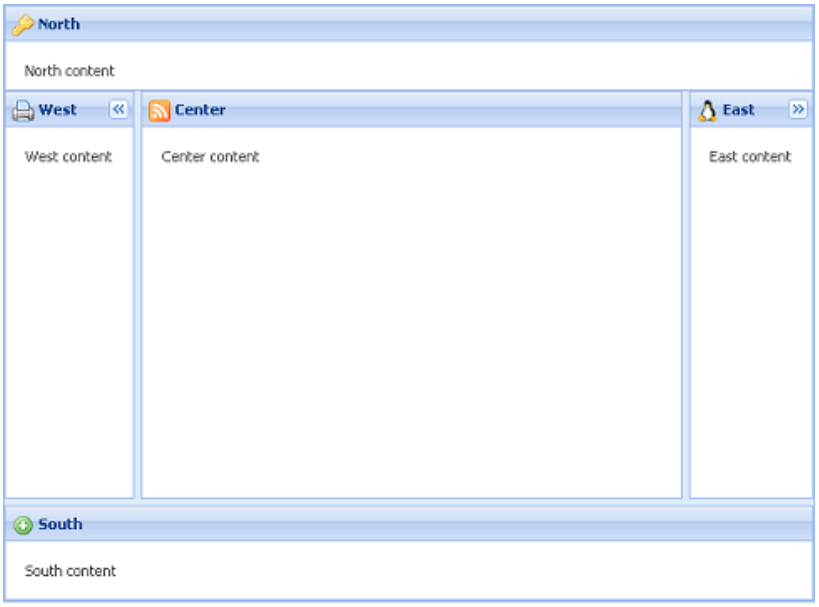
- url – pot z imenom funkcije, ki jo želimo poklicati.
- params – parametre, katere želimo podati.
- callback – funkcija, ki se izvede ob koncu zahtevka.
- scope – vidljivost zahtevka AJAX.

### 4.3.1 Izdelava komponente

Izdelava komponente v večini primerov temelji na razširitvi že obstoječih Ext JS komponent. Ext JS komponenta je JavaScript objekt, ki ima poleg osnovnih lastnosti, funkcij in dogodkov, še dodatne, katere mu podamo v konstruktor. Konstruktor je JavaScript objekt s katerim inicializiramo komponento. Ext JS komponento lahko tudi prepišemo ali razširimo. Za večino komponent, ki smo jih izdelali smo razširjali Ext JS komponente.

Izdelali smo prvo komponento, imenovano viewport, ki je tudi razširitev razreda Ext.Viewport. V konstruktor komponente smo ji podali otroke, kot polje novih objektov (Ext JS komponent). Ext.Viewport razred je komponenta, katera ima lahko največ pet otrok, vsak izmed otrok pa mora predstavljati svojo stran. Za pravilno izrisovanje otrok, moramo v konstruktor viewport komponente podati »border« postavitev (layout: »border«) in vsaki otrokovi komponenti, stran (ang. region), ki mora biti med otroci različna. Možne strani so: »north«, »south«, »west«, »east« in »center«. Primer Ext.Viewport komponente s petimi otroci lahko vidite na sliki 4.

Vsak otrok ima lahko še svoje otroke, itd. Če ima starš več otrok, se ti samodejno prikažejo navpično eden pod drugim. To se lahko uredi tudi z drugo vrsto postavitve (primer: layout: »vbox«, več podrobnosti v poglavju 2.2.1).




Slika 4: Primer vsebovalnika z »border« postavitvijo.

Znotraj Ext.onReady() funkcije ustvarimo viewport komponento (SM.Viewport). Nato sprožimo zahtevek AJAX, ki nam vrne privilegije v sistem prijavljenega uporabnika. Glede na privilegije uporabnika, lahko omogočimo vidnost drevesne plošče, ki se nahaja na levi strani viewport komponente. Drevesna plošča (Ext.tree.TreePanel) je navigacijska plošča, ki prikazuje svoje otroke kot drevo elementov. Prvi otroci so privzeto prikazani kot mape in vsi nadaljnji otroci kot datoteke. Na začetku smo vsem otrokom v konstruktor podali konfiguracijo »requiredPrivilege«, ki nosi ime privilegija, ki je potreben za vidnost določenega otroka. V konstruktor smo jim tudi podali zastavico za skrite otroke (hidden: true). Z zahtevkom AJAX smo nato preverili, če se otrok, ki nosi svoj »requiredPrivilege« ujema s katero izmed privilegijev, ki so prišli nazaj in so »true«. V primeru da se privilegij ujema, prikažemo otroka.

Nato pride na vrsto delovni prostor (ang. workspace), ki je razširitev navadne plošče (Ext.Panel). Ta ima postavitev nastavljeno na »card«, kar pomeni, da se vsi njegovi otroci obnašajo kot karte (več o tem je opisano v poglavju 2.3.2). Znotraj delovnega prostora smo zgradili vse komponente, ki jih potrebujemo na portalu. Tej komponenti nastavimo tudi poslušalca za globalne dogodke, katere sprožimo s klikom na katerikoli list (vidno kot datoteka) na navigacijski plošči. Zraven imena dogodka podamo še identifikator (ang. id) od komponente, ki jo želimo prikazati. Poslušalec na miškin klik dogodek na drevesni plošči sproži funkcijo, znotraj katere preverimo za kateri identifikator lista drevesne plošče gre in aktiviramo ter prikažemo ustreznega otroka delovnega prostora kar nam omogoča »card« postavitev.

Vsaki komponenti moramo pred inicializacijo zagotoviti »namespace« (prazne objekte oziroma vsebovalnik). Primer: Ext.namespace(»Timetracking«); ustvari prazen objekt imenovan Timetracking. S tem se rešimo napake praznih objektov, v tem primeru »Timetracking is null«. V navadi Ext JS programiranja je tudi, da za vsako komponento registriramo njen »xtype«. To storimo z Ext.reg(»xtimetrackingpanel«, Timetracking.Panel). Funkcija »reg« registrira našo komponento Timetracking.Panel z imenom, ki ga podamo kot

drugi parameter, v našem primeru »xtimetrackingpanel«. Kar pomeni, da kadar ustvarjamo komponento jo ne rabimo klicati z »new Timetracking.Panel()«, temveč lahko enostavno ustvarimo nov objekt { xtype: "xtimetrackingpanel" }, ki ima za xtype xsmviewport.

**Izgled komponente**

Sam izgled komponente se lahko spremeni z uporabo slogovnih predlog in lastnosti kot so:

- cls – niz CSS razreda, ki se doda na HTML element komponente,
- title – naslov komponente, v primeru da se ga ne poda v konfiguracijo, se title vsebovalnik ne kreira,
- bbar, tbar – spodnja/zgornja orodna vrstica plošče, ki je lahko Ext.Toolbar objekt ali polje gumbov,
- border – true/false, za prikaz robov komponente,
- autoScroll – true/false, za avtomatski prikaz drsnih trakov, če je vsebina komponente večja od njene višine,
- flex – število, ki predstavlja razmerje koliko prostora bo komponenta zasedla, le če ima njen starš layout »vbox« ali »hbox«,
- height/width – število, ki predstavlja višino oziroma širino, v primeru da flex ni bil podan, drugače se ignorira.

### 4.3.2 Razširitev ali prepis razreda

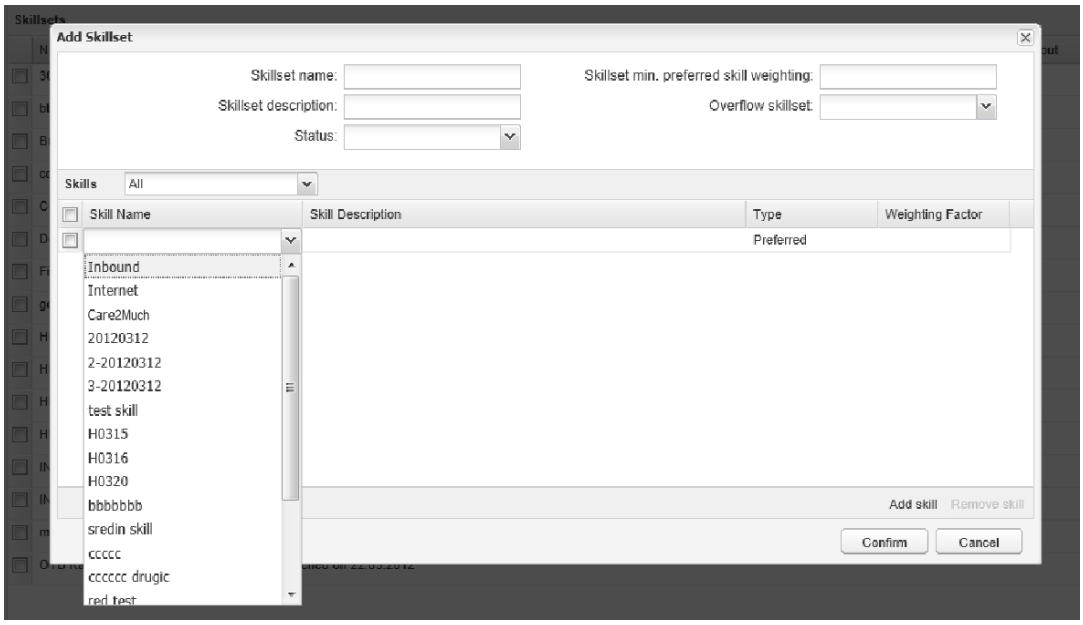
Razširitev (ang. extend) naredi nov podrazred na podlagi obstoječega razreda. Medtem, ko prepis (ang. override) uredi obstoječi razred brez ustvarjanja novega.

JavaScript prototip deluje na osnovi objektnega modela. Kar Ext naredi je povsem enako le, da programerju olajša razširjanje in prepisovanje razredov [3]. Slika 5 prikazuje kako lahko vzamemo prototip neke funkcije v JavaScript programskem jeziku in znotraj te prepisane funkcije s pomočjo Ext JS, uporabimo vse argumente prototipu te funkcije.

```
(function () {
    var origExpand = Ext.form.ComboBox.prototype.expand;

    Ext.override(Ext.form.ComboBox, {

        expand: function () {
            origExpand.apply(this, arguments);

            // do other things here
        }
    });
})();
```

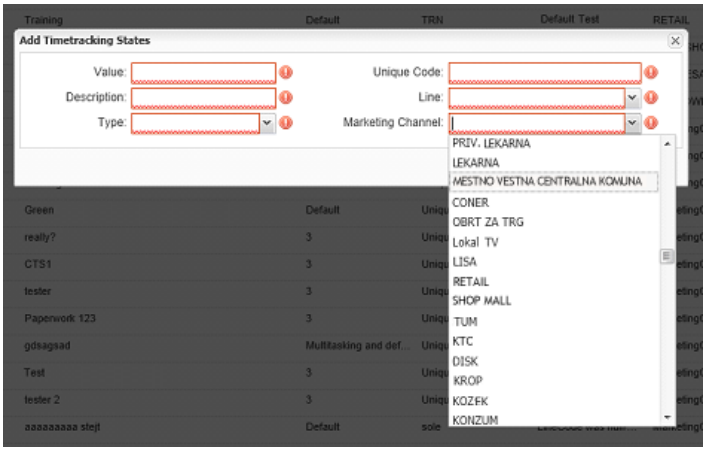
Slika 5: Prepisovanje komponente z uporabo privzete funkcije.

