Универзитет у Београду

Математички факултет

Мастер рад

Реализација веб-апликације за креирање распореда часова употребом RichFaces и EJB окружења

Студент:

Сања Петровић

Ментор:

др Филип Марић

Београд,

септембар 2017.

Ментор:

др Филип Марић

Математички факултет

Универзитет у Београду

Чланови комисије:

др Душан Тошић

Математички факултет

Универзитет у Београду

др Владимир Филиповић

Математички факултет

Универзитет у Београду

Задатак мастер рада

Циљ овог рада је да представи употребу савремених *Java* веб-технологија које укључују *JSF (JavaServer Faces), RichFaces* и *EJB (Enterprise JavaBeans)*, а које омогућавају брзо и поједностављено креирање веб-апликација. Такође, задатак рада је и имплементација веб-апликације коришћењем наведених технологија, која би се користила за ручно (или аутоматско) креирање распореда часова.

Први део апликације служи за прикупљање и складиштење неопходних података (нпр. поделе наставе, жеља наставног особља) од стране корисника за израду распореда часова. Други део апликације врши приказивање готовог распореда, проверу свих услова његове коректности, а администратору пружа могућност удобне модификације израђеног распореда.

Независно од саме апликације, у позадини се користи готов програм који решава проблем креирања распореда часова на основу прикупљених података. Апликација овом програму као улаз прослеђује одговарајућу датотеку, а излаз парсира у формат неопходан за приказивање распореда кориснику.

Детаљан опис наведених технологија, које се користе у раду, представља добар основ за учење свима који се раније нису сусретали са њима, а сама апликација приказује њихову практичну примену, која је данас све шира.

Садржај

1 Увод

Софтвер за креирање распореда часова састоји се од неколико независних компоненти: графички кориснички интерфејс за спецификовање распореда, систем *ArgoTimetabling* за аутоматско распоређивање и компоненте за приказ распореда.

Систем *ArgoTimetabling* развијен је на Математичком факултету, Универзитета у Београду, пре почетка рада на овом мастер раду.

Графички кориснички интерфејс (енг. *Graphic User Interface, GUI*) којим корисник задаје све релевантне информације за креирање распореда часова биће израђен употребом *Java* веб-технологија. Ово подразумева коришћење оквира *JSF (JavaServer Faces)* и *RichFaces* за креирање веб-страница, односно клијентског дела, док ће се спецификација *EJB (Enterprise JavaBeans)* користити за дефинисање модела серверских компоненти, у комбинацији са спецификацијом *JPA (Java Persistence API)* за управљање подацима. Резултат интеракције корисника са овом компонентом је текстуална датотека која садржи спецификацију распореда часова у формату **.tts* (*time-table specification*) погодном за аутоматски систем распоређивања *ArgoTimetabling*. Овај формат дефинисан је у склопу развоја *ArgoTimetabling* система. Кориснички интефејс је дизајниран у складу са форматом **.tts*, али у неким случајевима је проширен како би олакшао посао кориснику.

Резултат рада система *ArgoTimetabling* је текстуална датотека са распоредом у формату **.ttbl (timetable)*. На основу ове датотеке компонента за приказ распореда парсира добијене информације, складишти их у базу података и генерише документе распореда појединачно за све сале, наставнике или групе. Такође, могућ је преглед свих претходно креираних распореда на основу школске године и семестра.

2 Технологије

У овом поглављу биће објашњене технологије које су коришћене за израду корисничког интерфејса софтвера за аутоматско распоређивање часова.

2.1 *JSF (Java Server Faces)*

Сервлети су једна од технологија за генерисање динамичких веб-садржаја, тачније сервлет је *Java* класа која наслеђује стандардну класу *HttpServlet*. Основна мана употреба сервлет технологија за динамичко генерисање веб-страница је та што је генерисање текста смештено унутар сервлет класе. Тако тешко можемо раздвојити посао веб-дизајнера и програмера: обе групе аутора морају да модификују исте датотеке са *Java* изворним кодом. Ипак, концепт сервлета има добрих особина, на пример, преносивост на различите рачунарске платформе и веб-сервере, ефикасно извршавање, итд.

Идеја која се налази у основи технологије *Java Server Pages (JSP)*, а на којој је базиран и оквир *Java Server Faces (JSF)*, је да се омогући паралелан, или бар независан, рад дизајнера и програмера. *JSP* странице су текстуалне датотеке које садрже *HTML* документе. При томе се за писање таквих докумената могу користити стандардни *HTML* елементи, али и посебни тагови који омогућавају додавање динамичких елемената у страницу. Овако написана страница ће послужити као основ за динамичко генерисање сервлета који ће произвести управо онакву *HTML* страницу како је то специфицирано у *JSP* страници.

Дакле, *JSF* представља оквир (енг. *framework*) намењен веб-апликацијама базираним на *Java* технологији. *JSF* поједностављује развој корисничког интерфејса веб-апликација дефинисањем модела компоненти и његов се кôд извршава на серверској страни. Међутим, ово не значи да није могуће укључити кôд који се извршава на клијентској страни. Шта више, *JSF* компоненте могу садржати *JavaScript* кôд и *JSF* такође има подршку за *AJAX*. Највећа предност овог оквира је та што је базиран на компонентама, што омогућава вишеструку употребу кода који је независан. Под компонентама се подразумева било који део веб-апликације, могуће је да буде једноставан и представља само компоненту за унос текста, док може бити врло комплексан и садржати комплетну табелу са опцијама за сортирање података и пагинацију.

