UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Framework pre výuku Java EE

Diplomová práca

Bratislava, 2018 **Bc. Lívia Kupčuliaková**

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Framework pre výuku Java EE

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Mgr. Pavel Petrovič, PhD.

Bratislava, 2018 **Bc. Lívia Kupčuliaková**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Lívia Kupčuliaková
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Framework pre výuku Java EE
Framework for Teaching Java EE

Cieľ: Technológie Java EE pre tvorbu náročnejších aplikácií majú za sebou viac ako 15 rokov vývoja. Svojím rozsahom tvoria väčší výukový celok, ktorý sa na tejto fakulte preberá na magisterskom predmete Java EE. Odprezentovanie a precvičenie jeho technológií samostatne je možné len za cenu zotrvania pri zjednodušených príkladoch, ktoré demonštrujú jednotlivé technológie. Pre účely predmetu študent v tejto práci vytvorí väčšiu ucelenú aplikáciu, na ktorej budú študenti môcť vidieť a na praktických cvičeniach aj modifikovať a programovať viacero technológií v koncerte rozsiahlejšej aplikácie určenej pre aplikačný server. Súčasťou práce je overenie a vyhodnotenie navrhnutého frameworku a zapracovanie spätnej väzby. Študent zároveň v prehľade spracuje porovnanie rozličných frameworkov nad rámec bežného rozsahu kurzu a spracuje ich do podoby samostatne prezentovateľného cyklu dvoch prednášok s ukážkami príkladov.

Vedúci: Mgr. Pavel Petrovič, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 03.10.2016

Dátum schválenia: 17.10.2016

prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestné vyhlásenie

Čestne vyhlasujem, že som túto diplomovú prácu vypracovala samostatne pod vedením vedúceho diplomovej práce, s použitím citovaných zdrojov.

V Bratislave dňa 03.05.2018 _____

meno priezvisko

Poďakovanie

Chcela by som sa poďakovať môjmu školiteľovi, Mgr. Pavel Petrovič, PhD., za jeho čas, trpezlivosť, pomoc, ochotu a cenné rady pri písaní a programovaní diplomovej práce.

Abstrakt

Vytváranie webových aplikácií v programovacom jazyku Java prináša potrebu ovládania JavaEE technológií. Keďže tvoria väčší celok, výučba je bez možnosti využitia ucelenejšej aplikácie, v ktorej sú použité zložitejšia. V tejto práci sme analyzovali vyučovanie na iných kurzoch, kde sme sa inšpirovali hlavne technológiami a postupom ich predstavovania. Následne sme navrhli a vytvorili systém, v ktorom sú použité tieto technológie. Aplikácia Kalendár je určená na vytváranie kalendárov a zobrazovanie udalostí prihláseného používateľa. Využívame komunikáciu servera a klienta cez websocket a tiež spracovanie dát v podobe XML a JSON. Do aplikácie je možné importovať celé kalendáre, ale aj jednotlivé udalosti. Druhá aplikácia, Domácnosť, slúži na správu domácnosti. Domácnosť tvorí skupina používateľov, ktorí majú možnosť zobrazovať a upravovať recepty, nákupné zoznamy priradené domácnosti. Aplikácia na generovanie stránky využíva technológiu JSF. Aplikácie sú navzájom prepojené a tak sprostredkúvajú medzi sebou svoje funkcionality. Príkladom takejto funkcie je vyhľadávanie voľných časových úsekov pre viacej používateľov súčasne, či vytváranie udalostí. Prepojenie je realizované pomocou JMS a webových servisov. Aplikácie na pripojenie k databáze využívajú JPA. Aplikácia môže byť využitá pri demonštrácii použitia vyučovaných technológií ako aj pri ich cielenom precvičení. Následne sme vytvorili množinu zadaní úloh, ktoré majú slúžiť na precvičenie.

Kľúčové slová: JavaEE, JavaEE technológie, systém, aplikácia

Abstract

Creating web applications in the Java programming language implies the need to master JavaEE technologies. Because they form a larger package, they are more difficult to teach without the use of a more complex application in which they are used. In this work, we analyzed the content and form of similar courses at various educational institutions. We focused on the selection of technologies presented in those courses and the order in which they are presented. Next, we designed and created a system that uses these technologies. Calendar is for creating calendars and viewing logged-on events. We use server and client communication through the websocket, as well as data processing in the form of XML and JSON. Users can import entire calendars as well as individual events into their own calendar. Household is the second web application designed as a housekeeping assistant. Household manages a list of users who have the ability to view and edit recipes and shopping lists associated with a household. The page generation application uses JSF technology. These two applications are interconnected, and their functions communicate with each other. An example of such a feature is to search for free time slots for multiple users at the same time or to create events. The connection is realized through JMS and web services. Both applications use JPA to connect to the database. Application has been successfully used to explain the technologies. In addition, it provides a framework that can be extended through a series of exercises and assignments. Subsequently, we have created a set of student task assignments to help them learn how to use the technologies in an integrated and efficient manner.

Keywords: JavaEE, JavaEE technology, system, application

Obsah

Zoznam obrázkov

Úvod

Popularita webových aplikácií stúpa a stavajú sa neoddeliteľnou súčasťou nášho osobného aj pracovného života. Aplikácie, ktoré používame, častokrát v pozadí obsahujú zložitú logiku spracovania dát. Z tohto dôvodu sa programátori snažia pri vývoji využívať technológie, ktoré im to čo najviac zjednodušia.

JavaEE nám ponúka možnosť vytvárania komplexných aplikácii, pri ktorých sa môžeme viac zamerať na ich biznis logiku. Práve základné technológie JavaEE sa na Fakulte matematiky, fyziky a informatiky Univerzity Komenského vyučujú v rámci kurzu Pokročilé programovanie v Jave (JavaEE). Rozsiahlosť technológií JavaEE môže sťažovať ich výuku. Preto sme sa rozhodli vytvoriť rozsiahlejší systém, ktorý pri tom môže pomôcť.

Použitie väčšieho systému využívajúceho vyučované technológie umožňuje zameranie sa priamo na práve preberanú látku. Nie je nutné zaoberať sa časťami systému, ktoré práve nie sú podstatné. Študentom tiež umožní zamerať sa priamo na precvičenie si preberaných technológií bez nutnosti riešenia ostatných častí systému.

V prvej časti práce sa pozrieme na špecifikácie vývoja JavaEE aplikácií a postupne si rozoberieme všetky JavaEE technológie, ktoré sú súčasťou výučby spomínaného kurzu. Pozrieme sa na to ako prebieha takáto výučba inde a čím by sme sa mohli inšpirovať.

V druhej časti práce si popíšeme návrh aplikácie. Pozrieme sa na funkcionalitu, ktorú nám bude systém ponúkať a špecifikujeme si využitie jednotlivých technológii v aplikácii. Na záver si popíšeme databázový model, s ktorým budeme pracovať.

Tretia časť práce slúži ako akýsi manuál, pre výučbu pomocou vzniknutého systému. Popíšeme tam priebeh hodín zameraných na jednotlivé technológie. Vysvetlíme na čo a kde v aplikácii sa používajú a spíšeme zadanie jednotlivých domácich úloh.

V poslednej časti si popíšeme výsledok nasadenia aplikácie do výučby. Budeme sa zaoberať reakciami študentov a zhodnotíme čo nám takéto testovanie prinieslo.

1 Úvod do problematiky práce s JavaEE technológiami

Úlohou tejto kapitoly je oboznámiť nás so základnou motiváciou pre túto diplomovú prácu a ďalej sa budeme venovať už samotnej platforme JavaEE a jej technológiám.

V súčasnosti sa webové aplikácie stávajú každodennou neoddeliteľnou súčasťou či už pracovného, alebo súkromného života. Keďže internetové pripojenie je dnes dostupné snáď všade a ľudia aktívnejšie využívajú svoje mobilné zariadenia na rôzne účely, vývoj webových aplikácií sa stáva atraktívnejším riešením. Webové aplikácie nám ponúkajú nezávislosť od platformy, odbremeňujú svojich používateľov od inštalácie a v neposlednom rade sú im dostupné všade tam, kde je dostupný internet. JavaEE nám ponúka možnosti na vytváranie webových aplikácií, ktoré sú schopné obslúžiť veľké množstvo používateľov súčasne, preto ju môžeme považovať za akúsi odpoveď na tento rastúci trend. JavaEE je tradične spustená na aplikačnom serveri.

Aplikačný server je špecializovaný na prevádzkovanie nejakej zdieľanej aplikácie. Používa sa pre náročnejšie aplikácie, ktoré sa skladajú z dvoch častí. Jedna je spúšťaná na aplikačnom serveri a druhá je spúšťaná u používateľa. Aplikačný server na seba preberá veľkú časť funkcionality aplikácie. Zabezpečuje aplikačnú logiku, spracováva operácie medzi klientskou časťou a databázou a tiež zaisťuje bezpečnosť. Týmto odľahčuje softvér na koncovom zariadení a zabezpečuje, aby dáta na serveri boli bezpečné. Taktiež slúži ako súbor komponentov dostupných pre vývojárov, prostredníctvom štandardných API definovaných pre vybranú platformu, v našom prípade Javu.

Využitie aplikačného servera na spúšťanie JavaEE aplikácií a využívanie rôznych API, ktoré vývojárom ponúka, považujeme za najväčší rozdiel prístupu vývoja takýchto webových aplikácii oproti vývoju založenom na framework-och, či už serverových alebo klientskych.

1.1 Motivácia

Na Fakulte matematiky, fyziky a informatiky UK sa vyučuje kurz Pokročilé programovanie v Jave (JavaEE). Súčasťou výuky tohto predmetu sú aj viaceré JavaEE

technológie, ktoré ponúkajú širokú škálu využitia. Nakoľko nie je možné dostatočne predstaviť ich obšírnosť a možnosť spoločného využitia ukážkou menších aplikácii, vznikla myšlienka vytvoriť komplexnejšiu JavaEE aplikáciu, ktorá obsiahne všetky vyučované JavaEE technológie. V budúcnosti tak bude možné v rámci kurzu demonštrovať ich použitie v kontexte rozsiahlejšej aplikácie.

Pretože ide o softvér určený na výučbu, rozhodli sme sa jeho biznis logiku zvoliť tak, aby bola každému ľahko pochopiteľná a svojou zložitosťou neodpútavala pozornosť od implementácie jednotlivých komponentov. Ako tému aplikácie sme zvolili domácnosť, jednoduchú správu členov domácnosti, nákupných zoznamov, receptov či plánovaných udalostí. Vzniknutá aplikácia, Systém domácnosť, bude rozdelená do dvoch samostatne stojacich aplikácií, Kalendár a Domácnosť. Aplikácia Kalendár bude slúžiť na zobrazovanie a správu udalostí a voľného času prihláseného používateľa. Aplikácia Domácnosť bude slúžiť na pridávanie a správu bežnej agendy v domácnosti ako sú napríklad recepty či nákupné zoznamy.

Na začiatku by sme si ešte mali povedať čo myslíme pod slovami aplikácia, systém, používateľské rozhranie či modul.

Aplikácia a systém bude v texte tejto diplomovej práce predstavovať program, ktorý bude pracovať nad nejakými dátami uloženými v databáze.

Pod pojmom používateľské rozhranie máme na mysli Java Enterprise webovú aplikáciu vytvorenú pomocou Java Server Faces (viac v časti 1.3.2). Používateľské rozhranie je preto v tomto kontexte súbor webových stránok obsahujúcich grafické prvky, ako sú tlačidlá, formuláre a podobne.

Modulom máme na mysli jednu časť používateľského rozhrania. Spravidla každý modul aplikácie Domácnosť predstavuje webovú stránku, ktorá je spojená s operáciami nad jednou entitou databázy a jej atribútmi.

1.2 JavaEE aplikácie

Platforma JavaEE je navrhnutá tak, aby programátorom pomáhala vytvárať prenosné, viacúčelové, škálovateľné, spoľahlivé a bezpečné serverové aplikácie v jazyku Java. Vlastnosti, ktoré robia enterprise aplikácie spoľahlivými, ako napríklad bezpečnosť, škálovateľnosť a flexibilita, často spôsobujú ich zložitosť. Keďže ide o súhrn štandardizovaných aplikačných rozhraní (API), ktoré využívame pri vývoji takýchto aplikácií, umožňuje nám to znížiť zložitosť vývoja a umožniť programátorom sústrediť sa hlavne na funkčnosť.

Vo viacúčelových aplikáciách je ich funkcionalita rozdelená do viacerých izolovaných funkčných oblastí, ktoré nazývame vrstvy. Spravidla sa skladajú z klientskej, strednej a dátovej vrstvy. Klientska vrstva pozostáva z klientskeho programu, ktorého úlohou je prezentovanie dát používateľovi a posielanie jeho požiadaviek na strednú vrstvu. Strednú vrstvu môžeme ešte rozdeliť na webovú a biznis vrstvu. Tu sú spracovávané požiadavky klientov, spracované a ukladané dáta do trvalého dátového úložiska v dátovej vrstve.

JavaEE nám ponúka širokú škálu rôznych štandardov a technológií, ktoré sa využívajú v jednotlivých vrstvách a uľahčujú tak programátorom ich prepojenie, ale aj spracovanie či uchovávanie dát. Vývoj takýchto aplikácií sa zameriava hlavne na strednú vrstvu, aby sa uľahčilo riadenie enterprise aplikácií, bolo robustnejšie a spoľahlivejšie.

1.2.1 Klientská vrstva

Táto úroveň pozostáva z aplikačných klientov, ktoré pristupujú na server a prevažne sa nachádzajú na inom počítači ako aplikačný server. Klient podáva požiadavky na server, ten ich spracuje a vráti odpoveď späť klientovi. Klientom môže byť webový prehliadač, samostatná aplikácia, alebo dokonca iný server bežiaci na inom počítači ako JavaEE server.

1.2.2 Webová vrstva

Skladá sa z komponentov, ktoré dokážu vytvoriť interakciu medzi klientskou a biznis vrstvou. Hlavné úlohy tejto vrstvy sú:

- Dynamicky generovať obsah v rôznych formátoch pre klienta
- Zozbierať vstupy z používateľského rozhrania a vrátiť zodpovedajúce hodnoty komponentov pre biznis vrstvu
- Ovládať postupnosť screen-ov a stránok na klientovi
- Uchovávať stav dát pre session používateľa
- Vykonávať základnú logiku a dočasne ukladať dáta v managed bean-och

JavaEE technológie, ktoré sa využívajú pri vývoji práve v tejto vrstve, nám popisuje tabuľka 1:Technológie pre webovú vrstvu.

Technológia	Účel
Servlets	Triedy v jazyku Java, ktoré dynamicky spracovávajú požiadavky a vytvárajú odpovede, zvyčajne pre HTML stránky
JavaServer Pages (JSP)	Textové dokumenty, ktoré sú kompilované do servletov a definujú možnosť dynamického obsahu na statických stránkach ako sú napr. HTML stránky
JavaServer Faces (JSF)	Framework pre používateľské rozhranie webových aplikácii, ktorý do nich vkladá komponenty, konvertuje, overuje a ukladá údaje komponentov a udržiava ich stav
JavaServer Faces Facelets	Facelets aplikácie sú JSF aplikácie, ktoré používajú skôr XHTML stránky namiesto JSP stránok
Expression Language	Množina výrazov na dynamický prístup k údajom z komponentov JavaBeans
JavaBeans Components	Objekty, ktoré slúžia nie len na dočasné ukladanie údajov pre stránky aplikácie, ale celkovo na automatizované rozhranie medzi frontendom a backendom

Tabuľka 1: Technológie pre webovú vrstvu

1.2.3 Biznis vrstva

Skladá sa z komponentov, ktoré zabezpečujú tzv. biznis logiku aplikácie. Biznis logika je vyjadrená kódom, ktorý riadi komunikáciu medzi používateľským rozhraním a databázou. Hlavnými zložkami biznis logiky sú pravidlá a postupy. Pravidlo popisuje konkrétny postup. Postup pozostáva z úloh, procesných krokov, požadovaných vstupných a výstupných informácií a nástrojov potrebných pre každý krok tohto postupu. Biznis logika opisuje postupnosť operácií spojených s údajmi v databáze na vykonávanie pravidla.

Na vývoj tejto vrstvy nám JavaEE ponúka technológie, ktoré sú zhrnuté v tabuľke 2: Technológie pre biznis vrstvu.

Technológia	Účel
Enterprise JavaBeans (EJB)	Enterprise bean-y sú riadené komponenty, ktoré sú určené pre zjednodušenie vývoja prenosných a robustných aplikácií
JAX-RS RESTful web services	Rozhranie na vývoj webových služieb, ktoré reagujú na http metódy. Webové služby JAX-RS sa vyvíjajú podľa princípov REST
JAX-WS web service endpoints	Rozhranie pre vývoj webových služieb, ktoré sú založené na štandarde SOAP(Simple Object Access Protocol)
Java Message Service	Standard poskytujúci JavaEE aplikáciám komunikáciu prostredníctvom posielania správ prostredníctvom schránok alebo frontov.
WebSocket	Umožňuje komunikáciu medzi serverom a klientom v reálnom čase reagovať na zmeny
Java EE managed beans	Spravované komponenty, ktoré umožňujú poskytovať biznis logiku aplikácie

Java Persistence API entities	Rozhranie pre prístup k údajom v dátových úložiskách a mapovanie týchto údajov na objekty programovacieho jazyka Java
Java API for XML processing (JAXP)	Rozhranie pre spracovanie XML dokumentov
Java API for JSON Processing	Rozhranie pre spracovanie dát v podobe JSON

Tabuľka 2: Technológie pre biznis vrstvu

1.2.4 Dátová vrstva

Pozostáva z databázových serverov, systémov plánovania biznis zdrojov a iných zdrojov dát. Tieto zdroje sa môžu nachádzať na inom počítači ako beží JavaEE server a komponenty k nim pristupujú v biznis vrstve.

Aj pre túto vrstvu nám JavaEE ponúka podporu pri vývoji v podobe technológií spomenutých v tabuľke 3: Technológie pre dátovú vrstvu.

Technológia	Účel
The Java Database Connectivity API (JDBC)	Rozhranie s nízkou úrovňou prístupu k údajom v dátových úložiskách. Bežne použitie je robenie SQL dotazov na konkrétnej databáze
The Java EE Connector Architecture	Rozhranie pre pripojenie k iným enterprise zdrojom dát
The Java Persistence API (JPA)	Rozhranie pre prístup k údajom v dátových úložiskách a mapovanie týchto údajov na objekty programovacieho jazyka Java. JPA je oveľa vyššej úrovne ako JDBC a skrýva zložitosť JDBC pred používateľom
The Java Transaction API (JTA)	Rozhranie na definovanie a riadenie transakcií vrátane distribuovaných alebo

	tých, ktoré sa prelínajú viacerými základnými zdrojmi dát

Tabuľka 3: Technológie pre dátovú vrstvu

1.3 JavaEE technológie

Už sme si vyššie popísali z akých vrstiev by sa mali skladať enterprise aplikácie a aké technológie nám JavaEE pre jednotlivé vrstvy ponúka. V tejto časti sa pozrieme viac na jednotlivé technológie, ktoré nám JavaEE ponúka a zameriame sa hlavne na tie, ktoré sú súčasťou kurzu, pre ktorý je táto práca tvorená [1, 2, 4, 6].

