BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM

VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
MÉRNÖK INFORMATIKUS SZAK

Rendszerfejlesztés szakirány

Tudományos Diákköri Konferencia

Webes keretrendszerek vizsgálata és osztályozása

Készítette:

Demény Fruzsina Gyöngyi (HBS63L)

Konzulens(ek):

Dr. Goldschmidt Balázs

IRÁNYÍTÁSTECHNIKA ÉS INFORMATIKA TANSZÉK

2014

TARTALOMJEGYZÉK

1. BEVEZETÉS 1

2. A MEGVIZSGÁLT KERETRENDSZEREK 2

2.1. Google Web Toolkit (GWT) 2
2.2. Vaadin 3
2.3. Kendo UI 4
2.4. Spring MVC 6

3. OSZTÁLYOZÁS 7

3.1. Szerver és kliens oldal 7
3.1.1. Google Web Toolkit 7
3.1.2. Vaadin 8
3.1.3. Kendo UI 8
3.1.4. Spring MVC 9
3.1.5. Összegzés 9
3.2. Kommunikáció 9
3.2.1. Google Web Toolkit 9
3.2.2. Vaadin 11
3.2.3. Kendo UI 11
3.2.4. Spring MVC 11
3.2.5. Összegzés 13
3.3. Adatbáziskezelés 13
3.3.1. Google Web Toolkit 13
3.3.2. Vaadin 14
3.3.3. Kendo UI 15
3.3.4. Spring MVC 15
3.3.5. Összegzés 15
3.4. Felhasználói felület 15
3.4.1. Google Web Toolkit 15

3.4.2. Vaadin....16
3.4.3. Kendo UI....18
3.4.4. Spring MVC....19
3.4.5. Összegzés....19
3.5. Robusztusság....19
3.5.1. Google Web Toolkit....19
3.5.2. Vaadin....20
3.5.3. Kendo UI....20
3.5.4. Spring MVC....20
3.5.5. Összegzés....21
3.6. Tesztelhetőség....21
3.6.1. Google Web Toolkit....21
3.6.2. Vaadin....21
3.6.3. Kendo UI....21
3.6.4. Spring MVC....22
3.6.5. Összegzés....22
3.7. Dokumentáltság....22
3.7.1. Google Web Toolkit....22
3.7.2. Vaadin....22
3.7.3. Kendo UI....23
3.7.4. Spring MVC....23
3.7.5. Összegzés....23
3.8. Megtanulhatóság, fejlesztés bonyolultsága....23
3.8.1. Google Web Toolkit....24
3.8.2. Vaadin....24
3.8.3. Kendo UI....24
3.8.4. Spring MVC....25
3.8.5. Összegzés....25
4. A KERETRENDSZEREK ELŐNYEI ÉS HÁTRÁNYAI....27
4.1. Google Web Toolkit (GWT)....27
4.1.1. Előnyök....27

4.1.2. Hátrányok 27
4.2. Vaadin 28
4.2.1. Előnyök 28
4.2.2. Hátrányok 28
4.3. Kendo UI 29
4.3.1. Előnyök 29
4.3.2. Hátrányok 29
4.4. Spring MVC 30
4.4.1. Előnyök 30
4.4.2. Hátrányok 30
5. A VIZSGÁLATOK EREDMÉNYEI TÁBLÁZATBA SZEDVE 31
6. ÖSSZEFOGLALÁS ÉS TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK 33
7. IRODALOMJEGYZÉK ÉS HIVATKOZÁSOK 34

Absztrakt

Az informatikai alkalmazások világában rohamosan növekszik a webes alkalmazások száma, egyre több szoftvert alakítanak át vékony kliens alkalmazássá illetve új szoftverek készítésénél is sokszor azonnal felmerül az internetes elérés kérdése.

A webes technológia viszonylag újnak mondható, emiatt nincsenek olyan jól kiforrott nyelvek és technológiák, mint az asztali alkalmazások fejlesztésénél. A webtechnológiában jelenleg alkalmazott módszerek használatával sok esetben nehéz megfelelő minőségű szoftvert fejleszteni, amely minden webböngészőben ugyanúgy viselkedik. A vékonykliens alkalmazások fejlesztésének nehézségét adja a böngészők különböző megvalósításai a weben használt, jelenleg még nem teljesen szabványos nyelvekre. A felhasználói élmény növelésére és a fejlesztés megkönnyítésére egyre több webes keretrendszer jelenik meg, ezek egy nagyobb része Javascript és Java nyelven használható.

A rengeteg keretrendszer közül analizáltam négyet, a jelenleg legnépszerűbb webes keretrendszereket, ezek név szerint a Google Web Toolkit, a Vaadin, a Kendo UI és a Spring MVC. Különböző szempontok alapján összehasonlítást végeztem rajtuk, azzal a céllal, hogy megkönnyítsem a fejlesztés kezdeti fázisában felmerülő problémát, miszerint melyik keretrendszer segítségével készüljön egy új szoftver. A megfelelő keretrendszer kiválasztásának egyszerűsítése lecsökkenti a szoftverfejlesztés előkészítési fázisára fordított időt, így az egyébként kutatásra, különböző keretrendszerek megismerésére szánt idő fordítható tervezésre, fejlesztésre vagy tesztelésre, amely növeli a szoftver minőségét.

Jelenleg nincs olyan keretrendszer, amely minden típusú alkalmazásnál egyértelműen a legjobbnak mondható, ezért végeztem több szempont alapján az összehasonlítást, például kliens és szerver oldal kommunikációja, adatbáziskezelés, felhasználói felület kialakítása, testreszabhatóság, bonyolultság, mekkora alkalmazás készítésére tervezték. Ezek alapján már könnyen kiválasztható az adott projekthez legjobban megfelelő keretrendszer.

Abstract

In the world of computer applications the number of existing web application frameworks increases rapidly, more and more softwares are transformed into thin client applications, not to mention that when new softwares are created, the question of on-line reachability comes up soon.

Web technology is considered to be relatively new, therefore there aren't any well-developed languages and technologies like by desktop applications. Creating a software that has the appropriate quality and works well in all browsers, is usually a challenging task with the methods most commonly used in today's web technology. The difficulty of developing thin client applications is caused by the different implementations of the not entirely standard web programming languages. In order to increase user experience and to make developing easier, a lot of web application frameworks appeared, from which most of them are written in JavaScript and Java.

I analyzed the 4 most widely used frameworks among the many existing frameworks, these are Google Web Toolkit, Vaadin, Kendo UI and Spring MVC. I performed comparison on these frameworks based on different viewpoints with the goal to make the decision of choosing a framework in the early phase of development easier. It reduces the time spent on the preparing phase of software development, so this time can be spent on programming and testing to increase quality of the software instead of wasting it on researching web application frameworks.

At present there isn't any framework that can be declared to be the best choice by every type of applications, therefore I used different points of view by the comparison, for example client and server side communication, database management, user interface, designing, complexity, how big of an application it is designed for. Based on these viewpoints it is easy to choose the best suitable framework for the given project.

1. BEVEZETÉS

Az iparban jelenleg sok szoftverfejlesztő cég szembesül azzal a problémával, hogy egyre növekszik az igény a vékony kliens alkalmazások fejlesztése iránt, az új szoftvereket eleve webes klienssel tervezik illetve már létező vastag kliens alkalmazásokat vékony klienssé kell átalakítani. Ezekre a problémákra nyújtanak megoldásokat a több nyelven elérhető sokféle webes keretrendszer.

Jelen dolgozatnak az a célja, hogy megkönnyítse a fejlesztő cégek számára a webes keretrendszerek közötti választást.

Minden keretrendszerben készítettem egy alkalmazást, amivel mélyebben ki tudtam próbálni az egyes keretrendszereket, majd pedig tapasztalatok és kutatások alapján különböző szempontok alapján megvizsgáltam a keretrendszereket és az adott területen a vizsgálatok eredményeit összefoglaltam és kiemeltem a legjobban teljesítőt. Ezen osztályozás segítségével minden cég a számukra az adott projektben legfontosabb szempontok figyelembe vételével könnyen tudn választani a legnépszerűbb Java és JavaScript alapú webes keretrendszerek közül.

A rengeteg keretrendszer közül négyet analizáltam, a jelenleg legnépszerűbb webes keretrendszereket, ezek név szerint a Google Web Toolkit, a Vaadin, a Kendo UI és a Spring MVC. A dolgozat elején röviden bemutatom a keretrendszereket, a szempontok alapján részletesen megvizsgálom és értékelem őket, majd a könnyebb és gyorsabb összehasonlítás érdekében az eredményeket táblázatba gyűjtöttem.

2. A MEGVIZSGÁLT KERETRENDSZEREK

Először röviden bemutatom a keretrendszereket, mielőtt az egyes szempontok alapján osztályoznám őket. A vizsgálatokról és az eredményekről könnyebb úgy beszélni, ha legalább egy rövid bemutató erejéig ismerjük a keretrendszereket.

2.1. Google Web Toolkit (GWT)

A Google Web Toolkit (GWT) [7] egy fejlesztői eszközkészlet, mely lehetővé teszi webalkalmazások írását Java, Python, PHP és Go nyelveken. A GWT fordító optimalizált Javascript kódot készít az alkalmazásunk kliens oldali kódjából, melyet minden elterjedten használt böngésző képes értelmezni és megjeleníteni.