JSF долази са одређеним бројем већ развијених компоненти, као што су компоненте за унос и приказ текста, компоненте за приказ слика, падајуће листе, радио дугмад итд.

Пример компоненте за приказ дугмета:

```
<h:commandButton action="#{departmentController.addDepartment()}"
        value="#{msg['element.save']}" styleClass="btn btn-md btn-primary">
        <f:ajax event="click" execute="@form" render="@form" />
</h:commandButton>
```

У претходном примеру атрибут *action* дефинише методу која ће бити извршена након клика на дугме. Атрибут *value* садржи текст који ће бити исписан на дугмету, док атрибут *styleClass* дефинише *CSS* класе које се односе на дугме. Елемент *f:ajax* биће објашњен касније у склопу *AJAX* функционалности.

Пример компоненте за представљање радио дугмади:

```
<h:selectOneRadio id="userType" required="true" value="#{usersController.newUser.type}">
        <f:selectItem id="profesor" itemLabel="#{msg['user.profesor']}" itemValue="1" />
        <f:selectItem id="admin" itemLabel="#{msg['user.administrator']}" itemValue="2" />
</h:selectOneRadio>
```

Пример компоненте падајуће листе:

```
<h:selectOneMenu id="timetableTeachers" value="#{timetableController.selectedTeacher}"
        converter="#{teacherConverter}" styleClass="col-sm-10 margin10">
        <f:selectItem itemLabel="#{msg['select.teacher']}" itemValue="#{null}" />
        <f:selectItems value="#{timetableController.teacherList}" var="teacher"
                itemLabel="#{teacher.toString()}" itemValue="#{teacher}" />
</h:selectOneMenu>
```

У претходна два примера за дефинисање елемената унутар листе, тј. групе, коришћене су компонента *f:selectItem,* за дефинисање једног елемента и компонента *f:selectItems* за дефинисање листе елемената. Ове компоненте могу бити угњеждене у било коју другу компоненту која омогућава одабир једног или више елемената.

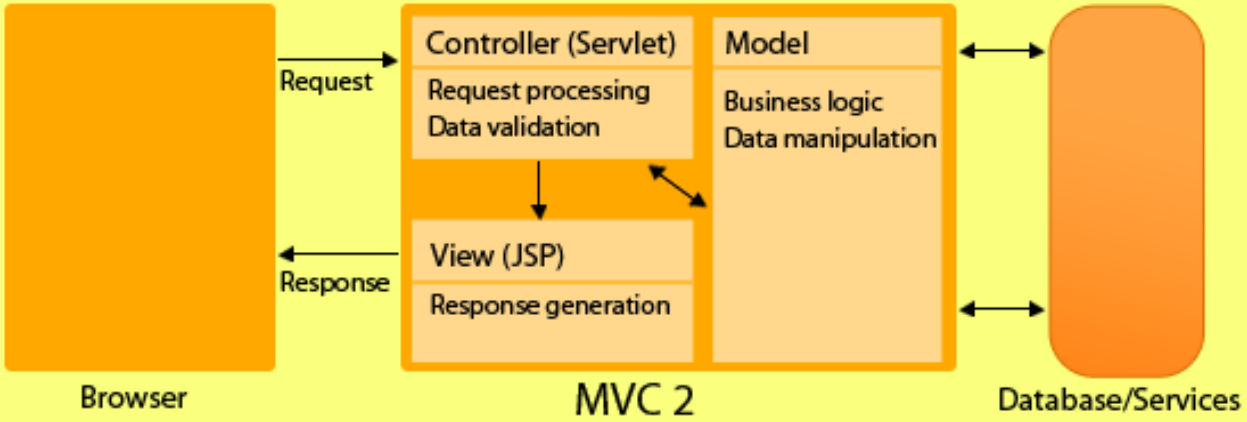
Такође су доступне и библиотеке које је могуће укључити са великим бројем већ спремних компоненти. Једна таква је библиотека *RichFaces* о којој ће бити више речи у даљем тексту.

2.1.1 Основне карактеристике

С обзиром на то да *JSF* представља апстракцију постојећег оквира који се користи за изградњу веб-апликација базираних на архитектури *MVC2 (Model-View-Controller, Model 2)*, кључне компоненте тог приступа су:

1. **Компонента модела** моделира податке и процесе. Она обавља интеракцију са базом података и извршава операције над подацима. Исправно дизајниран модел раздваја податке и процесе од презентационог слоја. Улогу модела у овом случају обављају компоненте које се називају *Java* зрна (енг. *JavaBeans*). *JSP* омогућава једноставно преузимање параметара странице, вредности унетих од стране корисника, и њихово смештање у својства одговарајућег *Java* зрна.
2. **Презентациони слој** (енг. *View*) визуализује податке из модела. За ову компоненту није битно одакле су дошли подаци. Презентацију обављају *JSP* странице.