Na sliki 6 lahko vidimo učinek prepisa funkcij onTriggerClick() in expand() izbirnega polja. Privzeto funckija onTriggerClick() ne naredi ničesar, ker je prazna funkcija namenjena prepisovanju po lastni želji, zato niti ne potrebuje dodajanja prototipa te funkcije [8]. Prepis funkcije omogoča, da izbirno polje registrira konfiguracijo »triggerAction«, če ji podamo niz »ignore«. Privzeto »triggerAction« registrira na dva niza: »query« in »all«. V našem primeru »ignore« nastavitev ignorira vse poizvedbe na podatkovno skladišče in naloži zapise v izbirno polje, ter postavi miškin kurzor na njegov element DOM.




Slika 6: Učinek klica funkcije onTriggerClick() ob kliku na gumb 'Add skill'.

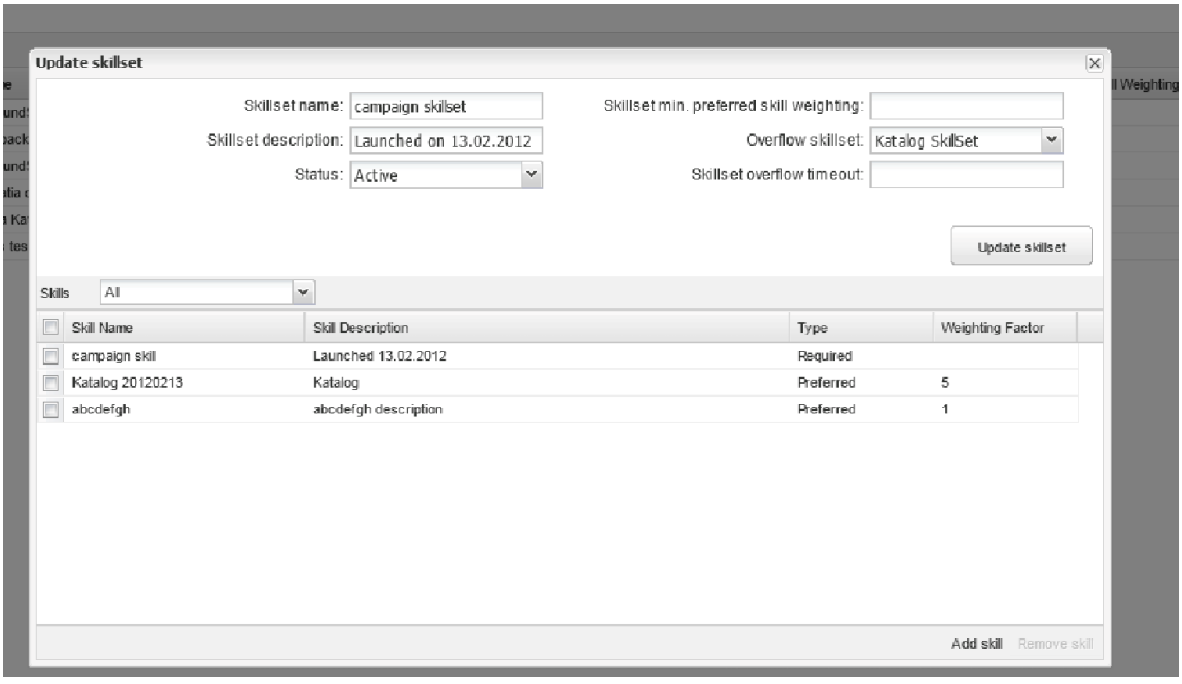
Pri prepisani expand() funkciji najprej na prototip te funkcije naložimo vse argumente, ki se zraven podajo. Znotraj funkcije expand() želimo spremeniti dolžino seznama izbirnega polja glede na dolžino podanih zapisov v podatkovnem skladišču. Velikost prilagodimo na širino pikslov najdaljšega zapisa. Tu smo uporabili drugo komponento imenovano Common.Helper za računanje potrebnih pikslov. Komponenta je enoinstančni razred, ki jo ustvarimo znotraj Ext.onReady() funkcije. Kar komponenta naredi je, da ustvari div element na document.body objekt in ga skrije. Za tem zapiše tekst, ki mu ga podamo v konstruktor v div element, ter vrne njegovo dolžino elementa izraženo v pikslih. Dolžino elementa dobimo s funkcijo getWidth(). Z dobljeno dolžino nato razširimo izbirno polje tako, da bo širina seznama tega polja malenkost večja od širine najdaljšega zapisa v njej. Na sliki 7 lahko vidimo učinek funkcije expand(). Primer kode prepisovanja izbirnega polja je v dodatku A.4.




Slika 7: Učinek funkcije expand() ob kliku na izbirno polje.

### 4.3.3 Večkratna uporaba komponent (JavaScript objekti)

Večkratna uporaba objektov je vedno dobrodošla. Če potrebujemo enako komponento na večih mestih, naredimo njen primerek (ang. instance) le enkrat in ga večkrat uporabimo. Najboljši primer je komponenta Window, ki smo jo uporabili pri dodajanju in urejanju sklopov veščin. Slika 8 prikazuje okno za dodajanje in urejanje sklopa veščin.




Slika 8: Okno, ki se prikaže s klicem funkcije Skillset.Maintenance.GetWindow.

Komponenta Window je ločena v svoji datoteki v kateri se nahaja:

- Skillset.Maintenace.EditWindow – razširitev razreda Ext.Window. Window vsebuje dva otroka, obrazec in tabelo.
- Skillset.Maintenance.WindowInstance – Skillset.Maintenace.EditWindow primerek.
- Skillset.Maintenance.GetWindow – funkcija, ki kot parametre sprejme objekt, ki je potreben za posodobitev obrazca in tabele, funkcijo ki se pokliče ob zaključku (ang. callback function) in obseg (ang. scope).

Vsakič, ko želimo prikazati okno (ang. window) kličemo Skillset.Maintenance.GetWindow funkcijo katera preveri, če Skillset.Maintenance.WindowInstance obstaja. Če ne obstaja, jo naredi in v obeh primerih osveži podatke, ki smo jih podali kot parametre v funkciji. Za urejanje podamo funkciji naslednje parametre:

- Zapis, ki ga dobimo iz podatkovnega skladišča preko poslušalca na miškin klik nad tabelo.
- Zastavico, ki nam pove ali gre za urejanje ali dodajanje novega sklopa veščin.
- Komponento iz katere pokličemo to funkcijo.
- Njeno zaključno funkcijo, ki se izvede ob potrditvi okna.

V zaključni funkciji moramo urediti ali na novo dodati podatke v tabeli. Slika 9 prikazuje funkcijo GetWindow, ki nam prikaže okno.

```
Skillset.Maintenance.WindowInstance = null;

Skillset.Maintenance.GetWindow = function(skillsetData, isActionAdd, scope, scopeFn) {

    if (!Skillset.Maintenance.WindowInstance) {
        Skillset.Maintenance.WindowInstance = new Skillset.Maintenance.Window();
    }

    Skillset.Maintenance.WindowInstance.updateData(skillsetData, isActionAdd, scope, scopeFn);

    return Skillset.Maintenance.WindowInstance;
};
```

Slika 9: Primer funkcije za prikaz okna za dodajanje ali urejanje sklopov veščin.

Window vsebuje tudi orodno vrstico v kateri sta dva gumba, eden za potrditev (v nadaljevanju Confirm) in drugi za preklic okna (v nadaljevanju Cancel). S klikom na gumb Cancel, ne naredimo ničesar, le skrijemo okno s klicem funkcije hide() nad oknom. S klikom na gumb Confirm, vzamemo vse trenutne podatke iz obrazca in tabele ter jih še enkrat validiramo. Ob uspešni validaciji naredimo zahtevek AJAX kot je viden na sliki 10. Za parameter objekta podamo identifikator sklopa veščin in funkcijo, ki se pokliče ob končanem zahtevku. Nato naredimo novo okno za prikaz sklopa veščin s podatki, ki smo jih dobili s strežnika.

```
Ext.Ajax.request({
    url: Urls.GetSkillsetDetails,
    params: { id: skillsetId },
    scope: this,
    callback: function (params, success, response) {
        if (!success) { return; }
        var resultMessage = Ext.util.JSON.decode(response.responseText);

        var data = resultMessage.data.skillset;

        var window = new Skillset.Maintenance.GetWindow(
            data,
            false,
            this,
            function (data, isUpdated) {
                this.lastSelectedRec = data;
                if (!isUpdated) {
                    return;
                }

                var gridView = this.gridPanel.getView();
                gridView.refresh();
                this.skillsetStore.reload();
                var row = gridView.getRow(rowIdx);
                new Ext.Element(row).highlight();
            });

        window.setTitle(getString('JS_SKILL_MAINTENANCE_WINDOW_TITLE_EDIT'));
        window.show();
    }
});
```

Slika 10: Primer funkcije Ext.Ajax.request v kateri prikažemo okno za dodajanje ali urejanje sklopov veščin.

### 4.3.4 Medsebojna odvisnost

Ext JS ima svoje podatke shranjene večinoma v podatkovnih skladiščih. Če želimo uporabiti en primerek podatkovnega skladišča (npr. za sklope veščin) na večih mestih, moramo uporabo le tega ustrezno prilagoditi. Podatkovna skladišča, ki jih želimo večkrat uporabiti so shranjena v globalnih spremenljivkah (kot je npr: Stores.Skillsets) [2]. Slika 11 prikazuje podatkovno skladišče sklopa veščin.

```
// using in campaigns, skillset editform combo
Portal.Stores.AllSkillsets = new Ext.data.JsonStore({
    remoteSort: true,
    idProperty: 'Id',
    root: 'data.skillsets',
    url: Urls.GetAllSkillsets,
    autoLoad: false,
    fields: [
       { name: 'Id', mapping: 'Id' },
       { name: 'Name', mapping: 'Name' },
       { name: 'Description', mapping: 'Description' },
       { name: 'MinSkillWeight', mapping: 'MinSkillWeight' },
       { name: 'OverflowTimeout', mapping: 'OverflowTimeout' },
       { name: 'OverflowSkillsetId', mapping: 'OverflowSkillsetId' },
       { name: 'OverflowSkillsetName', mapping: 'OverflowSkillsetName' },
       { name: 'Skills', mapping: 'Skills' }
    ]
});
```

Slika 11: Primerek podatkovnega skladišča sklopa veščin.