Én a Java nyelvű verzióját vizsgáltam meg. A GWT Java SDK sok GUI elemet tartalmaz a felhasználói felület elkészítéséhez, továbbá lehetőséget nyújt a kliens kód debuggolására. A Google Web Toolkit-et önmagában is lehet alkalmazni, de lehetőség van a Google App Engine-nel való összeintegrálására. A Google App Engine egy Google által nyújtott cloud szolgáltatás, amely teljes funkcionalitású, NoSQL adatbázissal ellátott szervert nyújt a webalkalmazásunkhoz. A GWT keretrendszert elterjedten használják manapság is, továbbá új a GWT alapjaira épülő keretrendszerek kifejlődését is elősegítette, mint például a Vaadin keretrendszer.

A GWT-ben a kliens és szerver oldalt is Java nyelven fejleszthetjük, de már kód szinjén is erősen elválik a kettő. A kliens oldalon a beépített GWT-s GUI objektumokon kívül Java-script-ben írt elemeket is használhatunk, lehetőség van ezek Java kódból való elérésére.

A GWT-ben írt alkalmazás szerver oldalán bármilyen adatbázist és Java-ban használható adatbáziskezelőt alkalmazhatunk, azonban amennyiben Google App Engine backend-et használunk, úgy kapunk egy NoSQL adatbázist, amelyhez egy a Google által nyújtott Low-level Datastore API-t, JDO-t vagy JPA-t használhatunk adatbáziskezelőként.

A GWT saját RPC (Remote Procedure Call – Távoli Eljárás Hívás) módszert alkalmaz a kliens és szerver oldali kommunikáció megvalósítására, ahol az összetett objektumok szerializálása is implementálva van, nincs szükség saját konverter írására. Természetesen lehetőség van szervletek alkalmazására is.

A GWT specialitása, amelyet a másik három keretrendszer nem nyújt: felhasználók azonosítása és hitelesítése. A GWT beépítve tartalmazza a Google felhasználói fiókokkal való bejelentkezés megvalósítását, így nincs szükség saját bejelentkeztetési folyamat implementálására. Természetesen a Google a fiókokról nem ad ki információt, összesen az e-mail címet tudjuk meg, de sok esetben ez is elegendő a felhasználók kezelésére.

A GWT MVP architektúrának a használatára is nyújt lehetőséget, lehet úgy nevezett Place-eket definiálni, amik segítségével az oldalon belüli irányítás, oldal váltás könnyen megvalósítható.

A fejlesztés megkönnyítésére Eclipse plugin is letölthető, illetve a Google készített tutorial-okat és API dokumentációt is.

Ez a keretrendszer már hosszabb ideje elérhető, ma is folyamatosan fejlesztik így egy új keretrendszer kiválasztásánál érdemes megnézni az általa nyújtott szolgáltatásokat.

2.2. Vaadin

A Vaadin [8] egy OpenSource Java alapú webes keretrendszer, a böngészők nagy részével kompatibilis. Célja a Rich Internet Application (RIA) készítésének megkönnyítése.

Szerver oldali architektúra jellegű, tehát a program logikájának nagy része a szerveren fut. A kliens oldalon az ajax technológiát alkalmazza a felhasználói élmény növelésére. A Vaadin a Google Web Toolkit-re épít, illetve ezzel lehet bővíteni a funkcióit, például az alap komponenskészlet kiegészíthető CSS segítségével testreszabott GWT widget-ekkel.

Java nyelvet alkalmazva készíthetjük el a webes alkalmazásainkat, ami könnyítést jelenthet, mivel nem kell a szokásos és néha kissé bonyolult JavaScript és HTML nyelveken programozni. A keretrendszer eseményvezérelt programozást és widget-ek alkalmazását is lehetővé teszi, így kicsit messzebb áll a hagyományos HTML / JavaScript webfejlesztéstől. A Vaadin a Google Web Toolkit használatával rendereli ki az eredményül kapott weboldalt. A Vaadin alkalmazások futtathatóak Java szervlet formájában bármilyen Java szerveren beleértve a Google App Engine-t is.

A Vaadin-t JAR file-ok kollekciójaként adták ki (közvetlen letöltés vagy Maven és Ivy integráció segítségével szerezhető meg), így bármilyen standard Java eszközökkel készített Java webprojekthez hozzáadható. A Vaadin alkalmazások fejlesztésének megkönnyítése

céljából léteznek Vaadin plugin-ok is Eclipse és NetBeans fejlesztőkörnyezetekhez, továbbá közvetlen támogatása van Maven-en keresztül.

Sok beépített komponense van, de lehetőség van saját design-olásra is CSS illetve SCSS használatával, témákat definiálhatunk ezzel egyedivé téve a widget-jeinket és így az alkalmazásunkat is.

A Vaadinban írt alkalmazások érdekessége, hogy kódolása hasonlít egy sima Java desktop alkamazáséra, mivel a fejlesztést elősegítendő elrejti a kliens – szerver architektúrát, ezzel a kettő közötti kommunikáció problémáját is.

Adatbáziskezelési módszerek beépítve nem találhatóak benne, de addon-ként letölthető a `JPAContainer`, ami segítségével az adatbázis műveletek SQL írása nélkül is elvégezhetőek illetve a grafikus elemekbe az adatok könnyen beszivárogtathatóak.

A fejlesztést elősegíti, hogy teljes dokumentáció található az interneten az API-hoz a docs.oracle.com-hoz hasonló formában. A weboldalukról letölthető egy teljes könyv [4], amely bemutatja a Vaadin keretrendszer működését, a widget-ekről leírást és használatukról példakódot tartalmaz.

A Vaadin jelenleg a 7-es verziónál tart, sok változtatást végeztek rajta, így módosítás nélkül a Vaadin 6 programok nagy része nem konvertálható Vaadin 7-re, viszont útmutatót adnak a konvertáláshoz, hogy pontosan miket kell máshogyan írni, így könnyen átírható egy projekt.

2.3. Kendo UI

A Kendo UI [9] web- és mobilfejlesztő környezetet, adatvizualizációs eszközöket és szerver wrapper-eket biztosít. Egy szimpla programozói interfész, biztosítja a szükséges eszközöket HTML5 és JavaScript alkalmazások fejlesztéséhez a Kendo UI Web segítségével.

Sokféle eszközön alkalmazható, vannak speciálisan mobilra készült komponensek, különböző beviteli opciókkal, minden GUI elem teljes mértékben alapból támogatja az érintőképernyős kezelést, még a drag-and-drop lehetőségét is biztosítja érintőképernyőn keresztül ezzel elkerülve a kódduplikációt, ami a különböző beviteli módok kezelésénél szokott előkerülni. Én a Kendo UI Web részével foglalkoztam, a mobilos rész programozása hasonló, csupán az elemek és a konfigurációs lehetőségei mások.

A megvizsgált keretrendszerek

Készítése során a fejlesztők célja kezdettől fogva a JavaScript alkalmazás teljesítmény maximálizálása volt, a beépített lightweight sablon könyvtártól, az optimalizált animációkig, amik CSS3 hardver gyorsítást (ahol lehetséges) és fejlett UI virtualizációt használnak. Jelenleg még nem minden böngésző támogatja a HTML5-öt, ami sokszor problémát okoz, ezért a Kendo UI különböző technikákat alkalmaz, hogy a régebbi böngészőkben engedélyezze a HTML5 funkcióit, máshol meg a programot "amortizálja" le, hogy fusson ezeken a böngészőkön is. Ennek segítségével a legtöbb böngészővel kompatibilis, például: Internet Explorer 7+, Firefox ESR, Chrome, Safari 5+, és Opera 11+.

A teljes API JavaScript nyelven íródott, a kliens oldal fejlesztésének megkönnyítésére találták ki. A keveredések elkerülése végett minden osztály nevének kezdetén "k-" áll, így nem okozhat problémát a névduplikáció.

Többféle grafikus elemet, úgynevezett Grid-eket nyújt, amelyeknek sokféle konfigurációs lehetőségei vannak, amik segítségével egyedi funkciókat is megvalósíthatunk. Ezen kívül segítséget biztosít a szerver oldalról érkezett adatok Grid-ekbe történő betöltéséhez is.

A HTML kód használatában is hozott újítást, úgynevezett Template-eket definiálhatunk, amelyeket JavaScript segítségével tölthetünk be a HTML kódba, így egy nagyobb GUI esetén jobban szétválasztható a HTML kód.

A Kendo UI nem csak JavaScript nyelven érhető el, folyamatosan fejlesztik a TypeScript nyelven íródott verzióját is. A TypeScript a Microsoft fejlesztése, JavaScript nyelvre fordul le, szintaktikája hasonló a JavaScript-éhez, de lehetőség van modulok, osztályok írására, az objektum-orientáltság fő jellemzői megtalálhatóak benne. A TypeScript viszonylag új nyelv, rendszeresen változik, így a Kendo UI fejlesztői nem tudják azonnal követni a változásokat és egy újabb verziót kiadni, így néha szükség van a Kendo UI forráskódjának szintaktikai javítására, például TypeScript korábbi verziójában nem volt kötelező típust megadni tömbhöz, egy későbbiben már kötelező volt legalább egy "any" típust adni a változónak (az "any" típus jelzi, hogy bármilyen típust felvehet később, majd értékadáskor derül ki a típusa).

A Kendo UI nem teljes mértékben ingyenes, ha minden részét használni akarjuk, akkor fizetni kell érte, én a webes részét használtam (Kendo UI web), amely ingyenesen elérhető.

2.4. Spring MVC

A Spring [10] keretrendszer-család rengeteg projektet tartalmaz, a programozás sokféle feladatára készítettek keretrendszert, például Spring MVC, Spring Data, Spring Security, Spring Mobile stb.

A Spring egy open source Java nyelven íródott keretrendszer. Az alapvető funkciók bármilyen Java alkalmazás esetén használhatóak, de vannak olyan kiegészítések amelyek webalkalmazásokhoz készültek Java EE platform-ra.