3. **Компонента контролер** управља презентационим слојем и моделом. Сви захтеви корисника иду преко ове компоненте, а она одређује који сегмент апликације ће бити приказан. Контролер је реализован преко сервлета. Ова компонента представља главну разлику у односу на претходни модел *(MVC, Model 1)* који није имао јасно издвојен контролер за управљање захтевима.



Слика 1: Архитектура *MVC2* [7]

Ова архитектура подразумева да сви захтеви (на пример, клик на дугме) углавном воде на сервлет који одлучује коју ће *JSP* страницу да прикаже кориснику. Иако *JSF* укључује две библиотеке тагова за представу компоненти на *JSP* страници, *JSP* није обавезна презентациона технологија. Такође, *JSF* омогућава креирање сопствених нових компоненти директном употребом постојећих, као и могућност генерисања излаза за више типова прегледача (енг. *browser*-а).

2.1.2 Животни циклус *JSF* апликације

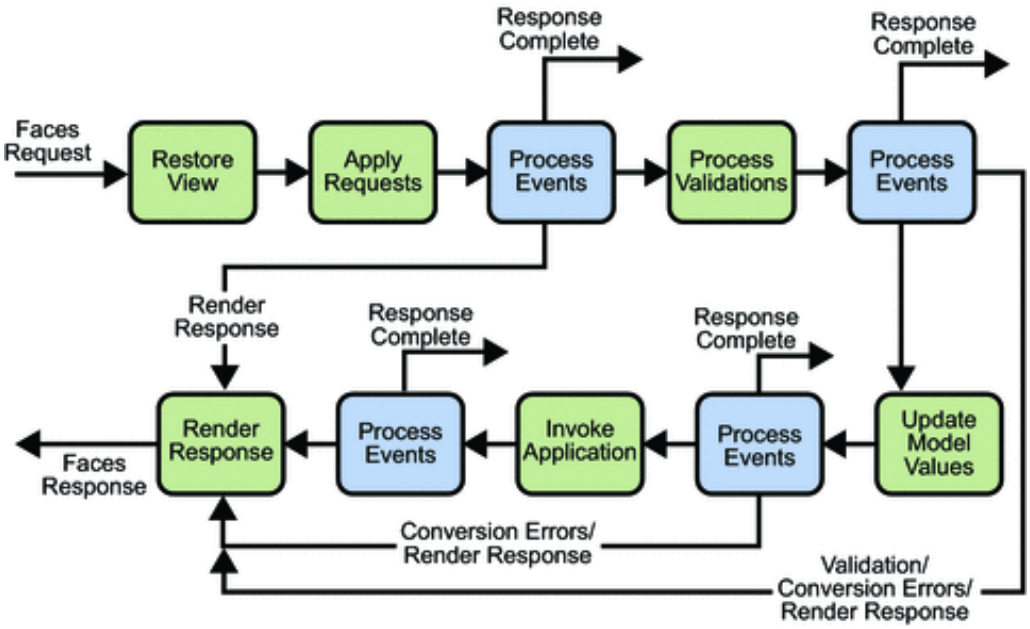
JSF дефинише апстрактну класу *javax.faces.context.FacesContext* за представљање свих контекстуалних информација везаних за обраду долазног захтева и креирање одговарајућег одговора.

Сваки пут када корисник кликне на линк или кликне на дугме *submit*, генерише се нови захтев и започиње животни циклус обраде захтева. *JSF* спецификација дефинише шест различитих фаза животног циклуса и њихов редослед извршавања током обраде захтева:

1. **Обнављање погледа** (енг. *Restore view*) – Прва фаза животног циклуса наступа одмах након што *JSF* прими захтев. Уколико се страница први пут приказује, конструише се ново стабло компоненти и чува у оквиру *FacesContext* инстанце. Уколико је страница већ приказана, преузима се постојеће стабло компоненти,

такође из *FacesContext* инстанце. Ако захтев не садржи улазне податке, аутоматски се прелази на завшну фазу, односно пружање одговора. До овога долази када се страница први пут приказује. Иначе, *JSF* имплементација прелази на другу фазу животног циклуса.

2. **Примена вредности захтева** (енг. *Apply request values*) – У овој фази свака компонента из стабла компоненти преузима одговарајућу вредност из захтева и чува је. Сачуване вредности у компонентама називају се локалне вредности.
3. **Валидација процеса** (енг. *Process Validation*) – Током дизајнирања *JSF* странице за компоненте се могу везати одговарајући валидатори и конвертори. На пример, валидација да ли је нека вредност унета у одговарајућем формату или конверзија ниске у датум одговарајућег формата. У овој фази долази до извршавања свих валидација и конверзија над локалним вредностима. Уколико се све валидације успешно заврше, *JSF* имплементација прелази на четврту фазу. Уколико дође до грешке, она се чува у оквиру *FacesContext* инстанце и одмах се прелази на фазу задужену за пружање одговора, где се врши поновно исцртавање странице како би корисник имао могућност да понови унос.
4. **Ажурирање вредности модела** (енг. *Update model values*) – Након провера којима се утврђује да су све вредности валидне, пролази се кроз све компоненте и врши промена вредности својства зрна које су повезане са компонентом.
5. **Позив апликације** (енг. *Invoke application*) – У овој фази извршавају се акције дугмета или линка које најчешће садрже бизнис логику. Такође, креирају се ниске које се прослеђују навигацији како би се прешло на наредну страницу.
6. **Пружање одговора** (енг. *Render response*) – Последња фаза животног циклуса захтева кодира одговор од стране сервера и испоручује га веб-прегледачу.



