DIPLOMAMUNKA

Nagy László

Debrecen

2010

Debreceni Egyetem

Informatika Kar

Kliensfüggetlen felhasználói felület megvalósításának vizsgálata, egy példa implementációval

Témavezető:

Dr. Juhász István

Egyetemi adjunktus

Készítette:

Nagy László

Programtervező Matematikus

Debrecen

2010

Tartalom

1. Bevezetés

Az Internet elterjedésével a Web alkalmazások szerepe is megnőtt. Népszerűségük oka, hogy használatukhoz csak Internet elérésre és egy böngésző programra van szükség és a szolgáltatásaikat a világon bárhonnan igénybe lehet venni. Legelterjedtebb Web alkalmazások a web áruház, web mail, közösségi oldal, blog, fórum, online aukciós oldal, netbank és még nagyon rengeteget találhatunk az Interneten. A Web alkalmazás fejlesztés napjaink egyik legfelkapottabb ága az informatikának. Ezért választottam ezen területhez kapcsolódó témát a Diplomamunkámnak.

Egy web alkalmazás felhasználói felülete – a különböző technológiák segítségével – nagyon sokoldalú megjelenéssel és funkcionalitással rendelkezhet. Diplomamunkám célja, egy olyan felhasználói felület megtervezése és megalkotása, amely megfelel egy mai Web alkalmazás igényeinek. Megvizsgálom azokat a problémákat, amelyek felmerülhetnek egy felhasználói felület megtervezése és létrehozása közben. Munkám közben a felhasználói felületeknek nem a kinézetére helyeztem a hangsúlyt, hanem a működésükre. Azt vizsgáltam, hogy mi történik egy-egy felhasználói interakció esetén és azt, hogy hogyan jut el az adat a felhasználói felületre és hogyan jut el az információ a felhasználótól az alkalmazás üzleti rétegéig.

A következő fejezetekben először a web alkalmazások általános jellemzőivel és lehetőségeivel foglalkozom. Majd a felhasználói felületek felépítését és funkcionalitását vizsgálom meg. Végül az általam létrehozott web alkalmazás és a hozzá kapcsolódó technológiák leírása következik.

A Debreceni Egyetemen folytatott tanulmányaim során a Java nyelv kiemelkedő szerepet játszott, ezért választottam a Java webes technológiáit az alkalmazásom megírásához. Választásom a Struts2 keretrendszerre esett, ami ezelőtt számomra ismeretlen volt. Ennek a keretrendszernek a segítségével, hoztok létre egy web alkalmazást, ami egy adminisztrációs rendszer autó szervizek számára.

Munkám során a NetBeans 6.7.1 fejlesztőkörnyezetet használom. Alkalmazás szerverként a Glassfish 2.1 vállalati alkalmazás szervert veszem igénybe. Az alkalmazás adatbázisának menedzselésében a Postgresql 8.3 adatbázis kezelő nyújt majd segítséget. Alkalmazásom a két legnépszerűbb böngésző programon az Internet Explorer 8-on és Mozilla Firefox 3.6-on valamint a kevésbé elterjedt Google Chrome 4.1 böngészőn próbáltam ki.

Elvárásaim szerint, diplomamunkám végére érve, képes leszek olyan alkalmazások fejlesztésére a Struts 2 kertrendszer segítségével, amelyek kliensfüggetlen felhasználói felülettel rendelkeznek. Reményeim szerint, egy web alkalmazás fejlesztés iránt érdeklődő tanuló programozó számára is hasznos olvasmány.

2. Fogalmak

Ebben a fejezetben a web alkalmazás fejlesztéssel kapcsolatos fogalmakat, definíciókat tárgyalom.

- **HTML:** A HTML (HyperText Markup Language) egy leíró nyelv, mely weboldalak leírására fejlesztettek ki, és mára már internetes szabvánnyá vált. Segítségével egy strukturált dokumentumot hozhatunk létre, melyek tartalmazhatnak bekezdéseket, linkeket, felsorolásokat, táblázatokat. Beágyazhatunk képet vagy más objektumot is. Interaktív lapok készítésére is alkalmas. A HTML dokumentumokba beágyazhatóak szkript nyelvek, melyek segítségével a web oldalak funkciói bővíthetők (JavaScript) és megjelenésük megadható (CSS).

- **XHTML:** „Az XHTML a HTML megfogalmazása XML-ben. Gyakorlatilag nincs jelentős eltérés a két nyelv között, csak a formai követelmények lettek szigorúbbak.” [Wikipédia – HTML szócikk]

- **HTML DOM:** HTML Document Object Model definiálja a szabványos módját a HTML dokumentumok elérésének és kezelésének. A DOM fastruktúrában ábrázolja A HTML dokumentumot. Objektumokként ad meg minden dokumentum elemet tulajdonságaikkal és metódusaikkal.

- **CSS:** „A CSS (Cascading Style Sheets) a számítástechnikában egy stílusleíró nyelv, mely a HTML vagy XHTML típusú strukturált dokumentumok megjelenését írja le. Ezen kívül használható bármilyen XML alapú dokumentum stílusának leírására is. A CSS-t a weblapok szerkesztői és olvasói egyaránt használhatják, hogy átállítsák vele a lapok színét, betűtípusait, elrendezését, és más megjelenéshez kapcsolódó elemeit. A tervezése során a legfontosabb szempont az volt, hogy elkülönítsék a dokumentumok struktúráját (melyet HTML vagy egy hasonló leíró nyelvben lehet megadni) a dokumentum megjelenésétől (melyet CSS-sel lehet megadni). Az ilyen elkülönítésnek több haszna is van, egyrészt növeli a weblapok használhatóságát, rugalmasságát és a megjelenés kezelhetőségét,

másrészt csökkenti a dokumentum tartalmi struktúrájának komplexitását.” [Wikipédia – CSS szócikk]

- **JavaScript:** az Internet szkript nyelve. Web oldalak milliói használják, hogy segítségével bővítsék funkcionalitásukat. A legnépszerűbb szkript nyelv az Interneten és minden jelentősebb böngészőn működik, mint például az Internet Explorer, Mozilla Firefox, Google Chrome, Opera és Safari. A JavaScript, mint programozási nyelv egy objektumalapú szkript nyelv. A JavaScript kód vagy közvetlenül a HTML fájlban vagy külön szövegfájlban van. A JavaScript esetében a futási környezet jellemzően egy web böngésző. Bár a nyelvet szabványosították, mégis részben különbözően implementálják a JavaScriptet a különböző böngészők.

- **DHTML:** A Dinamikus HTML (Dynamic HTML) nem egy nyelv. A DHTML egy kifejezés a dinamikus és interaktív web oldalak készítésének módjára. Kombinálja a HTML-t JavaScriptet, HTML DOM-ot és a CSS-t.

- **XML:** „Az XML (Extensible Markup Language, Kiterjeszthető Leíró Nyelv) a W3C által ajánlott általános célú leíró nyelv, speciális célú leíró nyelvek létrehozására.” [Wikipédia – XML szócikk] Szabályokat tartalmaz, amelyek dokumentumok elektronikus leírására vonatkoznak. Tervezésekor a fő szempontok az egyszerűség, az általánosság és az Interneten keresztül való használat volt. Támogatja az Unicode karakterformátumot, így a világ bármelyik nyelvén írhatunk XML dokumentumot. Az elsődleges célja strukturált szövegek és információk megosztása az Interneten keresztül.