A Spring MVC a Spring keretrendszer része, mely kérés-alapú és a Model-View-Controller architektúrát alkalmazza webalkalmazások készítéséhez. Interfészeket definiál mindenre, amelyeket egy kérés-alapú modern webes alkalmazásnak kezelnie kell. A megvalósításoknál cél volt az egyszerűség, így saját interfészeket könnyebben lehet definiálni a létezők átírásával. A keretrendszerben nagy az absztrakció aránya, mivel a cél nem az volt, hogy kivegyék az irányítást az ember kezéből, hanem hogy megadják az alapokat saját megfelelő webes alkalmazás írásához, így sokféle egyedi modelleket, kontrollereket definiálhatunk.

Természetesen ez a szabadság nehezíti a megtanulhatóságot, például a projekt megfelelő beállításai, az XML-ben történő konfigurációk bonyolultabbak, mint a többi keretrendszernél.

A Spring MVC a szerver oldal és a kliens oldal elválasztására, kommunikációjára, struktúráltságára koncentrál, így a kliens oldalon található felhasználói felület programozására kevés eszközt nyújt, érdemes egy JavaScript keretrendszerrel összeintegrálni a szebb felület kialakítás érdekében. Én például a Kendo UI keretrendszerrel integráltam össze.

A keretrendszer létrehozásánál figyelembe vették a tesztelés fontosságát, így a Spring Eclipse plugin segítségével készített alkalmazás már létrehozáskor tartalmazza a megfelelő könyvtárakat a JUnit teszteléshez, továbbá minden tutorial is kiemeli a tesztelés fontosságát és példákat nyújt a megfelelő tesztesetek írásához.

3. OSZTÁLYOZÁS

Ebben a fejezetben a korábban már röviden bemutatott 4 keretrendszert vizsgálom meg részletesen, az általam kidolgozott szempontrendszer szerint. Az elemzések végén az adott szempont alapján a legjobban teljesítő keretrendszert ki is emelem.

3.1. Szerver és kliens oldal

Az egyik legfontosabb kérdés egy webalkalmazásnál, hogy mit helyezzünk szerver (backend) és mit kliens (frontend) oldalra, a kód szintjén mennyire válnak el illetve az, hogy a két oldal hogyan tud kommunikálni. Először a kód szintű elválasztást vizsgálom meg, majd pedig a következő pontban azt, hogy mennyire jól használható kommunikációs lehetőségeket biztosítanak.

3.1.1. Google Web Toolkit

A szerver és a kliens oldal élesen elválik ebben a keretrendszerben, és ez már a package szerkezeten is látszik, jelölni kell, hogy szerver, kliens vagy mindkét oldalon használt osztályokat tartalmaz-e az adott package. Kötelezően kell legyen egy `.server` és egy `.client` package és opcionálisan használható a `.shared` package. Az API szintén követi ezt a szerkezetet, a `.server` package-ben található osztályokat csak szerver oldalon, a `.client` package-ben található osztályokat pedig csak kliens oldalon használhatjuk. A `.shared` package célja, hogy lehetőséget nyújtson egy osztály kliens és szerver oldali használatára is.

Ezt a struktúrát követni kell, és nem lehet keverni a kliens és a szerver oldal kódját, amennyiben mégis megtesszük, nem fog lefordulni az alkalmazásunk.

A `.server` package-ben található osztályok fognak futni a választott Java szerverünkön, a kliens oldali java kód pedig optimalizált JavaScript kódra fordul le.

A GWT láthatóan nagy figyelmet fordít arra, hogy kód szintjén is tisztában legyen a fejlesztő azzal, hogy bár mindent Java-ban fejleszt, nem egy helyen fognak futni, így fontos megtartani az elválasztást, nem ugyanúgy kell kezelni, ahogyan egy desktop alkalmazást.

GWT-ben implementálva van az MVP architektúra is, ennek használata azonban nincs kikényszerítve, így ez nagyon hasznos funkció, ajánlott is alkalmazni, de alapból egy

GWT-ben írt alkalmazásról nem feltételezhetjük, hogy biztosan alkalmazza az MVP architektúrát.

3.1.2. Vaadin

A Vaadin keretrendszer saját maga dönt arról, hogy mit kell a szerver és mit a kliens oldalra helyezni, így különösebben nincsen beleszólásunk, és látszólag nincs is elválasztás a két oldal között, legalábbis az alkalmazás kódjának szintjén nem látható.

Először nagyon meglepő úgy írni egy kliens-szerver architektúrájú programot, hogy kódban nincsen szétválasztva a kettő. Természetesen azért sejthető, hogy mi lesz kliens és szerver oldalra helyezve, de programkód szintjén egy desktop alkalmazás kódjához hasonló kód készül.

Alapvetően szerver-oldali architektúra jellegűre alakították ki a keretrendszert, így az alkalmazás kódjának nagy része a szerveren fut, csak a szükséges elemeket helyezi kliens oldalra, ajax technológiát és Google Web Toolkit-et alkalmaz kliens oldalon.

Alapvetően a keretrendszer kicsit megköti a fejlesztő kezét, mivel a háttérben megoldja a kód szétválasztását és a megfelelő osztályok JavaScript-re fordítását, ez egy tapasztalt Java EE és web fejlesztőnél zavaró lehet, viszont annak, aki nem jártas a webes technológiában, nagy segítséget nyújt.

3.1.3. Kendo UI

A Kendo UI keretrendszer célja a kliens oldali felhasználói felület minél hatékonyabb programozása, így a keretrendszer csak a kliens oldal fejlesztésére szolgál, tehát nem kell sokat foglalkozni az elválasztás kérdésével, egy más nyelven írt szerver oldallal fogjuk amúgy is megtámogatni, így az elválasztás triviális.

Bármilyen nyelven írt szervert adhatunk hozzá, segítség főleg Java-ban és ASP.NET-ben való fejlesztéshez van. Az újabb verziókhoz már készültek szerver wrapperek, de ezek is a kliens oldali GUI programozását könnyítik meg.

Programozása hasonlít a szokásos JavaScript / HTML programozáshoz, csupán be kell illeszteni a megfelelő JavaScript és CSS fájlokat a fő HTML vagy JSP fájlunkba.

Láthatóan az éles elválasztás adódik abból, hogy a keretrendszert JavaScript-ben a kliens oldal programozására fejlesztették ki.

3.1.4. Spring MVC

A Spring MVC a Model-View-Controller szoftverarchitektúra mintát valósítja meg, így nagyon erősen elválnak az egyes funkciók, ami alapvetően fontos cél egy webes alkalmazás fejlesztése során. A szerver oldalon találhatóak a Kontroller-k, amelyeket Java nyelven írhatunk meg, ezek állítják össze a Modellt, amelyet átadnak majd a kliens oldalnak (View). A kliens oldal HTML és JavaScript nyelven íródik, így itt is már a szerver és kliens oldal különböző nyelvű programozásából adódóan jól látszódik az elválasztás, továbbá ebben a keretrendszerben az MVC architektúra miatt a szerver oldali logika és az adatok is erősen elválnak.

3.1.5. Összegzés

A szerver és kliens oldal elválasztásának szempontjából a legegyszerűbben a Vaadin kezelhető, de ezzel együtt megköti a fejlesztő kezét. A Kendo UI csak kliens oldalra használatos, így a szokásos elválasztási módszer alkalmazandó. A Google Web Toolkit a már package szinten megjelenő elválasztás miatt előnyös, és lehetőség van MVP használatára, de az MVC architektúra szigorú használata miatt a **Spring MVC** a legelőnyösebb ebből a szempontból, hiszen ekkor biztosan érvényesül az MVC-ben definiált elválasztás.

3.2. Kommunikáció

A kliens és a szerver oldal különálló elemként való viselkedése miatt előáll a kommunikáció kérdése a két oldal között, az egyes keretrendszerek erre nyújtott megoldásait vizsgálom meg a következőkben.

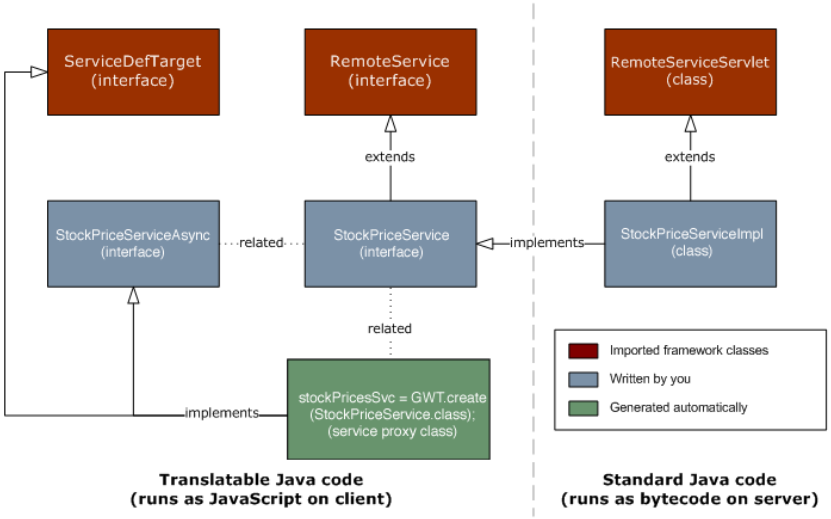
3.2.1. Google Web Toolkit

A GWT távoli eljárás hívásokkal (Remote Procedure Call - RPC) nyújt lehetőséget arra, hogy a futó webes alkalmazás a szerver és a kliens oldal között Java objektumokat küldjön HTTP-n keresztül. A kliens kódban egy automatikusan generált proxy osztályt használunk, hogy hívásokat generáljunk. A továbbítandó Java objektumok szerializálásáról a GWT gondoskodik, hátránya viszont, hogy az RPC hívások felépítése miatt hamar nagyon sok fájlból fog állni az alkalmazásunk.