Vývoj JavaEE aplikácií sa nezaobíde bez používania bean-ov, preto si najskôr povieme o tom čo sú bean-y, aké ich skupiny máme a aký je medzi nimi rozdiel.

1.3.1 Spravovanie bean-ov

Bean-y sú v podstate inštancie tried napísaných v jazyku Java, ktoré spĺňajú isté konvencie. Je to teda Java objekt, ktorý je serializovateľný, má konštruktor bez parametrov a prístup k jednotlivým atribútom poskytuje len pomocou metód get a set. Používajú sa hlavne na zapuzdrenie viacerých dát do jedného objektu – bean-u. Takto programátorom umožňujú odkazovanie iba na jeden, namiesto viacerých objektov. Každý bean by mal mať definovaný rozsah svojej existencie, väčšinou ide o aplikačné alebo session bean-y. Definujeme to podľa toho či má existovať iba jedna inštancia v celej aplikácii – aplikačný, alebo pre každého klienta (session) je automaticky vytvorená nová inštancia tejto triedy. Jej životnosť pretrváva počas viacerých http requestov klienta na server. Na najjednoduchšie jednorázové prepojenie biznis logiky a prezentačnej vrstvy môžeme použiť request bean, ktorého platnosť je obmedzená iba na jeden http request. V jazyku JavaEE sú poskytované tri typy spravovania bean-ov, ktoré si teraz bližšie popíšeme.

1.3.1.1 Spravovanie pomocou JSF

JSF bean-y sú používané na uchovávanie hodnoty komponentov v JSF stránke (viac v časti 1.3.2). Najčastejším takýmto bean-om je trieda s anotáciou

@javax.faces.bean.ManagedBean, ktorá ma voliteľný parameter meno. Toto meno sa môže použiť na odkazovanie v JSF stránke na tento bean. Pri spravovaní bean-ov ide o to, ako sú vytvárané a inicializované. JSF používa takzvaný „lazy initialization model“ čo znamená, že bean je vytvorený a inicializovaný nie vtedy keď sa začal rozsah jeho existencie, ale až keď je prvýkrát potrebný. V dôsledku toho je ich využitie zúžené len pri jednoduchých JSF stránkach, ktorých hodnoty nemusia byť zdieľané aj v iných častiach aplikácie. Je možné sa medzi sebou odkazovať na JSF bean-y pomocou anotácie @javax.faces.bean.ManagedProperty, kde parameter value je nastavený na meno zdieľaného bean-u. Príklad takéhoto použitia môžeme vidieť na obrázku 1. Takéto odkazovanie funguje len medzi JSF bean-mi navzájom bez toho, aby programátori museli dopĺňať kód. V najnovšej verzii Java EE8 sa používanie JSF bean-ov už ani neodporúča a nahrádzajú sa CDI bean-mi.

```
@ManagedBean(name = "detektiv")
@SessionScoped
public class Detektiv {
  @ManagedProperty(value = "#{detektivka}")
  Detektivka detektivka;
```

Obrázok 1: Bean spravovaný JSF

1.3.1.2 Spravovanie pomocou Contexts and Dependency Injection (CDI)

CDI je systém spravovania bean-ov a závislostí, ktorý obsahuje úplné komplexné riešenie na spravovanie bean-ov. Bean-y spravované pomocou CDI sú oveľa pokročilejšie a pružnejšie ako tie od JSF. Má podporu pre Interceptor, Decorator, Stereotype, Event a mnoho ďalších návrhových vzorov, ktoré z neho robia oveľa silnejší nástroj ako sú bean-y v JSF. Tieto funkcionality sú však nad rámec toho čo v tejto práci potrebujeme, preto sa nimi nebudeme bližšie zaoberať. Nám stačí použitie bean-ov v rámci JSF stránky a to dosiahneme pomocou anotácie @Named. Môžeme jej dať aj meno, pomocou ktorého sa budeme naň v stránke odkazovať. Odkazovať sa na tieto bean-y môžeme pomocou anotácie @Inject nielen v iných bean-och, ale aj v iných triedach aplikácie. Na obrázku 2 môžete vidieť CDI bean s odkazom na iný CDI bean. Čo sa týka vytvárania a inicializácie bean-ov, tak v tomto type spravovania sa hneď po spustení aplikácie spustí kontajner CDI, ktorý najprv zaregistruje všetky rozšírenia a potom začne so skenovaním tried. Potom budú všetky

informácie uložené v kontajneri CDI. Dôvodom takéhoto skenovania je detekcia chýb už pri spustení a tiež výrazne zlepšenie výkonu pri behu aplikácie.

```
@Named("detektiv")
@SessionScoped
public class Detektiv {
  @Inject
  Detektivka detektivka;
```

Obrázok 2: Bean spravovaný CDI

1.3.1.3 EJB bean-y

Tieto bean-y môžeme považovať za predchodcov CDI bean-ov. V niektorých prípadoch sú si veľmi podobné no v niektorých úplne odlišné. Rozdielne sú hlavne v tom, že EJB sú transakčné, schopné využívať časovače a môžu byť asynchrónne. Môžu byť dvoch typov, stateless a stateful. Stateless môžeme považovať za bezpečné jednorazové bean-y, ktoré neukladajú žiadny stav medzi dvomi požiadavkami. Stateful uchovávajú stav a môžu byť vytvorené a existovať tak dlho ako sú potrebné, až kým nie sú zrušené. Tieto bean-y definujeme jednoducho použitím anotácie @ javax.ejb.Stateless, alebo @ javax.ejb.Stateful. Na integrovanie EJB bean-u používame anotáciu @EJB. Výhodou je, že prepájať EJB a CDI bean-y je možné bez akýchkoľvek obmedzení. Príklad takéhoto prepojenia môžete vidieť na obrázku 3.

```
@Stateless
public class Detektiv {
  @Inject
  Detektivka detektivka;
  @EJB
  Servis servis;
```

Obrázok 3: EJB bean

1.3.1.4 Použitie bean-ov

V podstate môžeme povedať, že bean-y spravované JSF by sme už nemali používať. Vo všeobecnosti by sme mali používať CDI bean-y, ak nepotrebujeme pokročilé funkcie dostupné len v EJB bean-och ako napríklad transakčné funkcie. Je možné doprogramovať funkcionalitu pre CDI bean-y, aby boli transakčnými, ale je jednoduchšie využiť rovno EJB bean-y.

Napríklad ak by sme potrebovali sledovať či nejako upravovať údaje počas commit-u transakcie použijeme v EJB bean-e anotácie @AfterBegin, @BeforeCompletion a @AfterCompletion. Jednoduchý príklad takéhoto použitia môžete vidieť na obrázku 4.

```
@Stateful
@TransactionManagement(value = TransactionManagementType.CONTAINER)
public class Servis {
  private String meno;

  @AfterBegin
  private void afterBegin() {
    System.out.println("Zacala nova transakcia.");
  }

  @BeforeCompletion
  private void beforeCompletion() {
    System.out.println("Transakcia bude komitnuta");
  }

  @AfterCompletion
  private void afterCompletion(boolean committed) {
    if (committed) {
      System.out.println("Transakcia bola dokoncena uspesne.");
    } else {
      System.out.println("Transakcia skoncila ale bol na nej vzkonanz rollback.");
    }
  }
```

Obrázok 4: Ukážka transakčného EJB bean-u

1.3.2 JavaServer Faces (JSF)

JSF je webový MVC framework na strane servera, ktorý umožňuje vytváranie webových aplikácií založených na Java technológiách. Je určené na zjednodušenie vývoja integrácie webových používateľských rozhraní. JSF nám ponúka nasledovné funkcionality:

- API na reprezentáciu komponentov a správu ich stavov
- Spracovanie udalosti, validáciu na strane servera a konverziu údajov
- Definovanie navigácie stránky
- Podporu internacionalizácie

JSF poskytuje dobre definovaný programovací model a knižnice tagov, ktoré implementujú tagy komponentov a rozlišujú sa podľa menného priestoru (viac tabuľka 4: Typy menových priestorov knižníc).

Knižnica	Prefix	Popis
JSF Facelets	ui:	Tagy pre tvorbu šablón
JSF HTML Tag Library	h:	Tagy komponentov používateľského rozhrania
JSF Core Tag Library	f:	Tagy s funkcionalitou nezávislou od RenderKitu
JSTL Core Tag Library	c:	Tagy jadra JSTL – cykly, podmienky a iné
JSTL Functions Tag Library	fn:	Tagy JSTL pre funkcie – napríklad toUpperCase a iné

Tabuľka 4: Typy menových priestorov knižníc

Tieto funkcie nám výrazne zmenšujú problém pri údržbe webových aplikácii s používateľským rozhraním na strane servera. Pomocou JSF môžeme za minimálneho úsilia vytvárať webové stránky, priraďovať dáta uložené na serveri jednotlivým komponentom, pridávať udalosti, ktoré sú naprogramované na strane servera či upravovať dáta podľa aktuálneho stavu servera.

Ako sme už vyššie poznamenali JSF je MVC framework čo znamená, že oddeľuje logiku od prezentácie. Návrhový vzor MVC navrhuje aplikáciu pomocou troch samostatných modulov, model, view a controller. Model môžeme pokladať za nositeľa dát, view nám zobrazuje používateľské rozhranie a controller riadi spracovanie požiadaviek od používateľa.

JavaBeans preberajú na seba úlohu modelu, keďže obsahujú funkcie a údaje špecifické pre aplikáciu. Vo verziách Java 7 a nižších využívali JSF stránky ManagedBean-y, ktoré sú priamo súčasťou JSF frameworku. V novej verzii sa už však prechádza na využívanie @Named bean-ov, ktoré sú spravované skupinou CDI. Najväčším dôvodom tejto zmeny bol problém s odkazovaním sa na referencie týchto bean-ov v ďalších častiach aplikácií.

Rozdielmi medzi týmito skupinami bean-ov a informáciami o nich sme sa zaoberali v kapitole 1.3.1.

Úlohu view-u preberajú Facelety, ktoré sú hlavným šablónovacím systémom JSF. Facelets je jazyk webového framework-u pre tvorbu používateľského rozhrania webových aplikácií založený na štandarde XML. Práve táto vlastnosť je veľkou výhodou pre programátorov, pretože na základnú tvorbu JSF stránok stačí využiť pár základných XML tagov. Facelet súbor je vlastne bežný xhtml kód s rôznymi tagmi knižníc, ktoré sme si popísali v tabuľke 4: Typy menových priestorov knižníc. Tagy nereprezentujú len komponenty, ale zabezpečujú aj validáciu, obsluhu udalostí či navigácie. Jednoduchú ukážku použitia JSF tagov môžeme vidieť na obrázku 5. Ide o pridávanie položiek reprezentovaných textovým reťazcom do zoznamu a ich následné zobrazovanie. Položky sú v bean-e uložené v inštancii triedy List<String>.

```
<h:body>
  List of item
  <h:form id="listform">
    <h:inputText id="newitem" label="Enter new item:"
                 value="#{itemadder.value}" required="true"/>
    <h:commandButton action="index" value="Add"/>
    <br/>
    <ui:repeat value="#{itemadder.allItems}" var="item" >
      <h:outputText value="#{item}" escape="false"/>
      <br/>
    </ui:repeat>
  </h:form>
</h:body>
```

Obrázok 5: Jednoduchá JSF webová stránka

Životný cyklus JSF aplikácie pozostáva zo šiestich fáz, v ktorých JSF spracúva formulár. Popíšeme si tieto fázy v poradí, v ktorom sú realizované počas spracovania udalostí. Tieto prechody môžeme názorne vidieť aj na obrázku 6.

1. **Obnovenie zobrazenia**

   Táto fáza začína hneď po kliknutí na odkaz alebo tlačidlo a JSF dostane požiadavku. Počas tejto fázy sa vytvorí zobrazenie, validácia na jednotlivé komponenty a pripojí sa obsluha udalostí.

2. **Spracovanie hodnôt požiadavky**

Pri vytvorení, alebo obnovení stromu dokumentu používa každý komponent metódu na reprezentáciu novej hodnoty z parametrov požiadaviek. Ak konverzia hodnoty zlyhá, generuje sa chybová správa. Táto správa sa zobrazí vo fáze renderovania odozvy. Z tejto fázy je možné skočiť rovno do poslednej, ak o to požiada niektorá z udalostí.

3. **Proces validácie**

Ako už názov hovorí tak tu sa validujú hodnoty všetkých komponentov v strome podľa pravidiel, ktoré sú preň zadané. Ak je niektorá z hodnôt neplatná tak sa vytvorí chybová správa a pokračuje sa ďalej do poslednej fázy, kde sa zobrazí rovnaká stránka s chybovou hláškou.

4. **Aktualizácia hodnôt modelu**

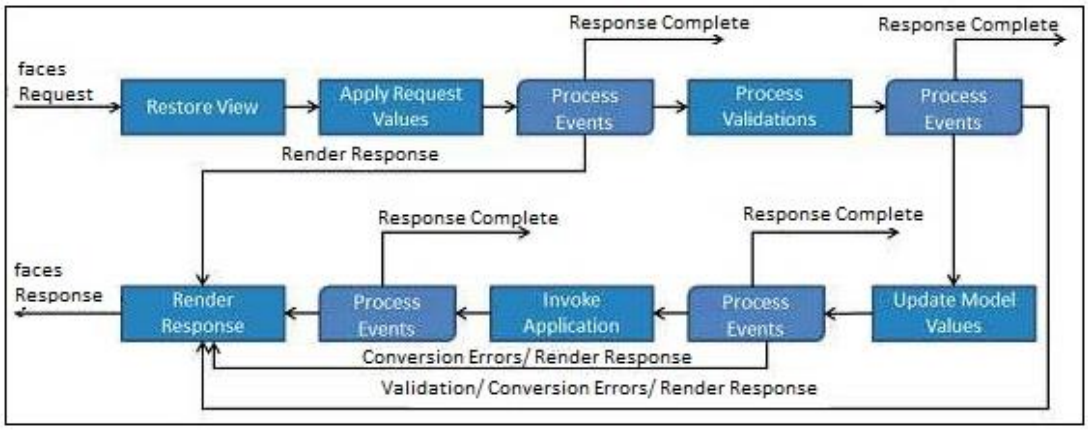
Po validácii sa pre všetky komponenty nastavia príslušné dáta na strane servera podľa ich lokálnych hodnôt. JSF tak aktualizuje atribúty bean-ov, ktoré zodpovedajú vstupným hodnotám komponentov.

5. **Vyvolanie aplikácie**

Počas tejto fázy spravuje JSF akékoľvek udalosti na úrovni aplikácie, ako napríklad prepnutie na inú stránku alebo odoslanie formulára.

6. **Renderovanie odpovede**

JSF požiada aplikačný server o vykreslenie stránky. Ak ide o prvú požiadavku na tejto stránke tak sa komponenty pridajú do stromu, inak sa znovu nepridávajú. Po vykreslení obsahu stránky sa stav odpovede uloží, aby bol prístupný pre ďalšie požiadavky a je možné ďalej pokračovať fázou 1.



Obrázok 6: Životný cyklus aplikácie JSF

1.3.3 Expression language (EL)

Táto technológia sa používa v JSF a JSP na umožnenie komunikácie s aplikačnou vrstvou, teda managed bean-mi. Umožňuje programátorom používať jednoduché výrazy na dynamický prístup k údajom z atribútov JavaBeans, ktoré môžu byť rôznych dátových typov či objektov. JSF využíva EL na vyhodnocovanie výrazov, nastavovanie či získavanie dát a tiež na volanie public alebo static metód. EL vieme využiť aj na dynamické vykonávanie aritmetických operácii. Čiže vieme priamo na stránke písať jednoduché programy, na čo nám EL poskytuje aritmetické, logické, vzťahové a podmienkové operátory. Samozrejme obsahuje aj sadu rezervovaných slov, ktoré môžeme nájsť v Java dokumentácii[1,2].

Táto technológia podporuje okamžité, alebo oneskorené vyhodnocovanie výrazov. Okamžité vyhodnotenie znamená, že výraz sa vyhodnotí a výsledok sa vráti hneď po vykreslení stránky. Oneskorené vyhodnotenie znamená, že výraz môže byť vyhodnotený kedykoľvek počas existencie stránky, kedykoľvek je to vhodné. Výrazy, ktoré majú byť vyhodnotené okamžite používajú syntax ${} a výrazy s neskorším vyhodnotením zase #{}.

Vďaka tomu, že JSF má viacfázový životný cyklus, používa väčšinou oneskorené vyhodnocovanie výrazov. Počas celého životného cyklu sa spracúvajú hodnoty a udalosti komponentov a údaje sa kontrolujú. Preto je dobré vyhodnocovanie odložiť až do príslušného stavu životného cyklu.

Výrazy používajúce okamžité vyhodnocovanie môžu byť použité iba ako súčasť šablónového textu, alebo ako hodnoty atribútov v tagoch. Sú to vždy hodnoty určené len na čítanie, teda nie je možné dáta upravovať.

Výrazy s oneskoreným vyhodnotením môžu byť vyhodnotené v rôznych fázach životného cyklu stránky. Najčastejšie sú použité v tagoch, ktoré slúžia na zdanie vstupných údajov od používateľa. Pri odoslaní požiadavky na stránku v rôznych fázach jej životného cyklu je hodnota získaná z požiadavky, overená a odovzdaná do príslušného bean-u. Tento typ vyhodnocovania sa najčastejšie používa pri hodnotách určených nielen na čítanie, ale aj na zápis a pri výrazoch s volaním metód.

V EL sú definované dva druhy výrazov: hodnotové výrazy a výrazy s volaním metód. Hodnotové výrazy môžu buď získavať, alebo nastavovať hodnotu nejakého atribútu beanu. Výrazy s volaním metód vyjadrujú referenciu na metódu, ktorá môže byť zavolaná a môže vrátiť hodnotu.

Hodnotové výrazy možno ďalej rozdeliť do výrazov rvalue a lvalue. Hodnoty rvalue môžu čítať údaje, ale nemôžu ich zapisovať a teda sú výrazmi s okamžitým vyhodnocovaním. Na druhej strane výrazy lvalue sú oneskorene vyhodnocované, preto môžu aj čítať, aj zapisovať údaje. Označenie je preto rovnaké, pre výrazy rvalue je to ${} a pre lvalue je to #{}. Výrazy hodnôt môžu obsahovať referencie na nasledujúce typy objektov a ich properties alebo atribúty:

- JavaBeans
- Collections
- typy Java Standard Edition
- implicitné objekty

Pri odkazovaní sa na tieto objekty používame názvy objektov a definovanie atribútov môžeme vykonať dvoma spôsobmi, a to bean.atribut alebo bean[atribut]. Rvalue môžu byť použité v statickom texte alebo ako hodnoty atribútov tagov. Na druhej strane výrazy lvalue môžeme použiť iba na nastavenie hodnoty atribútu tagu.