- **AJAX:** „Az Ajax (Asynchronous JavaScript and XML) interaktív web alkalmazások létrehozására szolgáló webfejlesztési technika. A weblap kis mennyiségű adatot cserél a szerverrel a háttérben, így a lapot nem kell újratölteni minden egyes alkalommal, amikor a felhasználó módosít valamit. Ez növeli a honlap interaktivitását, sebességét és használhatóságát. Az Ajax nem egy technológia önmagában, hanem egy kifejezés közösen használt technológiákra. A következő technikák kombinációja:

    - XHTML (vagy HTML) és CSS a tartalom leírására és formázására.

    - DOM kliens oldali szkript nyelvekkel kezelve a dinamikus megjelenítés és a már megjelenített információ együttműködésének kialakítására.

- XMLHttpRequest objektum az adatok aszinkron kezelésére a kliens és a webszerver között. Néhány Ajax keretrendszer esetén és bizonyos helyzetekben IFrame-et használnak XMLHttpRequest objektum helyett.

XML formátumot használnak legtöbbször az adattovábbításra a kliens és a szerver között, bár más formátumok is megfelelnek a célnak, mint a formázott HTML vagy a sima szöveg.”

[Wikipédia – Ajax szócikk]

3. Web alkalmazások

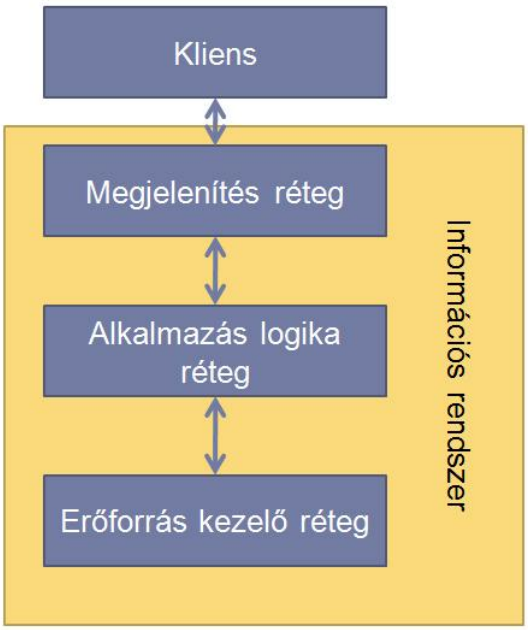
Szoftverfejlesztési szempontból a web alkalmazások olyan programok, amelyek az Interneten vagy intraneten keresztül érhetőek el. Manapság szinte minden számítógépen rendelkezésre áll valamilyen böngésző program, mely lehetővé teszi a web alkalmazások használatát.

Nagy előnyük, hogy karbantarthatóak a kliens gépek szoftvereinek módosítása nélkül. Korábbi kliens-szerver architektúrák esetén, minden egyes számítógépnek rendelkezni kellett az alkalmazás kliens programjával, amely azt igénybe akarta venni. Ezeket a klienseket külön-külön kellett telepíteni és karbantartani. Ezért az alkalmazás frissítése kliens oldalon nem volt hatékony. Ezzel ellentétben egy web alkalmazás dinamikusan generál szabványos HTML vagy XHTML oldalakat, melyeket a hálózaton küld át a kliensnek. Ezek statikus dokumentumokként jutnak el a kliens számítógépre, de interaktív felhasználói élményt nyújtanak, olyan eszközök segítségével, mint a JavaScript, AJAX, DHTML, CSS. Segítségükkel a web alkalmazások felhasználói felülete akár egy operációs rendszeréhez hasonló lehet. Megoldható velük a billentyűzet és egér kezelése, lejátszhatnak hangot vagy megjeleníthetünk felugró ablakokat.

Az oldalakat a böngésző program jeleníti meg, így az alkalmazás független a kliens gép operációs rendszerétől. A HTML és a fent említett eszközök a W3C (World Wide Web Consortium) szabványai. A böngésző programok néhol eltérnek a szabványoktól, valamint a beállításaik is befolyásolhatják a működésüket, ezeket az alkalmazás fejlesztése és támogatása során figyelembe kell venni. Ezen tulajdonságok miatt szinte minden platformon képesek lesznek működni a web alkalmazások.

3.1. Web alkalmazások felépítése

Egy tipikus web alkalmazás a háromrétegű architektúra tervezési minta alapján épül fel. Ez a három réteg a megjelenési réteg, az alkalmazás logikai réteg és az erőforrás kezelő réteg, melyeknek egymástól független a fejlesztésük.



1. ábra: Háromrétegű architektúra

Az 1. árba szemlélteti a rétegeket és a rétegek közti kommunikációt. Alul helyezkedik el az erőforrás kezelő réteg vagy más néven adatbázis réteg, mely tartalmazza az adatbázis szervert, ahol az adatokat érhetjük el és tárolhatjuk azokat. Függetlenül és a többi réteg szemszögéből ismeretlen módon működik. Az elkülönítettség növeli a skálázhatóságot és a teljesítményt. Az adatbázis réteg felett helyezkedik el az alkalmazás logikai réteg, ami az alkalmazás működését szabályozza. Megvalósítja az üzleti logikának és szabályoknak megfelelő folyamatokat. Legfelül a megjelenítési réteg található, ami megjeleníti azokat az információkat és a szolgáltatásokat, amelyeket az alkalmazás nyújt a felhasználónak.

A háromrétegű architektúra működése lineáris, a rétegek csak a szomszédos rétegekkel kommunikálhatnak. A megjelenítési rétegkéréseket küld az alkalmazás logikai rétegnek, melyeket kiszolgál azáltal, hogy adatot kérdez le az adatbázis rétegből vagy módosításokat hajtat végre ott, majd az eredmény visszaküldi a megjelenítési rétegre.

3.2. Web alkalmazások fejlesztése

Egy mai, nagyobb üzleti web alkalmazás fejlesztéséhez szinte nélkülözhetetlen valamilyen fejlesztői keretrendszer. Számos előnnyel járhat egy cég számára a keretrendszerek használata. Egy keretrendszer bizonyos egyszerű feladatok megoldását leegyszerűsíti, automatizálja, így a fejlesztő nem fárad bele a hétköznapi feladatokkal való bajlódásba. A programozó eltérhet ezektől a megoldásoktól, de egy jó keretrendszer esetében ezek a megoldások nehezen visszautasíthatóak. Ha egy elterjedt keretrendszert alkalmaz a cég, akkor könnyen talál képzett munkaerőt, aki tapasztalt az adott keretrendszerben történő fejlesztésben.

Egy-egy nagy web alkalmazás fejlesztése csoportos projekt munkában történik. Ezért a munka során fontos szerepet kap az alkalmazás dokumentációja. Az egyes csoportoknak külön feladatkörökön kell dolgozniuk. A háromrétegű architektúra lehetővé teszi, hogy az egyes rétegeket fejlesztőknek nem kell ismerniük a többi réteg implementációját.

4. Felhasználói felületek

A felhasználói felület az alkalmazás azon része, amely a felhasználóval történő közvetlen kommunikációért felel. Fontos szerepe van a rendszer használhatósága szempontjából. Az alkalmazás lehet hatékony és gyors, de ha a felhasználó nem tudja kezelni, akkor ez mind mit sem ér. Az üzleti alkalmazásokat általában nem programozók használják. ezért az alkalmazásnak „emberi” nyelven kell kommunikálnia a felhasználóval. A felhasználók különböző képességűek és tudásúak lehetnek, ezért a felhasználói felületnek egyértelműnek és könnyen kezelhetőnek kell lennie.

A következőkben a HTML alapú felhasználói felületek eszközeit és tulajdonságaikat vizsgálom meg. Egy felhasználói felületen az adatbevitel, az adatmegjelenítés és végül a vezérlés eszközeit veszem sorra.