Слика 2: Фазе животног циклуса захтева [8]

2.1.3 Модел компоненти корисничког интерфејса

JSF компоненте корисничког интерфејса су конфигурабилни и поново употребљиви делови кода од којих је састављен интерфејс *JSF* апликација. У оквиру архитектуре оквира *JSF* садржани су:

1. *UIComponent* – класа за спецификацију стања и понашања компоненти
2. *Rendering* – модел који дефинише приказивање компоненти
3. *Event-listeners* – модел који дефинише руковање догађајима
4. *Conversion* – модел који дефинише како се конвертер података региструје за компоненту
5. *Validation* – модел који дефинише како се валидатор података региструје за компоненту

Приказ компоненте дефинише се рендерерима. Захваљујући овом приступу могуће је понашање компоненте дефинисати једном, а развити више рендерера од којих сваки дефинише другачији приказ компоненте истом или различитим клијентима.

Објекат *Event* идентификује компоненту која је изазвала догађај, а при томе чува податке и о самом догађају. Неопходно је да апликација садржи имплементацију класе *Listener* и регистровати је као ослушкивач за догађај. Ово је један од најчешћих примера употребе обрасца ”Посматрач” (енг. *Observer pattern*) који је, иначе, веома заступљен у програмском језику *Java*.

Аутоматска конверзија података извршиће се када је тип података својства зрна подржан од стране компоненте са којом је повезан. На пример, уколико је компонента *h:selectBooleanCheckbox* повезана са својством зрна типа *java.lang.Boolean* компонентини подаци ће аутоматски бити конвертовани из типа *String* у *Boolean*, од стране оквира *JSF*. Понекад постоји потреба за конверзијом типа података из компоненте у неки тип који није стандардан за ту компоненту и тада је немогуће извршити аутоматску конверзију. На пример, уколико је потребна конверзија података компоненте из типа *String* у *Java* објекат и обрнуто. *JSF* омогућава регистровање специфичних конвертера, где је неопходно имплементирати интерфејс *Converter*. Пример:

```
<h:inputText id="exampleInput">
    <f:converter converterId="exampleConverter"/>
</h:inputText>
```

JSF поседује и могућност валидације података унетих преко компоненти од стране корисника система. Постоји скуп стандардних и најчешће коришћених валидатора, попут валидатора минималне и максималне дозвољене вредности унесеног нумеричког податка. Опет, као и код конвертера, могуће је дефинисати и сопствени валидатор, на пример, да ли је текстуални унос одговарајућег формата. Неопходно је имплементирати интерфејс *Validator*.

Пример:

```
<h:inputText id="exampleInput">
    <f:validator validatorId="exampleValidator"/>
</h:inputText>
```

2.1.4 Библиотеке *JSF* елемената

Унутар оквира *JSF* дефинисане су две библиотеке елемената. Библиотека *JSF Core* са основним *JSF* елементима, чији је именски простор дефинисан на следећи начин:

xmlns:f="*http://java.sun.com/jsf/core*"

Друга библиотека садржи *HTML* елементе, тј. библиотека *JSF HTML* чији је именски простор:

xmlns:h="*http://java.sun.com/jsf/html*"

2.1.5. Фејслети

Библиотека *Facelets* је развијена како би се *JSF* апликацијама омогућио механизам шаблона (енг. *templating*), тј. механизам бољег и ефикаснијег искоришћења програмског кода раздвајањем комплетне странице на делове који се могу искористити више пута у оквиру исте апликације. То могу бити, не пример, заглавље или подножје стране које треба укључити на свакој страници. Због уочених предности ове библиотеке уведена је у саму спецификацију доносећи могућност коришћења механизма шаблона у оквиру стандардне спецификације. Именски простор ове библиотеке уводи се на следећи начин:

xmlns:ui="*http://java.sun.com/jsf/facelets*"

2.1.6 *AJAX*

AJAX (*Asynchronous JavaScript and XML*) је технологија језика *JavaScript* која омогућава веб-апликацијама да шаљу и примају податке са сервера асинхроно, без мењања тренутног приказа и понашања странице.

JSF спецификација садржи интегрисану подршку за *AJAX* функционалности, највише у домену парцијалног извршавања странице и сличних аспеката асинхроне комуникације.