Výrazy s volaním metód sa používajú na zavolanie ľubovoľnej public metódy bean-u, ktorá vykonáva nejaké spracovanie komponentu a môže vrátiť výsledok. Sú potrebné na spracovanie udalostí, ktoré sú generované komponentami, alebo na validáciu hodnôt komponentov. Keďže metóda môže byť vyvolaná v ktorejkoľvek fáze životného cyklu stránky, musí byť definovaná len v #{}. Výrazy s volaním metód môžu byť použité len ako atribúty tagov. Metódy, ktoré majú byť zavolané, môžeme definovať dvoma spôsobmi #{bean.metoda} alebo #{bean[metoda]}. Príklady použitia môžete vidieť na obrázku 5.

1.3.4 Websocket

Súčasťou Java EE je aj API pre WebSocket, ktoré poskytuje podporu pre vytváranie aplikácií s jeho použitím. Ide o aplikačný protokol, ktorý umožňuje úplnú obojsmernú komunikáciu pomocou protokolu TCP. V tradičnom modeli využívajúcom protokol http bolo vždy potrebné, aby klient poslal požiadavku na server a až vtedy mohol server odpovedať poslaním nejakých dát. Keďže využívanie webov výrazne pokročilo a používatelia očakávajú vysoký stupeň interakcie, je potrebné, aby sa mohol obsah dostatočne rýchlo meniť. V kombinácii s inými klientskymi technológiami, ako sú napríklad JavaScript a HTML5, WebSocket umožňuje webovým aplikáciám poskytovať používateľom inak chýbajúcu interaktivitu.

V aplikáciách s WebSocket-om je na serveri na určenej URL adrese tzv. endpoint a klient sa k nemu pripája. Po vytvorení spojenia môžu počas celej jeho existencie či už klient, alebo server, posielať správy a tiež môžu kedykoľvek ukončiť spojenie. Klient sa zvyčajne pripája na jeden server, ale na jeden server môžu byť pripojení aj viacerí klienti. WebSocket podporuje textové správy, ktoré sú kódované ako UTF-8, a tiež binárne správy.

Reprezentácia endpoint-u na klientskej strane má nasledujúcu podobu:

- ws://host:port/path?query
- wss://host:port/path?query

Schéma popísaná ws predstavuje nešifrované pripojenie a schéma wss predstavuje šifrované pripojenie. Port je voliteľný no predvolené číslo portu je 80 pre nešifrované a 443 pre šifrované pripojenie. Path označuje umiestnenie endpoint-u v rámci servera a query je

voliteľné. Jednoduché vytvorenie spojenia z javascript-oveho klienta môžete vidieť na obrázku 7.

```
var wsocket;
function connect()
{
  wsocket = new WebSocket("ws://localhost:8080/Connect4/majak");
  wsocket.onmessage = onMessage;
  wsocket.onopen = function (e) {
    console.log("Connection established!");
  };
}
```

Obrázok 7: Spojenie pomocou websocket-u

WebSocket-ový endpoint je inštancia triedy javax.websocket.Endpoint. Java API nám umožňuje vytvoriť endpoint dvoma spôsobmi. Jeden spôsob je, že rozšírime triedu Endpoint a implementujeme override metódy. Druhý spôsob je triede pridať anotáciu @javax.websocket.server.ServerEndpoint a na metódy tejto triedy použijeme niektoré z anotácií @OnMessage, @OnOpen, @OnClose a iné. Triedu reprezentujúcu endpoint môžete vidieť na obrázku 8.

```
@ServerEndpoint("/majak")
public class Majak {
  Session mySession;
  @Inject
  Connect4 connect;

  @OnOpen
  public void open(Session session, EndpointConfig conf) {
    mySession = session;
  }
  @OnMessage
  public void onMessage(String text, Session session) {
    final RemoteEndpoint.Basic remote = session.getBasicRemote();
    System.out.println(text);
  }
  @OnClose
  public void onClose(Session session, CloseReason reason) throws IOException {
    //prepare the endpoint for closing.
  }
```

Obrázok 8: Trieda reprezentujúca websocket-ový endpoint

1.3.5 Spracovanie JSON

JSON je formát na výmenu údajov, ktorý je široko používaný vo webových službách a iných pripojených aplikáciách. Java EE poskytuje API, ktoré umožňuje analýzu, transformáciu a dožadovanie JSON údajov pomocou objektového alebo streaming modelu. Predpokladáme, že JSON je známy textový formát, a preto si o ňom povieme naozaj len základné informácie.

JSON definuje len dve dátové štruktúry: objekt, ohraničený {}, a polia, ohraničené []. Objekt je súbor dvojíc názov-hodnota a pole je zoznam hodnôt. Obsahuje len hodnoty typov reťazec, číslo, objekt, pole, true, false a null. Dáta majú stromovú štruktúru.

Pri spracovaní pomocou objektového modelu sa vytvorí strom, ktorý reprezentuje dáta z JSON v pamäti. Tento strom potom možno analyzovať, navigovať a upravovať. Tento prístup je flexibilný a umožňuje spracovanie, pri ktorom je potrebná celá štruktúra dát. Častokrát je však pomalší ako streaming model a vyžaduje viac pamäte. Generovanie prebieha tak, že sa vytvára celý strom naraz. Príklad spracovania JSON dát pomocou objektového modelu nájdeme na obrázku 9.

```
public static void navigateTree(JsonValue tree, String key) {
  switch (tree.getValueType()) {
    case OBJECT:
      JsonObject object = (JsonObject) tree;
      for (String name : object.keySet()) {
        navigateTree(object.get(name), name);
      }
      break;
    case ARRAY:
      JsonArray array = (JsonArray) tree;
      for (JsonValue val : array) {
        navigateTree(val, null);
      }
      break;
    case STRING:
    case NUMBER:
    case TRUE:
    case FALSE:
    case NULL:
      System.out.println(tree.getValueType().toString());
      break;
  }
}
```

Obrázok 9: Spracovanie JSON pomocou objektového modelu

Streaming model používa syntaktický analyzátor založený na udalostiach, ktorý naraz prečíta jeden element. Analyzátor generuje udalosti a zastavuje spracovanie, keď narazí na začiatok, alebo koniec objektu či poľa, nájde kľúč alebo hodnotu. Každý prvok môže byť spracovaný, alebo odstránený pomocou programu a potom analyzátor prejde na ďalšiu udalosť. Tento proces zodpovedá postupnému spracovaniu, ktoré si nevyžaduje informácie o všetkých dátach. Generovanie dát prebieha postupne vždy pre každý element zvlášť. Spracovanie JSON dát pomocou streaming modelu môžeme vidieť na obrázku 10.

```
public static void navigateTree(String jsonData) {
  JsonParser parser = Json.createParser(new StringReader(jsonData));
  while (parser.hasNext()) {
    JsonParser.Event event = parser.next();
    switch (event) {
      case START_ARRAY:
      case END_ARRAY:
      case START_OBJECT:
      case END_OBJECT:
      case VALUE_FALSE:
      case VALUE_NULL:
      case VALUE_TRUE:
        System.out.println(event.toString());
        break;
      case KEY_NAME:
        System.out.print(parser.getString() + " - ");
        break;
      case VALUE_STRING:
      case VALUE_NUMBER:
        System.out.println(parser.getString());
        break;
    }
  }
}
```

Obrázok 10: Spracovanie JSON pomocou streaming modelu

1.3.6 Spracovanie XML

Java API pre spracovanie XML (JAXP) slúži na spracovanie XML údajov pomocou java aplikácií. JAXP používa nasledujúce štandardy analyzátora Simple API pre spracovanie XML (SAX), Document Object Model (DOM) a Streaming API pre XML (StAX). Hlavné rozhrania sú definované v balíku javax.xml.parsers, ktorý obsahuje triedy SAXParserFactory, DocumentBuilderFactory a XMLInputFactory.

SAX je mechanizmus založený na sériovom prístupe, ktorý je riadený udalosťami. Dokument sa rozdelí na časti podľa začiatočných či koncových tagov, obsahov elementov a podobne. Postupne sa načítavajú údaje a sú pre tieto udalosti, ktoré hovoria o nájdení konkrétnej časti dokumentu sú volané callback metódy. Spracovanie udalostí je už na samotnom programátorovi. Ukážku spracovania pomocou SAX nájdeme na obrázku 11.

```
public class ParserDovoleniek extends DefaultHandler {
  @Override
  public void startDocument()
  {...3 lines }
  @Override
  public void startElement(String uri, String localName, String qName, Attributes atts)
  {...13 lines }
  @Override
  public void endElement(String uri, String localName, String qName)
  {...16 lines }
  @Override
  public void characters(char[] ch, int start, int length)
  {...9 lines }
```

Obrázok 11: Spracovanie XML pomocou SAX

DOM poskytuje známu stromovú štruktúru, ktorá je uložená v pamäti. Toto spracovanie je ideálne pre interaktívne aplikácie, ktoré potrebujú pracovať s celou štruktúrou dát, ktorá je tu prístupná. To vyžaduje načítanie celej štruktúry a jej uchovávanie v pamäti čo je náročne nie len na pamäť, ale aj na CPU. Ako môže vyzerať spracovanie pomocou DOM môžeme vidieť na obrázku 12.

```
private void readDoc(Document doc) {
  doc.getDocumentElement().normalize();
  NodeList nList = doc.getElementsByTagName("dovolenka");
  for (int temp = 0; temp < nList.getLength(); temp++) {
    Node nNode = nList.item(temp);
    if (nNode.getNodeType() == Node.ELEMENT_NODE) {
      Element eElement = (Element) nNode;
      System.out.print(eElement.getElementsByTagName("miesto").item(0).getTextContent());
    }
  }
}
```

Obrázok 12: Spracovanie XML pomocou DOM

StAX ponúka jednoduchší prístup ako je SAX a efektívnejší ako je DOM. Poskytuje alternatívu pre vysoko výkonné filtrovanie, spracovanie a modifikovanie s nízkym využitím pamäte a obmedzením na rozširovanie tried. Príklad ako využiť StAX na spracovanie XML nájdeme na obrázku 13.

```
XMLInputFactory factory = XMLInputFactory.newInstance();
InputStream is = HolidayXML.class.getResourceAsStream("/data.xml");
XMLEventReader eventReader = factory.createXMLEventReader(is);
while (eventReader.hasNext()) {
  XMLEvent event = eventReader.nextEvent();
  switch (event.getEventType()) {
    case XMLStreamConstants.START_ELEMENT:
      StartElement startElement = event.asStartElement();
      String name = startElement.getName().getLocalPart();
      System.out.println(name + " : ");
      break;
    case XMLStreamConstants.CHARACTERS:
      Characters characters = event.asCharacters();
      System.out.println(characters.getData());
      break;
    case XMLStreamConstants.END_ELEMENT:
      break;
  }
}
```

Obrázok 13: Spracovanie XML pomocou StAX

1.3.7 Java Message Service (JMS)

Správy sú spôsob komunikácie medzi softvérovými komponentami alebo aplikáciami. Každý klient sa pripája k takzvanému agentovi, ktorý umožňuje vytváranie, odosielanie, prijímanie a čítanie správ. Takáto komunikácia vytvára voľnú väzbu a preto je z hľadiska integrácie aplikácií vhodnou metódou. Nie je potrebné, aby odosielateľ a prijímateľ boli k dispozícii v tom istom čase za účelom komunikácie a tiež, aby o sebe vedeli akékoľvek informácie. Potrebné je, aby odosielateľ a prijímateľ poznali formát správ a ich cieľ.

Java Message Service je Java API, ktoré umožňuje aplikáciám vytvárať takúto komunikáciu pomocou správ. Ponúka nielen komunikáciu s voľnou väzbou, ale aj asynchrónnu, klient nemusí požadovať správu aby ju mohol prijať, spoľahlivú, so zárukou doručenia správy len raz. Najdôležitejším účelom JMS API umožniť prístup aplikáciám k existujúcemu messaging-oriented middleware (MOM) systému [3]. Aplikačný klienti, komponenty EJB a webové komponenty môžu odosiela, alebo prijímať JMS správy. Message-driven bean od EJB umožňuje asynchrónnu spotrebu správ a ich implementáciou môžeme implementovať súbežné spracovanie správ. Odosielanie a prijímanie správ tiež

môže byť súčasťou transakcií, čo zabezpečí, aby operácie JMS a prístup do databázy prebiehali v jednej transakcii. JMS API poskytuje podporu pre distribuované transakcie.

Ukážka triedy reprezentujúcej Message-driven bean je na obrázku 14. Ide o bean na strane klienta, preto je direction nastavené na ToClient. Podľa tejto hodnoty vie bean, ktoré správy patria jemu a má ich spracovať. Túto hodnotu nastavujeme pri odosielaní správy, ako môžete vidieť na obrázku 15.

```
@MessageDriven(activationConfig = {
  @ActivationConfigProperty(propertyName = "clientId", propertyValue = "jms/topic"),
  @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue = "jms/topic"),
  @ActivationConfigProperty(propertyName = "subscriptionName", propertyValue = "jms/topic"),
  @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Topic"),
  @ActivationConfigProperty(propertyName = "messageSelector",
      propertyValue = "Direction = 'ToClient'")
})
public class ResultReader implements MessageListener {
  public ResultReader() {
  }
  @Inject
  ListOfPendingComputes appBean;

  @Override
  public void onMessage(Message message) {...8 lines }
}
```

Obrázok 14: Message-driven bean na strane klienta

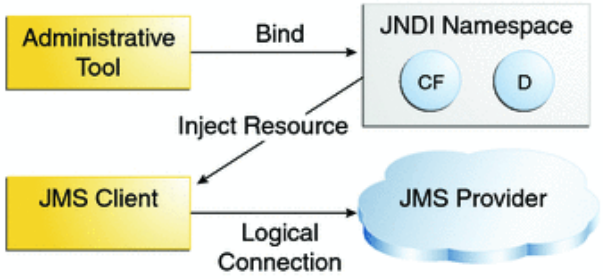
JMS aplikácia pozostáva z nasledujúcich častí

- Poskytovateľ systému na posielanie správ, ktorý implementuje JMS interface, poskytuje administratívne a riadiace funkcie.
- JMS klienti sú java programy, alebo komponenty, ktoré vytvárajú a spotrebúvajú správy. Každá zložka JavaEE aplikácie môže byť JMS klientom.
- Správy sú objekty, ktoré prenášajú informácie medzi klientami.
- Spravované objekty sú predkonfigurované JMS objekty vytvorené administrátorom pre klientske použitie. Poznáme dva typy destinations a connection factories, ktoré sú súčasťou JMS.

```
Message msg = context.createTextMessage(response);
msg.setJMSCorrelationID(originalMsgId);
msg.setStringProperty("Direction", "ToClient");
context.createProducer().send(topic, msg);
```

Obrázok 15: Odoslanie JMS správy so servera na klienta

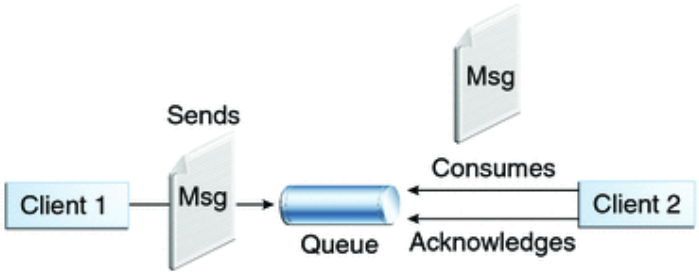
Obrázok 16 znázorňuje spôsob interakcie týchto častí. Nástroje na administráciu nám umožňujú viazať description a connection factories na JNDI (Java Naming and Directory Interface) menný priestor. Keďže v distribuovanej aplikácii potrebujú komponenty prístup k iným komponentom, JavaEE umožňuje pomenovanie jedinečným a pre používateľa prijateľným menom. Takéto meno označujeme ako JNDI meno. Všetky JNDI mená sú zhromaždené v JNDI mennom priestore, kde je definované, ku ktorému menu prislúcha aký komponent. Klient potom môže použiť resource na prístup k spravovaným objektom v mennom priestore a potom vytvoriť logické spojenie s rovnakými objektami prostredníctvom JMS poskytovateľa.



Obrázok 16:Architektúra JMS API

Na posielanie správ poznáme dva rôzne prístupy a to point-to-point a publish/subscribe prístupy. Väčšina produktov, ktoré nepoužívajú JMS API, podporovala len jeden z prístupov. S JMS API nám prišla možnosť používať ich naraz, pretože pre každý prístup poskytuje doménu. Väčšina implementácií podporuje obe domény a JMS klient kombinuje ich používanie v jednej aplikácii.

Point-to-point doména je založená na koncepte frontov správ, odosielateľov a príjemcov. Správa je vždy adresovaná do konkrétneho frontu a prijímajúci klient vyberá správy z fronty vytvorenej na zadávanie správ. Fronty obsahujú všetky odoslané správy kým nie sú spracované alebo nevypršia. Každá správa môže mať len jedného odosielateľa a jedného príjemcu, ktorý nemajú na seba nijaké časové obmedzenia. Preto je možné, aby prijímateľ prečítal správu, ktorá mu bola odoslaná v dobe, kedy nebol spustený. Vždy po prijatí správy prijímateľ potvrdzuje jej prijatie. Tento proces vidíme znázornený aj na obrázku 17.



Obrázok 17: Odosielanie správ point-to-point

V publish/subscribe(pub/sub) doméne klienti adresujú správy na topic, čo funguje podobne ako tabuľka. Zadávatelia a spracovávatelia sú anonymní a môžu dynamicky zadávať alebo spracovať hierarchický obsah. Topic uchováva správy len tak dlho ako trvá ich distribúcia aktuálnym spracovávateľom. V tejto doméne môže mať jedna správa aj viacerých prijímateľov. Zadávatelia a spracovávatelia sú v tomto prípade časovo závislí. Klient, ktorý chce spracovať topic, môže spracovávať iba správy, ktoré boli zaslané po tom ako sa klient pripojil. Tento typ posielania správ je dobre používať, keď môže byť každá správa spracovaná ľubovoľnému množstvu spotrebiteľov. Proces spracovávania správ týmto spôsobom vidíme na obrázku 18.

Obrázok 18: Odosielanie správ publish/Subscribe

1.3.8 Webové služby

Sú to klientske a serverové aplikácie, ktoré medzi sebou komunikujú. Poskytujú štandardné prostriedky pre spoluprácu aplikácii bežiacich na rôznych platformách a rámcoch. Webové služby je možné navzájom prepájať či spájať čím dosiahneme sofistikované služby s pridanou hodnotou.