4.1. Adatbevitel

Az űrlap a legalapvetőbb beviteli eszköz egy web oldalon. Egy-egy űrlap összetartozó adatok bevitelére szolgál, például egy felhasználó adatainak regisztrálása, melyeket adatbázisban tárolunk le. Lehetnek olyan mezők, melyek kitöltése kötelező. Ennek tényét fel kell tüntetni a felületen, hogy a felhasználó is tudja, mely adatot kötelező megadnia. Általában az ilyen mezőket egy csillaggal vagy a többi mezőétől különböző színnel jelölik. Egy beviteli mező jellemzője az is, hogy egy adatot milyen módon lehet bevinni a segítségével, valamint az, hogy a bevitt adatnak milyen ellenőrzéseken (validáción) kell átesnie. Rendelkezik továbbá címkével, ami a mező által képviselt adatot nevezi meg. Az űrlap különböző típusú beviteli mezőkből épül fel. A következőkben ezeket a típusokat és jellemzőiket veszem sorra.

4.1.1. Szövegmező

A szövegmező rövid karakteres adatok bevitelére szolgál. A bevitt adat, reprezentálhat egyszerű karakterláncot, számot vagy dátumot. Abban az esetben, ha karakterláncként szeretnénk kezelni, nem igényel további átalakítást mielőtt letárolnánk, azonban a tartalmát előfordulhat, hogy ellenőriznünk kell. Például, a felhasználó címének tárolásakor a város nevének egy létező városnak kell lennie, helyesen írva. Ezt biztosíthatjuk már az adat bevitelekor, oly módon, hogy a mező kitöltésekor segítséget nyújtunk a felhasználónak.

Megoldás lehet az automatikus kiegészítés (auto completer). Ekkor miközben a felhasználó gépeli be a város nevét, megjelennek az addig beírt szótöredéknek megfeleltethető városnevek. A megjelenő városnevekből választva a mező kitöltésre kerül a választott várossal. Másik eszköz az adatbevitel segítésére a súgó (fieldhelp). Azt hogy a mezőhöz tartozik súgó, azt egy a mező melletti ikon jelzi, melyre kattintva megjelenik a súgó képernyő. A súgó képernyőn megjelenő értékek listájából választva lehet a mezőt kitölteni. Ha lehetséges értékek száma túl nagy, szükség van szűrőmezőkre, melyek segítségével csökkenthetjük a megjelenő lista hosszát. Súgót általában akkor használnak, ha az kereséshez összetettebb feltételeket akarnak biztosítani. Például, ha . Ezek a megoldások nem biztosítják, hogy ne lehessen, olyan értéket bevinni, amely nem valós vagy nem megfelelő, ezért a bevitt értéket ellenőrizni, validálni kell.

Ha a mező értékének típusa szám, akkor a bevitt karakterláncot át kell alakítani szám típusúvá. Ennek sikerességét, valamint azt is ellenőriznünk kell, hogy a megfelelő érték tartományba esik-e. Például a felhasználó életkora nem lehet negatív.

Dátum típusú mező esetén is hasonló vizsgálatokat kell elvégezni. A dátum beviteli formátumának megfelelő karakterláncot kell bevinnie a felhasználónak, hogy a dátummá való átalakítás egyszerűen elvégezhető legyen. Erre megoldás, ha a dátum elemeit, azaz az évet, a hónapot és a napot külön mezőben lenyíló lista segítségével lehet bevinni. Ez biztosítja az egyértelműséget, hogy mely érték mely adatot tükrözi. Elterjedt megoldás a mai web alkalmazásokban a grafikus dátumsúgó, mely segítségével a felhasználó az egérrel választhatja ki a kívánt dátumot egy naptárszerű kis képernyőn. Ezután a súgóképernyő a megfelelő formátumban adja meg a mező számára a dátumot. Ha biztosítjuk azt, hogy a dátumot csak a dátumsúgó segítségével lehessen kitölteni, akkor az ellenőrzés során nem kell

vizsgálni a megfelelő formátumot. Ellenkező esetben a dátum megfelelő intervallumba esése mellett a dátum típusú adattá történő átalakítás sikerességét is kell vizsgálnunk.

4.1.2. Szövegdoboz

Nagyobb terjedelmű szöveges adat bevitelére alkalmas a szövegdoboz. Megjegyzéseket, hosszabb leírásokat, üzeneteket olvashatunk be segítségével. Tartalmának jellege miatt nem tudjuk tartalmi ellenőrzés alá vetni, legfeljebb csak helyesírás-ellenőrzést végezhetünk. Sok mai alkalmazásban lehetőségünk van formázott szöveg bevitelére, a szövegdobozhoz rendelt egyszerűsített szövegszerkesztővel.

4.1.3. Lenyíló lista

Ha egy adatbeviteli mező által bevitt adat csak kis számú, előre meghatározott értékből kerülhet ki, akkor alkalmazzuk a lenyíló listát. Például a felhasználó nemének megadása. A mező kezdetben üres és a lenyíló listát egy nyíl jelzi a mező mellett. A beviteli területre vagy a nyílra kattintva megjelenik a lehetséges értékek listája, melyből a felhasználó kiválaszthatja a megfelelőt. Ennél a típusú mezőnél csak a kitöltés tényét kell ellenőriznünk, ha az kötelező.

4.1.4. Rádió gomb

Kis számú lehetséges érték közüli választást tesz lehetővé a rádió gomb. A képernyőn minden választható elem megjelenik és közülük csak egyet választhat a felhasználó az értékek mellett megjelenő kis gombra kattintva. Ha egy másik értéket választunk egy összetartozó csoportból, akkor az előzőleg kiválasztott mező kiválasztása megszűnik. Jól használható a rádió gomb csoport egy tesztkérdés válaszainak megjelenítésére. A helytelen adatbevitelt a rádió gombnál sem kell ellenőrizni.

4.1.5. Jelölő négyzet

Több lehetőség közül akár többet is kiválaszthat a felhasználó a jelölő négyzet (check box) segítségével. Az értékek mellett megjelenő kis négyzetekbe kattintva választhatunk közülük. A kiválasztást a négyzetben megjelenő pipa jelzi.

Logikai típusú adatok bevitelére is alkalmas. Például, segítségével a felhasználó beállíthatja bizonyos szolgáltatások ki és bekapcsolását.

4.1.6. Gomb

Általában az űrlapokon a gomb a kitöltés végét, azaz az űrlap elfogadására szolgál. A felhasználó a kitöltést követően a gombra kattintva jelzi a bevitel végét. Ezután küldi el az űrlap az adatokat feldolgozásra az üzleti rétegbe. Ekkor történik az adatok validálása. Sikeres validálás után hajtódik végre az adatok tárolása vagy a már tárolt adatok módosítása.

4.1.7. Validálás

A validálás a bevitt adatok érvényesítését jelenti. Az kitöltött adatbeviteli űrlap validálására azért van szükség, hogy a felhasználó ne tudjon véletlenül vagy akár szándékosan, olyan adatot bevinni a rendszerbe, amely nem valós vagy hibás. Az ilyen értékek hibás működést okozhatnak az alkalmazásunkban.

Abban az esetben, ha a bevitt érték nem megfelelő, azaz nem esik bele az elfogadható érték tartományba vagy az átalakítás nem volt sikeres szám vagy dátum típusra, a felhasználó számára jeleznünk kell ezt, egy hibaüzenet formájában. A hibaüzenetnek egyértelműnek kell lennie és köthető kell legyen ahhoz a mezőhöz, amely értéke a hibát okozta. A nem megfelelő vagy a felhasználó számára érthetetlen, félreérthető hibaüzenetek összezavarhatják a felhasználót. Hosszú távon a rossz hibaüzenetek elidegeníthetik az alkalmazástól a felhasználót. Végső esetben akár meg is tagadhatja annak használatát.