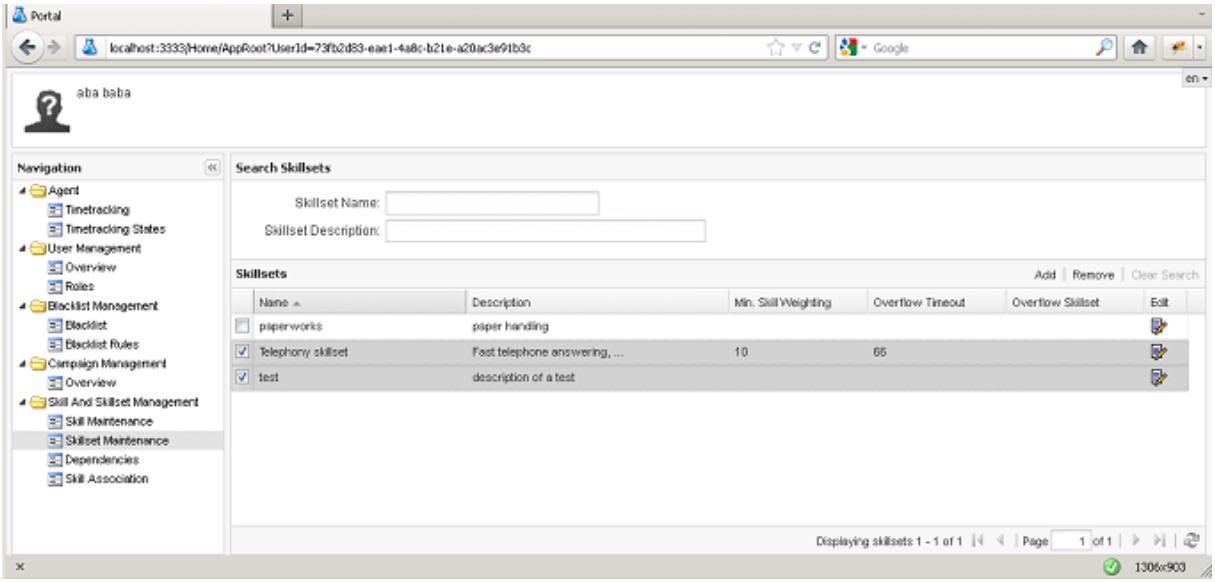
V primeru da osvežimo en sklop veščin v oknu, ki je bil opisan v prejšnjem poglavju, moramo v podatkovnem skladišču zabeležiti, da so se zapisi spremenili (čeprav v tem primeru le eden). Tako lahko to podatkovno skladišče uporabimo tudi v ostalih komponentah z osveženimi podatki.

Primer: Vsakič ko zamenjamo centralni pogled, ki prikazuje neko komponento s podatki (primer: tabela) je najbolje, da nad vsemi podatkovnimi skladišči, ki nastopajo na tej komponenti, kličemo funkcijo ensureData(), ki kot samo ime pove, zagotovi sveže podatke. Klic te funkcije je implementiran v razširitvi razreda Ext.data.Store. V funkciji ensureData() z zastavico preverimo, če so podatki spremenjeni in sprožimo klic funkcije load(), kateri naloži nove podatke v podatkovno skladišče. Funkcija ensureData() sprejme zastavico, katera je lahko »true« ali »false«. Zastavica »true« v vsakem primeru izvede klic funkcije load(), ne glede na to ali se trenutno podatki že osvežujejo. Slika 12 prikazuje implementacijo dveh funkcij ensureData() in forgetData().

```
dataLoaded: false,
dataLoading: false,
waitForEvent: true,
autoLoad: false,
clearOnException: true,
forgetData: function(){
    this.invalidateCache();
},

ensureData: function(force){
    if (this.dataLoading) {
        return;
    }

    if (force === true || !this.dataLoaded) {
        this.dataLoading = this.load();
    }
},
```

Slika 12: Prikaz implementacije funkcij ensureData() in forgetData().

Enako se zgodi v primeru, če želimo iz tabele zbrisati določene sklope veščin. Označimo vrstice katere želimo pobrisati in kliknemo na gumb za odstranitev (v nadaljevanju Remove). Ta sproži klic funkcije onRemoveClick() v kateri se naredi zahtevek AJAX. Če je zahtevek uspešen, kličemo nad podatkovnim skladiščem funkcijo forgetData() (implementirana v razširitvi objekta), katera zastavico »dataLoaded« označi na »false« tako, da se bo naslednjič osvežilo z novimi podatki. Slika 13 prikazuje označene sklope veščin v tabeli in omogočen gumb za brisanje (ang. Remove).




Slika 13: Brisanje sklopov veščin.

Za tem ročno izbrišemo vse zapise iz podatkovnega skladišča z naslednjimi klici:

- remove(Ext.data.Record/Ext.data.Record[] rec) – kot parametere sprejme enega ali več zapisov.
- removeAt(Number index) – kot parameter sprejme index mesta kjer se nahaja zapis.

Funkcija forgetData() se kliče vsakič, ko se podatki spremenijo v podatkovnem skladišču. Le tako zagotovimo najbolj sveže podatke na komponentah. Funkcija ensureData() pa se kliče vsakič, ko se zamenja pogled komponent (primer: ob kliku na neki podmeni v navigacijskem oknu) . Slika 14 prikazuje funkcijo, ki se izvede ob kliku na gumb Remove.

```
onRemoveClick: function () {
    var items = this.gridPanel.getSelectionModel().getSelections();
    var ids = [];

    Ext.each(items, function (itm) {
        ids.push(itm.data.Id);
    }, this);

    if (ids.length) {
        Ext.Ajax.request({
            url: Urls.RemoveSkillset,
            params: { ids: ids.join(",") },
            scope: this,
            callback: function (params, success, response) {
              if (!success) { return; }
              this.skillsetStore.forgetData();
              this.skillsetStore.ensureData();
              Portal.Stores.AllSkillsets.forgetData();
              this.removeBtn.setDisabled(true);
            }
        });
    }
}
```

Slika 14: Akcija, ki se sproži ob kliku na gumb Remove.

V omenjeni funkciji onRemoveClick() želimo izbrisati vse izbrane vrstice iz tabele. To storimo tako, da najprej nad tabelo kličemo funkcijo getSelectionModel(), ki nam vrne izbirni model (objekt) tabele. Izbirni model je model, ki posluša na vse dogodke, ki se zgodijo ob miškinem kliku na tabelo, njeno vrstico ali celico. Na sliki 12 ga lahko vidimo izrisanega v prvem stolpcu tabele. Ko imamo izbirni model, kličemo njegovo funkcijo getSelections(), ki nam vrne izbrane zapise iz podatkovnega skladišča sklopov veščin. Nato se sprehodimo čez vse zapise in shranimo identifikator vsakega zapisa v začetku funkcije inicializirano prazno polje. V primeru, da se v polju nahaja vsaj en identifikator, naredimo zahtevek AJAX, v katerem kot parameter podamo združeno polje v niz identifikatorjev ločenih z vejico. Če se je zahtevek uspešno izvršil, pokličemo funkcijo forgetData() pri vseh odvisnih podatkovnih skladiščih Odvisna podatkovna skladišča so skladišča ki se uporabljajo v večih komponentah. Podatkovnemu skladišču, ki je vezano na tabelo katera prikazuje sklope veščin, pokličemo še funkcijo ensureData() za takojšnjo zagotovitev svežih podatkov. Gumbu, ki je sprožil opisane akcije, pokličemo funkcijo setDisabled() s parametrom »true«, ki ga onemogoči do nadaljnega označevanja vrstic v tabeli.

## 4.4 Upravljanje z agenti (Agent Management)

Uporabniki lahko znotraj tega menija pregledujejo delovne naloge medtem, ko imajo administratorji še pregled nad stanji delovnih nalog.

### 4.4.1 Delovne naloge

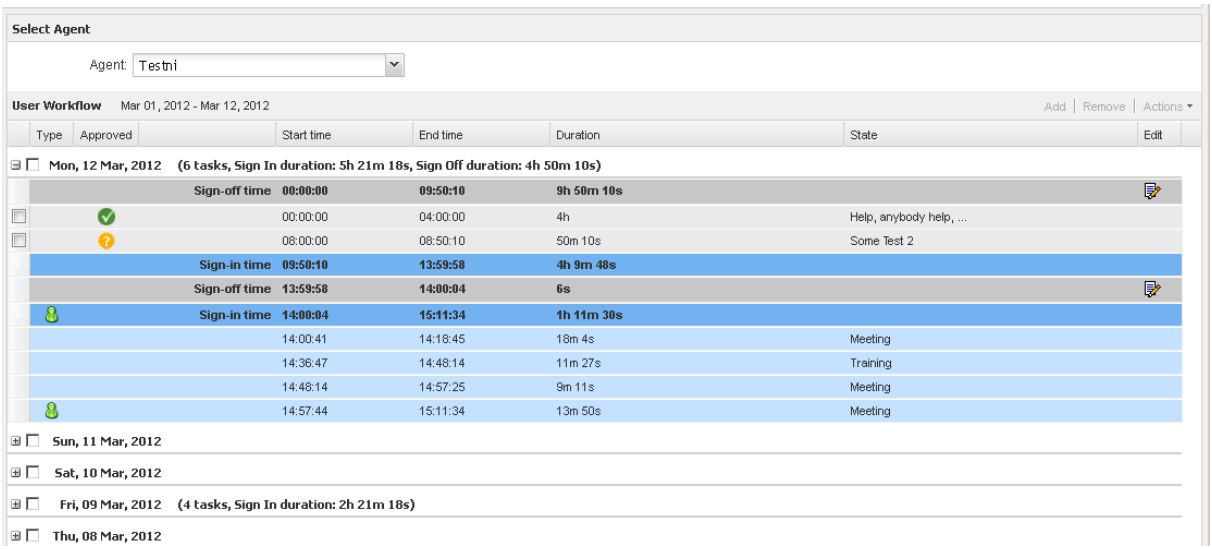
Delovne naloge so za agente in nadzornike zapisane v tabeli, če pa je v sistem prijavljen administrator, pa je zraven tabele še obrazec za izbiro agenta. Obrazec ima eno potrditveno polje vseh agentov ter orodno vrstico z gumbi za dodajanje, brisanje in spreminjanje statusa nalog. Obrazec administratorjem omogoča izbiro agenta za prikaz njegovih delovnih nalog v tabeli.