Osztályozás

A kliens oldalon két interfészt kell elhelyezni, az egyik a szinkron, a másik az aszinkron változat. Szigorú szabály, hogy a függvények ugyanazok legyenek mindkettőben, kivéve, hogy az aszinkron változatban minden függvényhez még egy függvényparamétert hozzá kell adni, ami a callback-et adja meg, amit a hívás végeztével a szerver visszahív. Továbbá nem számít, hogy mi a neve a szinkron interfésznek, azonban az aszinkron névadás ehhez van kötve: ha a szinkron interfész neve `TestService`, akkor az aszinkron változaté `TestServiceAsync`.

A szerverben kell implementálni a szinkron interfészt. Megkötés, hogy az implementáló osztály nevét szintén a szinkron interface alapján kell megválasztani: ha a szinkron neve `TestService`, akkor az implementáló osztályé `TestServiceImpl`. Az implementációnak továbbá örökölnie kell a `RemoteServiceServlet` osztályból.



1. ábra Az RPC hívások felépítése GWT-ben

RPC hívások implementálása során az 1. ábrán kék színnel jelzett osztályokat és interfészeket kell nekünk megírni, a pirossal jelzettek alapból megtalálhatóak az API-ban. A zöld rész pedig azt mutatja, hogy az RPC hívás használatatakor mit generál le a GWT.

A GWT használata esetén a JavaEE-ben használatos szervleteket is alkalmazhatjuk, ilyen szempontból nincsen megkötve a programozó keze, a GWT nyújt egy jól kidolgozott lehetőséget a kommunikációra, de nem korlátozza le a lehetőségeinket annak a használatára.

3.2.2. Vaadin

A Vaadin keretrendszer elrejti a távoli eljárás hívásokat (RPC - Remote Procedure Calls) és a protokollokat. Ez megkönnyíti a fejlesztést, a fejlesztőnek a kód írásához nem szükséges alapos protokoll és hálózat ismeret, bár nyilván nem hátrány. Ez a fajta elrejtés korlátot is szabhat, például problémás lehet egy biztonságkritikus rendszernél, hiszen a kommunikáció esetén bíznunk kell a Vaadin RPC megvalósításának biztonságában, mivel nem mi írjuk meg a kommunikáció megvalósítását. Az RPC hívások lezajlása egyedül akkor látszik, ha valamilyen programozói hiba miatt (pl. túlindexelés) az RPC hívás meghiúsul, ekkor `RPCInvocationException`-t dob a program.

Csak kliens oldalon alkalmazza a Google Web Toolkit-et, így biztonsági problémák merülhetnek fel, ezért a Vaadin szerver-oldali adat-validációt is alkalmaz minden egyes műveletnél, így ha a kliens oldali adatot módosították, a szerver ezt észleli és nem engedi tovább a műveletet [4].

Egy tapasztalt web és JavaEE fejlesztőnek zavaró lehet a kommunikáció elrejtése, azonban egy kezdő fejlesztőnek nagy segítséget jelenthet.

3.2.3. Kendo UI

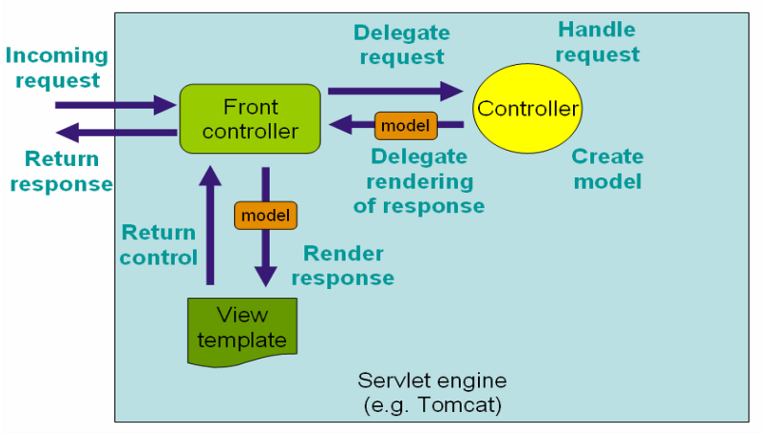
A Kendo további osztályokat / metódusokat nem biztosít a kommunikációra, a sima JavaScript és Java közötti POST és GET hívásokat kell alkalmazni, például ajax hívások és szervletek használatával.

Problémás lehet, hogy milyen formátumban küldjük az információt és azt hogyan alakítsuk át a számunkra szükséges nyelvi elemekre, erre sincsen a Kendo UI-nak saját megoldása, azonban egyéb library-k használatával könnyíthetünk a problémán.

3.2.4. Spring MVC

A Spring MVC esetén modell (`Model`) segítségével kommunikál a kontroller (`Controller`) a különböző nézetekkel (`View`), így adatok eljuttatása a kliens oldalra könnyen megoldható. A modell alapból implementálva van, ami tökéletesen alkalmas az adatok továbbítására, de lehetőség van a `Model` osztályból való örökléssel új, a számunkra

szükséges plusz funkciókat biztosító modellek létrehozására is, így nem szükséges az XML és JSON formátumok és egyéb objektumok közötti konvertálás.



2. ábra A Controller, Model és View kommunikációja Spring MVC-ben

A 2. ábrán látható az MVC egyes elemei között lefolyó kommunikáció. Látható, hogy a `Front controller` feldolgozza a kérést, átirányítja az általunk írt controller-nek, ami pedig összeállítja a modellt és visszaadja a `Front controller`-nek, ami értesíti a `View`-t a frissülésről majd pedig válaszol a kliensnek.

A kontrollernek van egy alap elérhetőségi útvonala, amelyet annotációban kap meg, ehhez szintén annotációkkal kaphatnak az egyes függvényei relatív útvonalakat, abban az esetben, ha annotációk használatával adjuk hozzá az adott osztályhoz a `Controller` tulajdonságot. Ekkor minden függvényhez megadható a relatív útvonalon kívül az is, hogy milyen típusú kéréseket szolgálnak ki (pl.: GET vagy POST). Az egyes kéréseket kezelő függvények visszatérési értéke leggyakrabban `ModelAndView` vagy `String` típusú. `ModelAndView` esetén a kliensnek szánt adatot és a használandó `View` forrásfájljának elérhetőségét adjuk vissza `ModelAndView` típusú objektumban. `String` visszatérési érték esetén a `String` tartalmazza a `View` forrásfájljának útvonalát, adatot pedig a paraméterként kapott `Model` típusú objektumba helyezéssel adhatunk át a `View`-nak.

Másik megoldás a `Controller` interfész megvalósítása, ekkor annotációban csak a `Controller` útvonalát kell megadni, ekkor két függvényt kell megvalósítani, az egyik a GET a másik a POST kéréseket kezeli, hasonlóan a szerveletekhez.

Többeknek az annotációs megoldás jobban megfelelhet, mivel ekkor akármennyi függvénye lehet egy kontrollernek, amelyek a megadott url-lel elérhetőek kliens oldalról, és

a jobb átláthatóság miatt több függvényt is lehet definiálni a kliens oldali kérések kezelésére.

A HTML / JavaScript nyelven írt `View`-kban a kontroller által a modellbe rakott adatokat HTML-ben egyszerűen a `${<változónév>}` szintaktikával lehet elérni. Adatok átvétele a szerver oldaltól ezzel a megoldással sokkal könnyebben megoldható, mint a szokásos szervletekkel.

3.2.5. Összegzés

Könnyedség szempontjából a Vaadin keretrendszer a legkényelmesebb, de ez korlátozza a fejlesztőt, mert nincs igazán az irányítása alatt ennek a megvalósítása, így a kommunikáció megvalósításának szempontjából a **Spring MVC** és a **Google Web Toolkit** van a legjobban kidolgozva, ezeknél a keretrendszereknél van könnyítés a kommunikáció terén, de lehetőség van saját kommunikációs megvalósítás implementálására is.

3.3. Adatbáziskezelés

Manapság már szinte nem készül olyan alkalmazás, amihez nincs adatbázis, hiszen a perzisztens tárolás elengedhetetlen funkció, ezért fontos megvizsgálni a keretrendszerek adatbáziskezeléshez nyújtott szolgáltatását.

3.3.1. Google Web Toolkit

A *Google Web Toolkit*-hez az App Engine keretein belül készült adatbázis és ahhoz használható API, de beintegrálhatunk saját adatbázisokat és azokhoz használt adatbáziskezelő módszereket is, mint a JPA vagy a JDO.

Az App Engine adattára a High Replication Datastore (HRD) egy NoSQL adatbázis, ehhez készült a Low-level Datastore API, amely beépített függvényeket tartalmaz az adatbáziskezeléshez, de használhatjuk a JDO-t vagy JPA-t is az adatok elérésére.

Az adatokat objektumok, másnéven entitások formájában menti el. Minden entitásnak (`entity`) van egy kulcsa (`key`), ami egyértelműen azonosítja őt. Az két entitás között szülő-gyermek (`parent-child`) viszony definiálására is lehetőség van, így hierarchikus struktúra alakítható ki. A szülő nélküli entitásokat gyökér (`root`) entitásnak nevezzük, az

entitások entitáscsoportokat alkothatnak (`entity group`). Az egy entitáscsoporton végzett lekérdezések mindig friss és helyes adatokkal térnek vissza, míg a több entitáscsoporton végzettek hibás eredményeket adhatnak vissza. Adatbázis kapcsolat nyitását és lezárását külön nem kell elvégeznünk, ezt az App Engine magától lekezeli.