Z pohľadu koncepčnej úrovne sú webové služby poskytované prostredníctvom koncového bodu prístupného v sieti. Spotrebiteľ a poskytovateľ služieb používajú správy na

výmenu informácií o požiadavkách a odpovediach vo forme vlastných dokumentov, ktoré majú málo informácii o technologických možnostiach prijímateľa. Z pohľadu technickej úrovne môžu byť implementované rôznymi spôsobmi. Poznáme dva typy webových služieb, založené na SOAP a RESTful, ktoré si bližšie popíšeme.

Java API pre XML webové servisy (JAX-WS) je technológia pre vytváranie webových služieb a klientov, ktorí komunikujú pomocou XML. V JAX-WS je vyvolanie operácie webovej služby reprezentované protokolom založeným na XML, ako je Simple Object Access Protocol (SOAP). SOAP definuje štruktúru obálok, pravidlá kódovania a konvencie na reprezentáciu žiadostí a odpovedí webových služieb. Tieto správy sa prenášajú ako SOAP správy cez protokol HTTP. Hoci sú tieto správy zložité, JAX-WS API skrýva túto komplexnosť pred programátormi. Na strane servera určuje operácie webových služieb definovaním metód, ktoré majú anotáciu @WebMethod, a ich implementáciu v Java triedach s anotáciou @ WebService. Ukážku triedy reprezentujúcu webový servis SOAP môžeme vidieť na obrázku 19.

```
@WebService(serviceName = "Faktorial")
public class Faktorial {
    @WebMethod(operationName = "faktorial")
    public long faktorial(@WebParam(name = "n") int n) {
        long result = 1;
        while (n > 1) result *= n--;
        return result;
    }
}
```

Obrázok 19: Definícia SOAP webového servisu

Na klientskej časti je potrebné vytvoriť lokálny objekt reprezentujúci službu a potom len jednoducho volať jeho metódy. Ukážka takéhoto volania je znázornená na obrázku 20. Týmto spôsobom programátor ani nevytvára, ani nerozpoznáva správy SOAP, pretože práve JAX-WS systém konvertuje tieto správy. Veľkou výhodou je možnosť úplnej nezávislosti od platformy programovacieho jazyka Java, teda je možné z klienta pristupovať k webovej službe, ktorá nie je v jazyku Java a naopak. Táto flexibilita je možná, pretože JAX-WS používa technológie definované W3C: HTTP, SOAP a WSDL.WSDL určuje XML formát na popísanie služby ako množiny koncových bodov pracujúcich na správach.

```
@WebServiceRef(wsdlLocation
    = "http://localhost:8080/FaktorialService/Faktorial?wsdl")
private Faktorial_Service faktorialService;

public void compute() {
  Faktorial faktorial = faktorialService.getFaktorialPort();
  result = faktorial.faktorial(n);
}
```

Obrázok 20: Zavolanie lokálneho objektu na strane klienta

RESTful webové služby sú vytvorené tak, aby fungovali najlepšie na webe. Representational State Transfer(REST) je architektonický štýl, ktorý špecifikuje obmedzenia, ako je jednotné rozhranie, ktoré pri aplikovaní na webové služby prináša požadované vlastnosti ako výkon, škálovateľnosť a modifikovateľnosť. V tomto architektonickom štýle sa údaje a funkcionalita považujú za zdroje a sú prístupné pomocou Uniform Resource Identifiers (URIs), väčšinou odkazom na webe. Je to vlastne súbor zdrojov, ktoré určujú ciele interakcie s klientom. Na popis tohto cieľa používame anotáciu @Path. URI poskytuje globálny priestor na adresovanie pre vyhľadávanie zdrojov a služieb. Prostriedky sú riadené pomocou súboru jednoduchých, dobre definovaných operácií. REST obmedzuje architektúru na klient-server architektúru a používanie komunikačného protokolu, najčastejšie HTTP. Klient a server si vymieňajú reprezentácie zdrojov pomocou štandardizovaného rozhrania a protokolu. Jednoduchý príklad použitia takéhoto typu webovej služby nájdeme na obrázkoch 21 a 22.

```
@Path("delitelne")
public class Delitelne {
  @GET
  @Path("{delenec}/{delitel}")
  public boolean jeDelitelne(@PathParam("delenec") int delenec,
      @PathParam("delitel") int delitel) {
    return (delenec % delitel == 0);
  }
}
```

Obrázok 21: Definícia webového servisu typu RESTful

```
Client client = ClientBuilder.newClient();
client.register(Delic.class);
try {
  vysledok = client
             .target("http://localhost:8080/rs/webresources/delitelne")
             .path("{delenec}/{delitel}")
             .resolveTemplate("delenec", delenec)
             .resolveTemplate("delitel", delitel)
             .request()
             .get(Boolean.class);
} catch (Exception e)
{
    err = "chyba: " + e.toString();
}
```

Obrázok 22: Volanie webového servisu typu RESTful

RESTful webové služby sú vhodné pre integráciu cez web a SOAP webové servisy sú vhodné pre enterprise aplikácie, ktoré majú požiadavky na kvalitu služieb. JAX-WS poskytuje štandardy pre bezpečnosť, spoľahlivosť a spolupracuje s ostatnými klientami a servermi, ktoré zodpovedajú požiadavkám WS. Pre JAX-RS by sme sa rozhodli v prípade webovej aplikácie, pretože je pre mnoho typov klientov jednoduchšie spracovávať RESTful webové služby, zatiaľ čo serverová strana sa môže rozvíjať a škálovať. Klienti sa môžu rozhodnúť, či využijú všetky, alebo len niektoré služby, ktoré webová služba ponúka.

1.3.9 Java Persistence API

Java persistence by sme mohli definovať ako ukladanie čohokoľvek, do akejkoľvek úrovne uloženia pomocou programovacieho jazyka Java. Keďže je to veľmi obsiahla definícia, my sa budeme zaoberať ukladaním Java objektov do relačných databáz. Existuje mnoho spôsobov ako pristupovať k relačnej databáze z Javy, no my sa zameriame na využitie Java Persistence API (JPA).

JPA je framework programovacieho jazyku Java na prístup a správu dát medzi Java objektami/triedami a relačnou databázou. Ako konkurenčná technológia k rozšírenému frameworku Hibernate je JPA považované za štandardný prístup pre Object to Relational Mapping (ORM) [4] v Jave. JPA definuje súbor rozhraní, ktoré môžu napĺňať rôzne implementácie. Umožňuje objektovo-relačné mapovanie objektov, ktoré dosiahneme po použití štandardných anotácii alebo XML súborov, ktoré definujú ako sa majú mapovať

triedy na tabuľky v relačných databázach. Na spracovanie dopytov a transakcií objektov voči databáze JPA definuje EntityManager API. Taktiež definuje jazyk dopytov na úrovni objektov, JPQL, ktorý umožňuje vytváranie dopytov na objekty v databáze.

Jedným z výrazných prínosov JPA je jeho minimálna potreba mapovania. Je potrebné definovať, ktoré objekty môžu byť udržiavané, v triede musí byť definovaný aj primárny kľúč, alebo jedinečný identifikátor. My sa zameriame na mapovanie použitím anotácií. JPA umožňuje umiestniť anotáciu buď k atribútu triedy alebo k jej get metóde. Pomocou anotácií je možné definovať aj závislosti medzi tabuľkami. V tabuľke 5 nájdete prehľad základných anotácii potrebných pre používanie JPA prístupu k dátam v relačnej databáze.

Anotácia	Použitie
@Entity	Definícia, že trieda reprezentuje dáta z databázy
@Table	Definuje mapovanie na tabuľku databázy
@Column	Definuje mapovanie na databázový stĺpec
@Id	Definuje primárny kľúč tabuľky
@GeneratedValue	Definuje nastavenie generovania kľúču
@Basic	Definuje mapovanie na databázový stĺpec, prevažne primitívneho typu
@Transient	Atribút triedy, ktorý nemá byť mapovaný do databázy
@Temporal	Rozširujúca anotácia definujúca aký typ má byť použitý na uchovanie časového údaju
@Enumerated	Rozširujúca anotácia definujúca atribút typu Enum
@OneToOne	Definícia vzťahu OneToOne
@OneToMany	Definícia vzťahu OneToMany
@ManyToOne	Definícia vzťahu ManyToOne
@ManyToMany	Definícia vzťahu ManyToMany
@JoinColumns/@JoinColumn	Definuje vložené cudzie kľúče/kľúč
@PrimaryKeyJoinColumn	Definuje vložený cudzí kľúč
@NamedQuery	Označuje dopyt v jazyku JPQL

Tabuľka 5: Typy anotácií

Ukážka triedy, v ktorej sú použité niektoré zo spomenutých anotácii, reprezentujúcej dáta uložené v databáze môžeme vidieť na obrázku 23.

```
@Entity
@Table(name = "eventInCalendar")
@XmlRootElement
@NamedQueries({
  @NamedQuery(name = "Eventincalendar.findById",
      query = "SELECT e FROM EventInCalendar e WHERE e.id = :id"),
  @NamedQuery(name = "Eventincalendar.findByEventId",
      query = "SELECT e FROM EventInCalendar e WHERE e.event = :eventId")})
public class EventInCalendar implements Serializable {
  private static final long serialVersionUID = 1L;
  @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Basic(optional = false) @Column(name = "id")
  private Integer id;
  @JoinColumn(name = "event", referencedColumnName = "id") @ManyToOne
  private Event event;
  @JoinColumn(name = "calendar", referencedColumnName = "id") @ManyToOne
  private Calendar calendar;
```

Obrázok 23: Entita reprezentujúca dáta uložené v databáze

Java Persistence Query Language (JPQL) je dopytový jazyk podobný SQL, ktorý definuje JPA. Najdôležitejší rozdiel od SQL je práca nad objektami a ich atribútmi namiesto tabuľkami a ich stĺpcami. Môže byť použitý na čítanie (SELECT), aktualizáciu (UPDATE) a mazanie (DELETE) údajov. JPQL môžeme využívať prostredníctvom NamedQuery alebo v dynamických dopytoch pomocou EntityManagera.

JPA poskytuje runtime API definované v balíku javax.persistence, ktorého hlavnou triedou je EntityManager. EntityManager poskytuje rozhranie na vytváranie dopytov, prístup k transakciám a vyhľadávanie, udržovanie, upravovanie a mazanie objektov. Možnosti jeho vytvorenia sú dva: pomocou EntityManagerFactory alebo vložením inštancie do EJB.

1.4 Existujúce riešenia

Na výučba JavaEE či už na univerzitách, alebo v rámci rôznych kurzov sa využívajú ukážky existujúcich aplikácií. Tento fakt má svoje logické odôvodnenie a to, že rozprávanie o tom ako niečo naprogramovať pomocou nejakej technológie nie je tak efektívne ako ukázať a vysvetliť to na existujúcom riešení. Tento trend sme si overili na základe

informačných listov a popisov viacerých kurzov [11, 12, 13, 14]. Nájsť však informácie o tom, či tieto ukážky tvoria jeden väčší ucelený celok alebo sú to menšie čiastkové aplikácie nebolo jednoduché. Podarilo sa nám nájsť dva zahraničné kurzy, kde súčasťou vyučovania boli postupne rozširujúce sa aplikácie [11, 12]. Nakoľko boli tieto informácie zistené podľa popisov a informácií o kurzoch, nie je možné zanalyzovať dopad na kvalitu či efektívnosť takejto výuky. Z tohto dôvodu sme sa rozhodli, že sa na analýzu výučby JavaEE technológií pozrieme z hľadiska vyučovaných technológií.

JavaEE ponúka rozsiahlu množinu technológií, a preto nie je možné v rámci jedného semestra odprezentovať všetky. Na základe prehľadu technológií vo vyššie spomínaných kurzoch a kurzu, pre ktorý je táto práca určená sme zistili zoznam tých najzakladanejších. Technológie môžeme rozdeliť podľa toho, o ktorú z hlavných a nevyhnutných časti systému sa starajú. Týmito časťami je spojenie s databázou, generovanie zobrazenia stránky a prepojenie s inými systémami. JPA, viac 1.3.9, tvorí základnú technológiu na prepájanie aplikácie s databázou. Čo sa týka zobrazovania a generovania webovej stránky najčastejšie sú vyučované JSF, viac 1.3.2, a JSP. Prepojenie s inými systémami je možné pomocou JMS, viac 1.3.7 , alebo dnes stále využívanejšími webovými servismi, viac 1.3.8.

V praxi existuje viacero rôznych framework-ov, ktoré programátorom umožňujú prácu s vyššie spomínanými technológiami, ale všetky sú zamerané skôr na zjednodušenie použitia jednotlivých technológií a nie na ich výučbu či prezentáciu použitia. Zdrojový kód týchto framework-ov pravdepodobne nebol vytváraný so zámerom na jednoduchú demonštráciu použitia spomínaných technológií. Práve preto nie sú vhodné na výučbu, ale môžu nám slúžiť ako východiskové riešenie, ktoré je potrebné upraviť tak, aby bol kód ľahšie čitateľnejší a využitie vybraných technológii dostatočne demonštrované[7, 8, 9, 10].

1.4.1 Spring

Spring je populárny open-source framework pre vývoj J2EE aplikácii. Môže byť využitý ľubovoľnou Java aplikáciou. Tento framework je možné využívať ako alternatívu k Enterprise Java Beans (EJB), alebo ako jeho nadstavbu. Základným dôvodom vzniku Springu je zjednodušenie vývoja enterprise aplikácií. Umožňuje voľbu implementácie (EJB, POJO viac nižšie) biznis vrstvy pre aplikačnú architektúru a nie naopak. Podporuje

implementáciu komponentov pre prístup k dátam. Taktiež priniesol abstrakciu, ktorá viedla k zjednodušeniu používania ďalších častí JavaEE, ako napríklad JMS, JMX, JavaMail, JDBC, JCA alebo JNDI.

EJB sú riadené, serverové komponenty umožňujúce modulárnu tvorbu aplikácií. Ich cieľom je oddeliť biznis logiku aplikácie od prezentačnej a perzistentnej vrstvy, ale tiež zistiť predpoklady pre integráciu s ostatnými technológiami, viac v časti 1.3.1.3.

Spring využíva Plain Old Java Object alebo teda POJO triedy. Takáto trieda reprezentuje jednoduchú triedu, alebo odľahčenú entitu, na ktorú nie sú kladené žiadne špeciálne požiadavky. Takáto trieda nemusí implementovať žiadne rozhranie, ani dediť od žiadnej inej triedy. Tieto triedy sa vyznačujú jednoduchosťou, sú jednoduché na čítanie, pochopenie a jednoduché na spravovanie.

Spring sa skladá z častí organizovaných do asi dvadsiatich modulov, ktoré sú rozdelené do skupín:

- Core Container
- Data Access/Integration
- Web
- AOP
- Instrumentation
- Test

Väčšina z nich zaobaľuje funkcionalitu, ktorá sa veľmi neodlišuje od modulov JavaEE až na jeden. Modul AOP a jeho využitie prelínajúce sa celým framework-om je jedna z najsilnejších vlastností Springu. V tomto module nám prináša podporu pre trošku iný typ programovania, ktorým je aspektovo orientované programovanie. Toto nám umožňuje oddeľovať časti kódu prelínajúce sa celou aplikáciou, ako napríklad autorizácia, logovanie, transakcie, do takzvaných aspektov a ich následnú aplikáciu na akýkoľvek POJO objekt.

1.4.2 Vaadin

Vaadin je softvérový framework pre tvorbu webových aplikácií. Tieto aplikácie sa v internetových prehliadačoch zobrazujú a správajú rovnako, akoby išlo o desktopové aplikácie. Kód je písaný v Jave a následne pomocou Google Web Toolkit-u (GWT)[5] prekladaný do JavaScriptu a ten je následne interpretovaný v internetovom prehliadači. Vaadin obsahuje veľkú sadu bežne používaných komponentov pre tvorbu webových aplikácií. Pre Eclipse ponúka plugin, ktorý umožňuje aj vytváranie vlastných komponentov podľa svojho návrhu a nápadu. Špecifikom je využívanie lazy loading, s ktorým sú dáta nahrávané až vtedy, keď sú potrebné. Tým je dosiahnutá rýchla komunikácia medzi klientom a serverom, keďže je všetka práca presunutá na server. Aplikácie vytvorené vo Vaadine majú vďaka použitiu GWT podporu vo všetkých najpoužívanejších internetových prehliadačoch, čo umožňuje programátorom zabudnúť na ladenie aplikácie pre rôzne prostredia.

2 Návrh systému

V tejto kapitole bližšie popíšeme našu aplikáciu, Systém Domácnosť. Charakterizujeme jej funkcionalitu, používateľov, popíšeme architektúru aplikácie a vysvetlíme jednotlivé moduly, z ktorých sa bude skladať. K jednotlivým častiam aplikácie priradíme technológie Java EE. Vhodné zvolenie technológií a ich čo najširšie pokrytie je jeden z podstatných cieľov tejto práce, keďže aplikácia má slúžiť ako ukážková aplikácia ich praktického využitia na reálnom príklade.

Systém Domácnosť bude pozostávať z dvoch nezávislých Java EE webových aplikácií, Kalendár a Domácnosť. Aplikácia Kalendár bude slúžiť na zobrazovanie a spravovanie časových aktivít prihláseného používateľa. Aplikácia Domácnosť bude slúžiť na zjednodušenie uchovávania receptov, organizáciu nákupov a to všetko bude ponúkať pre všetkých členov domácnosti súčasne.

2.1 Funkcie systému

V tejto časti kapitoly si popíšeme aké funkcie budú obe aplikácie ponúkať používateľom.

2.1.1 Aplikácia Kalendár

Aplikácia umožní používateľom prihlásiť sa a tak využívať všetky možnosti systému. Hlavnou úlohou tohto systému bude zobrazenie kalendára aj s udalosťami, ktoré ma prihlásený používateľ naplánované na zobrazované obdobie. Nad zobrazeným kalendárom bude možné robiť viaceré funkcie:

a) **Import –** do aplikácie bude možné importovať údaje o udalostiach v dvoch rôznych formátoch, XML a JSON
b) **Export –** z aplikácie bude možné exportovať dáta vo formátoch XML a JSON
c) **Zmeny zobrazenia –** možnosť zmeniť zobrazenie kalendára podľa ponuky
d) **Posúvanie v kalendári –** posúvanie sa v kalendári na ďalší alebo predchádzajúci týždeň, mesiac, rok

Súčasťou tejto aplikácie bude aj funkcia na vyhľadávanie voľného časového úseku pre viacerých používateľov súčasne.

2.1.2 Aplikácia Domácnosť

Aplikácia umožní používateľom prihlásiť sa do aplikácie a využívať všetky jej funkcie. Prihlásený používateľ môže vytvárať nové domácnosti, čo mu umožní neskôr upravovať informácie o tejto domácnosti. Autor domácnosti má možnosť pridávať existujúcich používateľov, alebo vytvoriť nových. Po tom ako sa používateľ stane členom domácnosti môže využívať ďalšie funkcie systému.

V časti systému zameranej na recepty bude používateľ môcť pridávať, upravovať a mazať recepty, prepočítavať množstvo ingrediencií receptov na zadaný počet porcii, alebo vytvoriť nákupný zoznam podľa potrebných ingrediencií.