4.2. Adatmegjelenítés

A felhasználói felületeken szükség van olyan eszközökre, amelyek képesek megjeleníteni a felhasználó számára adatokat. Egy alkalmazás napi használata közben szükség lehet bizonyos adatok megjelenítésére. Például egy web áruházban az adott napon leadott rendelése és azok adatainak megjelenítésére. A következőkben az erre alkalmas eszközök és jellemzőik leírása olvasható.

4.2.1. Áttekintő képernyő

Az áttekintő képernyő valójában egy űrlap, amit nem adatbevitel céljából használnak, hanem összetartozó adatok megjelenítésére. Az adat mezők nem szerkeszthetőek és rendelkeznek kezdeti értékkel. Gyakran használják egy űrlap kitöltése után arra, hogy a felhasználó ellenőrizni tudja az általa bevitt adatokat. Például a web áruházban leadott rendelés véglegesítése előtt még a felhasználó ellenőrizheti a szállítási címet, a rendelt mennyiséget és a végösszeget. Ha rendben találja az adatokat, elfogadhatja azokat vagy javíthatja a hibás értékeket.

4.2.2. Lista

Egyszerű adatok felsorolására alkalmasak a listák. Lehetnek számozottak és nem számozottak. A listák egymásba ágyazhatók, így allisták is létrehozhatóak. A lista elemeihez eseményeket rendelhetünk, így alkalmasak menük kialakítására vagy egy a lista elemeihez tartozó művelet meghívására. Például a felhasználók listájából kiválasztva egy felhasználót módosíthatjuk az adott felhasználó adatait.

4.2.3. Táblázat

A táblázatok segítségével összetett adatokat jeleníthetünk meg. A táblázat egy-egy sora egy-egy elem adatait tartalmazza. Alkalmas nagyobb mennyiségű adat megjelenítésére is. Az

adatok táblázatba történő szervezése, jól áttekinthetővé teszi a megjelenített információkat. A táblázat fejlécében az oszlopok tartalmát jelző címkéket helyezhetünk el. Sok elemet tartalmazó táblázat nagyon hosszú lehet, ezért vagy maximalizáljuk a megjelenített sorok számát vagy képessé tesszük a lapozásra. Ha a felhasználó egy elemet és annak adatait szeretné megtalálni a táblázatban, akkor nagy mennyiségű elem közül nehezen, sok keresés után találhatja meg. Ennek elkerülésére használhatunk a táblázathoz szűrőmezőket, melyek segítségével a táblázatban megjelenő sorok számát csökkenthetjük. Például az alkalmazottak adatait tartalmazó táblázathoz az alkalmazott nevéhez egy szűrőmezőt rendelve, a felhasználó ki tudja szűrni a nagy mennyiségű alkalmazott közül azt, amelyiket ő keres vagy csak kevés a keresett alkalmazott nevéhez hasonló alkalmazott adatai jelennek meg. Az átláthatóságot és a kiválasztást elősegíthetjük valamely oszlop értékei szerinti rendezéssel. A listához hasonlóan a táblázat soraihoz is rendelhetünk műveleteket, ezen felül akár egy soron belül az egyes cellákhoz külön-külön is rendelhetünk funkciókat.

4.3. Menü

A menük az alkalmazáson belüli navigálást, az egyes funkciók elérését szolgálják. A felhasználó számára egyértelműnek kell lennie a menünek. A menüpontokon belül lehetnek almenük, melyek segítségével kialakítható egy hierarchikus menürendszer. A menü hierarchia felépítését következetesen kell megtervezni. Egy menüponthoz tartozó almenüben csak logikailag a menüponthoz tartozó funkciók kaphatnak helyet. Ellenkező esetben a felhasználó nehezen találhatja meg az egyes menüpontokat. Akár össze is zavarhatja egy következetlen menü a felhasználót. A menük elrendezése és pozíciója változatos lehet, de a fő szempont a menük kialakításánál az átláthatóság és a könnyű kezelhetőség.

5. Implementáció

Ebben a fejezetben az általam írt web alkalmazást és az elkészítéséhez használt technológiákat és az implementáció lépéseit mutatom be. Az alkalmazás neve e-Car Sevice és egy autó szerviz adminisztrációs feladatait látja el. A legfontosabb cél az alkalmazásom felhasználói felületének kliensfüggetlen megvalósíthatóságának vizsgálata volt, ezért az alkalmazás funkcionalitása nem teljes, de tovább fejlesztve képes lehet egy valódi autó szerviz munkafolyamatainak segítésre.

5.1. JSP

A JSP (Java Server Pages) HTML oldalak dinamikus, szerveroldali készítését lehetővé tevő technológia. A JSP oldalak működése a servletekéhez hasonló. Mégis van egy fontos különbség, mégpedig az, hogy amíg a servletek esetén Java kódba ágyazunk HTML kódot, addig a JSP oldalak HTML kódjába helyezhetünk el Java kódokat. A következő elemek szerepelhetnek egy JSP oldal forráskódjában:

- statikus HTML kód
- szkriptek (pl.: JavaScript)
- direktívák
- megjegyzés
- akcióelemek

Megjegyzések <%-- --%> jelek között helyezhető el az oldalon. Ezeket az elemeket figyelmen kívül hagyja a JSP fordító. A direktívák a JSP konténernek szóló utasítások. Alakjuk: <%@ direktíva_név attr_1=”érték_1” attr_2=”érték_2” . . . %>. A direktívák nem módosítják az előállított oldal szövegét. Három fajtája létezik: page, taglib és include. A page direktívával az egész oldalra vonatkozó tulajdonságokat állíthatunk be. Az include direktíva

adott fájl fordítás előtti statikus beillesztésére szolgál. Az állomány beillesztése karakterről karakterre másolással történik. A taglib direktíva segítségével a JSP által felismert akcióelem készlet bővíthető. A JSP oldalak servletekké fordulnak le az első hívásukkor, ezért ekkor lassabb lesz a betöltés. A későbbi hívások lényegesen gyorsabban kapnak választ. Az akcióelemek segítségével átadhatjuk lapok közt a vezérlést vagy szerver oldali eszközöket használhatunk.

Alkalmazásom felületének megvalósításához JSP oldalakat vettem igénybe. A későbbiekben látni fogjuk miként szolgálnak kényelmes megoldásként a megjelenítés számára.