Пример:

```
<h:commandButton action="#{availabilitiesController.addTeacherAvailability()}"
    value="#{msg['element.save']}" styleClass="btn">
    <f:ajax event="click" execute="@form" render="result"/>
</h:commandButton>
```

Елемент *f:ajax* омогућава да се *AJAX* функционалност уведе у било коју *JSF* компоненту. На овај начин се, након *AJAX* захтева, не учитава комплетан садржај странице поново, већ само оно што нам је неопходно да би приказали промену. Ово постижемо уз помоћ атрибута *render* тако што му као вредност наводимо идентификатор елемента странице, или елемената раздвојених зарезима, које треба поново учитати. Дакле, не мења се изглед и понашање постојеће странице, а и *CSS* датотеке се учитавају само једном. Атрибут *execute* садржи један или листу идентификатора компоненти које ће бити укључене у *AJAX* захтев. У овом примеру коришћена је предефинисана вредност *@form* која означава да ће сви елементи са форме у којој се налази и дугме бити укључени у захтев. Још неке од предефинисаних вредности овог типа су: *@all*, *@this*, *@none*, а које се могу користити и као вредности атрибута *render*. У великим системима ова могућност може значајно повећати функционалност и брзину апликације.

2.1.7 Зрна подршке (енг. *Backing bean*)

JSF апликација упарује сваку страницу у апликацији са одговарајућим зрном подршке (енг. *Backing bean*). Зрно подршке реализовано је преко *Java* класе која се извршава на серверу и дефинише атрибуте и методе које су придружене свакој компоненти која се користи на страници. Садржај самог зрна може бити повезан са инстанцом компоненте или са њеном вредношћу.

Методе које су имплементиране у зрнима подршке могу обављати различите функције везане за компоненту као што су: валидација података компоненте, обрада догађаја које изазива компонента, конверзију података итд.

Атрибут *value* дефинисан на компоненти повезује вредност компоненте са садржајем зрна. Атрибут *binding* повезује садржај зрна са инстанцом компоненте.

Да би се зрно подршке означило као такво, неопходно је испред дефиниције класе додати анотацију *@ManagedBean* након које је могуће навести и име зрна подршке које ће касније користити компонента за повезивање. Уколико име није експлицитно наведено, биће употребљено име класе. Опсег зрна подршке може се такође нагласити неком од наредних анотација:

1. **Опсег захтева** (*@RequestScoped*) – зрно ће постојати исто онолико колико постоје и *HTTP* захтев и одговор. Оног тренутка када се направи *HTTP* захтев креира се и зрно и постоји све док *HTTP* одговор иницран претходним захтевом не нестане. Дакле, приликом сваког захтева креира се нова инстанца зрна. Овај

опсег није одговарајући уколико је потребно користити неку вредност на више страница.

2. **Опсег погледа** (*@ViewScoped*) – зрно ће постојати све док корисник не мења поглед у претраживачу. Инстанцира се оног тренутка када се креира и *HTTP* захтев и постоји све док корисник не промени поглед. Овај опсег највише одговара апликацијама које користе *AJAX*.
3. **Опсег сесије** (*@SessionScoped*) – Зрно подршке постоји све док постоји и *HTTP* сесија. Креира се оног тренутка када се креира и први *HTTP* захтев који се тиче зрна и постоји све док *HTTP* сесија не постане невалидна. Овај опсег се најчешће користи за чување података о пријављеном кориснику.
   Најпростији метод за праћење сесија је колачић. У овом случају назив колачића је *JSESSIONID*. Приликом сваког захтева клијента шаље се и вредност колачића *JSESSIONID*, па тако сервер тачно зна који је клијент у питању. Уколико је клијент искључио подршку за колачиће, податак о сесији ће се слати кроз *URL*.
4. **Опсег апликације** (*@ApplicationScoped*) – Зрно постоји све док постоји и апликација. Креира се када и први *HTTP* захтев и постоји све док се апликација не уклони са сервера.

2.1.8 Анотације животног циклуса зрна

Коришћењем одговарајућих анотација могуће је дефинисати методе које ће се извршити одмах по креирању зрна или на крају опсега:

1. Анотација *@PostCostruct* најчешће се користи када је потребно довући неке податке из базе и припремити их за приказ одмах при иницијализацији странице.
2. Анотација *@PreDestroy* ће навести методу да се изврши пре самог изласка зрна из опсега. Овакве методе су корисне уколико је неопходно очистити неке податке на крају животног циклуса зрна и сл.

2.1.9 Навигација

Навигација у *JSF* апликацији одређена је скупом правила која дефинишу на коју страницу ће корисник бити упућен након клика на дугме или линк.

Најпростија навигација је статичка навигација. У оквиру атрибута *action*, дефинисаног на компоненти, наводи се име странице за редирекцију која ће кориснику бити прослеђена након изазване одговарајуће акције. Пример:

```
<h:commandButton value="Stranica 2"  action="stranica2" />
```

Навигација такође може бити и динамичка. На основу података унетих у форму и одговарајућих израчунавања могуће је проследити различите странице. Пример:

```
<h:commandButton value="Navigacija"  action="#{bean.navigacija}" />
```

Кликом на дугме позива се одговарајућа метода, у овом случају *navigacija* која враћа име странице:

```
public String navigacija() {
        if (uslov) {
                return "stranica1";
        } else {
                return "stranica2";
        }
}
```

2.2 *RichFaces*

RichFaces је радни оквир отвореног кода чија је основна примена проширивање скупа компоненти и уградња подршке за *AJAX* у оквир *JSF*. Дакле, могуће је развити апликације које користе *AJAX* без употребе језика *JavaScript*. Такође, омогућава лако манипулисање изгледом компоненти и апликације уопште употребом скинова (енг. *skin*), што се може посматрати као проширење *CSS*-а на високом нивоу, о чему ће бити више речи касније.