V časti venovanej nákupným zoznamom bude používateľom poskytnutá možnosť vytvárať nové nákupné zoznamy, pridávať nové produkty, meniť stav vybavenosti produktov alebo pridávať udalosť, ktorá reprezentuje tento nákup, do kalendára.

2.2 Používatelia systému

Pre obe aplikácie môžeme používateľov rozdeliť do dvoch základných skupín:

1. Neprihlásený používateľ
2. Prihlásený používateľ
    a. Používateľ bez domácnosti
    b. Člen domácnosti
    c. Administrátor domácnosti

2.2.1 Neprihlásený používateľ

Pre neprihláseného používateľa bude v aplikácii Kalendár a Domácnosť prístupná len možnosť prihlásiť sa. Iné moduly a funkcionality nebudú používateľovi prístupné.

2.2.2 Prihlásený používateľ

V aplikácii Kalendár bude môcť prihlásený používateľ využívať všetku funkcionalitu systému. V aplikácii Domácnosť budeme prihlásených používateľov rozdeľovať do 3 skupín.

1. Používateľ bez domácnosti

Tomuto typu používateľa je oproti neprihlásenému používateľovi sprístupnený modul domov. Nemá ešte svoju domácnosť, a preto si v module domov môže vytvoriť vlastnú domácnosť a stane sa z neho administrátor domácnosti. Takýto typ používateľa by sa v aplikácii mal vyskytovať len v prípade, že bola odstránená domácnosť, v ktorej bol predtým členom.

2. Člen domácnosti

Tento typ používateľa môže využívať všetky funkcionality aplikácie okrem úpravy a spravovania domácnosti.

3. Administrátor domácnosti

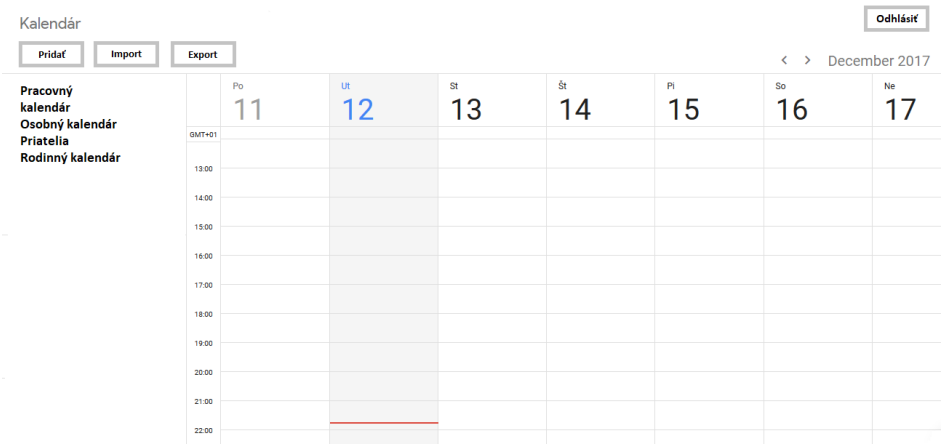
Pre tento typ používateľa je dostupná všetka funkcionalita aplikácie.

2.3 Používateľské rozhranie

V tejto časti kapitoly sa budeme venovať návrhu používateľského rozhrania.

2.3.1 Aplikácia kalendár

Táto aplikácia bude jednostránková Java EE webová aplikácia, ktorú môžeme rozdeliť do troch častí.



Obrázok 24: Návrh používateľského rozhrania pre aplikáciu Kalendár.

Horná časť

V ľavom hornom rohu aplikácie sa bude nachádzať názov aplikácie a v pravom rohu možnosť odhlásenia. Pod tým používateľ bude môcť nájsť tlačidla na prácu s kalendárom ako napríklad import, export a iné.

Ľavá časť

V ľavej časti budú tlačidlá s názvami kalendárov, ktoré patria používateľovi a budú poskytovať možnosť meniť výber zobrazovaného kalendára.

Pravá časť

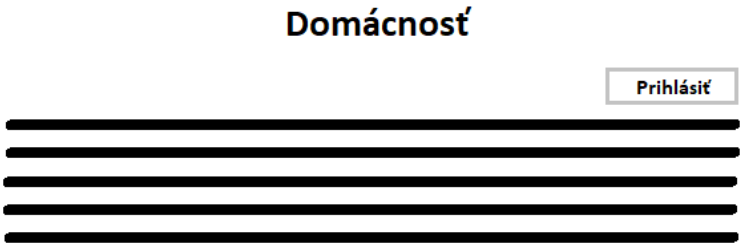
V tejto časti bude zobrazený kalendár, v ktorom budú zvýraznené jednotlivé udalosti, aj s možnosťami zmeny jeho zobrazenia.

2.3.2 Aplikácia domácnosť

Táto aplikácia sa bude skladať z viacerých podstránok, budeme ich nazývať moduly.

Hlavný modul

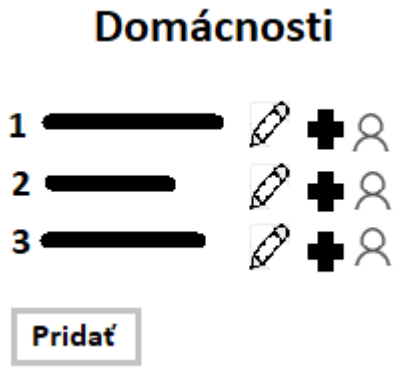
Táto časť aplikácie bude hlavnou stránkou, na ktorej bude možné sa dozvedieť základné informácie o aplikácii. Ďalej bude poskytovať možnosť prihlásenia a odhlásenia zo systému. Po prihlásení sa používateľovi otvorí modul domov, viac nižšie.



Obrázok 25: Návrh úvodnej stránky pre aplikáciu Domácnosť.

Modul domov

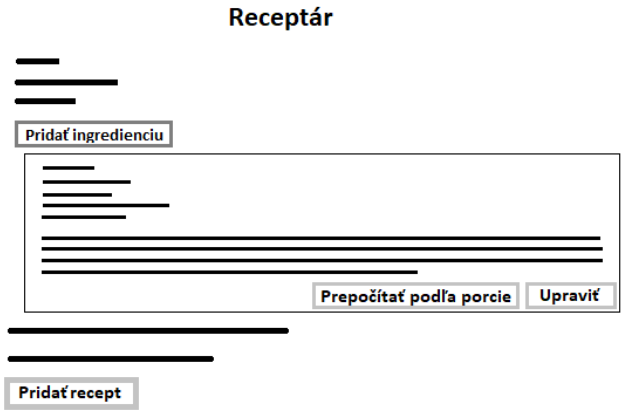
Stránka, ktorá sa používateľovi zobrazí po prihlásení. Zobrazujú sa tu všetky domácnosti v aplikácii a informácie o nich. Používateľ, ktorý je administrátorom domácnosti, má tu zobrazenú aj možnosť na úpravu informácií o domácnosti a tiež možnosť spravovania členov domácnosti. Pre používateľov, ktorí nie sú členmi žiadnej domácnosti, sa tu nachádza tlačidlo s možnosťou vytvorenia novej domácnosti. Pre konkrétnu domácnosť tu bude možné aj naplánovanie spoločnej akcie pre všetkých členov domácnosti.



Obrázok 26: Návrh stránky pre základnú stránku.

Modul receptár

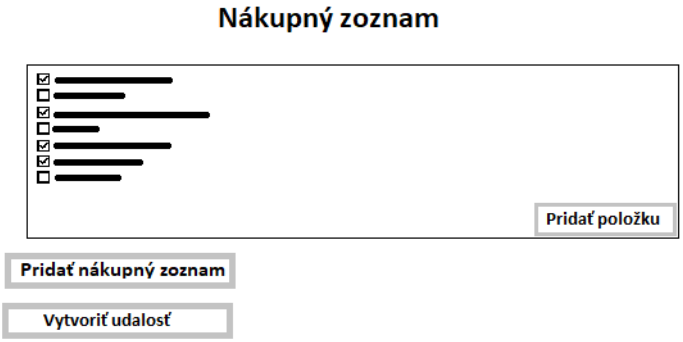
V tomto module budú zobrazené recepty a tlačidlá, s možnosťou vytvorenia nového a úpravy existujúceho receptu. Tiež bude možné pridávať alebo odstraňovať ingrediencie.



Obrázok 27: Návrh stránky pre zobrazovanie receptov.

Modul nákupný zoznam

V tomto module budú zobrazené všetky nákupné zoznamy vybranej domácnosti. Tiež bude možné vytvárať, vymazávať a upravovať nákupné zoznamy. Pre jednotlivé nákupné zoznamy, tu používateľ môže pridávať produkty a meniť ich stav vybavenia. Pre vybraný nákupný zoznam môže vytvoriť udalosť do svojho kalendára, ktorý bude reprezentovať nakupovanie položiek tohto nákupného zoznamu.



Obrázok 28: Návrh stránky pre zobrazovanie nákupných zoznamov.

2.4 Využitie technológií

V tejto časti kapitoly si navrhneme, ktoré Java EE technológie budeme, v ktorej časti aplikácií používať. Výsledné aplikácie budú pracovať nad rovnakou databázou. Prácu s databázou zabezpečíme vďaka JPA. Aplikácie budú medzi sebou komunikovať cez JMS a webový servis.

2.4.1 Aplikácia Kalendár

Zobrazenie tejto stránky bude pomocou JSF a komunikáciu so serverom zabezpečíme pomocou websocketu. Načítavanie pravidelných udalostí v roku budeme načítavať pomocou technológii SAX a StAX. Zobrazovanie udalostí v kalendári zabezpečíme pomocou údajov, ktoré uložíme do formátu JSON. Import bude možný v dvoch formátoch, XML a JSON. Pri spracovaní importovaných dát v podobe XML použijeme DOM(Document Object Model) a pri dátach v podobe JSON rozoberieme dáta využitím streaming API. Funkciu na vyhľadávanie voľného času vytvoríme ako metódu, ktorú budeme potom volať z aplikácie Domácnosť pomocou JMS.

2.4.2 Aplikácia Domácnosť

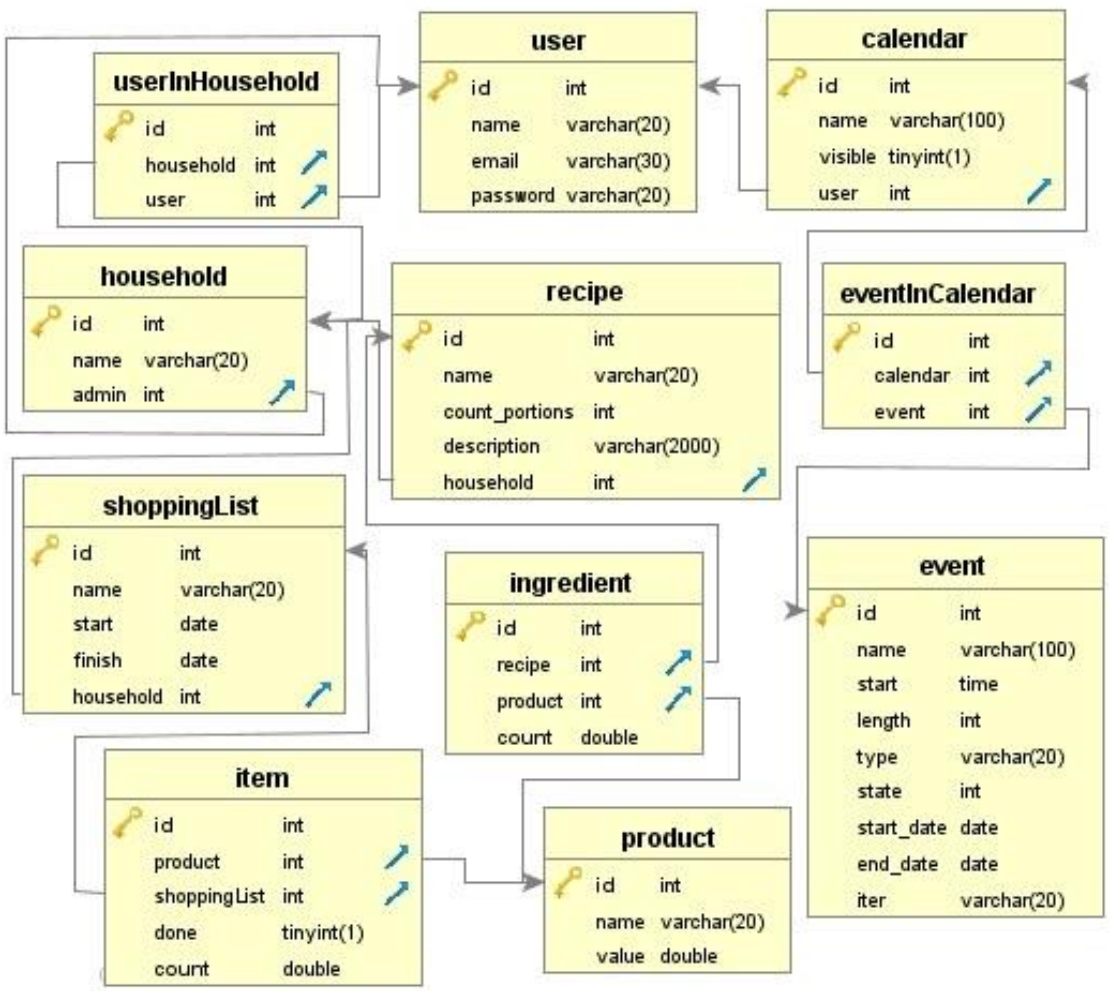
Všetky moduly, z ktorých sa táto aplikácia skladá budeme zobrazovať pomocou technológie JSF. Na komunikáciu so serverom použijeme ajax. V časti receptár budeme volať funkciu na prepočet množstva ingrediencií podľa zadaného počtu porcií. Táto funkcia bude vytvorená ako webový servis. V module domov budeme vytvárať komunikáciu s aplikáciou Kalendár pomocou JMS.

2.5 Návrh databázového modelu

V tejto kapitole si bližšie popíšeme databázový model. Databáza bude vytvorená z jedenástich tabuliek, ktorých prepojenie môžeme vidieť na obrázku 29.

2.5.1 EER diagram

Diagram zobrazuje databázový model systému Domácnosť, štruktúru jeho tabuliek a dátové typy jednotlivých stĺpcov. Obrázok kľúča pri názve stĺpca určuje primárny kľúč, šípka určuje cudzí kľúč.



Obrázok 29: Diagram databázového modelu systému Domácnosť

2.5.2 Popis modelu

V tejto časti kapitoly si popíšeme, aké dáta uchovávajú jednotlivé tabuľky a ich parametre.

Tabuľka user

V tejto tabuľke budeme zaznamenávať informácie o všetkých zaregistrovaných používateľoch systému.

- id (int) – jedinečný identifikačný údaj používateľa
- name (varchar) – meno používateľa, ktoré sa bude zobrazovať v aplikácii
- email (varchar) – prihlasovacie meno do aplikácie

- password (varchar) – prihlasovacie heslo do aplikácie

Tabuľka Calendar

V tejto tabuľke budú zaznamenané všetky v aplikácii vytvorené kalendáre a informácie o nich.

- id (int) – jedinečný identifikačný údaj kalendára
- name (varchar) – názov kalendára
- user (int) – id používateľa, ktorému patrí tento kalendár
- visible (boolean) – prístupnosť kalendára ostatným používateľom

Tabuľka Event

Tu budeme zaznamenávať všetky udalosti vytvorené v aplikácie aj s informáciami o nich.

- id (int) – jedinečný identifikačný údaj udalosti
- name (varchar) – meno udalosti
- start (time) – čas začiatku udalosti
- length (int) – dĺžka trvanie v hodinách
- type (varchar) – typ udalosti
- state (int) – stav udalosti
- start_date (date) – dátum začiatku udalosti
- end_date (date) – dátum konca udalosti
- iter (varchar) – opakovanie udalosti

Tabuľka EventInCalendar

Táto tabuľka bude slúžiť na prepojenie kalendárov s udalosťami.

- id (int) – jedinečný identifikačný údaj
- event (int) – id udalosti
- calendar (int) – id kalendára, do ktorého udalosť patrí

Tabuľka Household

V tejto tabuľke sa budú ukladať všetky v aplikácii vytvorené domácnosti a informácie o nich.

- id (int) – jedinečný identifikačný údaj domácnosti
- name (varchar) – názov skupiny
- admin (int) – id používateľa, ktorý vytvoril domácnosť

Tabuľka UserInHousehold

V tejto tabuľke budú záznamy o prepojení používateľov s jednotlivými domácnosťami.

- id (int) – jedinečný identifikačný údaj
- household (int) – id domácnosti
- user (int) – id používateľa

Tabuľka Product

Všetky v aplikácii vytvorené produkty a informácie o nich budú uložené v tejto tabuľke.

- id (int) – jedinečný identifikačný údaj produktu
- name (varchar) – názov produktu
- value (double) – objem balenia

Tabuľka Recipe

Informácie o všetkých receptoch, ktoré v aplikácii vzniknú, alebo budú do nej importované budú uložené v tejto tabuľke.

- id (int) – jedinečný identifikačný údaj receptu
- name (varchar) – názov receptu
- count_portions (int) – počet porcií
- description (varchar) – popis/postup receptu
- household (int) – id domácnosti, ktorej recept patrí

Tabuľka Ingredient

V tejto tabuľke sa uchovávajú všetky ingrediencie a informácie o nich.

- id (int) – jedinečný identifikačný údaj
- recipe (int) – id receptu
- product (int) – id produktu
- count (double) – množstvo produktov v recepte

Tabuľka ShoppingList

V tejto tabuľke sa uchovávajú všetky nákupné zoznamy a informácie o nich.

- id (int) – jedinečný identifikačný údaj
- name (varchar) – názov nákupného zoznamu
- start (time) – kúpiť od
- finish (time) – kúpiť do
- household (int) – id domácnosti, do ktorej patrí

Tabuľka Item

V tejto tabuľke sa uchovávajú všetky položky a informácie o nich.

- id (int) – jedinečný identifikačný údaj
- product (int) – id produktu
- shopping List (int) – id nákupného zoznamu
- count (double) – množstvo produktov v nákupnom zozname

3 Využitie aplikácie Domácnosť vo vyučovaní

V tejto kapitole sa budeme venovať priebehu vyučovacích hodín, na ktorých bude využitý systém Domácnosť. Postupne si prejdeme jednotlivými technológiami a popíšeme ich využitie v aplikácii. Súčasťou tejto kapitoly budú aj zadania domácich úloh pre jednotlivé technológie, ktoré sme vytvorili pre študentov. Budeme postupovať podľa poradia, v ktorom odporúčame, aby boli technológie vyučované.