Tabela je sestavljena iz naslednjih stolpcev:

- Potrditveno polje (če je v sistem prijavljen administrator) - omogoča spreminjanje statusa delovne naloge izbranemu uporabniku. Potrditveno polje se nahaja tudi v sami grupi, ki predstavlja dneve. Ob kliku slednjega potrditvenega polja, se označijo vse naloge, ki jim administrator lahko spremeni status.
- Tip (ang. Type) - prikazuje trenutno delovno nalogo, ki jo uporabnik opravlja.
- Status (ang. Status, prikazano Approved) – prikazuje delovno nalogo katero lahko administrator odobri, zavrne ali spremeni na čakajoč status. Spremeni lahko le na portalu ustvarjene delovne naloge. Uporabniki pa lahko čakajoče naloge tudi izbrišejo.
- Vrsta prijave - prikazuje sliko, če je uporabnik prijavljen na delovno nalogo v drugem sistemu.
- Začetni čas (ang. Start time) - prikazuje čas, ko je agent začel opravljati nalogo.
- Končni čas (ang. End time) - prikazuje čas, ko je agent prenehal z opravljanjem naloge.

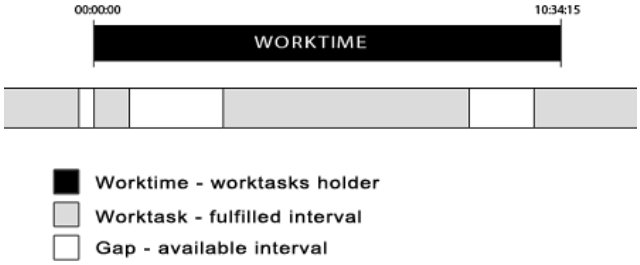
- Trajanje (ang. Duration) - prikazuje trajanje delovne naloge.
- Stanje (ang. State) - prikazuje opis delovne naloge, ki se je opravljala.
- Editiranje (ang. Edit) – stolpec z gumbom za urejanje delovne naloge.

Slika 15 prikazuje administratorski pogled na delovne naloge.



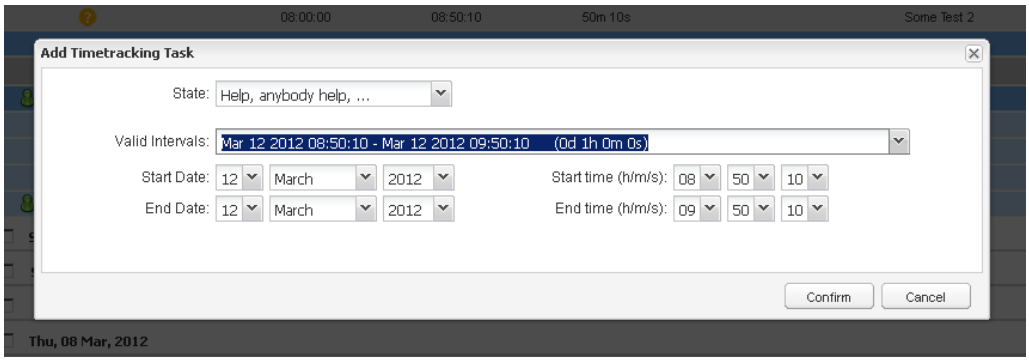

Slika 15: Administratorski pogled na delovne naloge izbranega uporabnika.

Agent si lahko napiše ure s pomočjo vseh stanj delovnih nalog, ki jih ima na razpolago. Ob kliku na gumb za dodajanje ali urejanje delovne naloge se uporabniku odpre okno. Okno tvori obrazec in dva gumba, prvi za potrditev in drugi za preklic okna. V obrazcu imamo obvezna izbirna polja za stanja delovnih nalog: Začetni datum, končni datum, začetno uro in končno uro. Vse to mora uporabnik izpolniti za vnos nove ali urejanje stare delovne naloge. V obrazcu se nahaja še izbirno polje, ki vsebuje vse veljavne intervale, kar uporabniku pomaga pri vnašanju datuma in ure. Datum, ure, minute in sekunde, tako začetne kot končne, se morajo ujemati z že obstoječim časovnim intervalom, ki se ne sme prekrivati z drugimi nalogami. Delovne naloge lahko dobimo tudi iz drugega zunanjega sistema zato zna biti računanje veljavnosti intervalov težavno. Slika 16 prikazuje skico dveh intervalov, ki sta na voljo (ang. available interval) znotraj prikazanega delovnega časa (ang. worktime).




Slika 16: Primer enega izmed mnogih časovnih trakov, prihajajočih iz drugega sistema.

Pri vnašanju začetnih in končnih podatkov (datuma, ure, minute in sekunde), ki tvorijo nek časovni interval mora ta ležati na enemu izmed veljavnih intervalov, ki so na voljov izbirnemu polju. Validacija se sproži ob vsaki spremembi, ki se zgodi v obrazcu. Slika 17 prikazuje uporabnikov obrazec za vnos delovne naloge.

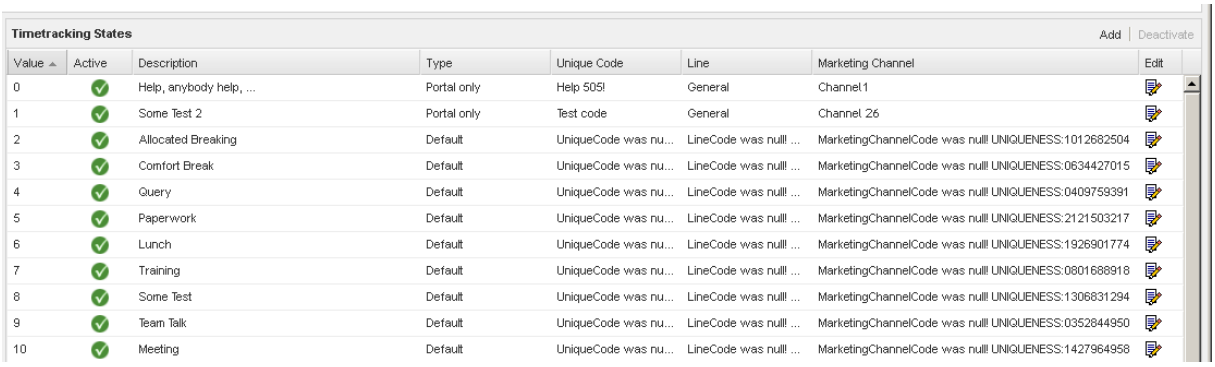



Slika 17: Pravilno izpolnjen obrazec za vnos delovne naloge.

## 4.4.2 Stanja delovnih nalog

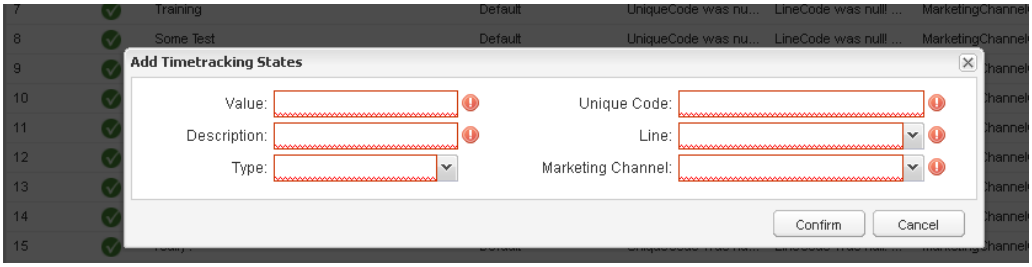
Tabela stanj delovnih nalog nam omogoča pregledovanje, dodajanje in spreminjanje aktivnosti stanja delovnih nalog. Sestavljena je iz stolpcev: Vrednosti, aktivnosti, opisov, tipov, unikatne kode, linij in marketinškega opisa.

Tabela ima še orodno vrstico, na kateri je gumb za dodajanje novih stanj (Add) in gumb za spreminjanje aktivnosti stanja (Edit). Slika 18 prikazuje tabelo stanja delovnih nalog.




Slika 18: Pregled stanja delovnih nalog.

Ob kliku na gumb Add se odpre novo okno, ki vsebuje obrazec. Na obrazcu so vnosna polja enaka kot stolpci v tabeli. Posebnost obrazca je v tem, da se morajo unikatna koda, linija in marketinški opis razlikovati v vsaj enem izmed zapisov drugih stanj. Ko je uporabnik uspešno izpolnil vsa polja, lahko potrdi vnos s klikom na gumb Confirm. Slika 19 prikazuje potrjen prazen obrazec stanja delovnih nalog.



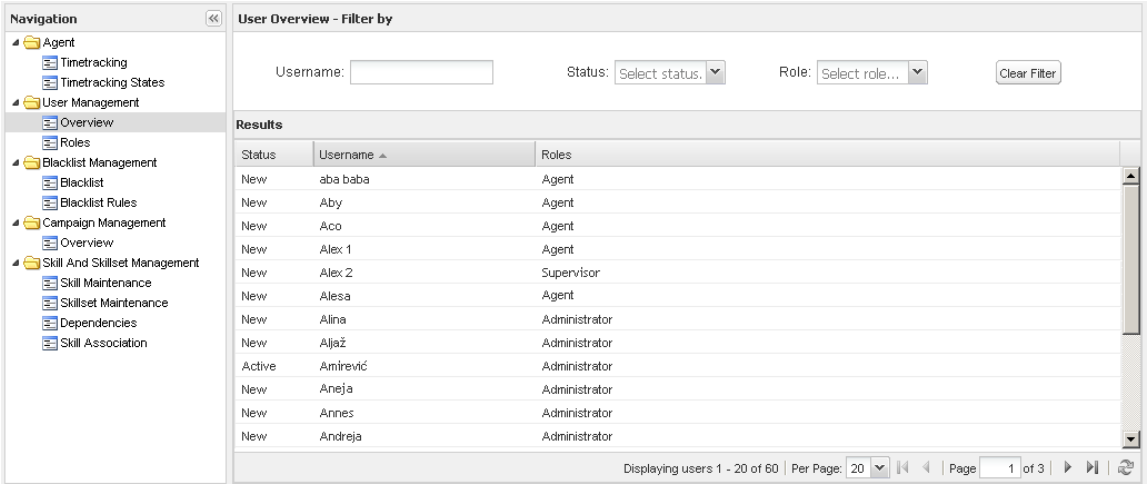

Slika 19: Sprožena validacija na obrazcu za dodajanje stanja delovnih nalog.