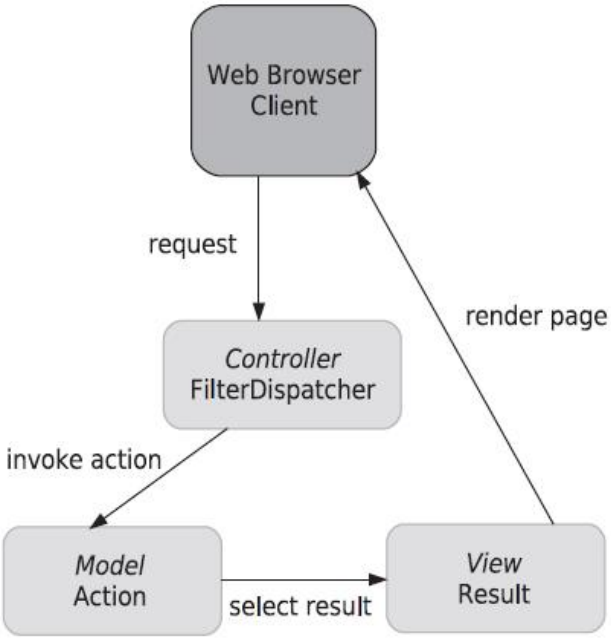
5.2. A Struts 2 keretrendszer

Egy mai nagyobb üzleti web alkalmazás megírása elképzelhetetlen lenne valamilyen keretrendszer használata nélkül. Egy web alkalmazás fejlesztő keretrendszer egy strukturált szoftver, ami beépített architekturális megoldásokat és a gyakori feladatok automatizálását szolgáltatja, amelyeket könnyen örökölhet egy alkalmazás, amit a keretrendszerben implementálnak. Olyan feladatokra kaphatunk beépített megoldásokat, mint a HTTP karakterláncok átalakítása Java adat típusokra, az üzleti réteg és az adat réteg hívások szétválasztása a webes megjelenítési feladatoktól, nemzetköziesítés, és a felület megjelenítése. Egy jó keretrendszer elegáns megoldásokkal szolgál a programozónak a hétköznapi feladatok megkönnyítésére. Ezeknél a feladatoknál talán még fontosabbak a keretrendszer strukturális jellemzői. A keretrendszer használatával, ezekkel a jellemzőkkel az alkalmazásunk is rendelkezni fog, így a keretrendszer választásával az alkalmazásunk architektúráját is megválasztjuk.

5.2.1. A Struts 2 szerkezete

A Struts 2 tervezése és implementálása a Modell-Nézet-Vezérlő (Model-View-Contoller) tervezési mintát követve történt. A MNV minta három különálló részre osztja az alkalmazást, amelyek jól illeszkednek a web alkalmazásokra. Ezek a részek a modell, nézet és a vezérlő. A Struts 2-ben az *Action, Result* és a *FilterDispatcher* valósítják meg a MNV elemeit. Az 2.

ábrán látható, hogyan implementálja a Struts 2 keretrendszer az MNV mintát, és hogy hogyan kezeli a munkafolyamát (workflow) egy web alkalmazásnak.



2. ábra: Struts 2 MNV elemei: Action, Result, FilterDispatcher

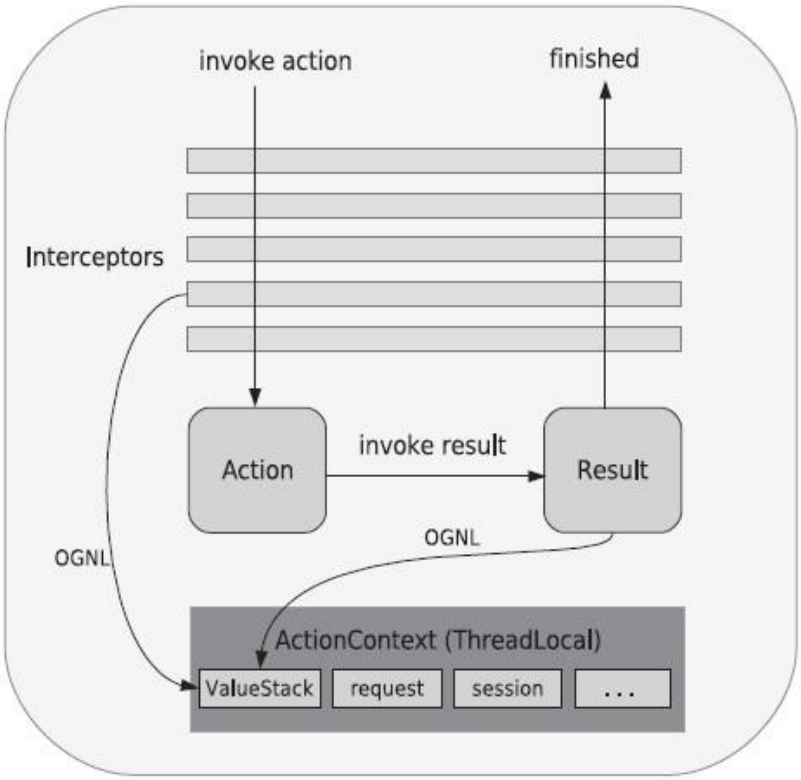
A vezérlő feladata a kérések eseményekre való leképezése. Egy web alkalmazásban a beérkező HTTP kérések felfoghatók úgy, mint a parancsok amelyeket a felhasználó kiad az alkalmazás számára. Az egyik alapvető feladat egy web alkalmazás számára a beérkező kéréseknek a megfelelő eseményekhez való rendelése. A vezérlő szerepét a FilterDispatcher tölti be a Struts 2 keretrendszerben. Ez a fontos objektum egy servlet szűrő (servlet filter), amely megvizsgál minden beérkező kérést, hogy eldöntse melyik Struts 2 action-nek kellene kezelnie a kérést. A keretrendszer minden munkát elvégez, nekünk csak arról kell informálnunk, hogy melyik kérés URL-jéhez melyik Action-t rendelje. Ezt megtehetjük XML alapú konfigurációs állományokkal vagy Java annotációkal.

A modell a legködösebb a Modell-Nézet-Vezérlő hármasból. A modell egy fekete doboz, amely tartalmazza az alkalmazás lényegét, mind az üzleti logikát mind az adat réteget. Más szemszögből a modell az alkalmazás belső állapota. A Struts 2 Action-je valósítja meg a

modellt. Egyik szerepe az, hogy becsomagolja a hívásokat egyszerű feladatokká az üzleti logika számára. Ezenkívül az adatmozgatás színhelyéül is szolgál.

A vezérlő – miután fogadja a kérést – megvizsgálja, hogy melyik Action-nek kellene kezelnie a kérést. Ha megtalálta a megfelelőt, átadja a vezérlést a kérés feldolgozására az Action meghívásával. Amikor az Action befejezi munkáját, itt az ideje a felhasználó számára megjeleníteni az eredményt. Így az Action dolga végeztével, továbbítja a megfelelő eredményt a Struts 2 nézet összetevője, a Result felé. A nézet a megjelenítés eszköze az MNV mintában. Az 2. ábrára visszatekintve láthatjuk, hogy a Result visszaküld egy oldalt a web böngészőnek. Ez az oldal a felhasználói felület, amely az aktuális állapotát jeleníti meg az alkalmazásnak. Ezek az oldalak általában JSP oldalak vagy más megjelenítési rétegbeli technológiák. Az Action választja meg, mely Result jelenítse meg eredményét. A Result-ok száma tetszőleges sok lehet. Általánosan, ha az Action sikeresen végrehajtódik a „success”, hiba esetén az „error” eredménnyel tér vissza és a hozzátartozó Result jelenítődik meg.

5.2.2 A Struts 2 működése



3. ábra: A kérés feldolgozásának munkafolyamata

A Struts 2 az MNV elemeken kívül más összetevőkkel is rendelkezik. A keretrendszer egy tisztább implementációja az MNV mintának. Ez csak néhány egyéb architekturális összetevő segítségével válik valóra, amelyek részt vesznek minden kérés feldolgozásában. Legfontosabbak ezek közül az interceptorok, az OGNL és a ValueStack.

A 3. ábrán látható a kérések feldolgozásának munkafolyamata. Az ábra azt az állapotot szemlélteti, amikor a FilterDispatcher már elvégezte vezérlő feladatát, azaz kiválasztotta a megfelelő Action a kérés feldolgozására. Az ábra azt mutatja be, hogy mi történik valójában amikor a vezérlő meghívja az Action-t.