Poradie vyučovania jednotlivých technológií sme sa snažili zvoliť tak, aby na seba nadväzovali a aby študenti z hodiny na hodinu vedeli vytvoriť stále rozšírenejšiu a multifunkčnejšiu webovú aplikáciu. Najskôr sa zameriame na formáty, v ktorých sú častokrát reprezentované dáta, ktoré aplikácie spracúvajú, posielajú či dostávajú na vstupe. Takže si ukážeme spracovanie XML a JSON. V ďalšom kroku im chceme ukázať možnosť vytvorenia jednoduchej webovej stránky, ktorá vie spracovať udalosti po kliknutí na ľubovoľné tlačidlo a zobrazovať aktuálne údaje. Na vytvorenie takejto webovej stránky si ukážeme technológiu JSF. Takúto stránku chceme ďalej rozširovať o možnosť dynamického vytvárania jej obsahu, a preto študentom ukážeme ako na stránku pridať ajax. V súčasnosti už aplikácie so statickým obsahom sa vyskytujú len ojedinelo a nahrádzajú ich stránky s dynamickým obsahom, ktorý je prevažne uložený v databázach. Zvolili sme reprezentáciu dát v relačných databázach, ich spracovanie a využitie vo webových aplikáciách. Rozhodli sme sa, že študentom ukážeme pracovanie s dátami uloženými v databáze pomocou technológie JPA. Pri možnosti stáleho sa menenia dát je potrebné aj ich dynamické zobrazovanie na stránkach. Aby sme vždy nemuseli čakať na nejakú akciu od používateľa, po ktorej by sa upravilo zobrazenie, do aplikácii pridávame technológiu websocket, ktorá nám umožňuje dynamické generovanie obsahu stránky. Preto ako ďalšia technológia, o ktorú rozšírime obzory študentov bude práve websocket. Ako reakciu na dnes stále viac rozširujúci sa trend prepájania viacerých aplikácií, sme sa rozhodli do výučby zakomponovať aj rôzne spôsoby komunikácie dvoch rôznych aplikácii medzi sebou. Zvolili sme technológie JMS, JAX-RS a JAX-WS. Na posledných hodinách kurzu študentom ukážeme iné Java framework-y, Vaadin a Spring, ktoré tiež umožňujú pracovať s vyššie spomenutými technológiami. Cieľom je študentom predstaviť aj iné možnosti ich využitia, ktoré je častokrát aj jednoduchšie. Počas kurzu sa študenti venujú aj iným technológiám ako

napríklad synchrónne či asynchrónne sockety a servlety, Java streams a batch. Java streams síce v našej aplikácii viackrát využívame, ale ani z ďaleka nepokrývajú tieto príklady všetky možnosti ich využitia, a preto ich nepoužívame pri ich výučbe ako hlavný zdroj príkladov a nevenujeme im viac pozornosti z tejto práci. Ostatné spomínané technológie nie sú súčasťou tejto práce nakoľko sme im nenašli dostatočné uplatnenie a tiež máme k dispozícii dostačujúce zdroje ukážkových aplikácii.

Súčasťou každej hodiny bude teoretický úvod do práve preberanej technológie, v prípade záujmu a dostupnosti ukážka, príkladov ich využitia na príkladoch z predchádzajúcich rokov. Keďže to bude súčasťou každej hodiny nebudeme to ďalej pravidelne spomínať. Taktiež sme pre študentov pripravili zadania domácich úloh k jednotlivým technológiám, ktoré majú zabezpečiť ich precvičenie. Veríme, že študenti viac pochopia a zachovajú si informácie, ktoré o jednotlivých technológiách dostali, keď si vyskúšajú prácu s nimi sami a nebudú len o tom počuť.

3.1 Spracovanie XML

Spracovanie dát v podobe XML sa používa v aplikácii Kalendár na import dát a pri prenose dát zo servera na javascript-ového klienta, ktorý následne dáta zobrazuje na stránke. V aplikácii máme použité všetky tri metódy načítania údajov z XML a jeden spôsob využijeme aj na vytvorenie XML štruktúry.

Načítavanie XML štruktúry údajov pomocou metódy SAX, sa používa v časti **`XML.MeninyXML`**. Zo súboru udalosti.xml, ktorý nájdete v **`WebApplication/src/java`**, sa načítajú udalosti, ktorých type je meniny. Keďže tieto informácie potrebujeme dostať v podobe JSON, tak sa z načítaných dát rovno vytvárajú objekty JSON, ktoré sú neskôr vrátené ako výstup tohto spracovania. Štruktúra dát je nami navrhnutá pre potreby tejto práce a preto je možné si jej reprezentáciu pozrieť na obrázku 30. V tejto štruktúre sa nachádzajú tri tagy:

- udalost – reprezentuje jednu udalosť a jej parameter type označuje o aký typ udalosti sa jedná
- nazov – označuje názov udalosti

- datum – predstavuje dátum, kedy má udalosť prebiehať

Ďalšou použitou metódou spracovania dát v podobe XML je StAX. Používame ho v časti **`XML.SviatkyXML`**, kde z rovnakého súboru ,udalosti.xml, načítavame udalosti, ale v tomto prípade filtrujeme iba tie, ktorých typ je sviatok. Rovnako ako v predchádzajúcom prípade sa z nich vytvárajú objekty JSON, ktoré vraciame na výstupe. Keďže ide o ten istý vstupný súbor tak aj štruktúra dát je rovnaká a môžeme ju vidieť na obrázku 30.

```
<?xml version="1.0" encoding="UTF-8"?>
<timetable version="0.1" name="sviatky">
    <udalost type="sviatok">
        <nazov>Deň vzniku Slovenskej republiky</nazov>
        <datum>01.01.</datum>
    </udalost>

    <udalost type="meniny">
        <meno>Emil</meno>
        <datum>31.01.</datum>
    </udalost>
</timetable>
```

Obrázok 30: Udalosti reprezentované v štruktúre XML

Načítavanie pomocou metódy DOM je použité na importovanie udalosti do aplikácie. V triede **`com.controllers.ApplicationManager`** sa nachádza metóda s názvom **`importXmlData`** so vstupnými parametrami, názov súboru a id práve prihláseného používateľa. Dáta, ktoré je možné takto importovať by mali byť vyexportované zo systému Candle, ktorý študenti našej fakulty poznajú ako systém, v ktorom nájdu rozvrhy. Tento import umožňuje pridávať nové udalosti pre práve zobrazený kalendár a príklad ich štruktúry môžeme vidieť na obrázku 31. Práve zobrazený kalendár znamená, že bolo kliknuté na jedno z tlačidiel s názvami kalendárov a sú zobrazené udalosti, ktoré mu patria. Udalosti sa do databázy pridajú iba v prípade, že ešte neexistujú.

```
<timetable version="0.1" name="1mAIN">
  <lesson id="1252">
    <type>Seminár</type>
    <room>A</room>
    <subject>Projektový seminár (1)</subject>
    <day>Pi</day>
    <start>8:10</start>
    <end>9:40</end>
    <teacher>R. Ďurikovič</teacher>
    <teacher>A. Riečický</teacher>
  </lesson>
</timetable>
```

Obrázok 31: Export rozvrhu zo systému Candle

Vytvorenie štruktúry dát do podoby XML je zastúpená využitím metódy DOM. Zobrazenie aktuálnych kalendárov prihláseného používateľa je generovaný javascript-om a preto je potrebné tieto údaje tam zo serveru dostať. V časti **`XML.KalendarXML`** sa z databázy načítajú všetky kalendáre, ktoré majú byť prihlásenému používateľovi zobrazené. Následne sa spracujú do formátu XML, ktorého reprezentáciu môžete vidieť na obrázku 32. Tieto dáta sa potom pomocou websocket-u posielajú na klienta kde sú následne spracované a použité na zobrazenie zoznamu kalendárov. Spracovanie v javascript-e si môžete pozrieť v súbore **`calendar.js`**.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<timetable name="calendars" version="0.1">
<calendars>
        <calendar id="3">
                <name>Pracovný kalendár</name>
        </calendar>
</calendars>
</timetable>
```

Obrázok 32: Kalendáre vo formáte XML

3.1.1 Zadanie domácej úlohy o spracovaní XML

Táto domáca úloha pozostáva z dvoch častí, v jednej si precvičia spracovanie vstupu v podobe XML a v druhej vytvorenie dát do štruktúry XML.

Vašou úlohou je rozšíriť aplikáciu Kalendár o možnosť automatického exportu dát zo systému vo formáte rovnakom ako sú dáta z aplikácie Candle. Na export dát použite metódu DOM.

Prerobte štruktúru, ktorou sú dáta reprezentované v súbore **`import.json`**, tak aby to bola štruktúra vo formáte XML. Následne rozšírte aplikáciu Kalendár o možnosť automatického importu dát v tomto formáte. Na načítanie dát použite jednu z metód SAX a StAX.

3.2 Spracovanie JSON

Spracovanie JSON sa využíva v našej aplikácií Kalendár. Využívame ho na import dát do aplikácie a na prenos dát medzi serverom a javascript-ovým klientom, ktorý tieto dáta následne zobrazuje.

Vytvorenie reprezentácie dát v podobe JSON sa v aplikácii používa na odoslanie informácii o udalostiach vybraného kalendára do javascript-u, kde sa tieto dáta používajú na zobrazenie kalendárového komponentu s vyplnenými udalosťami. Kalendár zobrazujeme pomocou javaskript-ového objektu **`Fullcalendar`**, ktorý dostáva vstupné dáta v podobe JSON, ktoré reprezentujú jednotlivé zobrazované udalosti. Dáta sa spracúvajú do formátu, ktorého ukážku môžeme vidieť na obrázku 33.