Amennyiben nem használjuk az App Engine-t, hanem saját Java-s szervert teszünk a GWT mellé, akkor a szokásos Java-s JPA és Hibernate implementációk állnak rendelkezésre az adatbázis kezelésére.

3.3.2. Vaadin

A Vaadin nem rendelkezik beépített adatbáziskezelő API-val, azonban addon-ként letölthető a `JPAContainer`, aminek segítségével sokféle adatbázis kezelését beleépíthetjük a programunkba. Leginkább az EclipseLink-et támogatja, azonban bármilyen JPA vagy Hibernate implementációt használhatunk, ami többféle adatbázisra is épülhet pl. H2 vagy MySQL. A `JPAContainer` használatával nem szükséges a szokásos JPA implementációk kissé bonyolult szintaktikáját és SQL kódokat használnunk. A `JPAContainer` tartalmaz függvényeket ezek megvalósítására, cache-ként tárolja az adatbázis megfelelő táblájának az adatait a gyors elérés miatt és könnyű lehetőséget biztosít a widget-ek adatokkal való feltöltésére. Ha mégis szeretnénk használni a szokásos JPA megvalósításokat, akkor erre is lehetőség van, sőt a `JPAContainer`-rel együtt lehet használni, de ekkor figyelni kell a `JPAContainer` és az adatbázis szinkronban tartására. A `JPAContainer` implementációjában még vannak hiányosságok, jelenleg még nagy szükség lehet a szokásos `EntityManager` használatára, de még így is nagyon megkönnyíti az adatok kezelését.

A `JPAContainer` főleg azért hasznos, mert hamar el lehet érni az adatokat, és egyszerűen elvégezhetők segítségével bizonyos műveletek, például egy táblázat adatokkal való felöltése. Amennyiben a problémákat (például a hiányos implementáció) kijavítják, akkor az adatbáziskezelést és az adatok widgetek-be töltését is igencsak megkönnyítené.

3.3.3. Kendo UI

Csak kliens oldal programozásának megkönnyítésére tervezték, így nincs beépített adatbáziskezelője, hiszen az a szerver oldalon történik meg. A szervertől kapott információkat `DataSource` objektum segítségével feldolgozhatjuk és tárolhatjuk. Többféle formátumú információ feldolgozására is képes, pl. JSON vagy XML. A Grid-ek (GUI elemek) adatokkal való feltöltését is ezen keresztül tudjuk könnyen elvégezni.

3.3.4. Spring MVC

A Spring MVC beépítve nem tartalmaz adatábáziskezelő API-t, de JPA integrációt biztosít és a Spring Roo segítségével néhány parancs kiadásával felépíti az adatbázist. Amennyiben nincs Spring Roo telepítve, vagy saját magunk akarjuk felépíteni az adatbázist, akkor bármilyen Java-ban írt adatbáziskezelő módszer közül választhatunk , de ennek beintegrálását magunknak kell megoldani. Ezen túl létezik a Spring Data keretrendszer is a Spring keretrendszer-család részeként, amely az adatok megfelelő kezelésével foglalkozik, azonban ez alapból nem található meg a Spring MVC keretrendszerben, így ennek használatához is szükség van az integráció megvalósítására.

3.3.5. Összegzés

Összességében adatbáziskezelés szempontjából a **Vaadin** és a **Google Web Toolkit** a legkidolgozottabb, mivel a többi keretrendszer a meglévő implementációk miatt ezzel a területtel nem foglalkozik.

3.4. Felhasználói felület

A felhasználók az alkalmazásból a felhasználói felülettel tudnak kommunikálni, amennyiben ez a része az alkalmazásnak nincs jól megoldva, akkor hiába van tökéletes backend mögötte, az alkalmazásunk sosem lesz sikeres, ezért fontos a grafikus felhasználói felület programozhatóságát is megvizsgálni.

3.4.1. Google Web Toolkit

Fontos, hogy bár Java nyelven programozhatunk a Swing használata nem megengedett, de helyette a GWT saját elemeket definiál. A GWT GUI elemeit hasonlóképpen kell

használni, ahogyan a Swing-ben már megszokhattuk, a függvénynevek természetesen több esetben nem teljesen ugyanazok, és a `Layout`-ok is kissé mások, de az API dokumentáció segítségével ezek kikereshetőek és amennyiben ismerjük a Swing-et, a megtanulása egyszerű.

Alapvetően a beépített GUI elemek kombinációival sokféle felhasználói igény kielégíthető, de többször felmerülnek olyan igények amihez szükség van új GUI elem definiálására. Ez megoldható leszármaztatással, de nehezen lehet különleges testreszabási igényeket kielégíteni, ezek megvalósítása sokszor részletes ismereteket igényel a GWT GUI elemeinek működéséről.

A szokásos CSS fájlok segítségével lehet testreszabni az egyes GUI elemeket, erre sajnos nincsen külön funkcionalitás a GWT-ben, annyi segítség található, hogy Java-ban megadhatjuk a stílusban használt nevét az egyes objektumoknak vagy rakhatunk rájuk class-okat, így CSS-ben könnyen hivatkozhatunk rájuk.

3.4.2. Vaadin

Java osztályokat és függvényeket használva készíthetjük el a grafikus felületet a kliens oldalon, ez hasonló a Swing és a GWT programozásához, azonban jobban ki van dolgozva, többféle elemet definiáltak.

A Vaadin saját widget-eket is nyújt, amellett hogy a GWT-nél használt GUI elemek ebben a keretrendszerben is használhatóak, hiszen a kliens oldal a GWT-re épül.

A felület elemeit `Panel`-ekbe, `Window`-kba és `Layout`-okba tehetjük. `Panel`-ek közül nagyon hasznos tud lenni a GWT-ből átvett `HorizontalSplitPanel`, amelybe két elemet lehet elhelyezni, és az ablakot “félbevágva” a keletkező két oldalon helyezi el őket.

Sokat használtam `Layout`-okat (`Vertical` és `Horizontal`), ami sokat javít az oldal kinézetén. Definiáltak egy úgynevezett `FormLayout` osztályt, amely nagyon hasznos ha például a `TextField`-eket és az elnevezéseiket szeretnénk megjeleníteni egymás alatt, ekkor egy vonal mentén összerendezi őket, így akármennyire is eltérnek a `TextField`-ek nevei, a `TextField`-ek akkor is egy vonalban kezdődnek és ehhez nekünk külön beállításokkal nem kell foglalkoznunk.

Osztályozás

A TextField-eket átkonfigurálták, több funkciót helyeztek el bennük. Adhatunk nekik nevet, nem szükséges a szokásos Label és TextField páros használata, továbbá szimbólumokat helyezhetünk el a TextField-ek neveihez, melyre ToolTip-et is elhelyezhetünk. Ez nagyon hasznos funkció, mivel sok weboldalon jelölik meg az egyes mezőket, amelyeket kötelező kitölteni, vagy amihez segítség található ToolTip formájában.

Ami újdonság a Swing-hez képest és nekem nagy könnyedséget jelentett a felhasználói felület összerakása során, hogy van egy FieldGroup nevű osztály, amelyhez TextField-eket lehet kapcsolni, melyeket azonosítóval különböztetünk meg. Így nem kell egyesével foglalkozni a TextField-ekkel, azokat elhelyezni, adatukat kiolvasni. Továbbá lehet DataSource-t állítani a FieldGroup-hoz, ami az azonosítók alapján elhelyezi az adatokat a megfelelő TextField-be, így például egy táblázat sorának adatai könnyen összekapcsolhatóak TextField-ekkel.

Létezik egy úgynevezett Visual Designer, amivel egyedi komponenseket lehet gyártani viszonylag egyszerűen. Célszerűbb azonban témákat alkalmazni egyedi felület készítéséhez. Többféle beépített téma található, de sajátot is létrehozhatunk. Ekkor a Vaadin beimportálja a szükséges alap fájlokat, hogy jól jelenjenek meg a widget-ek, tehát csak a saját plusz szabályainkat kell hozzáírnunk. Sokszor a lefordított program böngészőben megnézhető HTML fájljának megnézése szükséges, hogy megállapítsuk az egyes widget-ek és azok részeinek class-ját. Ezzel részletesen változtathatjuk a widget-eket, ami elég nagy szabadságot ad, azonban nehezebbé is teszi a megfelelő testreszabást.

A Vaadin CSS és a CSS továbbfejlesztését, az SCSS fájlokat alkalmazza a témákban. Egy témát a Vaadin 6 keretrendszerben a setTheme() függvénnyel lehetett a UI-ra illeszteni, azonban ez megváltozott a Vaadin 7-re, ahol már az @Theme annotáció segítségével illeszthetjük rá.

A Vaadin keretrendszerben nagy hangsúlyt fektettek az egyedi témák kidolgozására, amiknek használata az alkalmazásunkat a felhasználó számára különlegesebbé teheti.

3.4.3. Kendo UI

Többféle lehetőség van ugyanannak a Grid-nek (GUI elem) a megírására, sokféle beállítás található egy Grid-hez, így nagy szabadságot biztosít.

Érdekesség, hogy jelenleg nem található (és nem is tervezik az implementálását) egyszerű `TextBox / TextField` a Kendo UI keretrendszerben. Ahhoz hogy egyszerű szöveges beviteli mezőket készítsünk, az ennél sokkal többet tudó `AutoComplete` Grid-et kell alkalmazni letiltva az animációkat és egyéb plusz funkciókat. `Button`-t sem implementáltak, vannak `Command` Grid-ek, melyből `Button` készülhet, vagy pedig a szokásos `Button` típussal rendelkező HTML input elemhez kell `click` függvényt definiálni és ez megkapja a Kendo UI design-ját a belinkelt CSS fájlokból.