Az ábrán látható, hogy az Action előtt egy verem helyezkedik el, amelyben az interceptorok kapnak helyet. Az Actoin hívása keresztül halad ezen a vermen és a megfelelő interceptorok végrehatják feladatukat. Minden Action-höz van egy verem rendelve. Az interceptorok meghívódhatnak az Action előtt vagy után is, habár általában az Action végrehajtása után lépnek működésbe. Az interceptorok lehetővé teszik a mindennapi, átfogó feladatok definiálását tiszta, újrafelhasználható összetevőkként, amelyeket elkülönítve tarthatunk az Action-től. Az interceptorok segítségével, olyan feladatokat végezhetünk el, mint például a naplózás. Segítségével elkülöníthetjük az Action feladataitól, de az Action minden hívásakor végrehajtódik a naplózás. A programozónak lehetősége van saját interceptorok létrehozására is.

A ValueStack(érték verem) egy tároló, ami a kérések feldolgozásához tartozó összes adatot magába foglalja. Az adatok bekerülnek a ValueStack-be a feldolgozás előkészítése során, itt módosítja az Action végrehajtás közben, majd innen lesz kiolvasva, amikor a Result, megjeleníti az eredményt. Az OGNL egy erős kifejező nyelv, mellyel hivatkozhatjuk és módosíthatjuk a ValueStack értékeit. A ValueStack és az OGNL egy szálbiztos környezetben az ActionContext-ben tárolódnak. Itt tárolódnak a keretrendszer belsőleg használt dolgai, mint például maga a kérés. Az ActionContext segítségével a végrehajtás során a ValueStack egy szálon belül bármikor elérhető.

5.3. Az implementáció első lépései

Ahhoz, hogy a Struts 2 keretrendszer segítségével fejlesszek a projektemhez hozzá adtam a keretrendszert tartalmazó jar fájlokat. A keretrendszer letölthető a http://struts.apache.org/ internetes oldalról. Majd az alkalmazás web.xml állományát kiegészítettem az alábbi sorokkal.

/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" . . . >

. . .

<filter>
    <filter-name>struts2</filter-name>
    <filter-class>
      org.apache.struts2.dispatcher.FilterDispatcher
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<listener>
    <listener-class>
      org.apache.struts2.tiles.StrutsTilesListener
    </listener-class>
</listener>
<context-param>
    <param-name>tilesDefinitions</param-name>
    <param-value>/WEB-INF/tiles.xml</param-value>
</context-param>

. . .

</web-app>
```

Először a struts2 nevű szűrőt (filter) definiálom, mely a servlet kéréseket és válaszokat szűri. Korábban tárgyaltam, hogy a Struts 2 esetén a FilterDispatcher végzi ezt a feladatot. A `<filter-mapping>` tag segítségével lehet erőforrásokhoz rendelni. Ebben az esetben minden kérést és választ ez a szűrő fog kezelni.

A `<listener>` tag eseménykezelőt definiál, ami ebben az esetben a StrutsTilesListener, ami Struts Tiles típusú események hatására hívódik. Végül a Tiles definíciókat tartalmazó tiles.xml-t adom meg konfigurációs paraméterként.

5.3.1. Struts Tiles

Az alkalmazásom megírásához igénybe vettem a Struts Tiles nevű szerver oldali felhasználói interfész komponens rendszert. Web alkalmazásom megjelenítési rétegében, azért használom, hogy egységes kinézetet kölcsönözzek egész alkalmazásomnak. Segítségével egyszerűen megvalósítható egy alkalmazás számára:

- azonos elrendezés,
- azonos menü,
- azonos fejléc és lábléc.

További előnye a könnyű karbantarthatóság. Az elrendezést teljes mértékben szétválasztja a tartalomtól, így ha változik az elrendezés, akkor azt csak egy helyen kell módosítani. Sablonok (Template) segítségével, valósítja meg a szétválasztást. Egy sablon egy JSP oldal, ami meghatározza az oldal elrendezését, a tartalom megadása nélkül. A sablonok definiálják a képernyő elemeket, melyekre névvel hivatkozhatunk.

A megjelenítés építő elemei a Tile-ok melyek definícióit a tiles.xml fájlban kell megadni. Egy Tile definíció rendelkezik névvel és tartalmazza a sablont. Továbbá az egyes képernyő elemekhez rendelt tartalmat. Az alábbi példában egy általam létrehozott Tile látható.

A példában a mainPage nevű Tile definíció látható. Létrehoztam egy MyTemplate nevű sablont, ami egy jsp oldal. Ebben a sablonban a képernyőt négy részre bontottam, ezek a header, a pagetitle, a body és a menu nevet kapták. A Tile definícióban a put-attribute direktíva segítségével adtam meg az egyes részekhez tartozó oldalakat, amik ebben az esetben jsp oldalak. Ahogy azt később látni fogjuk egy Struts 2 Action eredményeként adhatunk meg egy-egy Tile-t.

/WEB-INF/tiles.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE tiles-definitions PUBLIC
      „-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
      "http://tiles.apache.org/dtds/tiles-config_2_0.dtd">
<tiles-definitions>

      <definition name="mainPage" template="/layouts/myTemplate.jsp">
            <put-attribute name="header" value="/tiles/login.jsp" />
            <put-attribute name="pagetitle" value="" />
            <put-attribute name="body" value="/WEB-INF/jsp/index.jsp" />
            <put-attribute name="menu" value="/tiles/menu.jsp" />
      </definition>

. . .
</tiles-definitions>
```

5.3.2. Action definiálása

Az alkalmazás funkcióit, feladatait egy-egy Action reprezentálja. Az üzleti logika folyamatait valósíthatjuk meg segítségükkel. Egy Action létrehozásához először a strurs.xml fájlban kell definiálni magát az Action-t. Egy Action definíciója tartalmazza a nevét, az Action-t megvalósító Java osztályt és az Action-höz tartozó Result elemeket. Egy Result rendelkezhet névvel és típussal. Ezek határozzák meg, hogy mely az Action által visszaadott eredményhez tartozik és milyen típusú a megjelenített Result. Az alábbi példában az alkalmazásom néhány Action-e látható.

Az index nevű Action az alkalmazás nyitóképernyőjének megjelenítésére szolgál. A com.nlthesys.action.Index Java osztály valósítja meg. Az osztály forráskódja alább található. Egy Action-t reprezentáló Java osztálynak ki kell terjesztenie az opensymphony.xwork2.ActionSupport osztályt. Az általa végrehajtott feladatokat az ActionSupport osztály execute() metódusának felüldefiniálásával implementálhatjuk.

/struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
<package name="carservice" extends="tiles-default">

    <action name="index" class="com.nlthesys.action.Index">
      <result type="tiles">mainPage</result>
    </action>

    <action name="savecar" class="com.nlthesys.action.RegNewCar">
      <result name="success" type="tiles">carDetails</result>
      <result name="input" type="tiles">newCar</result>
    </action>

    <action name="gotoregnewcar" class="com.nlthesys.action.GoToRegNewCar">
      <result type="tiles">newCar</result>
    </action>
. . .

</package>
. . .