## 4.5 Upravljanje z uporabniki (User Management)

Upravljanje z uporabniki je omogočeno le administratorjem. Ti imajo celoten pregled nad uporabniki klicnega centra ter njihovim urejanj klicnih nastavitev. Poleg tega imajo še možnost urejanja privilegijev, kar vpliva na uporabniško dostopnost na portalu.

### 4.5.1 Pregled uporabnikov

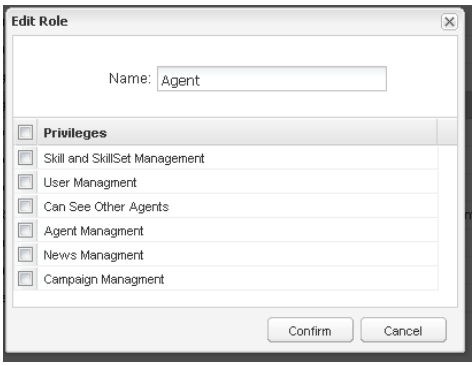
Pregled uporabnikov je sestavljen iz obrazca za iskanje uporabnikov in tabele za prikazovanje rezultatov iskanja. Obrazec je sestavljen iz vnosnega polja za iskanje po uporabniškemu imenu, dveh izbirnih polj (za status in vloge) in gumb za ponastavitev iskalnika. Tabela vsebuje stolpce: Status, uporabniško ime in vloge. Tabela ima tudi orodno vrstico za oštevilčenje strani, ki omogoča prikaz izbranega števila agentov na stran. Ob dvojnemu kliku na vrstico v tabeli se odpre okno, ki je sestavljeno iz treh zavihkov. Na prvem zavihku so vsi podatki o uporabniku, na drugem so klicne nastavitve (polje za vnos klicne številke, izbirno polje za nastavitev oddelka in potrditvena polja za zvočne nastavitve) in na tretjem se nahaja tabela za urejanje veščin uporabniku. Komponento te tabele smo uporabili tudi v poglavju 4.8.2. Slika 20 prikazuje administratorjev pogled na uporabnike.




Slika 20: Pregled uporabnikov z orodno vrstico za oštevilčenje strani.

### 4.5.2 Pregled vlog

Pregled vlog vsebuje tabelo, katera ima v orodni vrstici dodano polje za iskanje po imenu. Iskanje se sproži na »keyup« dogodek, ki se zgodi ob izpustitvi tipke na tipkovnici. Tabela ima naslednje stolpce: Ime vloge, ime skupine aktivne mape in privilegije. Ob kliku na tretji stolpec se odpre okno za urejanje privilegijev. Okno vsebuje tabelo, ki ima dva stolpca: potrditveni model izbiranja (CheckboxSelectionModel) in ime privilegije. Administrator lahko doda ali odstrani privilegije s klikom na potrditveno polje. Po končanem urejanju ima možnost shraniti oziroma preklicati to urejanje. S potrditvijo se privilegiji zapišejo v podatkovno skladišče vlog. Zatem se okno zapre. Slika 21 prikazuje okno kjer se lahko ureja privilegije.



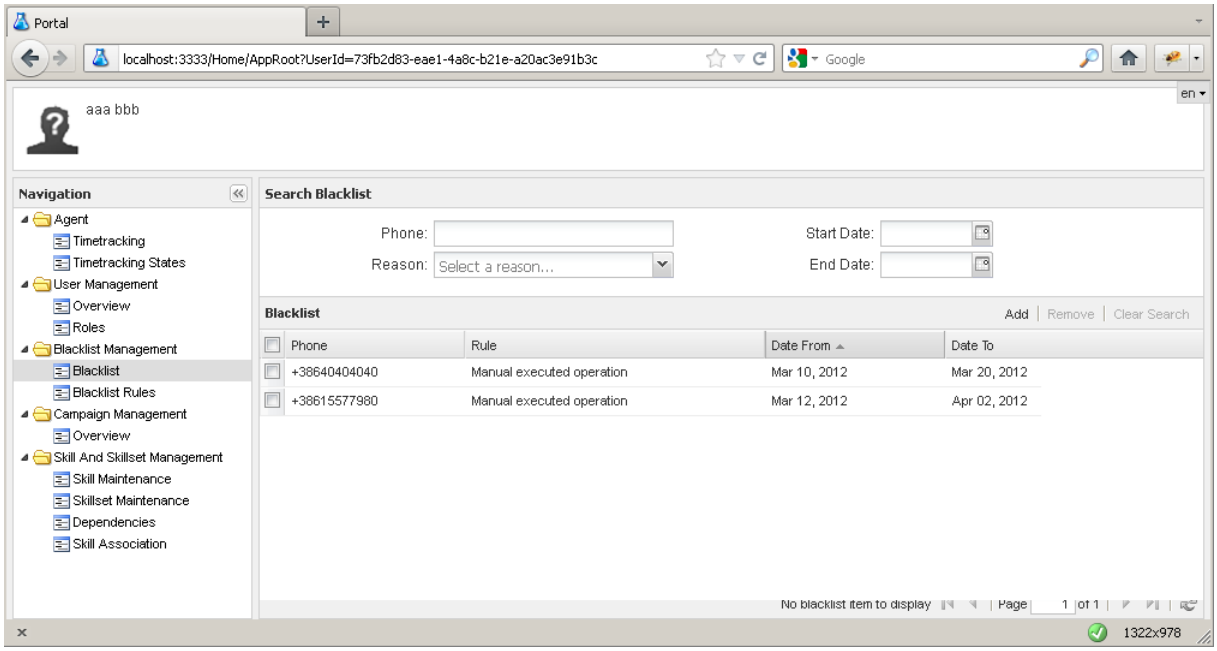

Slika 21: Okno za urejanje privilegijev.

## 4.6 Upravljanje z blokado številk (Blacklist Management)

Vsebovalnik omogoča administratorjem in nadzornikom dodajanja klicne telefonske številke na seznam blokiranih številk, spreminjanje le teh in dodajanje pravil za avtomatsko blokiranje številk.

### 4.6.1 Seznam blokiranih številk

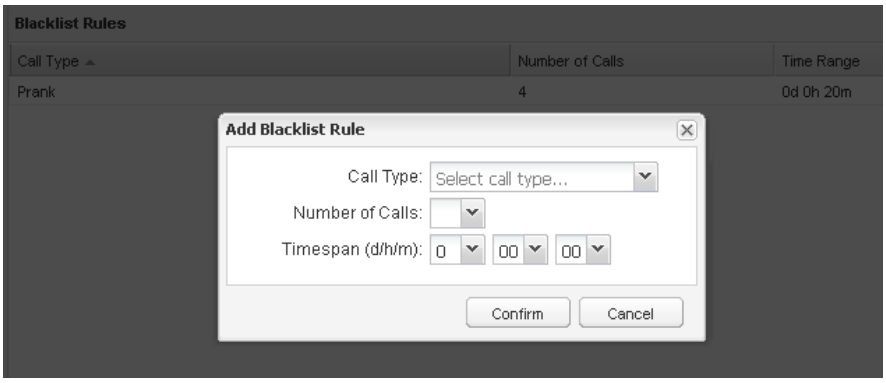
Seznam je sestavljen iz obrazca za iskanje in tabele za prikaz rezultatov iskanja. Obrazec vsebuje polja za vnos telefonske številke, razloga blokade, začetni dan in končni dan blokade. Obrazec ima tudi orodno vrstico z gumbi za dodajanje in brisanje številke in ponastavitev iskalnih parametrov. Tabela je sestavljena iz naslednjih stolpcev: potrditvenega polja, telefonske številke, pravila blokade telefonske številke ter datum začetka in konca blokade telefonske številke. Tabela vsebuje tudi orodno vrstico za prikaz oštevilčenja strani. Ob kliku na gumb Add, se odpre okno za dodajanje številke na seznam blokiranih številk. Okno vsebuje obrazec, ki je enak obrazcu za iskanje številk le da ima ta razlog že vnaprej izbran (»Manual executed operation«). Ko so vsa polja izpolnjena, lahko uporabnik dodajanje potrdi, s čimer se okno zapre in zapis shrani v podatkovno skladišče blokiranih številk. Na sliki 22 lahko vidimo implementacijo pogleda blokiranih številk.




Slika 22: Seznam blokiranih številk.

## 4.6.2 Pravila za blokado številk

Tabela pravil vsebuje: tip klica, število klicev in trajanje blokade telefonskih številk. V orodni vrstici se nahajata gumba za dodajanje in brisanje pravil. Ob kliku na gumb dodaj, se odpre novo okno z obrazcem za vnos pravila blokade številk. Obrazec vsebuje izbirna polja za izbiro tipa klica, ter število klicev in časovni razpon v dnevih, urah in minutah. Ob pravilno izpolnjenem obrazcu lahko nadzornik ali administrator potrdi dodajanje pravil. Ob kliku na gumb Confirm se zapis shrani v podatkovno skladišče pravil za blokado številk. Gumb Remove se omogoči ob kliku na vrstico v tabeli. Slika 23 prikazuje okno za dodajanje pravila, kjer lahko nadzornik ali administrator vneseta: tip klica, število klicev ter trajanje blokade telefonskih številk v dnevih, urah in minutah.

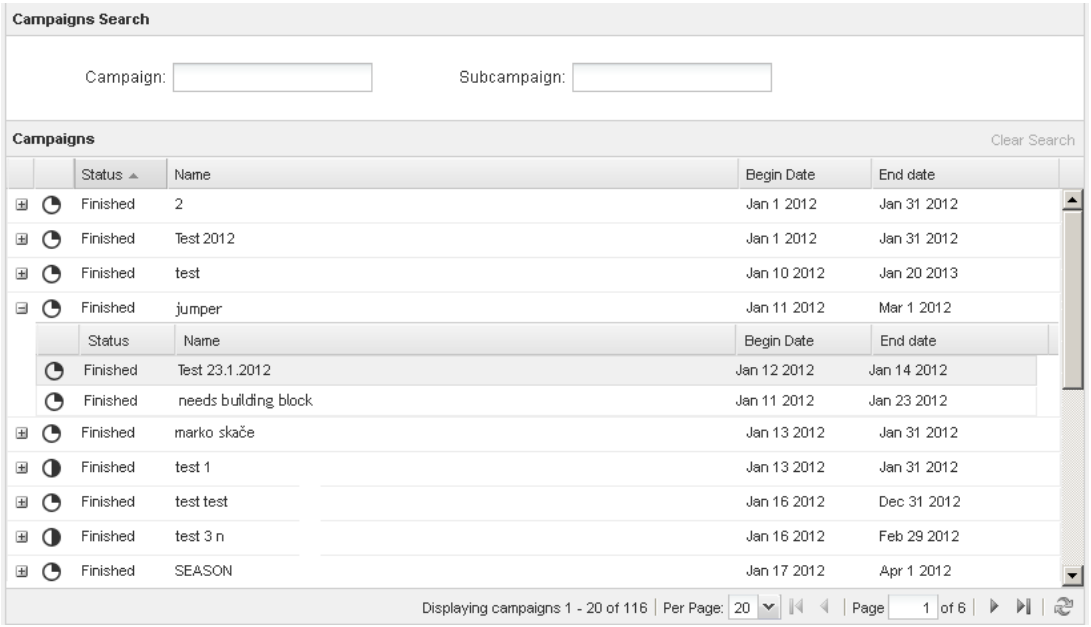



Slika 23: Prikaz okna za dodajanje pravila blokade številk.