Az egyes GUI elemeket HTML-ben definiáljuk majd a JavaScriptben tudjuk inicializálni és megadni a megfelelő tulajdonságaikat. Sokszor csak egy Grid-et definiálunk, nincs specifikus neve, azonban a beállítások során derül ki, hogy milyen elem lesz belőle. Például ha megadunk oszlopokat és tulajdonságaikat, akkor táblázatként fog megjelenni az alkalmazásunkban.

A Kendo-ban lehetőség van sablonok (templates) definiálára. Ezek arra valók, hogy a HTML-ben definiált Grid-eket egységbe zárja. Ennek segítségével rendszerezettebb kódot írhatunk, akár több fájlba is szétszedhetjük a HTML kódunkat. Az egyes sablonokat tmpl.htm kiterjesztésű fájlokba helyezhetjük, majd JavaScript-ben meg kell írnunk egy sablon betöltőt. Ez a sablon betöltő kód fut le legelőször, és ezután minden sablon elérhető és megjeleníthető.

Lehetőség van a HTML / JavaScript-ben is stílus tulajdonságokat megadni, például méretet állítani, de hatékonyabban elvégezhető bármilyen Grid testreszabása a szokásos CSS stílusokon keresztül.

A Kendo UI-ban található GUI elemek támogatják a témák és stílusok alkalmazását CSS-en keresztül. Alapból 5 témát tartalmaz: Default, Metro, Black, Blue Opal és Silver. A Kendo UI témák segítenek normál HTML elemek (pl. `input`-ok, stb.) testreszabásában is. Ha a beépített témák nem elégségesek, akkor a ThemeBuilder eszköz segítségével gyorsan készíthetünk testreszabott témákat.

3.4.4. Spring MVC

A Spring MVC inkább a szerver oldal szerkezetével és az adatok továbbításával foglalkozik, így kliens oldalon nem hoz igazán újításokat. Ettől függetlenül megemlítem ebben a fejezetben, hogy én milyen megoldással készítettem el a kliens oldalt.

Külön kliens oldali elemeket nem igazán ad a keretrendszer, néhány plusz HTML Tag-et adtak hozzá a szokásos elemekhez, így elég nehézkes elérni azt, hogy az elképzelt felületet megfelelően leprogramozzuk. Legtöbbször hasznos integrálni egy másik HTML / JavaScript nyelven írt keretrendszert a Spring MVC-ben írt projektünkhöz. Én a Kendo UI keretrendszert alkalmaztam, ugyan a kódon változtatni kell, de a Kendo UI JSP Tag-ekkel egyszerűen beilleszthető a különböző Grid-ekbe a modelltől kapott változók.

Az egyes elemeket a szokásos CSS fájlokban megadott stílusokkal lehet testreszabni, nincsen külön definiált téma vagy stílus lehetőség beépítve a keretrendszerbe. Amennyiben a kliens oldal programozásához külön beintegráltunk egy másik keretrendszert, akkor természetesen az adott keretrendszer testreszabási lehetőségeit alkalmazhatjuk.

3.4.5. Összegzés

Ugyan a Javascript nyelv nehézségei kezdetben problémát okozhatnak, de a **Kendo UI** segítségével lehet véleményem szerint a legkönnyebben és legszebben megvalósítani azt a kezelőfelületet a webes alkalmazásunkhoz, amit elképzeltünk, továbbá testreszabhatóság szempontjából is bár hasonló módszereket használnak az egyes keretrendszerek, a Kendo UI-nál a legkönnyebb az elemekre egyedi kinézetet illeszteni.

3.5. Robusztusság

Nagy alkalmazás írása esetén fontos szempont, hogy az egyes funkciók mennyire választhatóak el, hogyan lehet különböző metódusokra, osztályokra és package-ekre vagy fájlokra bontani, illetve mennyire lehet megoldani a bővíthetőséget.

3.5.1. Google Web Toolkit

A kód szokásos Java osztályokra és package-ekre bontását alkalmazhatjuk, azonban egy nagy alkalmazás esetén az egyes RPC hívásokhoz szükséges több interfész és osztály

miatt gyorsan beleütközhetünk abba a problémába, hogy a túl sok fájl miatt az alkalmazás átláthatatlan lesz.

A GWT lehetőséget ad MVP architektúra használatára is, ami segíthet az alkalmazás forrás kódjának karbantarthatóságában, de a fájlok száma ekkor is nagy mértékben megnőhet.

3.5.2. Vaadin

Mindig az UI osztályból leszármazott osztály `init()` függvényéből építjük fel a GUI-t, mivel ez a függvény hívódik meg amikor az alkalmazás url-jét beírják a böngészőbe. Ha csak ebbe az osztályba írhatnánk GUI elemeket, akkor hamar átláthatatlan lenne az osztályunk, de megoldást jelent, hogy más osztályokban építjük fel a felhasználói felület egyes részeit, majd ezekből az osztályokból tartalmaz egy-egy példányt a UI leszármazott osztálya. Az alkalmazás többi része ugyanúgy struktúrálható ahogyan bármilyen más Java alkalmazás.

3.5.3. Kendo UI

A JavaScript kódot fájlokra szedhetjük, de itt ütközhetünk olyan problémába, hogy az elválasztás egyáltalán nem triviális és a kódon esetleg sokat kell változtatni a szétválasztás megvalósításához. Célszerű a HTML kódból kiemelni a JavaScript részeket és ezeket külön fájlba tenni majd meghívni a HTML-ből. Továbbá HTML kód átláthatósága érdekében használjunk sablonokat, ezeket külön fájlokban is tárolhatjuk. Ekkor valamivel átláthatóbb kódot kapunk, de terjedelmes GUI esetén a Kendo UI alkalmazás forráskódja nagyon elburjánzhat.

3.5.4. Spring MVC

Jelentősen elválnak az egyes részek az MVC architektúra miatt, így a kód kordában tartható. Több kontrollert alkalmazhatunk, egy kontroller url-jét meghívva több függvényének adhatunk relatív URL-t így a nézetekből könnyen elérhetőek a kontrollerek egyes metódusai. A szerver oldal kódja szépen struktúrált lesz, azonban kliens oldalon itt is beleütközhetünk az átláthatatlan JavaScript és HTML kódokba.

3.5.5. Összegzés

Egy nagy alkalmazás implementálása esetén a szerver oldal kódja a legjobban a **Spring MVC**-vel tartható karban, de a felhasználói felület felépítése minden alkalmazásban könnyen átláthatatlanná válhat.

3.6. Tesztelhetőség

A fejlesztés során fontos kérdés a tesztelés lehetősége, ami a folyamat egyik legfontosabb része. Enélkül a fejlesztés minősége nem lesz megfelelő, a fejlesztési idő pedig indokolatlanul kitolódik.

3.6.1. Google Web Toolkit

Google Web Toolkit esetén a böngészőkre telepíthető kiegészítő segítségével a GWT SuperDev mód használható, mellyel nem kell folyamatosan újrafordítani a kódot, ha változtatást végzünk, a változás azonnal megjelenik a böngészőben (frissítés után) és a kód könnyen debuggolható segítségével. Tesztelésre plusz eszközt nem építettek bele, a JUnit beintegrálásával készíthetünk unit teszteket.

3.6.2. Vaadin

Nem biztosítottak beépített tesztelő eszközt, JUnit beintegrálásával valósíthatjuk meg, de ennek megvalósítása sokszor időigényes, ha a projekt alapból nem tartalmazza a JUnithoz szükséges beállításokat.

3.6.3. Kendo UI

A Kendo UI keretrendszer JavaScript-ben íródott, így a debuggolási és tesztelési lehetőségek korlátozottak. A Chrome böngésző "*inspect element*" funkciójával vagy a Firefox Firebug kiegészítőjének segítségével a böngészőben futás közben láthatjuk a kódot és kereshetjük meg a problémákat, illetve lehetőség van a konzolra iratni, itt láthatjuk, ha például `ParseException` dob egy Ajax hívás, így kevesebb idő megtalálni a problémát. Alapvetően azonban JavaScript-et nehéz tesztelni és debuggolni, könnyít a helyzeten például a Webstorm fejlesztőkörnyezetben fellelhető lehetőség JavaScript debuggolására, ez a környezet fizetős és nem alkalmas a szerver oldal fejlesztésére.

3.6.4. Spring MVC

A Spring MVC keretrendszerben nagy hangsúlyt fektettek a tesztelésre, a projekt létrehozásakor alapból legenerálódnak a JUnit teszteléshez szükséges fájlok, könyvtárak és projekt beállítások. A tutorial-ok során is sokat foglalkoznak a tesztek megírásával. A JavaScript oldal tesztelésére ugyanolyan módszereket alkalmazhatunk, mint a Kendo UI esetében.

3.6.5. Összegzés

A tesztelés területén a **Spring MVC** megoldása a legjobb, mivel itt a készítők figyeltek a tesztelés szükségességére. Debuggolhatóságot nézve azonban a **Google Web Toolkit** nyújtja a legtöbbet, mivel a kliens oldal is debuggolható a SuperDev mód segítségével.

3.7. Dokumentáltság

A keretrendszerről elérhető dokumentáció mennyisége és minősége fontos szempontja a keretrendszer kiválasztásának. A megbízható és hatékony fejlesztés alapja minden esetben a részletes és jó dokumentáció.