</struts>
```

com.nlthesys.action.Index.java

```
package com.nlthesys.action;

import com.opensymphony.xwork2.ActionSupport;

public class Index extends ActionSupport {

   public Index() { }

   @Override
   public String execute() throws Exception {
       return SUCCESS;
   }

}
```

Az index Action működése nagyon egyszerű. Az Index osztály execute() metódusának törzse egyetlen utasításból áll. Nem tesz mást mint visszatérési értékül a SUCCESS nevű nevesített String konstans adja vissza, amelyet az ActionSupport osztálytól örököl. Ez a visszaadot String határozza meg, hogy mely Result jelenítődjön meg. Ebben az esetben a struts.xml-ben, egy Result lett definiálva, melynek nincs név megadva, így minden eredmény esetén ez jelenítődik meg. Ez egy Tiles típusú Result mainPage nevű Tile-t jeleníti meg. Tehát ez az Action nem csinál mást csak a nyitó képernyőt jeleníti meg.

5.4. Menü

Az alkalmazás felületének elrendezését, úgy alakítottam ki a Tiles segítségével, hogy a menü mindig látható legyen a képernyő bal oldalán. A menü egy JSP oldal, amin listába szervezve jelennek meg a menüpontok. Egy menüpont egy hivatkozást tartalmaz egy Action-re. Ezek az Action-ök a korábban bemutatott Tiles típusú, index nevű Action-höz hasonlóan csak egy képernyő elrendezés megjelenítését végzik, azaz a megfelelő funkciókat tartalmazó képernyőket teszik elérhetővé.

/tiles/menu.jsp

```
. . .
<ul>
   <li><A href="gotoregnewcar.action" id="menuitem-message_subjects">Autó
regisztrálása</A></li>
   . . .
</ul>
. . .
```

Itt jegyezném meg, hogy az alkalmazásom kinézetét az http://www.freecsstemplates.org/ oldalról egy ingyenesen letölthet CSS sablon segítségével alakítottam ki. Azért nem tárgyalom hosszabban, mivel nem a felhasználói felületek kinézetének tanulmányozására helyeztem a hangsúlyt munkám során. Ahogy azt korábban tárgyaltam a böngésző programok a szabványoktól eltérően valósíthatják meg az egyes technológiákat. Megfigyeltem, hogy a böngészőkön a menü megjelenése nem teljesen egyforma. A Google Chrome böngészőn való megjelenítés hibátlannak tűnik, míg a Mozilla Firefox-on a menü fejlécének elrendezése nem

tökéletes. Az Internet Explorer által megjelenített menü fejlécének nem csak az elrendezése hibás, hanem a fejléc elemei nem látszódnak teljesen. Ezek apró hibák, amelyek ellenére az alkalmazás hibátlanul használható. Funkcionalitásban egyik böngésző által megjelenített menü sem tér el.

5.5. Űrlap

Egy autó regisztrálása a rendszerben egy űrlap segítségével történik. A következőkben az űrlap létrehozását és a hozzátartozó Action implementációját mutatom be. A Struts 2 keretrendszer által nyújtott eszközök segítségével egyszerűen hozhatunk létre. mint azt az autó regisztrálására szolgáló űrlapot megvalósító regnewcar.jsp forráskódjában megfigyelhető.

regnewcar.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
 <head>
  <title>Register New Car</title>
 </head>

 <body>
  <hr>
  <s:form action="savecar">
   <s:textfield name="carnumber" label="Rendszám" size="7"
maxlength="7"/>
   <s:textfield name="manufacturer" label="Gyártó" maxlength="25"/>
   <s:textfield name="type" label="Típus" maxlength="25"/>
   <s:textfield name="producingyear" label="Gyártás dátuma" size="11"/>
   <s:textfield name="color" label="Szín"/>
   <s:textfield name="volume" label="Hengerűrtartalom" size="5"/>
   <s:textfield name="owner" label="Tulaj neve" maxlength="50"/>
   <s:label name="owneradrress" value="Tulaj Címe:" />
   <s:textfield name="city" label="Város" maxlength="30"/>
   <s:textfield name="zipcode" label="Irányítószám" size="4"
maxlength="4"/>
   <s:textfield name="address" label="Utca, házszám" maxlength="50"/>
   <s:submit value="Elküld"/>
  </s:form>
  <hr>
 </body>

</html>
```

A <%@ taglib prefix="s" uri="/struts-tags" %> direktíva deklarálja a Struts 2 tag könyvtárat és az „s” előtaghoz rendeli őket. A Struts 2 felhasználói felület elemeket ezzel az előtaggal jelöljük.

Az alábbi példa azt szemlélteti, hogy egy Struts 2 felület elemhez milyen HTML kód rendelődik.

Egy Struts 2 szövegmező és a hozzátartozó HTML kód:

```
<s:textfield name="username" label="Username"/>

<td class="tdLabel">
<label for="Register_username" class="label">Username:</label>
</td>
<td>
<input type="text" name="username" value="" id="Register_username"/>
</td>
```

Egy felület elem több mint az elem HTML kódja. A következőket is elvégzi számunkra:

- hozzárendeli a HTML űrlap elemeket Java oldali változókhoz
- a keretrendszer típus konverziója alá kerül
- a keretrendszer validálása alá kerül
- a keretrendszer nemzetköziesítése alá kerül

Az űrlap <s:form> és </s:form> között adható meg és az action jellemzőnek értéket adva tudjuk Struts 2 Action-höz rendelni. A regnewcar.jsp űrlapjához a regnew car Action tartozik.

A savecar Action definíciója:

```
<action name="savecar" class="com.nlthesys.action.RegisterNewCar">
    <result name="success" type="tiles">carDetails</result>
    <result name="input" type="tiles">newCar</result>
</action>
```

Látható, hogy az Action két - „success” és „input” nevű - eredménnyel is rendelkezik. A „success” eredmény jelzi, ha az űrlap kitöltése sikeres. Ha a validálás hibával tér vissza, akkor az „input” eredményhez tartozó Result jelenik meg. Siker esetén egy áttekintő képernyő jelenik meg, amelyen az éppen regisztrált autó adatai jelennek meg. Hiba esetén az űrlap marad a képernyőn.

Az adatok bevitelére szövegmezőket használtam, amelyek rendelkeznek névvel, címkével és mérettel, de ezeken kívül még nagyon sok tulajdonságát be lehet állítani egy mezőnek. A legfontosabb tulajdonsága a neve, mivel a keretrendszer összeköti az Action Java osztályában található azonos nevű változóval. Nem kell mást tenni, mint a megfelelő névvel és típussal deklarálni a változókat valamint minden változóhoz lekérdező és beállító metódust létrehozni, és a keretrendszer gondoskodik róla, hogy a változók rendelkezzenek a felhasználó által bevitt értékekkel.

5.5.1. Az Űrlap validálása

Miután a felhasználó kitöltötte és elfogadta az űrlapot a keretrendszer elvégzi a típuskonverziókat. Ha a típuskonverzió nem volt sikeres az alkalmazás hibaüzenettel jelzi. Ezt a keretrendszer automatikusan hajtja végre számunkra.

Miután a típuskonverziók és azok ellenőrzése megtörténtek a programozó által írt ellenőrzés fut le. Az ellenőrzéseket a validate() metódus törzsében kell implementálni. Itt van lehetőség a mezők értékeinek validálására. A mezők értékeit a lekérdező metódusaikkal érhetjük el és az addFieldError() metódussal rendelhetünk a mezőhöz hibát. Az alábbi példában a hengerűrtartalom ellenőrzése látható

Hengerűrtartalom validálása:

```
public class RegisterNewCar extends ActionSupport {
    private int volume;
    public int getVolume() {
      return volume;
    }
    public void setVolume(int volume) {
      this.volume = volume;
    }
    . . .
    @Override
    public void validate() {
      if (getVolume() == 0) {
        addFieldError("volume", "Hengerűrtartalom megadása kötelező");
      } else if (getVolume() < 0) {
        addFieldError("volume", "Hengerűrtartalom értéke nem lehet
            negatív");
      }
      . . .
    }
}
```

Ha a validálás közben hiba rendelődött valamely mezőhöz, akkor az Action az „input” eredményt adja vissza és a hiba üzenet megjelenik az űrlapon a megfelelő mező mellett. A felhasználó ezután javíthatja az értéket és újra elfogadva az űrlapot ismét végrehajtódnak az ellenőrzések. Ha a validálás hiba nélkül ér véget az Action végrehajtja művelteit.