2.2.1 Основне карактеристике

RichFaces је изграђен на оквиру *Ajax4jsf*, који је настао са циљем да обједини *JSF* и *AJAX* функционалности и то је реализовано *JSF* компонентама које су руковале *AJAX* позивима аутоматски, без употребе језика *JavaScript*. Паралелно са оквиром *Ajax4jsf* развијао се *RichFaces* чија је основна сврха била проширење броја компоненти оквира *JSF*. Међутим, суштинска разлика је у томе што је *Ajax4jsf* омогућавао подршку за *AJAX* на целој страници, а не само за појединачне компоненте, што је случај код оквира *RichFaces*.

Пример:

```
<a4j:commandButton value="#{msg.cancel}"
        onclick="#{rich:component('deleteDepartmentConfirmation')}.hide(); return false;" />
```

У претходном примеру коришћено је дугме оквира *Ajax4jsf* где је атрибутом *onclick* дефинисано да се компонента са идентификатором *deleteDepartmentConfirmation* уклања са странице.

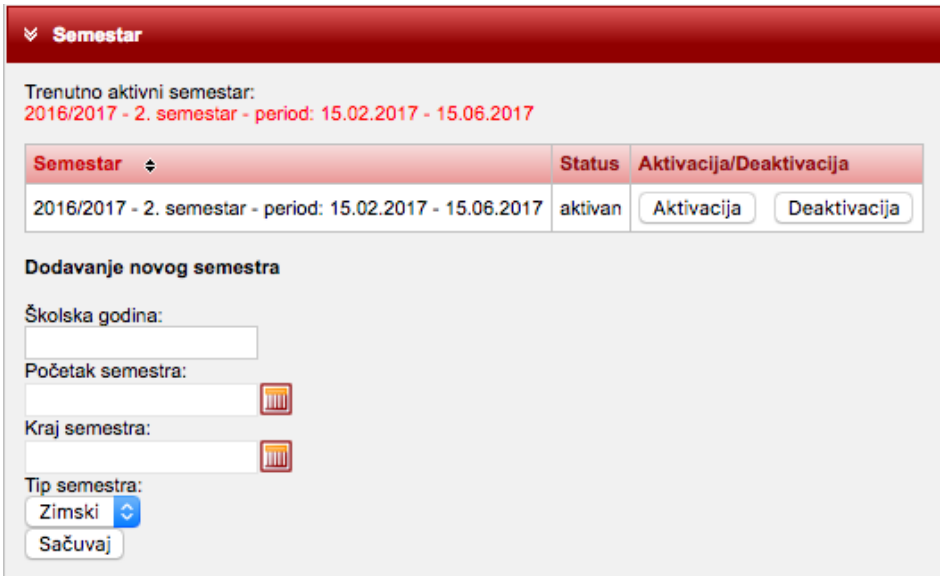
Убрзо су оба оквира доспела под окриље компаније *JBoss* и обједињени су у јединствен оквир назван само *RichFaces*. Ипак, компоненте које су проистекле из оквира *Ajax4jsf* задржале су префикс "*a4j*" на *JSF* страницама.

Велика предност оквира *RichFaces*, у односу на друге сличне оквире, је та што је у потпуности интегрисан у животни циклус оквира *JSF*. Док други оквири омогућавају само приступ зрнима подршке, *RichFaces* нуди и приступ ослушкивачима догађаја, позива конверторе и валидаторе на серверској страни током обраде захтева.

Једноставна промена скинова (енг. *Skinnability*) представља важно својство оквира *RichFaces* које упрошћава дизајнирање корисничког интерфејса и умањује потребу за дефинисањем великог броја елемената у *CSS* документима. Један скин дефинише палету боја и низ других параметара који се примењују на комплетан кориснички интерфејс, чиме се избегава понављање одређених параметара са идентичном вредношћу. Једном променом скина се мења комплетан изглед свих компоненти корисничког интерфејса. Скинови се дефинишу у конфигурационој датотеци *web.xml*:

```
<context-param>
        <param-name>org.richfaces.skin</param-name>
        <param-value>plain</param-value>
</context-param>
```

Слика 3: Пример изгледа странице употребом скина *wine*



Слика 4: Пример изгледа странице употребом скина *ruby*

RichFaces нуди листу предефинисаних скинова који су доступни за употребу. Неки од њих су: *wine*, *ruby*, *blueSky*, *deepMarine*,… Наравно, могуће је мењати особине постојећих скинова, као и креирати нове које је неопходно сачувати у директоријуму *META-INF/skins*.