3.7.1. Google Web Toolkit

A Google Web Toolkit-hez a Google sok példakódot biztosít magyarázatokkal, továbbá a docs.oracle.com-hoz hasonló teljes API dokumentáció érhető el ingyenesen interneten keresztül. Elterjedten és régebb óta használt keretrendszer, ezért az interneten keresgélve sok GWT-vel kapcsolatos problémára találhatunk megoldást.

3.7.2. Vaadin

A Vaadin-hoz a fejlesztői interneten keresztül ingyenesen elérhetővé tettek egy teljes könyvet (Book of Vaadin), amely leírja a Vaadin keretrendszer működését, segítséget nyújt a keretrendszer használatának megtanulásához. Szintén ingyenesen elérhető API dokumentáció áll rendelkezésre a docs.oracle.com Java-hoz írt API dokumentációjához hasonló formában. Több osztály forráskódja is megtalálható az interneten, ami segíthet egy

probléma megtalálásában, nekem a `JPAContainer` egy függvényével kapcsolatban a forráskód segített a legtöbbet. Több tutorial-t és példa programot is biztosít a Vaadin.

Saját fórumot is biztosít, amelyen a problémákra a Vaadin fejlesztői is válaszolnak.

3.7.3. Kendo UI

A Kendo UI honlapján található sok demo program az egyes Grid-ekhez, ezek mellett látható a forráskód is többféle megvalósítási lehetőséggel (HTML, ASP.NET, JSP). Sokat segít a honlapon található API dokumentáció, ami nem túl részletes, de találhatóak rövid példakódok az egyes Grid-ek függvényeinek, beállításainak használatához. Sok videót készítettek, amely segítségével meg lehet tanulni a keretrendszer használatát.

3.7.4. Spring MVC

A Spring MVC-hez step-by-step tutorial-okat biztosítanak, ami sokat segít annak, aki még sosem dolgozott ebben a keretrendszerben. Egyéb dokumentáció hiányos bár biztosítanak online dokumentációt, de nem a teljes API-ról. A Spring keretrendszernek nagyon sok része van, nem csak az itt bemutatott MVC, így egyes részek dokumentációinak kidolgozása elmaradtnak tűnik. Főleg a tutorial-ok segítségével tanulható meg, de interneten keresgélve több Spring különböző részeivel foglalkozó oldalakat találhatunk.

3.7.5. Összegzés

Összefoglalva a dokumentáció és a segédanyagok terén a **Vaadin** és a **Google Web Toolkit** keretrendszer nyújtja a legtöbbet.

3.8. Megtanulhatóság, fejlesztés bonyolultsága

Egy új keretrendszer választásánál bár nem a legfontosabb kérdés a megtanulhatóság és a bonyolultság, egyáltalán nem hátrány, ha a fejlesztők könnyen beletanulnak, hiszen ekkor hamarabb elkezdődhet a fejlesztés, illetve szívesebben is foglalkoznak vele az fejlesztők.

3.8.1. Google Web Toolkit

A Google Web Toolkit keretrendszerhez sok példakód, tutorial és segédanyag található az interneten, továbbá van hozzá API dokumentáció is, ráadásul teljes mértékben Java nyelven kell írni a kódot, viszont a plugin hibái miatt időnként nehéz volt kijavítani a problémákat és ez több órát is elvehet a fejlesztés idejéből, ha valaki nem találkozott még velü.

3.8.2. Vaadin

Egy Java fejlesztőnek kisebb probléma megtanulni ezt a keretrendszert, mivel Java nyelven, az RPC hívásokkal való foglalkozás nélkül feljeszthető webes alkalmazás, azonban némi webes múlt után kezdetben zavaró úgy programozni egy webes alkalmazást, mintha egy szokványos asztali alkalmazást készítenénk, nincs semmilyen szétválasztás a kód szintjén szerver és kliens oldal között.

Widgetek kezelése több esetben más, mint a szokásos Swing kezelése, megtanulása nem nehéz, de amíg nem ismertem, addig problémát okozott a könyvből és tutorial-okból a megfelelő kód összeállítása.

A `JPAContainer` új dolog, de miután megszoktuk, már kényelmes a használata, feltéve, hogy nem futunk bele még meglévő nagyobb hiányosságaiba. Például mikor hozzá akartam adni az adatbázishoz egy új rekordot a `JPAContainer`-en keresztül, akkor mindig `UnsupportedOperationException`-t kaptam. A `JPAContainer` forráskódját megnézve láttam, hogy a függvény, ami a beillesztést végezné, az nincs implementálva csak `Exception`-t dob.

`Filter`-ek használata `Container`-eken sem szokványos, logikája a szokásos logikai és/vagy műveleteken illetve SQL műveleteken alapul, de nekem meglepő volt, hogy például a “vagy” kapcsolat írása egy `OR` típusú objektum létrehozásával történik.

3.8.3. Kendo UI

A JavaScript nyelv nehézségei természetesen ennek a keretrendszernek a programozását is megnehezítik.

A Kendo UI alkalmazás fejlesztése és tanulása során nagy segítséget nyújtottak a demo programok és forráskódjaik, a JavaScript nehézsége és a kevésbé dokumentált API miatt azonban akadtak problémák a fejlesztés alatt, bár alapvetően a keretrendszer maga nem túl bonyolult.

Nehézséget okozott a szokatlan inicializálás, nincs mindig azonnal definiálva, hogy a Grid milyen típusú, hanem majd a beállításai és feltöltése után dől el.

Problémát okozott a `Button` alkalmazása, mivel HTML `input` definíció után már lehet hozzá `Listener`-t rendelni, azonban nincs konkrét `Button` Grid külön, vagy HTML-ben definiáljuk, vagy `Command`-ot alkalmazunk.

Sima `Textbox / TextField` nem található a keretrendszerben, ennek használatára megfelel az `Autocomplete`, azonban a dokumentációt kutatva először úgy tűnik, mintha `TextBox`-ot egyáltalán nem lehetne alkalmazni. Ha ismerjük az `Autocomplete` Grid-et, akkor hamar rá lehet jönni a trükkre, ha nem, akkor Google keresés során fogunk ráakadni erre a megoldásra.

3.8.4. Spring MVC

A dokumentáltság hiánya nagy problémát okozott, alapvetően nem olyan nehéz a keretrendszer felépítése, bár a step-by-step tutorial-ok nagyon jók az elején, de ez természetesen csak kis részét fedi le a keretrendszernek és csak egyféle megvalósítást mutat meg.

Elég sok nehézséget okozott a megfelelő konfigurációs beállítások elkészítése, többféle XML fájlban sok dolgot lehet beállítani, és emiatt kezdetben nehéz volt elindítani az alkalmazást.

Kliens oldal programozásánál sok probléma volt azzal, hogy az elképzelt felhasználói felületet elkészítsem, ezért összeintegráltam a Kendo UI keretrendszerrel, ami okozott egy kis nehézséget, de található volt néhány tutorial, hogy a modell-t hogyan lehet feldolgozni a Kendo UI keretrendszerben.

3.8.5. Összegzés

Számomra egyértelműen a **Vaadin** volt a legkönnyebben megtanulható keretrendszer, mivel Java nyelvben többet programoztam, mint Javascript-ben és a keretrendszer

használata egyszerű, a dokumentációk pedig sokat segítenek. Azonban aki Javascript-et alaposan ismeri, annak a **Kendo UI** megtanulása könnyen és gyorsan fog menni.

4. A KERETRENDSZEREK ELŐNYEI ÉS HÁTRÁNYAI

A fejezetben keretrendszerenkénti bontásban, a kísérleti fejlesztés során szerzett tapasztalatok alapján összegzem az egyes keretrendszerek előnyeit és hátrányait.

4.1. Google Web Toolkit (GWT)

4.1.1. Előnyök

- Kliens és szerver oldal élesen elválik.
- MVP architektúra használatára fel van készítve.
- Teljes API dokumentáció található hozzá, meg sokféle tutorial, így könnyen tanulható.
- Teljesen Java nyelven írhatjuk a webes alkalmazásunkat.
- Amennyiben szükséges, írhatunk kiegészítő HTML és Javascript kódokat is.
- App Engine-hez hasonló szervert ad localhost-ra, hogy ne kelljen minden apró átírásért feltölteni.
- WindowBuilder a GUI összeállításához.
- RPC hívások összetett objektumokat is tudnak beépítve kezelni.
- Kliens oldal debuggolhatósága.

4.1.2. Hátrányok

- Saját GUI elem létrehozása nehéz, nem erre vannak optimalizálva.
- Ha sok különböző típusú RPC hívást végzünk, akkor nagyon sok osztály / interfész szükséges.
- GWT Compile-lal kell végezni a fordítás, ez hosszabb, mint a sima Eclipse build.
- Néha probléma van a unit-cache mappával, ki kell törölni az előzőt, különben nem fordul le a program.

- Az Eclipse plug-in több bug-ot is tartalmaz egyelőre.

4.2. Vaadin

4.2.1. Előnyök

- Használatát nagyon egyszerű megtanulni, API dokumentáció, egy ingyenes könyv és rengeteg tutorial segíti a tanulást.
- Nem szükséges igazán a protokollok, speciális webes dolgok ismerete egy Vaadin webes alkalmazás készítéséhez, mindent elvégez a keretrendszer, még annak eldöntését is, hogy mi kerüljön a kliens és mi a szerver oldalra, továbbá minden beavatkozás nélkül elvégzi a biztonságos kommunikációt a kettő között.
- Nem szükséges JavaScript és HTML nyelvek ismerete, XML-re is csak minimálisan van szükség a konfigurációhoz, teljes mértékben Java nyelven írhatjuk a böngészőben működő alkalmazásunkat.
- Bármilyen Java alapú szerveren és nagyon sok böngészőben működik, még régebbiekben is.
- Új `widget`-eket alkalmaz a Swing-hez képest, amelyek segítik az egységes kezelést illetve csökkentik a feleslegesen sok változót a kódunkban például `Label` és `TextField` párosok csoportjai.
- Táblázat és egyéb adatmegjelenítő `widget` feltöltése adatokkal könnyen megoldható `Container`-ek segítségével, illetve a szűrés is viszonylag kevés kódból megoldható a `Container`-eken (így a megjelenített adatokon is).
- Témák hozzáadásával minden `widget` és az egyes `widget`-ek részei is testreszabhatóak tetszés szerint, továbbá beépített témák is alkalmazhatóak.