5.5.2. Az Űrlap működése

Az Action által végzett feladatokat az Action-t megvalósító Java osztály execute() metódusa végzi. A programozó a metódus törzsében kell implementálnia a kívánt működést. A példában, az autó regisztrációja esetén, itt történik az adatbázis rétegbe való letárolása az űrlapon kitöltött adatokkal. A sikeres végrehajtás után az Action a „success” eredményt adja vissza és az eredményhez tartozó Result jelenítődik meg.

Az űrlap működésében nem találtam hibát és eltérést a különböző böngésző programokon. A megjelenítésben viszont volt eltérés. Amikor az űrlapon sok hibaüzenet jelent meg, és ez által az űrlap hossza is megnőtt. A Mozilla Firefox és a Google Chrome nem növelte meg az

űrlapot tartalmazó képernyőrész méretét, így az űrlap bizonyos sorai rácsúsztak a láblécre. A mezők továbbra is használhatók maradtak, de a hiba elég szembetűnő. Az Internet Explorer kellő mértékben megnövelte az űrlapnak szánt területet, így a hiba ezen a böngészőn nem jelentkezett.

5.5.3. Az Űrlap kitöltését segítő eszközök

A Struts 2 keretrendszer tartalmaz sok megoldást, melyekkel a felhasználó gyorsabban és a megfelelő formátumban töltse ki az űrlap mezőit.

A dátum megfelelő formátumának biztosítására és a dátum bevitelének megkönnyítésére a Struts 2 keretrendszer által nyújtott dátumsúgót vettem igénybe. Ahhoz, hogy használni lehessen a súgót a JSP oldal fej részében a következő sort kell elhelyezni:

```
<s:head theme="ajax" />
```

Ez a sor emeli be az oldal fej részébe, azokat a Javascript kódokat, amelyeket a súgó használni fog. A gyártási dátum megadásához a datetimepicker nevű Struts 2 elemet hívtam segítségül. Ez egy olyan mezőt hoz létre, amely rendelkezik dátum súgóval. A következő módon definiálható:

```
<s:datetimepicker name="producingyear" label="Gyártás dátuma)"
displayFormat="yyyy-MM-dd"/>
```

A displayFormat tulajdonság segítségével állíthatjuk be a dátum formátumát. A megjelenő mező teljesen bolond biztos, mivel csak a súgó segítségével lehet értéket adni a mezőnek és az nem is szerkeszthető, így a felhasználó nem tud rossz formátumú dátumot beírni.

A város kitöltésének segítésére az automatikus kiegészítést használtam. A Struts 2 keretrendszer az autocompleter eszközt biztosítja erre a feladatra. A használatával a mező kitöltése közben az addig beírt karakterláncnak megfelelő városokat felajánlja kiválasztásra. A kiválasztható értékeket az űrlapot megjelenítő Action execute() metódusában kell megadni. Egy List típusú változóba elhelyezve tudjuk átadni a városokat. Ezt megtehetjük akár

adatbázisból lekérdezett értékekkel is. Ezen kívül egy lekérdező metódust is definiálni kell a változó számára. Az Action osztályában definiált List típussal és city névvel deklarált, valamint város nevekkel feltöltött változó esetén a következő elemeket kell a JSP oldalon definiálni:

```
<s:label name="cityName" value="Válassza ki a város nevét:" />

<s:autocompleter theme="simple" list="city" name="CityName"/>
```

5.6. Az alkalmazás tovább fejlesztése

Mivel a munkám során a felhasználói felületek vizsgálata volt a fő szempont, az általam írt alkalmazás nem rendelkezik minden funkcióval, ami egy valós autó szerviz mindennapi munkájához kellhet. Az alkalmazást kiegészítve a hiányzó funkciókkal akár egy eladható szoftver válhat belőle. A keretrendszer segítségével egy olyan alkalmazást hoztam létre amely egy nagyobb rendszer alapja is lehet.

A Struts 2 keretrendszer jól kombinálható olyan technikákkal, mint az Ajax vagy a Dojo toolkit. Segítségükkel az alkalmazás interaktivitása és felhasználói élménye növelhető. Viszont a kliensfüggetlenség rovására is válhatnak azáltal, hogy a különböző böngészők nem egyformán valósítják meg a technikák implementációját.

6. Összefoglalás

Diplomamunkám keretein belül számba vettem a web alkalmazások tulajdonságait és fényt derítettem azokra a tulajdonságaikra, amelyek miatt népszerűek és elterjedtek. Az egyik ilyen tulajdonság a kliensfüggetlenség. Mivel a klienshez csak egy web oldal jut el, amelyet egy böngésző program jelenít meg, a web alkalmazások felületeinek kliensfüggetlenségi vizsgálata volt a diplomamunkám középpontjában.

Először azokat a felhasználói felület elemeket és tulajdonságaikat vettem sorra, amelyek jellemzőek egy tipikus web oldalra. Hangsúlyt fektettem a felhasználói felületek beviteli eszközeire és a felületek felhasználó barát működésére, amelyek biztosíthatják a könnyű kezelhetőséget.

A kliensfüggetlenség megvalósításának vizsgálatára használtam az általam megírt alkalmazást. A program elkészítéséhez a Struts 2 keretrendszert vettem igénybe. Ezelőtt még nem használtam a keretrendszert, de a diplomamunkám végére érve képes lettem abban fejleszteni. A Struts 2-n keresztül megismertem a MNV mintát is, mely az alkalmazás munkafolyamát is maghatározta. Tapasztalatom szerint a Struts 2 keretrendszer kényelmes és hatékony megoldásokat nyújt, amelyek segítségével gyorsan és elegánsan lehet fejleszteni.

Az alkalmazásom különböző böngésző programokon való tesztelése során fény derült bizonyos hibákra. Ezek a hibák jellemzően csak a megjelenítésben fordultak elő. A munkám során nem találkoztam működési hibákkal, de egy nagyobb rendszerben nagy valószínűséggel, ilyen típusú hibák is előfordulnak. Egy web alkalmazás fejlesztése és támogatása során elkerülhetetlen a böngészők helyes megjelenítésének és működésének problémáinak kezelése. Az összes böngésző programra tökéletesíteni egy web nagy alkalmazást gyakorlatilag lehetetlen. Ezért a teljes kliensfüggetlenség is csak megközelíthető.

A web alkalmazás fejlesztésében mindig szerepet fognak játszani a böngészők. Gyakori megoldás erre, hogy egy fejlesztő cég a népszerű böngészők közül választ - akár többet is -, amelyre az alkalmazását fejleszti és az alkalmazás használatát ezen a böngészőn ajánlja.

Az ilyen típusú tapasztalat hatékonyabb programozóvá tehet egy web alkalmazást fejlesztő programozót. Tapasztalataimat felhasználva, a jövőben web alkalmazások fejlesztésével szeretnék foglalkozni.

7. Ábrajegyzék

8. Irodalomjegyzék

- Nyékiné G. Judit – J2EE Útikalauz programozóknak, 2002
- Donald Brown, Chad Michael Davis, Scott Stanlick – Struts 2 In Action; 2008
- http://www.webgo.hu/web-alkalmazas-web-application-webapp
- Wikipédia: CSS, HTML, JavaScript, Ajax szócikkek