```
{"events": [{
        "ranges": {
                "start":"2018-03-18",
                "end":"2018-09-18"},
        "start": "08:10",
        "end": "11:09",
        "title": "Programovanie paralelných a distribuovaných systémov",
        "dow": [1]
        }]
}
```

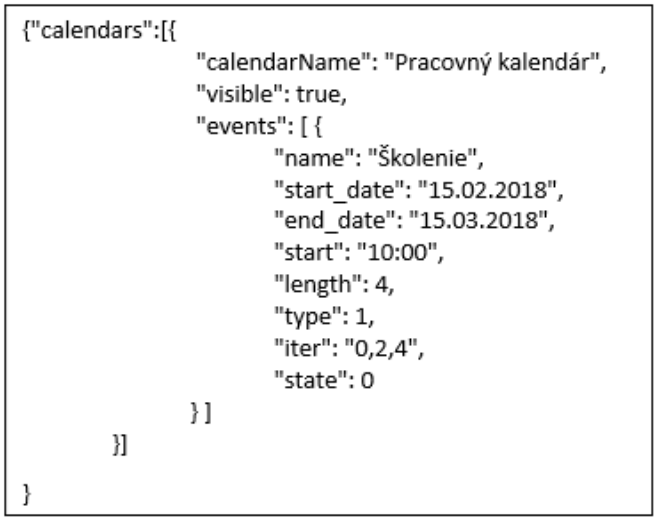
Obrázok 33: Reprezentácia udalostí vo formáte JSON

Trieda **`Event`** reprezentuje entity z databázy, ktoré sú uložené v tabuľke **`Event`**. Kľúč **`events`** reprezentuje zoznam udalosti, ktoré sú reprezentované objektami triedy **`Event`**. Každý objekt tohto zoznamu obsahuje nasledujúce hodnoty:

- title – vyjadruje názov udalosti, ktorý je v triede **`Event`** uložený v atribúte **`name`**.
- start – reprezentuje časovú informáciu kedy daná udalosť začína v rámci dňa, v triede **`Event`** je reprezentovaná atribútom **`start`**.
- end – znamená časový údaj kedy udalosť končí v rámci dňa, v triede **`Event`** táto hodnota nie je priamo uložená, ale dopočítava sa pomocou atribútov **`start`** a **`length`**.
- ranges – predstavuje obdobie, v ktorom sa má pravidelne udalosť opakovať. V triede **`Event`** sú tieto informácie reprezentované atribútmi **`startDate`** a **`endDate`**.
- dow – značí informáciu, v ktoré dni týždňa sa má udalosť opakovať. V triede **`Event`** v atribúte **`iter`** je uložený reťazec, ktorý reprezentuje toto opakovanie. Je to vlastne zoznam čísel, začínajúci od 0 (pondelok) do 6(nedeľa).

V aplikácii Kalendár v časti **`JSON.CallendarJSON`** sa pre vybraný kalendár načítajú udalosti z databázy a následne sa spracúvajú do podoby JSON, ktorý sme si vyššie popísali. Tu sa pridajú k udalostiam z databázy aj sviatky a meniny načítané zo súboru udalosti.xml. Tomuto načítaniu dát sme sa venovali v časti 3.1. Takto spracované dáta sú následne pomocou Websocketu odoslané do javascript-u, kde sa dáta nastavia ako jeden zo vstupných parametrov pre objekt **`Fullcalendar`**.

Ako ukážku načítania a spracovania dát v podobe JSON je vytvorená možnosť importovania kalendárov a udalostí do aplikácie. V aplikácii v časti **`com.controllers.ApplicationManager`** sa nachádza metóda s názvom **`importJsonData`**, ktorej vstupnými parametrami sú názov súboru a id používateľa, ktorý chce dáta importovať. Metóda slúži na vytvorenie celých kalendárov aj s udalosťami. Dáta sa do databázy zapíšu len v prípade, že už tam také neexistujú. Formát vstupných dát pre import je jednoduchá štruktúra, ktorú sme sami navrhli a jej ukážku môžete vidieť na obrázku 34. Testovací súbor s názvom **`import.json`**, na ktorom je možné vyskúšať túto technológiu nájdete priložený pri zdrojových kódoch na priloženom CD. Tieto dáta sú v aplikácii reprezentované objektami tried **`Calendar`** a **`Event`**.



Obrázok 34: Reprezentácia kalendárov a ich udalosti vo formáte JSON

Kľúč calendars obsahuje zoznam všetkých kalendárov, ktoré chceme pridať do aplikácie. Údaje kalendára sú reprezentované nasledovne:

- calendarName – označuje názov kalendára, ktorý je uložený v triede **`Calendar`** v atribúte **`name`**.
- visible – určuje či je kalendár viditeľný pre všetkých používateľov aplikácie. V triede **`Calendar`** je uložená táto hodnota v atribúte **`visible`**.
- events – reprezentuje zoznam udalostí, ktoré patria tomuto kalendáru. V triede **`Calendar`** nájdeme tieto informácie v atribúte **`eventInCalendarList`**, ktorý predstavuje zoznam objektov triedy **`EventInCalendar`**. Táto trieda reprezentuje prepojenie kalendárov a udalostí.
- name – značí názov udalosti, v triede **`Event`** je to atribút **`name`**
- start_date – označuje dátum začiatku obdobia, v ktorom sa bude udalosť pravidelne opakovať. Túto hodnotu reprezentuje atribút **`startDate`** v triede **`Event`**.
- end_date – predstavuje dátum konca vyššie spomínaného obdobia. V triede **`Event`** je to atribút **`endDate`**.
- start – je časový údaj, reprezentujúci začiatok udalosti v rámci dňa. Atribút **`start`** triedy **`Event`** reprezentuje túto hodnotu.
- length – vyjadruje dĺžku trvania udalosti, v triede **`Event`** túto hodnotu predstavuje parameter **`length`**

- type – určuje typ udalosti, ktorý je uložený v atribúte **`type`** v triede **`Event`**
- iter – znamená pravidelné opakovanie udalosti v jednotlivých dňoch týždňa. Ide o zoznam číselných reprezentácii dní, ktorý je uložený ako reťazec v atribúte **`iter`** v triede **`Event`**
- state – reprezentuje hodnotu, ktorá je uložená v triede **`Event`** v atribúte **`state`**

3.2.1 Zadanie domácej úlohy o spracovaní JSON

Vašou úlohou je rozšíriť aplikáciu **`KalendarDomacnost`** o možnosť automatického exportu dát zo systému vo formáte rovnakom ako je reprezentovaný v súbore **`import.json`**, teda ten, z ktorého je v aplikácii urobený import.

3.3 JSF

Zobrazenie webovej aplikácie **`Domacnost`** sa generuje pomocou technológie JSF. Každá jej podstránka je vytvorená ako .xhtml súbor.

Úvodná stránka slúžiaca na prihlásenie je jednoduchým príkladom využitia čisto len technológie JSF na zabezpečenie vykreslenia a funkcionality webovej stránky, **`users.xhtml`**. V ďalších podstránkach aplikácie už využívame aj technológiu ajax, ktorá nám po niektorých akciách zabezpečí renderovanie obsahu, ktorý bol akciou upravený. Príkladmi týchto stránok sú **`householders.xhtml`**, **`shoppingLists.xhtml`** a **`recipes.xhtml`**.

Pre využívanie dát zo servera na takto vytvorených webových stránkach potrebujeme tiež bean-y, v ktorých budú tieto dáta uložené. Všetky tieto Named bean-y nájdeme v časti jsf.. V systéme **`Domacnost`** používame dva prístupy k rozsahu existencie bean-ov, applikačný a session. Ukážkou aplikačného prístupu je trieda **`jsf.ApplicationManager`**, jeho úlohou je udržiavať potrebné informácie pre fungovanie celého systému. Udržiavame tu zoznam práve prihlásených používateľov a k nim prislúchajúce zobrazené domácnosti a websocket-ové endpoint-y, ktorých využitiu sa budeme venovať neskôr. Session bean-ov máme v aplikácií viac, **`UserController`**, **`HouseholdController`**, **`ShoppingListController`**,

`RecipeController a ProductController`. Každý z nich uchováva dáta potrebné pre prislúchajúcu podstránku pre jedného prihláseného používateľa.

3.3.1 Zadanie domácej úlohy

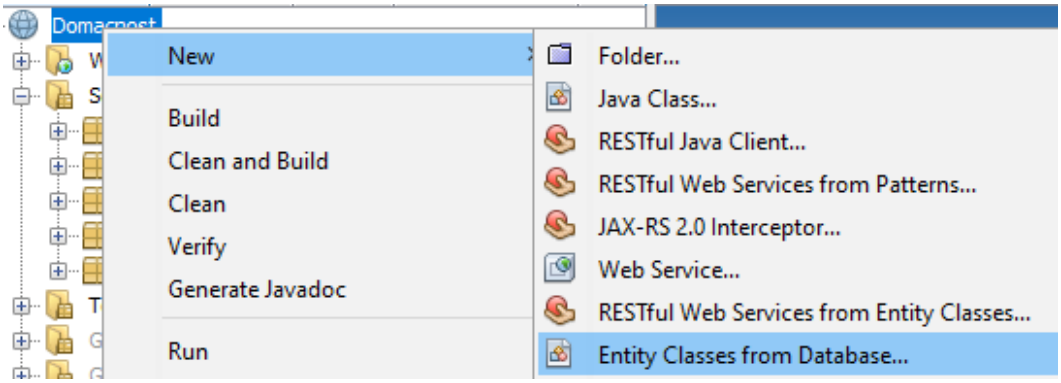
Pre túto technológiu sme študentom pripravili domácu úlohu, ktorá je však spojená aj s technológiou JPA, ktorej výučbu odporúčame o niečo neskôr a preto jej zadanie bude presne zadefinované až neskôr v časti 3.4.1.

3.4 JPA

Na prepojenie aplikácie Domácnosť s databázou sme využili technológiu JPA. Na reprezentovanie dát, ktoré prislúchajú jednotlivým technológiám používame objekty tried, na ktoré sú tieto dáta mapované. Všetky takéto triedy sú anotované pomocou @Entity. V aplikácii Domácnosť nájdete všetky entity prislúchajúce tabuľkám v databáze v časti entity.

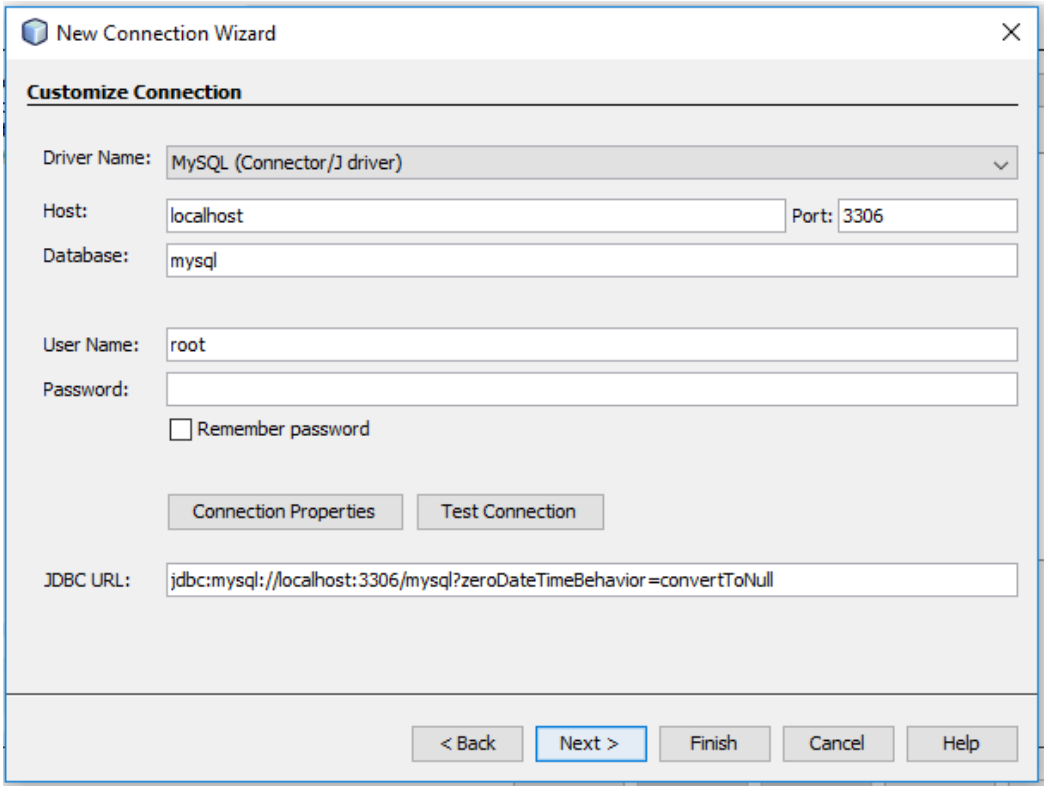
Využitie NetBeans-u nám umožňuje tieto entity vygenerovať automaticky a tým nás ušetriť zdĺhavému písaniu, aj entity v našej aplikácii boli vytvárané takýmto spôsobom. Teraz si tento postup podrobne popíšeme.

Ľavým tlačidlom klikneme na názov projektu, klikneme na možnosť New a potom z možnosti vytvorenia tried vyberieme Entity Classes from Database. Názorný postup je vidieť na obrázku 35. Je možné použiť aj možnosť Other a v časti Persistence vybrať Entity Classes from Database.



Obrázok 35: Pridanie entít pre databázu

V ďalšom kroku je potrebné vybrať data source, podľa ktorého sa načítajú databázové tabuľky. Môžeme zvoliť existujúci alebo vytvoriť nový. Na vytvorenie nového potrebujeme zadať JNDI name, pomocou ktorého budeme môcť ďalej tento data source používať v aplikácii. Ďalej je potrebné pridať databázové spojenie, pomocou ktorého sa pripojíme na databázu. Taktiež môžeme vybrať existujúce alebo vytvoriť nové. Na vytvorenie databázového spojenia je potrebné vyplniť údaje o serveri, na ktorom je databáza uložená. Na obrázku 36 môžete vidieť formulár, ktorý je pri vytvorení databázového pripojenia potrebné vyplniť.



Obrázok 36: Formulár na vytvorenie databázového pripojenia

Keď už máme vytvorené databázové pripojenie aj príslušný data source tak sa nám načítajú všetky tabuľky, ktoré v aplikácií existujú. Je potrebné vybrať tie, s ktorými chceme ďalej pracovať, postupne prejsť procesom vytvárania a nastaviť kam sa majú tieto triedy pridať a rôzne iné základné nastavenia. Pri databáze, ktorá obsahuje rôzne prepojené tabuľky sa automaticky do vytvorenia pridajú aj tabuľky, s ktorými je vybraná tabuľka prepojená. Po vytvorení nám pribudnú samotné entity ale tiež súbor v Configuration Files, persistence.xml. Tu je popísané spojenie s databázou a nastavené potrebné vlastnosti spojenia. Môžete si všimnúť, že pre data source je vyplnená hodnota, ktorú sme zadali ako JNDI name. Ostatné

jeho vlastnosti zase nájdeme v súbore **`glassfish-resources,xml`**. Samozrejme je možné tieto vlastnosti v týchto súboroch napísať aj ručne.

Keď už máme entity reprezentujúce dáta z databázy tak na prácu s nimi potrebujeme ešte EntityManager-a, ktorý nám umožní vykonávať dopyty nad ňou. V našej aplikácii máme pre každú entitu vytvoreného manažéra v triedach s názvomEntityFacade teda napríklad **`HouseholdFacade`**. Tieto triedy sú vytvorené ako EJB Stateless bean-y čo znamená, že sú vždy jednorazove. To znamená, že pre každý dopyt na databázu je vytvorená nová inštancia takejto triedy. Najčastejšie dotazy, ktoré používame sú vytvorenie, úprava a odstránenie dát z databázy. Na všetky tieto úkony sme používali priamo funkcie triedy **`EntityManager`**, persist na vytvorenie, remove na odstránenie a merge na úpravu. Na ostatné dotazy sme využívali možnosť vytvorenia NamedQuery, ktoré sú písane v expression language, viac 1.3.3.

3.4.1 Znenie domácej úlohy

Ako sme už vyššie spomínali táto domáca úloha bude spojená tiež s úlohou o technológii JSF. Presné znenie tejto spojenej úlohy je:

Vašou úlohou je do databázy domacnost vytvoriť novú tabuľku s názvom task_VaseMeno. Tabuľka bude obsahovať tieto stĺpce:

- id(integer) – jedinečný identifikačný údaj
- name(varchar(50)) – názov/popis úlohy
- state(integer) – označuje vybavenosť úlohy (0-nevybavená, 1-vybavená)
- user(integer) – id používateľa, ktorému táto úloha patrí

Následne do aplikácie Domácnosť vytvoríte entitu reprezentujúcu dáta v tabuľke. Vytvoríte tiež stránku, ktorá bude zobrazovať zoznam všetkých úloh prihláseného používateľa. Zoznam bude usporiadaný tak, aby nevybavené úlohy boli na začiatku. Nevybavenú úlohu bude možné odstrániť, editovať(zmeniť názov/popis) a zmeniť stav. Vybavenej úlohe bude možné iba zmeniť stav. Stránka bude obsahovať formulár na vytvorenie nových úloh pre prihláseného používateľa. Na vytvorenie tejto stránky použite

JSF. V časti **`householders.xhtml`** je tlačidlo s názvom List of tasks, ktorému pridáte akciu na zobrazenie vami vytvorenej JSF stránky. Ak na nejakú zaujímavú funkcionalitu využijete Ajax, môžete získať bonusové body.

3.5 Využitie technológie websocket

Hlavnú ukážku využitia websocket-u máme v aplikácii Kalendár kde je využitý na komunikáciu medzi serverom a javascript-ovým klientom. Všetky akcie, ktoré prihlásený používateľ v tejto aplikácii urobí vyvolajú odoslanie požiadavky na server pomocou websocket-u, ktorý po spracovaní prijatých dát odošle späť požadované dáta.

Trieda **`websocket.ServerEndpoint`** slúži ako websocket-ový endpoint, na ktorý sa potom v calendar.js pripájame. Táto vlastnosť triedy je dosiahnutá použitím anotácie @javax.websocket.server.ServerEndpoint a na označenie metód, ktoré zabezpečujú otvorenie alebo ukončenie spojenia či prijímanie správ, používame anotácie @OnOpen, @OnClose, @OnMessage.

Aby bolo jasné čo klient od servera požaduje a tiež server sprostredkúva klientovi zaviedli sme si protokol ich komunikácie. Keď klient požaduje od servera všetky udalosti pre práve zobrazený kalendár prihláseného používateľa tak na server odošle správu vo formáte events;calendarId;userId. Server si z prijatej správy vytiahne id kalendáre a id používateľa. Id používateľa použije na nastavenie práve zobrazeného kalendára prihlásenému používateľovi, tieto údaje udržiava aplikačný bean. Pomocou id kalendára si vyžiada JSON reprezentáciu všetkých udalostí prislúchajúcich tomuto kalendáru, ktorú mu poskytne trieda **`CallendarJSON`**, viac 3.2. Po získaní týchto dát server odošle správu späť klientovi v tvare **`calendarEvents;JSONdata`**. Celý protokol nájdete stručne popísaný v triede reprezentujúcej endpoint.

Websocket využívame aj v aplikácií Domácnosť na dynamické zobrazenie dát získaných z aplikácie Kalendár. Spojenie medzi týmito dvoma aplikáciami je zabezpečené pomocou technológie JMS, ktorej sa budeme venovať v časti 3.6. Po prijatí informácii z aplikácie Kalendár chceme, aby sa dáta hneď zobrazili používateľovi a preto ich server odošle klientovi cez websocket, kde sú následne v javascript-ovom klientovi spracované

a zobrazené používateľovi. To ako server vie kedy a kam má odoslať dáta si budeme popisovať v časti 3.6, teraz si však môžete pozrieť príslušný endpoint, ktorý nájdete v triede `websocket.FindWebsocket` a klientsku časť nájdete v householders.js.

3.5.1 Zadanie domácej úlohy

V triede `ApplicationManager` je mapa `users`, ktorá obsahuje zoznam aktuálne prihlásených používateľov. Pridajte do aplikácie Kalendár možnosť(formulár na stránke `calendars.xhtml`) vytvorenia novej udalosti v kalendári, ktorá na server pošle informácie o udalosti(dátum, názov, čas, dĺžka trvania v hodinách),server udalosť pridá do databázy a následne pošle klientovi správu calendarEvents, v ktorej sú všetky udalosti kalendára vrátane práve vytvorenej udalosti, ale okrem toho klienta, ktorý udalosť vytvoril, sa kalendár pošle všetkým ostatným klientom(používateľom), ktorí sú prihlásený a majú práve zobrazený daný kalendár. Aplikácia je navrhnutá tak, že sa predpokladá, že každý používateľ je prihlásený iba raz.

To znamená, že stačí pridať nový typ správy klient -> server o pridanej udalosti, protokol je dokumentovaný(zoznam správ) priamo v kóde v triede `websocket.ServerEndpoint`.

Na pridanie udalosti využite metódu `DataQuery.addEvent()` a následne na priradenie udalosti kalendáru `DataQuery.addCalendarEvent()`.

3.6 Využitie JMS

Technológiu JMS sme v našej aplikácii využili na komunikáciu aplikácie Kalendár s aplikáciou Domácnosť. V aplikácii Domácnosť sa nachádza možnosť na vyhľadanie voľného časového úseku pre všetkých členov systému. Účelom tejto funkcie je umožniť používateľom naplánovať spoločnú akciu pre všetkých členov domácnosti v čase, kedy majú všetci voľno.

Dáta o udalostiach sú spravované aplikáciou Kalendár, na ktorú sa v tomto prípade obracia aplikácia Domácnosť s požiadavkou na nájdenie takéhoto voľného času pre všetkých používateľov. V prípade, že nájdu interval, ktorý im bude vyhovovať môžu následne poslať

požiadavku na vytvorenie udalosti. Takúto komunikáciu medzi dvoma nezávislými aplikáciami sme zabezpečili pomocou technológie JMS.

Výpočet teda realizuje aplikácia Kalendár, ktorú môžeme považovať za výpočtový server. Zaregistruje sa na odoberanie správ doručených do JMS schránky s JNDI menom jms/topic. **`CalendarDomacnost.jms.FindReader`** je takzvaný MessageDriven bean, ktorý prijíma správy. Správa obsahuje informácie v akom časovom horizonte majú mať voľno, aké dlhé by malo byť a ktorých používateľov sa to týka, alebo kedy a o koľkej má udalosť prebiehať, ako dlho bude trvať, ako sa má volať a pre ktorých používateľov má byť pridaná. Po prijatí správy zavolá jednu z metód triedy **`com.controllers.ApplicationManager`**, findFreeTime, ktorá podľa vstupných parametrov vyhľadá časový úsek, v ktorom majú všetci členovia domácnosti voľno alebo **`addJoinEvent`**, ktorá vytvorí udalosť podľa zadaných vstupov.

Aplikácia Domácnosť zase umožňuje používateľom odosielať výpočtové úlohy na server. Odosielanie požiadaviek je ako akcia na stlačenie tlačidla. Túto akciu nájdeme v **`jsf.HouseholdController`** vo funkciách **`findFreeTime`** a **`createJoinEvent`**. Do kódu sme nad každú z týchto funkcií pridali komentár, ktorý popisuje tvar odosielanej správy. Na získanie výsledkov zo servera používame, podobne ako v aplikácií Kalendár, MessageDriven bean, jms.ResultReader. V časti jsf.ApplicationManager máme uložený zoznam referencií na otvorené websocket-ové spojenia s id používateľa, ktorý odoslal požiadavku. Po doručení správy zo servera z nej vytiahneme id používateľa, ktoré používame na jednoznačné určenie koho je potrebné pomocou websocket-u notifikovať, alebo komu treba preposlať prijatú správu. Podľa tohto id pohľadáme v mape referenciu na websocket, ktorý bol vytvorený v tej session, z ktorej správa odišla. Tento websocket následne požiadame, aby odoslal odpoveď, ktorú potom odchytí javascript-ový websocket-ový onMessage handler, household.js, ktorý bol zaregistrovaný po vytvorení websocketu. Tento handler následne spracuje správu a dynamicky upraví informácie zobrazené na stránke.

3.6.1 Zadanie domácej úlohy

V aplikácii Domácnosť sa JMS využíva na komunikáciu s aplikáciou Kalendár. Pri plánovaní spoločnej akcie členov domácnosti sa aplikácia Domácnosť pomocou správy cez JMS pýta aplikácie Kalendár na zoznam voľných časových úsekov daných používateľov v stanovenom časovom období. Pomocou ďalšej JMS správy sa Kalendáru pošle požiadavka na vytvorenie novej udalosti pre všetkých používateľov, ktorí sa udalosti zúčastnia. V tejto úlohe treba rozšíriť protokol JMS, aby poskytoval ďalšiu funkcionalitu.