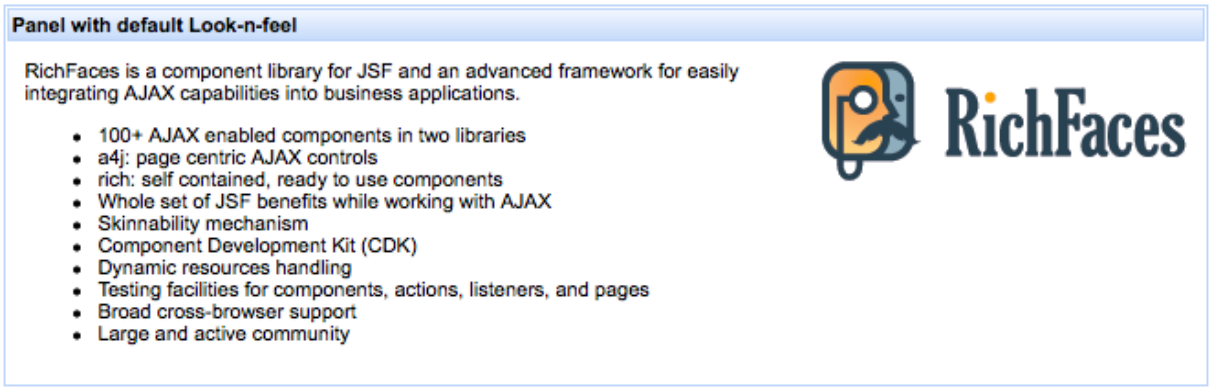
Унутар оквира *RichFaces* дефинисане су две библиотеке компоненти: библиотека *Core Ajax* која садржи скуп компоненти које уносе *Ajax* функционалности, без употребе *JavaScript* кода и библиотека *UI* која нуди читав низ компоненти корисничког интерфејса – табеле, календаре, различите панеле, меније итд.

2.2.2 Основне компоненте

rich:panel

Ова компонента креира област правоугаоног облика на страници која може садржати било коју другу компоненту, укључујући и друге панеле. Панел може имати заглавље, што је опционо, и тело.

```
<rich:panel header="Unos i izmena termina" rendered="#{user.type eq 2}"
        styleClass="panel panel-primary" headerClass="panel-heading"
        bodyClass="panel-body">
```



Слика 5: Компонента *Panel* са заглављем

Објашњења атрибута компоненте:

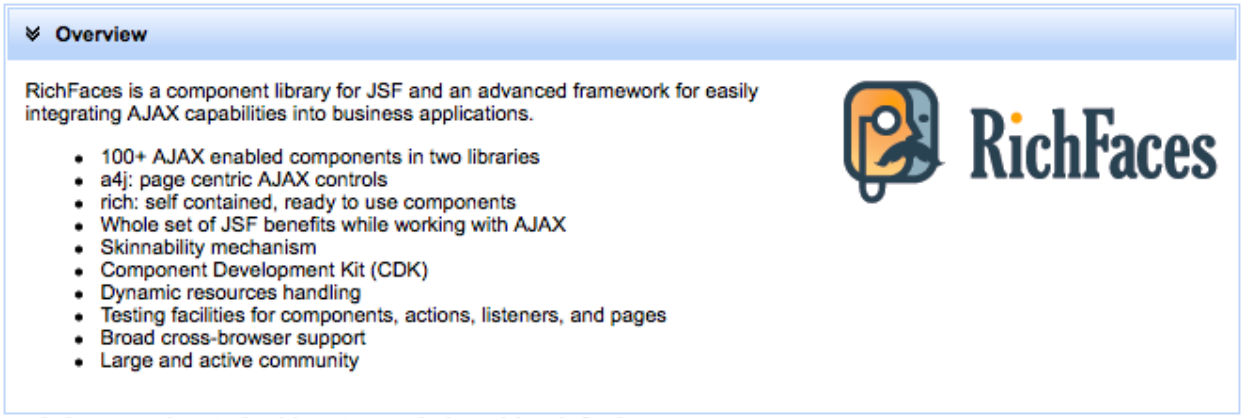
- *header* – текст који ће бити приказан у заглављу панела.
- *rendered* – овај атрибут очекује бинарну вредност, користи се за дефинисање. услова под којим ће компонента бити приказана.
- *styleClass* – име *CSS* класе која ће бити примењена на комплетан панел.
- *headerClass* - име *CSS* класе која ће бити примењена само на заглавље панела.
- *bodyClass* - име *CSS* класе која ће бити примењена само на тело панела.

rich:collapsiblePanel

Ова компонента у потпуности је аналогна претходној компоненти, с тим што овај панел може бити скупљен или раширен. Уколико је скупљен приказано је само заглавље, а када је раширен види се комплетан панел.

```
<rich:collapsiblePanel header="Izlistavanje časova"
        switchType="client" styleClass="panel panel-primary" headerClass="panel-heading"
        bodyClass="panel-body">
```

Као што се може видети, и овде уз помоћ одговарајућих атрибута компоненте, могуће је наводити *CSS* класе које ће се односити засебно на заглавље, а засебно на остатак, односно тело панела.