## 4.7 Upravljanje s kampanjami

Upravljanje s kampanjami je razdeljeno na obrazec za iskanje in tabelo za prikaz kampanj. Iskati je mogoče po imenu kampanje ali podkampanje. Iskalnik ima dodan gumb za ponastavitev obrazca (ang. Clear Search). Tabela kampanj je sestavljena iz gnezdene tabele, katera prikazuje podkampanje. Obe tabeli imata naslednje stolpce: stanje, status, ime ter začetni in končni datum. Tabela kampanj ima poleg vsega tega tudi ikono v prvem stolpcu, ki ob kliku nanj prikaže ali skrije notranjo tabelo podkampanj. Za boljši pregled vseh kampanj, ima še dodano orodno vrstico za oštevilčenje strani. Tabela podkampanj ima možnost klika na vrstico, kar sproži dogodek »rowclick« v katerem se odpre okno za izdelavo in dodajanje gradnikov podkampanji. Gradniki so glavni del kampanj in so sestavljeni iz naslova, opisa, ter lastnosti, ki jo določeni gradnik predstavlja (lahko ima možnost izbire izdelka za trženje, drugačen uvodni govor, ipd.).

Ob spremembi lastnosti podkampanj mora administrator podkampanjo shraniti in objaviti. Pri urejanju gradnikov podkampanj se ti avtomatsko shranijo. Slika 24 prikazuje kampanjo »jumper« z gnezdeno tabelo njenih podkampanj.



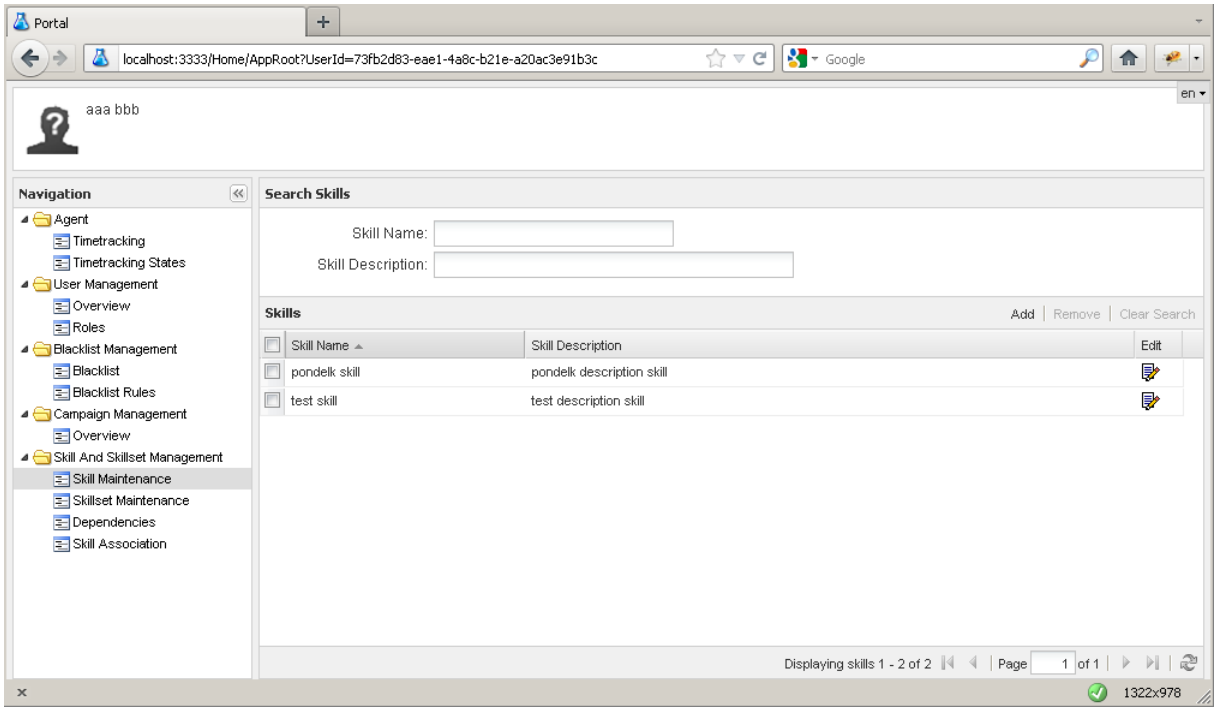

Slika 24: Pregled kampanj in podkampanj v gnezdeni tabeli.

## 4.8 Upravljanje z veščinami in sklopi veščin

Na portalu je prisotno mnogo veščin, te predstavljajo usposobljenost klicnih agentov. Lahko jih dodajamo in brišemo agentom, jih urejamo in z njimi ustvarjamo sklope veščin ter pregledujemo njihovo medsebojno odvisnost.

### 4.8.1 Vzdrževanje veščin

Vzdrževanje veščin omogoča iskanje, dodajanje, urejanje in brisanje veščin. Sestavljeno je iz obrazca za iskanje in tabele za prikaz najdenih zadetkov. Obrazec je sestavljen iz dveh polj, prvo za ime in drugo za opis veščine po kateri želimo iskati. Na obrazec so pripeti trije gumbi: dodajanje (Add), brisanje (Delete) in brisanje vpisanih podatkov iz obrazca (Clear Search). Tabela vsebuje naslednje stolpce: Potrditveno polje, ime, opis in gumb za urejanje veščine. Ob kliku na gumb za dodajanje ali ikono za urejanje se odpre novo okno z enakim obrazcem kot je pri iskanju. Obrazec v oknu ima vklopljeno validacijo, ki sproti preverja če polji nista izpolnjeni in če je ali ime ali opis že v podatkovnem skladišču veščin. V nasprotnem primeru omogoči dodajanje veščine. Ob izbiri na vsaj eno potrditveno polje iz tabele, se omogoči gumb za brisanje s katerim izbrišemo izbrane veščine. Slika 25 prikazuje obrazec in tabelo za vzdrževanje veščin.




Slika 25: Vzdrževanje veščin.

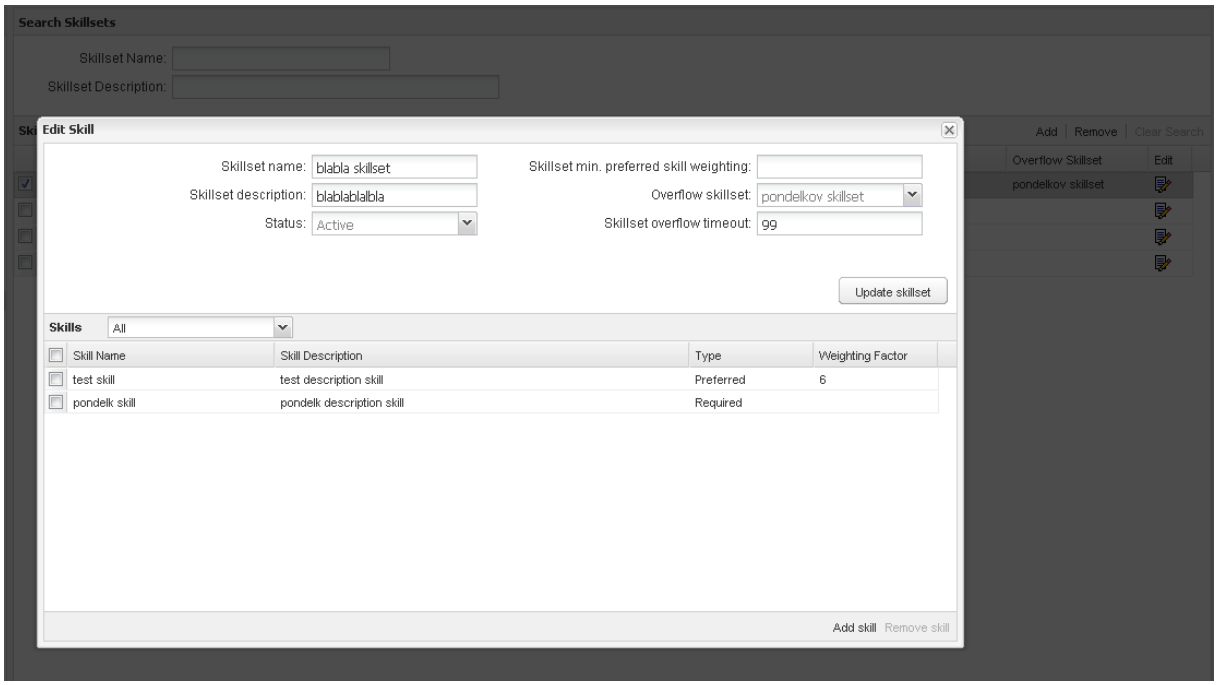
### 4.8.2 Vzdrževanje sklopov veščin

Vzdrževanje sklopov veščin je po izgledu in funkcionalnosti enako vzdrževanju veščin. Le da obrazec išče po sklopih veščin. V tabeli pa je razlika le, da prikazuje sklope veščin in ima dodane še tri stolpce:

- Minimalna teža veščin – za zagotavljanje minimalne teže zaželjenega faktorja teže veščin.
- Zasedenost sklopa veščin – v primeru da je ta sklop veščin zaseden se zagotovitovi dodatnega.
- Maksimalno dovoljeni čas zasedenosti – Maksimalen čas, ki je dovoljen za zasedenost sklopa veščin.