Na stránku List of tasks vytvorenej v úlohe o JSF a JPA by sa mala dať zvoliť jedna úloha, z ktorej sa má stať udalosť v kalendári. Pridajte formulár, kde sa bude dať stanoviť dátum, čas začiatku plnenia úlohy a časová dĺžka v hodinách, ako dlho sa má daná úloha plniť. Pomocou tlačidla sa aktivuje odoslanie správy na server, odkiaľ správa putuje ďalej do druhej aplikácie cez JMS. Po vybavení požiadavky sa zobrazí správa o tom, či pridanie udalosti do kalendára prebehlo úspešne. Táto správa musí pochádzať až z aplikácie Kalendár. Či použijete websocket alebo JSF tu nezáleží.

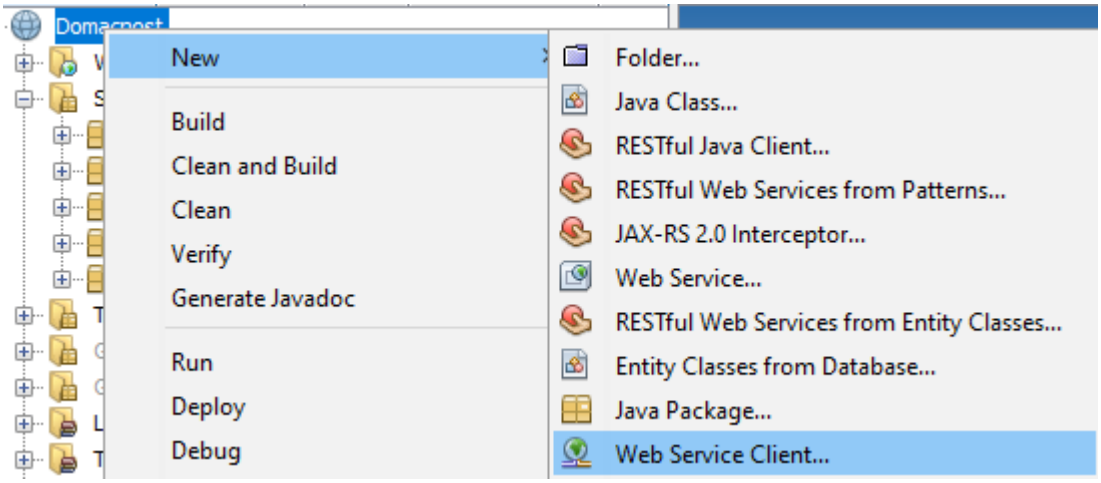
3.7 JAX-WS

Ako ďalšiu technológiu na prepojenie dvoch aplikácií študentom ukážeme webový servis s protokolom SOAP. Táto technológia nám umožní navzájom volať funkcie dvoch aplikácií, ktoré ani nemusia byť naprogramované v rovnakom jazyku a dokonca ani bežať na rovnakom serveri. Systém domácnosť sme rozšírili o ďalšiu aplikáciu **`PrepocetWS`**, ktorá slúži ako výpočtový server, od ktorého aplikácia Domácnosť požaduje výpočet. V aplikácii domácnosť na stránke, ktorá je venovaná receptom, nájdeme možnosť s prepočtom množstva ingrediencií podľa počtu porcií. Každý recept má jednoznačne určené na koľko porcií je určený no v prípade, že nám počet porcií nevyhovuje môžeme si dať ingrediencie prepočítať na nami zadaný počet porcií.

Aplikáciu **`PrepocetWS`** sme vytvorili ako bežnú webovú aplikáciu, ktorá obsahuje triedu **`server.Prepocet`** s anotáciou @WebService a public metódu **`prepocet`** s anotáciou @WebMethod, ktoré sú automaticky prístupné body pre všetkých klientov, ktorí chcú službu využiť. Automaticky sa vygeneroval XML súbor WSDL (Web Services Definition

Language), v ktorom je táto služba popísaná. Keď sme urobili deploy tejto aplikácie na server tak sa sprístupnila na príslušnej URL. URL prislúchajúcu aplikácii, ktorá ponúka webový servis, vieme zistiť z administratívneho rozhrania pre Glassfish, kde si v zozname aplikácií klikneme na príslušnú aplikáciu a potom ju nájdeme vo View Endpoint.

Do aplikácií Domácnosť sme následne pridali referenciu na túto webovú službu pomocou podpory, ktorú ponúka NetBeans. Popíšeme si tento postup. Je potrebné pravým tlačidlom kliknúť na názov projektu, ďalej kliknúť na možnosť New a v ponuke vybrať Web Service Client, tak ako vidíme na obrázku 37. Môžeme tiež použiť možnosť Other a v časti Web Services vybrať možnosť Web Service Client.



Obrázok 37: Vytvorenie referencie na webový servis

Po zobrazení formulára je potrebné zvoliť možnosť pomocou, ktorej túto referenciu vytvoríme. Na výber máme možnosti vybrať projekt, zdrojový súbor alebo WSDL URL. V našom prípade sme požili možnosť projekt. Po výbere a stlačení tlačidla Finish sa v aplikácii v časti Web Service References pridala referencia na náš webový servis a po dvojkliku na jeho referenciu sa nám otvorí XML reprezentácia tejto webovej služby. Taktiež sa vytvorila trieda **`webservice.client.Prepocet_Service`** s metódou **`getPrepocetPort`**, ktorá vždy vráti referenciu na analogickú triedu **`Prepocet`**.

V triede **`jsf.RecipeController`** je metóda **`calculateIngredient`**, ktorá sa zavolá po kliknutí na tlačidlo v stránke **`recipe.xhtml`**. Táto metóda zabezpečuje volanie metódy z webového servisu.

3.7.1 Zadanie domácej úlohy

Recept pozostáva z názvu, popisu a zoznamu potrebných prísad, čiže dvojíc názov prísady, potrebné množstvo. Prísady pre príslušný recept sú uložené v tabuľke **`ingredient`** v databáze, každý záznam v tabuľke má odkaz na recept, kde je použitý a id produktu, aby bolo jednoznačné určenie prísad k produktom, ktoré sa objavujú aj v nákupných zoznamoch.

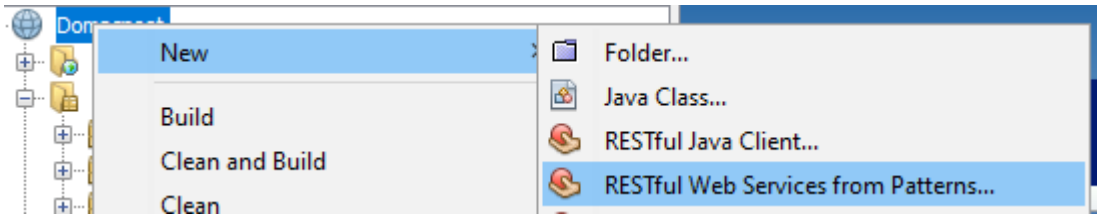
Aplikácia umožňuje evidovať a vytvárať nové recepty i nákupné zoznamy. V aplikácii sa dajú vytvárať, editovať, či rušiť aj nákupné zoznamy, čiže zoznamy produktov aj s určením množstva, ktoré sa majú nakupovať. K nákupnému zoznamu sa eviduje odkedy dokedy sa majú produkty nakúpiť a jednotlivo sa dá označovať, ktoré už sú nakúpené.

Na stránku **`recipes.xhtml`** pridajte tlačidlo, ktoré umožní z prísad nejakého zobrazeného receptu automaticky vytvoriť nákupný zoznam pomocou webového servisu, ktorý vytvoríte v samostatnej aplikácii. Teda váš kód v aplikácii Domácnosť sa obráti na vašu novú webovú aplikáciu s webovým servisom, ktorý vytvorí potrebný záznam v databáze, servis oznámi, či operácia prebehla úspešne (aplikácia správu následne zobrazí). Webovému servisu sa posiela id receptu a počet porcií. Webový servis sa obráti na už existujúci webový servis **`PrepocetWS`**, ktorý prepočíta zoznam produktov pre daný recept podľa počtu porcií. Webový servis musí byť vytvorený pomocou technológie JAX-WS.

3.8 JAX-RS

Študentom sme už predstavili webové služby s protokolom SOAP a teraz im chceme predstaviť RESTful webové služby. V aplikácii Domácnosť umožňujeme používateľovi vytvoriť udalosť do kalendára pomocou volania webovej služby, ktorá je definovaná v aplikácií Kalendár. Ide o udalosť reprezentujúcu nakupovanie položiek príslušného nákupného zoznamu, pre ktorý sme túto udalosť pridali.

Do aplikácie Kalendár sme pridali webový servis pomocou kliknutia pravým tlačidlom na názov projektu, klikli na možnosť New a ďalej zvolili možnosť RESTful Webservices from Patterns, ako môžete vidieť na obrázku 38. To isté dosiahneme keď zvolíme možnosť Other a v časti Web Services zvolíme RESTful Webservices from Patterns.



Obrázok 38: Pridanie RESTfull webového servisu

Takto sme pridali triedu **`service.CreateEventResource`**, ktorá slúži ako webový servis, a automaticky sa nám vygenerovala aj trieda **`service.ApplicationConfig`**, ktorá má použitú anotáciu @javax.ws.rs.ApplicationPath. Do triedy **`service.CreateEventResource`** sme pridali metódu create, ktorá vytvorí novú udalosť do kalendára podľa vstupných údajov. Na túto metódu sme použili anotácie @GET a @Path("{name}/{date}/{time}/{length}/{userId}"), ktoré nám definujú URL, na ktorom bude dostupná pre klientov a tiež akou metódou budú získavané hodnoty parametrov v nej. Každý parameter metódy má anotáciu @PathParam("name"), ktorá nám jednoznačne určuje, ktorú hodnotu z URL obsahuje. Po prevedení deploy na túto aplikáciu môžeme prejsť na použitie u klienta.

V aplikácií Domácnosť sme v **`shoppingLists.xhtml`** pridali formulár, v ktorom je možné vyplniť všetky údaje o tom, na ktorý deň a čas má byť udalosť o nákupe vytvorená. Po odoslaní formulára sa zavolá metóda **`createShoppingEvent`** v triede **`jsf.ShoppingListsController`**. Pomocou **`javax.ws.rs.client.ClientBuilder.newClient()`** vytvoríme novú inštanciu triedy javax.ws.rs.client.Client, ktorej následne nastavíme URL a ostatné parametre, ktoré hovoria o webovom servise, ktorý sa má použiť.

3.8.1 Zadanie domácej úlohy

Vašou úlohou je aplikáciu Domácnosť rozšíriť o webový servis poskytujúci informácie o nákupnom zozname. Vstupom preň bude názov nákupného zoznamu a výstupom bude zoznam položiek s prislúchajúcim množstvom pre nákupný zoznam určený zadaným menom nákupného zoznamu, ktoré je jedinečné pre každý nákupný zoznam. Webový servis musí byť vytvorený pomocou technológie JAX-RS.

V aplikácii Kalendár je v triede `com.controllers.ApplicationManager` vytvorená prázdna metóda `showShoppingListDetail`. Vašou úlohou je do tejto metódy napísať zdrojový kód, ktorý zavolá webový servis, ktorý ste pridali do aplikácie Domácnosť. Prijaté dáta je následne potrebné nejakým spôsobom zobraziť používateľovi, spôsob zobrazenia je len a len na vás.

3.9 Framework Spring

Na záver kurzu chceme študentom ukázať aj iné možnosti vývoja webových aplikácii pomocou programovacieho jazyka Java. Na ukážku sme si vybrali framework Spring, o ktorom sme pripravili prezentáciu so základnými informáciami a ukážkami zdrojového kódu, na ktorých demonštrujeme použitie niektorých technológií, ktorými sme sa v práci zaoberali, vo frameworku Spring. Vytvorená prezentácia je priložená na CD, ktoré je súčasťou príloh tejto práce, jej názov je Spring.pdf.

Po teoretickom úvode si rozoberieme ukážkovú aplikáciu s využitím websocketu, ktorú môžete nájsť na oficiálnej webovej stránke frameworku Spring na adrese http://spring.io/guides/gs/messaging-stomp-websocket/.

3.9.1 Domáca úloha

Pre študentov, ktorých zaujala možnosť programovania vo frameworku Spring sme pripravili aj zadanie bonusovej domácej úlohy.

Naprogramujte jednoduchú chat webovú aplikáciu pre viacerých používateľov pomocou frameworku Spring. Odporúčame využiť komunikačný protokol STOMP s Websocketmi podľa príkladu: spring.io/guides/gs/messaging-stomp-websocket/

3.10 Framework Vaadin

Rovnako ako pre framework Spring sme pripravili prezentáciu aj o Vaadin-e. Pripravená prezentácia študentom objasní rozdiel vo vývoji používateľského rozhrania využitím tohto frameworku. Prezentácia je rovnako priložená na CD a jej názov je Vaadin.pdf.

4 Testovanie a výsledky

V tejto kapitole si povieme ako a kedy prebiehalo testovanie aplikácie. Pozrieme sa na reakcie študentov, ktorí boli tohoto testovania zúčastnení a zhodnotíme aké prínosy malo testovanie pre nás.

Počas posledného semestra sme vytvorený systém Domácnosť už aplikovali do výučby kurzu, pre ktorý je určený. Na začiatku nás sprevádzali rôzne problémy spôsobené spúšťaním systému na databáze na serveri, ktoré boli spôsobené vývojom na lokálnej databáze a tiež rozdielnosťou systému, nad ktorým databáza pracuje. Postupne sme problémy odstránili a postupovali podľa plánu, v ktorom sme si určili, ktoré technológie budú kedy preberané. Jednotlivé hodiny na seba nadväzovali a študentom sme predstavovali vždy rozšírenejšiu aplikáciu. Týmto sme si overili, že štúdia podľa ktorej sme zvolili poradie výučby, bola správna.

Počas hodín, na ktorých bola aplikácia použitá študenti mohli sledovať jej použitie a tiež sme sa venovali jednotlivým prislúchajúcim častiam zdrojového kódu. Na konci každej časti kurzu, ktorého súčasťou bola jedna z technológií spracovaných v tejto práci, bola študentom zadaná domáca úloha. Aby sme získali rady ako a čo ešte vylepšiť požiadali sme pravidelne chodiacich študentov o spísanie krátkej spätnej väzby, ktorú sme následné zanalyzovali. Za nedostatky študenti považujú využívanie prostredie NetBeans pri práci so systémom a tiež jeho rôzne problémy pri spúšťaní. Na druhej strane však oceňujú túto myšlienku vytvorenia aplikácie, ktorá bude aj základom pre domáce úlohy. Veria, že tak budú môcť cielene precvičiť iba danú problematiku a nemusia riešiť konštrukciu aplikácie, ktorá nie je súčasťou precvičovanej látky.

Tieto názory študentov sme samozrejme zobrali do úvahy a ako reakciu sme vytvorili manuál, podľa ktorého je potrebné aplikáciu spúšťať. Taktiež sa vyskytla požiadavka na využitie iného programovacieho prostredia ako je NetBeans, no táto požiadavka je už na zvážení vyučujúceho. NetBeans bol zvolený z dôvodu jeho pravidelného využívania v rámci kurzu a tiež z dôvodu, že NetBeans a Glassfish je tzv. referenčná implementácia. Odhliadnuc od týchto nedostatkov študenti ocenili zjednodušenie precvičovania jednotlivých technológií čo bolo jednou z hlavných motivácii pre túto prácu.

V neposlednom rade je potrebné aby s aplikáciou bol spokojný aj samotný vyučujúci, a preto sme aj jeho požiadali o spätnú väzbu. Jeho reakcia na prácu so systémom:

> *„Predmet Java EE sa kryštalizoval niekoľko rokov do ako-tak ucelenej sady jednoduchých ukážkových projektov pre jednotlivé technológie Java EE. Napriek tomu som nebol spokojný s tým, že v jednotlivých projektoch nebolo ľahké demonštrovať previazanie týchto technológií do väčších celkov a aplikácií, pričom práve to je hlavnou silou celej technológie Java EE. Naviac, cítil som, že študenti by privítali aj iné príklady, na ktorých by mohli vidieť širšie spektrum rôznych use-case využitia technológií. Systém, ktorý vytvorila vo svojej práci Lívia Kupčuliaková jednoznačne obohacuje zdroje, ktoré mám k dispozícii na demonštrovanie technológií vyučovaných v predmete. Naviac to vytvára lepšie scenáre pre domáce zadania, kde študenti nie sú nútení kvôli každej domácej úlohe stavať celý nový projekt, ale len dopĺňajú vlastnosti relevantné pre príslušnú technológiu.*
>
> *V priebehu letného semestra 2017/2018 sme systém aktívne využívali jednak na hodine na ukážky technológií a jednak na formulovanie domácich zadaní pre študentov, výrazne mi uľahčil prácu a som diplomantke vďačný za veľmi pružnú spoluprácu a spoľahlivé plnenie všetkých dohodnutých úloh. Predpokladám využitie systému aj v budúcom akademickom roku.“*

Možnosť otestovať využitie práce už počas jej tvorby bola pre nás veľmi prínosná. Narazili sme na niekoľké nedostatky, ktoré sme mohli následne odstrániť a tak zabezpečiť jednoduchšie nasadenie použitia systému v ďalších rokoch.

5 Záver

Cieľom diplomovej práce bolo vytvoriť rozsiahlejšiu JavaEE aplikáciu využívajúcu hlavne technológie, ktoré sa vyučujú na kurze, pre ktorý je práca určená. Analyzovali sme spôsob výučby JavaEE technológií v rámci iných kurzov, z ktorých sme sa inšpirovali hlavne poradím, v ktorom jednotlivé technológie vyučovať. Hlavnou úlohou systému Domácnosť je demonštrovať využitie JavaEE technológií a ponúknuť tak možnosť sústrediť sa priamo na jednu z nich.

Aplikácie sú naprogramované podľa návrhu. V aplikácii Kalendár je možné vytvoriť nový kalendár, zobrazovať udalosti kalendára a importovať buď celé kalendáre aj s udalosťami, alebo len udalosti. Aplikácia tiež ponúka možnosť vyhľadávania voľného časové úseku, ktorý majú spoločný viacerí používatelia, vytváranie udalosti podľa vstupných dát. Tieto funkcionality aplikácia sprostredkováva pre aplikáciu Domácnosť, z ktorej sú volané webovými službami alebo pomocou JMS správ. Aplikácia Domácnosť umožňuje vytváranie a mazanie domácnosti, spravovanie jej členov, receptov a nákupných zoznamov. Obe aplikácie obsahujú aj časti, ktoré sú pripravené na rozšírenie študentami pri tvorbe domácich úloh. Aplikácia Prepočet zabezpečuje webový servis prepočítavajúci potrebné množstvo ingrediencií receptu podľa počtu porcií.

Myslíme si, že práca môže byť pre potreby kurzu, pre ktorý bola tvorená, prínosná. Aplikácie tvoria ukážku JavaEE aplikácií a ponúkajú možnosti rozširovania funkcionality podľa potrieb. Veríme, že práve možnosť stále ju rozširovať bude viesť k jej využitiu aj v ďalších akademických rokoch. V budúcnosti by bolo prínosné ju rozšíriť nie len o novú funkcionalitu, ale hlavne o využitie ďalších JavaEE technológií.

Téma tejto práce bola pre nás zaujímavá a jej implementácia veľmi prínosná. Dúfame, že aplikácie budú študentom zjednodušovať učenie sa JavaEE technológií a nie im ho komplikovať. Taktiež veríme, že vyučujúci bude so systémom spokojný a bude ho chcieť využívať a rozširovať čo najdlhšie.

6 Literatúra

[1] JENDROCK, Eric, et al. *The java ee 7 tutorial: Volume 1*. Addison-Wesley Professional, 2014, ISBN 978-0321994929

[2] JENDROCK, Eric, et al. *The java ee 7 tutorial: Volume 2*. Addison-Wesley Professional, 2014, ISBN 978-0321980083

[3] ORACLE CORPORATION AND/OR ITS AFFILIATES. Message-Oriented Middleware [online] 2010. [01.04.2018] Dostupné na docs.oracle.com/cd/E19316-01/820-6424/index.html

[4] YULI, Vasiliev. *Beginning Database-Driven Application Development in Java™ EE: Using GlassFish™*. Apress, 2008, ISBN 978-1-4302-0963-8

[5] Google Web Toolkit [online] [01.04.2018] Dostupné na www.gwtproject.org/overview.html

[6] GUPTA, Arun. *Java EE 7 essentials*. O'Reilly Media, 2013, ISBN 978-1-449-37017-6

[7] JOHNSON, Rod, et al. *The spring framework–reference documentation*. Interface 21, 2004

[8] PIVOTAL SOFTWARE. Spring guides [online] 2018. [2.5.2018] Dostupné na spring.io/guides#gs

[9] SHAN, Tony C., and WINNIE W. Hua. Taxonomy of java web application frameworks. In: *E-Business Engineering, 2006*. ICEBE'06. IEEE International Conference on (pp. 378-385). IEEE

[10] VAADIN LTD. Vaadin tutorial [online] 2018. [01.04.2018] Dostupné na https://vaadin.com/docs/v10

[11] JAMES STAFFORD. Enterprise Conputing with Java Course Homepage [online] 2014. [01.04.2018] Dostupné na apps.ep.jhu.edu/course-homepages/2764-605.784-enterprise-computing-with-java-stafford

[12] UNIVERSITY OF SOUTH FLORIDA. Systems Programming: Java EE [online] 2012. [01.04.2018] Dostupné na ugs.usf.edu/pdf/courses/0708/COP3601-01-02-2011.pdf

[13] GOPAS-IT školiace stredisko, Bratislava. Úvod do vývoja Java EE [online] 2018. [01.04.2018] Dostupné na www.gopas.sk/Kurzy/Katalog-kurzov/Programovanie/Nove-kurzy-programovania/Uvod-do-vyvoja-Java-EE-JJ2EE1.aspx?subpage=description

[14] LONDON METROPOLITAN UNIVERSITY. Oracle Certification Programme (OCP) - Java Enterprise Component Development (short course) [online]. [01.04.2018] Dostupné na www.londonmet.ac.uk/courses/short/oracle-java-enterprise-component-development/

7 Prílohy

Prílohou práce sú zdrojové kódy aplikácií, ukážkové súbory potrebné pre import, prezentácie určené na výuku o framework-ch Spring a Vaadin vo formáte .pdf, kompletná práca vo formáte .pdf a tiež manuál na spustenie aplikácií. Všetko je uložené na kompaktnom disku.