4.2.2. Hátrányok

- Adatbáziskezelés alap helyzetben nem található a keretrendszerben, más implementációkat kell beintegrálni, és csak egy addon alkalmazásával tudjuk

megkönnyíteni az adatbáziskezelést, azonban alkalmazása jelenleg még problémás, a `JPAContainer` implementációja kissé hiányos.

- Nem dönthetünk saját magunk, hogy mi kerül a szerver és mi a kliens oldalra.
- Meg kell bíznunk a Vaadin beépített adatvédelmében, mivel a két oldal közötti kommunikációhoz sem férünk hozzá, azt is a keretrendszer végzi.
- A testreszabásnál kissé bonyolult lehet, hogy a testreszabáshoz a generált HTML kódból (vagy szerencsés esetben a Vaadin könyvből) kell kikeresni az adott widget megfelelő részének `class` nevét.

4.3. Kendo UI

4.3.1. Előnyök

- Sokféle nyelven írt szerverhez alkalmazható kliens oldalként.
- Meglévő szerverhez könnyen csatlakoztatható, mivel szervert nem kell változtatni, amennyiben már az adatok kliens oldalra küldése implementálva van.
- Könnyen testreszabható.
- Nagyobb alkalmazás esetén is alkalmazható, mert a kód külön forrásfájlokra bontható.
- Jól különválik a szerver és a kliens oldal kódja.
- Sokféle demo és tutorial valamint dokumentáció áll rendelkezésre a Kendo hivatalos honlapján a megtanulás könnyítése érdekében.
- Nem csak webre, hanem mobilra való fejlesztésre is lehetőség van.

4.3.2. Hátrányok

- Kommunikációra nem definiál jobb megoldásokat, szokásos ajax hívásokkal történik a kommunikáció.

- Problémás az adatok bizonyos formátumban való küldése, majd ennek a szerver oldali átalakítása.
- Javascript nyelv alapos ismerete szükséges. Ha valaki nem dolgozott benne korábban eleget, hanem Java rendszerben fejlesztett és ahhoz keres keretrendszert, akkor a megtanulása kicsit nehéz.

4.4. Spring MVC

4.4.1. Előnyök

- Nagy alkalmazások esetén is jól alkalmazható az MVC architektúra miatt.
- Jól különvállnak az egyes funkciók (Model, View, Controller).
- Modellek használatával az adatok küldése a kliens oldalra egyszerűen megoldható formátumok közötti konvertálások megírása nélkül.
- Könnyen integrálható más kliens oldalt megcélzó keretrendszerekkel.

4.4.2. Hátrányok

- Sok konfigurációt kell végezni XML fájlokban.
- Nem koncentrál igazán a kliens oldalra, így azt nehezebb programozni, amennyiben nem használunk újabb keretrendszert mellé.
- A dokumentáltsága hiányos, főleg Google-ben való kereséssel találhatunk segítségeket.

5. A VIZSGÁLATOK EREDMÉNYEI TÁBLÁZATBA SZEDVE

A gyorsabb áttekinthetőség érdekében a korábban megvizsgált jellemzőket keretrendszerenkénti bontásban, táblázatban is összegyűjtöttem.

	GWT	**Vaadin**	**Kendo UI**	**Spring MVC**
Nyelv	Java	Java	Javascript (Typescript)	Java
Szerver és kliens oldal	Élesen elválnak, kommunikáció RPC-kel	Nem válnak el kódban, kommunikáció rejtett RPC-kel	Csak kliens oldalra tervezték, bármilyen nyelvű szerver lehet, kommunikáció Ajax hívásokkal, MVP	Elválnak, főleg szerver oldali, kommunikáció MVC architektúrával
Adatbázis-kezelés	App Engine Datastore API, JDO, JPA	JPAContainer, JPA, Hibernate	Nincs	Spring Data, JPA, Hibernate
GUI	Java-ban, Swinghez hasonló, de többféle elem, nehéz saját GUI elem készítése	Java-ban, Swing-hez hasonló, de többféle elem, GWT-sek is, saját widget készíthető	HTML kód kiterjesztése wrapper-ekkel vagy Javascripttel	Néhány plusz HTML tag, amúgy alap HTML tag-ek
GUI Designer	van	van	nincs	nincs
Testreszab-hatóság	CSS	Beépített és saját témák (SCSS), CSS	Beépített témák és CSS	CSS

A vizsgálatok eredményei táblázatba szedve

Biztonság	Google felhasználók védve, saját védelemre integrálható pl. Oauth / Spring Security	Beépített, RPC-hez validáció automatikusan biztosított	Nincs beépítve	Nincs beépítve, pl. Spring Security integrálható
Nagy alkalmazás esetén	Osztályokra, package-ekre bontható, de könnyen elburjánzó kód és túl sok fájl RPC-k miatt	GUI felépítése is osztályokra bontható, de könnyen elburjánzó kód	Template-ek miatt kliens oldal jól szétdarabolható	MVC architektúra miatt jól megoldható
Tesztelhető-ség	JUnit	JUnit	Nehezen megoldott, főleg böngészőben vagy pl. Webstorm-ban	Szerverhez beépített JUnit támogatás, kliens nehezen, kiegészítő megoldásokkal
Támogatott-ság	Teljes API dokumentáció, hivatalos tutorial-ok	Online ingyenes könyv, teljes API dokumentáció, hivatalos tutorial-ok	Demo programok, dokumentáció	Hiányos dokumentáció, hivatalos step-by-step tutorial-ok
Bonyolult-ság	RPC hívások szintaktikája kicsit nehézkes, többi könnyen tanulható	Egyszerű, gyorsan megtanulható	Javascript nyelv bonyolultságai miatt kissé nehéz megtanulni, amúgy egyszerű szintaktika	Szerkezet betartása fontos, konfiguráció xml fájlokban bonyolult

6. ÖSSZEFOGLALÁS ÉS TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

A sokféle keretrendszer közül négyet analizáltam, a jelenleg legnépszerűbb webes keretrendszereket, ezek név szerint a Google Web Toolkit, a Vaadin, a Kendo UI és a Spring MVC. Különböző szempontok alapján összehasonlítást végeztem rajtuk, azzal a céllal, hogy megkönnyítsem a keretrendszerek közüli választást.

Jelenleg nincs olyan keretrendszer, amely minden típusú alkalmazásnál egyértelműen a legjobb, sőt sok esetben projektenként változik az egyes funkciók fontossága, ezért az analizálás után nem egy összesített eredmény alapján rangsoroltam őket, hanem szempontonként jelöltem ki az azon a területen legtöbbet nyújtó keretrendszert.

Végeredményben a Google Web Toolkit a kommunikáció megvalósításában, az adatbáziskezelésben, a debuggolhatóságban és a dokumentáció részletességében bizonyult a legjobbnak. A Vaadin az adatbáziskezelés, dokumentáció és megtanulhatóság illetve bonyolultság területén teljesített a legjobban. A Kendo UI a felhasználói felület programozásában, a grafikus elemek testreszabhatóságában nyújtja a legtöbbet. Végül a Spring MVC a szerver és kliens oldal elválasztásánál, a kommunikációnál, a nagy alkalmazás készítésénél és a tesztelhetőségnél emelkedett ki.

Természetesen további keretrendszerekkel bővíthető ez az elemzés, hiszen rengeteg Java és JavaScript nyelven alkalmazható keretrendszer létezik és a webes kliensek iránti igények növekedésével a számuk egyre nő.

A szempontokat úgy választottam meg, hogy azok ne csak speciálisan erre a négy keretrendszerre legyenek alkalmazhatóak, hanem bármilyen webes keretrendszerre illeszthetők legyenek, programozási nyelvtől függetlenül, így ugyan én korlátoztam a vizsgálatatot a Java és JavaScript nyelvekre, beilleszthetőek .NET, Python vagy más nyelveken írt keretrendszerek is.

Az elemzésbe újabb keretrendszerek beillesztésén kívül lehetőség van még az egyes szempontok finomítására is, mivel viszonylag általánosan fogalmaztam meg az összehasonlítások alapjait, így bármilyen szakterületnél alkalmazhatóak, de szakterületspecifikus szempontokkal is bővíthető az osztályozás.

7. IRODALOMJEGYZÉK ÉS HIVATKOZÁSOK

[1] T. C. Shan, Hua W. W.: Taxonomy of Java Web Application Frameworks
e-Business Engineering, 2006

[2] James R. Lamar: Instant Kendo UI Grid
Packt Publishing, 2013

[3] John Adams: Learning Kendo UI Web Development
Packt Publishing, 2013

[4] Marko Grönroos: Book of Vaadin
Vaadin Ltd, 2014

[5] Ryan Dewsbury: Google Web Toolkit Applications
Pearson Education Inc, 2008

[6] Frederico Kereki, Essential GWT
Pearson Education Inc, 2011

[7] https://developers.google.com/web-toolkit/

[8] https://vaadin.com

[9] http://www.kendoui.com

[10] http://www.springsource.org

[11] http://docs.spring.io/