Okno za dodajanje in urejanje sklopov veščin ima obrazec za vnos podrobnosti sklopa veščin in tabelo za urejanje veščin. Obrazec vsebuje naslednja polja: Ime, opis, status (aktiven ali neaktiven), minimalna teža veščine, zasedenost sklopa veščin (izbirno polje vseh sklopov veščin) in maksimalno dovoljen čas zasedenosti. Tabela, ki prikazuje vse veščine pod sklopom veščin je sestavljena iz naslednjih stolpcev: Potrditveno polje, ime, opis, tip in faktor teže. Stolpca ime in tip je mogoče urejati s pomočjo izbirnega polja. V imenu se nahajajo imena vseh veščin, medtem ko tip (ki je lahko zaželjen ali zahtevan), če je izbran zaželjen, omogoči možnost urejanja tudi stolpcu, ki določa faktor teže. Slednji administratorju omogoča vnos številke od 1 do 128. Slika 26 prikazuje okno za urejanje sklopov veščin.

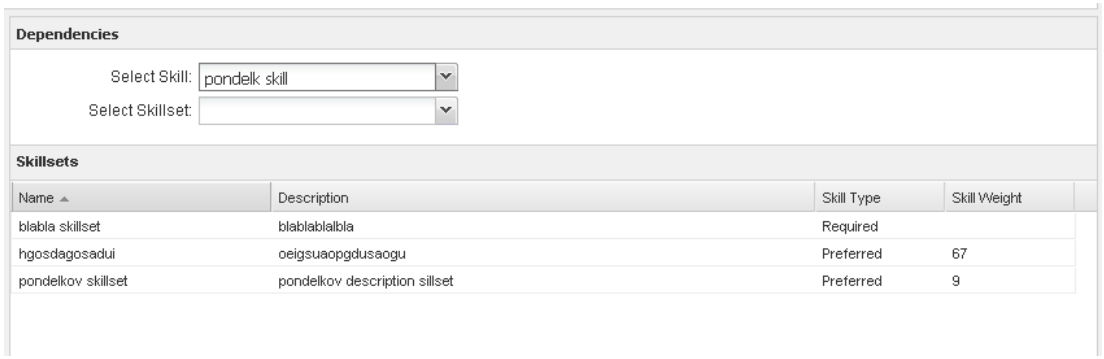
Pravilno izpolnjen obrazec urejanja sklopa veščin je, če ima izpolnjeno ime in opis ter, če je dodana vsaj ena veščina. Gumb Remove nad tabelo sklopov veščin ima enako funkcionalnost kot pri vzdrževanju veščin.




Slika 26: Prikaz okna za urejanje sklopa veščin.

### 4.8.3 Odvisnosti veščin in sklopov veščin

Administratorjem je omogočen pregled medsebojnih odvisnosti med veščinami in sklopi veščin. Pregled tvori obrazec in tabelo. V obrazcu se nahajata dve izbirni polji, eno za iskanje odvisnosti po veščini in drugo za iskanje odvisnosti po sklopu veščin. Ob izbiri elementa iz prvega polja, se prikaže tabela kot je za iskanje po sklopu veščin. Razlika je le v tem, da je na tabelo vezano podatkovno skladišče, ki kot iskalni parameter prejme identifikator iskalne (izbrane) veščine in kot rezultat vrne vse sklope, ki vsebujejo to veščino. Ob izbiri elementa iz drugega polja se prikaže tabela, ki je enaka tabeli za iskanje veščin le, da ima ta tabela podatkovno skladišče, ki vsebuje veščine izbranega sklopa veščin. Slika 27 prikazuje tabelo sklopov veščin ob izbiri veščine na izbirnem polju.




Slika 27: Prikaz odvisnosti veščin in sklopov veščin ob izbiri veščine.

### 4.8.4 Masovno asociiranje veščin

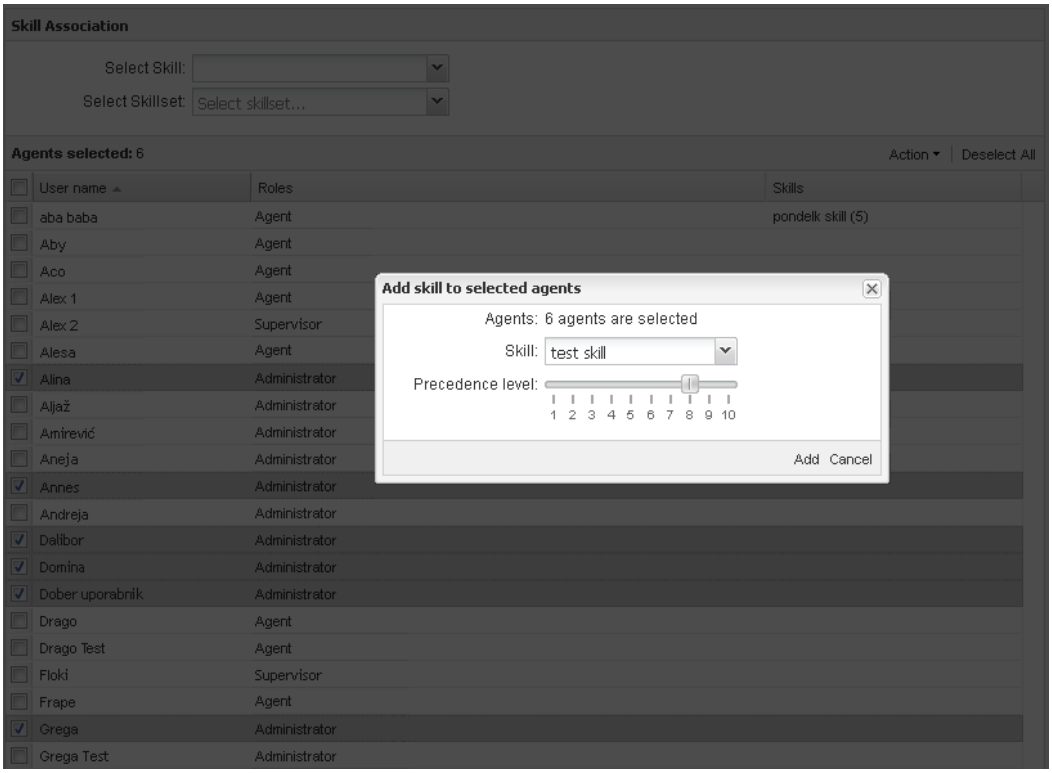
Pogled na masovno asociiranje veščin administratorjem omogoča iskanje in urejanje veščin uporabnikov. Iskanje je mogoče po veščini ali sklopu veščin. Ob izbiri veščine se prikažejo vsi agenti, ki so asociirani z izbrano veščino. Ob izbiri sklopa veščin pa se v tabeli prikažejo vsi agenti, ki so asociirani z veščinami izbranega sklopa veščin.Tabela omogoča prikaz uporabniških imen asociiranih agentov, njihovih vlog in veščin. Tabela ima orodno vrstico za oštevilčenje strani in potrditveno polje za masovno asociiranje veščin. Ob izbiri enega ali več agentov se omogočita gumba za akcije in izbris izbire, ki se nahajata na obrazcu. Gumb akcije omogoča dodajanje in brisanje veščin izbranim uporabnikom.

Ob kliku na gumb dodaj, se odpre okno z naslednjimi polji:

- Agents – število izbranih agentov.
- Skill – izbirno polje z vsemi veščinami.
- Precedence level – stopnja prednosti izbrane veščine.

Ob kliku na gumb Remove se odpre isto okno le, da je brez izbire stopnje prednosti. Izbirno polje veščin ima podatkovno skladišče s podatki, ki so unija veščin izbranih agentov.

Slika 28 prikazuje okno za dodajanje veščin izbranim agentom.




Slika 28: Masovno asociiranje veščin.

# 5 Sklepne ugotovitve

Diplomska naloga opisuje tehnologije in orodja, ki jih razvijalec potrebuje pri izdelavi spletnega portala s pomočjo enega izmed odprto kodnih spletnih ogrodij. Ogrodje je mogoče uvoziti na katerokoli spletno stran in začeti izdelovati atraktivno spletno aplikacijo z zelo dobrim znanjem HTML-ja, CSS-a in JavaScript-a.

Pri izdelavi spletnega portala smo upoštevali lastnosti kot so enostavnost uporabe in zanesljivost. Hiter razvoj ogrodij in pojavljanje vedno več novih tehnologij na trgu predstavlja breme razvijalcu. Razvijalcu se ni potrebno naučiti vseh tehnologij posebej, saj mu lahko moderna ogrodja veliko olajšajo razvoj spletnih aplikacij.

Pri izdelavi spletnega portala s pomočjo ogrodja Ext JS smo naleteli na nekaj težav, ki so bile najdene v ogrodju. Dobra primera za to sta »vbox« in »hbox« postavitvi, ter zagotavljanje svežih podatkov v podatkovnem skladišču. Hrošč je v kodi »vbox« postavitve.

Prvi problem lahko predstavimo pri vsebovalniku z dvema otrokoma, kjer ima prvi otrok nastavljeno fiksno višino (primer height: 200), drugi pa ima parameter flex nastavljen na 1. V tem primeru se plošča izriše pravilno. Takoj ko ostane prvi otrok brez vsebine in/ali je njegova višina enaka 0, je »vbox« postavitev raztegnila drugega otroka čez celotno višino, ob tem pa ga ni pravilno pozicionirala, ker je bil »top« parameter NaN (ang. not a number). To nastane zaradi napake pri računanju »topOffset«:

```
topOffset += calcs.height + childMargins.bottom;
```

Calcs.height je lahko neznan, kar je za Ext ogrodje pravilno, saj tako ve da ta komponenta ni računala svoje višine. Številska operacija z neznano vrednostjo (undefined) nastavi topOffset na NaN. Napako popravimo s preverjanjem ali obstaja calcs.height in po potrebi vzamemo 0:

```
topOffset += (calcs.height || 0) + childMargins.bottom;
```

Do problema zagotavljanja svežih podatkov je prišlo pri skladiščenju podatkov, ko smo dodali nov zapis. Skladišče ni nosilo podatka ali se je kak zapis dodal ali izbrisal. Prepisali smo razred Ext.data.Store, v katerem smo mu dodali informacijo ali skladišče vsebuje sveže podatke ali ne, ter nekaj pripadajočih funkcij, ki upravljajo s podano informacijo.

Razširitev spletnega portala je mogoča in zelo preprosta. Poljubno se lahko doda nove menije z novimi funkcionalnostmi, kot tudi dopolni k že obstoječim. Spremeni se lahko tudi celotno obliko, postavitev in izgled portala. Vendar je to potrebno narediti s previdnostjo pri prepisovanju Ext JS CSS datotek, saj nam lahko uniči privzeti izgled komponent.

# Viri

[1] Shea Frederick, C. Ramsay, S. 'Cutter' Blades, S. Blades: Learning Ext JS 3.2, Packt Publishing Ltd., October 2010

[2] Jesus Garcia: Ext JS IN ACTION, Manning Publications Co., 2011

[3] John Resig: Pro JavaScript Techniques, Apress Media LLC, 2006

[4] Paul Haine: HTML Mastery, Semantics, Standards, and Styling, December 2006

[5] John Allsopp: Developing with web standards, New Riders, 2010

[6] Andy Budd with C. Moll & S. Collison: CSS Mastery, Second edition, friendsofED, October 2009

[7] I. Novak, A. Velvart, A. Granicz, G. Balassy, A. Hajdrik, M. Sellers, G. Hillar, A. Molnar, J. Kanjilal: Visual Studio 2010 and .NET 4 Six-in-One (Wrox Programmer to Programmer), Wiley Publishing Inc., 2010

[8] (2012) Ext JS forum. Dostopno na: http://www.sencha.com/forum/forumdisplay.php?39-Ext-JS-Community-Forums-3.x

[9] (2012) Ext JS API. Dostopno na: http://docs.sencha.com/ext-js/3-4/

[10] (2012) MVC. Dostopno na: http://www.codeproject.com/Articles/383153/The-Model-View-Controller-MVC-Pattern-with-Csharp

# Kazalo slik

# DODATEK A

## A.1 Ext.onReady funkcija:

```
Ext.namespace('SM');
Ext.onReady(function () {
   try {
      Ext.Ajax.timeout = 3600000; //1 hour
      Ext.QuickTips.init();

      var viewport = new SM.Viewport();
      var languageSelectorControl = new SM.Common.LanguageSelector({
         languages: SM.InitData.AvailableLanguages,
         style: { position: "absolute", right: "0px", top: "0px" },
         renderTo: document.body
      });

      var revNumber = '<%= ConfigurationManager.AppSettings["RevisionNumber"] %>';
      var revTitle = '<%= ConfigurationManager.AppSettings["RevisionTitle"] %>';
      if (revNumber) {
         var rv = "R: " + revNumber;
         if (revTitle) {
            rv += "-" + revTitle;
         }

         var revisionPanel = new Ext.Panel({
            cls: "revision",
            html: rv,
            renderTo: document.body
         });
      }

      Ext.Ajax.request({
         url: SM.Urls.GetPrivileges,
         callback: function (params, success, response, data, isSmError, errMsg) {
            if (isSmError) { return; }

            var resultMessage = Ext.util.JSON.decode(response.responseText);
            var results = data.results;
            var treeNode = Ext.getCmp('treewidget').getRootNode();

            for (itm in results) {
               UserPrivileges[UserPrivilegesMapping[itm]] = results[itm];
            };

            var firstVisibleLeafId;

            var setRecursivelyToVisible = function (parent) {
               if (this.hasChildNodes) {
                  this.eachChild(setRecursivelyToVisible);
               }

               if (UserPrivileges[this.attributes.requiredPrivilege] == true) {
                  if (!firstVisibleLeafId) {
                     firstVisibleLeafId = this.id;
                     setInitialScreen(Ext.getCmp('treewidget'), firstVisibleLeafId);
                  }
                  this.hidden = false;
                  this.getUI().show();
               }
```

```
            };

            treeNode.eachChild(setRecursivelyToVisible);
            treeNode.hidden = false;
            UserPrivileges.loaded = true;

            SM.Event.EventManager.fireEvent('privilegesloaded');
         },
         scope: this
      });
   }
   catch (ex) {
      console.error(ex);
   }
});
```

## A.2 viewport.js:

```
Ext.namespace("SM");

SM.Viewport = Ext.extend(Ext.Viewport, {
   layout: 'border',
   forceFit: true,
   hideMode: "offsets",
   defaults: {
      collapsible: true,
      split: true,
      margins: '5px',
      autoScroll: true,
      border: true
   },

   navigationTreePnl: null,
   headerPnl: null,
   workspacePnl: null,

   constructor: function (cfg) {
      Ext.apply(this, cfg);

      this.navigationTreePnl = new Ext.tree.TreePanel({
         region: 'west',
         margins: '0 0 0 5',
         cmargins: '0 5 0 0',
         collapsible: true,
         id: 'treewidget',
         title: 'Navigation',
         minWidth: 160,
         width: 200,
         useArrows: true,
         split: true,
         loader: new Ext.tree.TreeLoader(),
         root: new Ext.tree.AsyncTreeNode({
            id: 'treeroot',
            hidden: true,
            expanded: true,
            defaults: { expanded: true, hidden: true },
```

```
            children: [{
               text: 'Agent Management',
               requiredPrivilege: 'AgentManagement',
               children: [
                  { hidden: true, text: 'Timetracking', id: "12", leaf: true },
                  {   hidden:   true,   requiredPrivilege:   'CanSeeOtherAgents',   text:
'Timetracking States', id: "13", leaf: true }
               ]
            }, {
               text: 'User Management',
               requiredPrivilege: 'UserManagement',
               children: [
                  { hidden: true, text: 'Overview', id: "20", leaf: true },
                  { hidden: true, text: 'Roles', id: "21", leaf: true }
               ]
            }, {
               text: 'Blacklist Management',
               requiredPrivilege: 'UserManagement',
               children: [
                  { hidden: true, text: 'Blacklist', id: "30", leaf: true },
                  { hidden: true, text: 'Blacklist Rules', id: "31", leaf: true }
               ]
            }, {
               text: 'Campaign Management',
               requiredPrivilege: 'CampaignManagement',
               children: [
                  { hidden: true, text: 'Overview', id: "40", leaf: true }
               ]
            }, {
               text: 'Skill and Skillset Management',
               requiredPrivilege: 'SkillManagement',
               children: [
                  { hidden: true, text: 'Skill Maintenance', id: "60", leaf: true },
                  { hidden: true, text: 'Skillset Maintenance', id: "61", leaf: true },
                  { hidden: true, text: 'Dependencies', id: "62", leaf: true },
                  { hidden: true, text: 'Skill Association', id: "63", leaf: true }
               ]
            }]
         }),
         rootVisible: false,
         listeners: {
            click: SM.Click.Handler,
            scope: this
         }
      });

      this.workspacePnl = new SM.Workspace({
         region: 'center',
         padding: '0',
         margins: '0 5 0 0',
      });

      this.items = [
         new Ext.Panel({
            region: 'north',
            height: 70,
            margins: '5 5 0 5',
            collapsible: false,
            split: false
         }),
         this.navigationTreePnl,
         this.workspacePnl
```

```
      ];

      SM.Viewport.superclass.constructor.apply(this, arguments);
   }
});

Ext.reg('xsmviewport', SM.Viewport);
```

## A.3 workspace.js:

```
SM.Workspace = Ext.extend(Ext.Panel, {
   collapsible: false,
   border: false,
   header: false,
   layout: 'card',
   layoutConfig: {
      deferredRender: true
   },

   constructor: function () {
      this.items = this.constructItems();

      SM.Workspace.superclass.constructor.apply(this, arguments);

      //register to global events
      SM.Event.EventManager.on('setCenterItem', this.activateItem, this);
   },

   activateItem: function (itm) {
      this.getLayout().setActiveItem(itm);
   },

   constructItems: function () {
      this.timetracking = new SM.Agent.Timetracking({
         itemId: 'agenttimetracking'
      });
      this.timerecordingstates = new SM.Agent.Timetracking.States({
         itemId: 'timerecordingstates'
      });
       this.useroverview = new SM.User.Management.Overview({
         itemId: 'useroverview'
       });
       this.userroles = new SM.User.Management.Roles({
         itemId: 'rolesoverview'
       });
       this.userblacklist = new SM.User.Management.Blacklist({
         itemId: 'blacklist'
       });
       this.userblacklistrules = new SM.User.Management.BlacklistRules({
         itemId: 'blacklistrules'
       });
       this.campaignoverview = new SM.Campaign.Management({
         itemId: 'campaignmanagement'
       });
       this.news = new SM.News({
         itemId: 'news'
       });
       this.skill = new SM.Skill.Maintenance({
```

```
        itemId: 'skillmaintenance'
      });
      this.skillsetmaintenance = new SM.Skillset.Maintenance({
        itemId: 'skillsetmaintenance'
      });
      this.dependencies = new SM.Dependencies({
        itemId: 'dependencies'
      });
      this.skillassociation = new SM.Skill.Association({
        itemId: 'skillassociation'
      });

      return [
        this.timetracking,
        this.timerecordingstates,
        this.useroverview,
        this.userroles,
        this.userblacklist,
        this.userblacklistrules,
        this.campaignoverview,
        this.news,
        this.skill,
        this.skillsetmaintenance,
        this.dependencies,
        this.skillassociation
      ];
   },

   destroy: function () {
      //Uregister from global events
      SM.Event.EventManager.un('setCenterItem', this.setActiveItem, this);

      delete this.timetracking;
      delete this.timerecordingstates;
      delete this.useroverview;
      delete this.userroles;
      delete this.userblacklist;
      delete this.userblacklistrules;
      delete this.news;
      delete this.skill;
      delete this.skillsetmaintenance;
      delete this.dependencies;
      delete this.skillassociation;

      SM.Workspace.superclass.destroy.call(this);
   }
});

Ext.reg('xsmworkspace', SM.Workspace);
```

## A.4 extension.combobox.js:

```
(function () {
   var origExpand = Ext.form.ComboBox.prototype.expand;

   Ext.override(Ext.form.ComboBox, {
      onTriggerClick: function () {
         if (this.readOnly || this.disabled) { return; }
         if (this.isExpanded()) {
            this.collapse();
            this.el.focus();
         }
         else {
            this.onFocus({});
            if (this.triggerAction == 'ignore') {
               this.onLoad();
            }
            else if (this.triggerAction == 'all') {
               this.doQuery(this.allQuery, true);
            }
            else {
               this.doQuery(this.getRawValue());
            }
            this.el.focus();
         }
      },

      expand: function () {
         origExpand.apply(this, arguments);

         if (this.list) {
            var maxlength = 0,
                maxLengthText = "";

            Ext.each(this.getStore().data.items, function (rec) {
               if (rec.data[this.displayField]) {
                  var len = rec.data[this.displayField].length;
                  if (len > maxlength) {
                     maxlength = len;
                     maxLengthText = rec.data[this.displayField];
                  }
               }
            }, this);

            var comboWidth = this.getWidth();
            var newWidth = SM.Common.Helper.measureWidth({
               minWidth: comboWidth,
               text: maxLengthText
            });

            this.doResize(newWidth);
         }
      },

      isValueInStore: function (v) {
         return this.getStore().find(this.valueField, v) >= 0;
      }
   });
})();
```