Слика 6: Компонента *CollapsiblePanel*

rich:dataTable

Ова компонента користи се за табеларни приказ података који су садржани у колекцији података у зрну подршке. Атрибути сваког од објеката из колекције се мапирају на одређене колоне у табели. Компонента омогућава једноставно сортирање и филтрирање података, а такође подржава и велики број атрибута којима се дефинише обрада догађаја над табелом.

```
<rich:dataTable id="subjectLessonsList" var="lesson" rows="10"
        value="#{lessonsController.subjectLessonsList}"
        iterationStatusVar="it" styleClass="table table-bordered">
        <rich:column>
                <f:facet name="header">
                        Ime predmeta
                </f:facet>
                <h:outputText value="#{lesson.subject.name}"/>
        </rich:column>
</rich:dataTable>
```

Објашњења атрибута компоненте:

- *var* – променљива помоћу које се у току итерације кроз колекцију приступа објекту чији се подаци представљају у тренутном реду табеле. Тип ове променљиве одговара објекту из колекције и не захтева додатно дефинисање игде у коду.
- *rows* – број редова у табели који ће бити приказан. Уколико има више података од дефинисаног броја, подаци ће бити подељени на странице.
- *iterationStatusVar* – променљива којом се у току итерације приступа редном броју објекта у тренутном реду табеле.
- *value* – колекција објеката који се представљају у табели. За сваки од атрибута наводи се засебна колона са одговарајућим заглављем.

State Flag	Sort by Capital Name	Sort by State Name	Sort by Time Zone
	Montgomery	Alabama	GMT-6
	Juneau	Alaska	GMT-9
	Phoenix	Arizona	GMT-7
	Little Rock	Arkansas	GMT-6
	Sacramento	California	GMT-8
	Denver	Colorado	GMT-7
	Hartford	Connecticut	GMT-5
	Dover	Delaware	GMT-5
	Tallahassee	Florida	GMT-5
	Atlanta	Georgia	GMT-5
	Honolulu	Hawaii	GMT-10
	Boise	Idaho	GMT-8

« « « 1 2 3 4 5 » » » » » »

Слика 7: Компонента *DataTable*

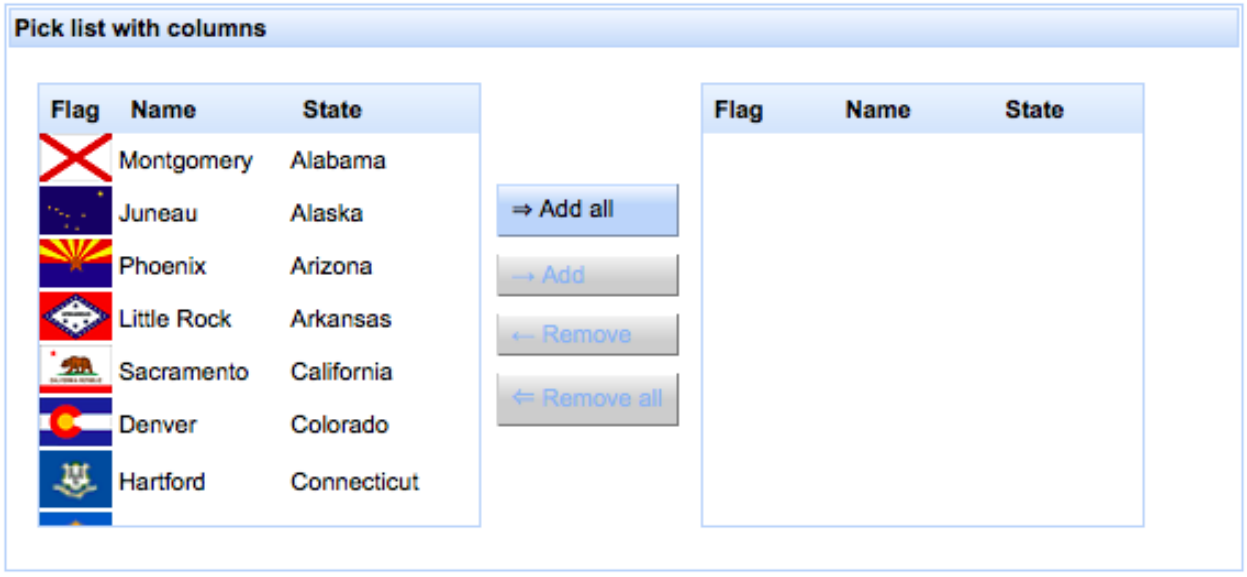
rich:pickList

Ова компонента омогућава вишеструки одабир елемената из дате листе објеката, могућност уклањања елемената из одабране листе, као и промену редоследа објеката. Такође, ова компонента омогућава представљање објеката кроз више колона, што омогућава детаљнији приказ кориснику.

```
<rich:pickList value="#{lessonsController.selectedRooms}"
        var="room" listHeight="200px" converter="#{roomConverter}" required="true">
        <f:selectItems value="#{lessonsController.listAllRooms()}" />
        <rich:column>
                <f:facet name="header">Naziv sale</f:facet>
                #{room.name}
        </rich:column>
</rich:pickList>
```

За ову компоненту неопходно је да у зрну подршке буду дефинисане две колекције објеката. Прва која ће представљати све објекте, односно објекте за одабир, док друга колекција објеката треба да буде празна и у њој ће се чувати сви одабрани објекти од стране корисника из прве листе у компоненти.

JSF компонента *f:selectItems* дефинише листу свих елемената који ће бити излистани у компоненти.



Слика 8: Компонента *PickList*

2.3 *EJB (Enterprise Java Beans)*

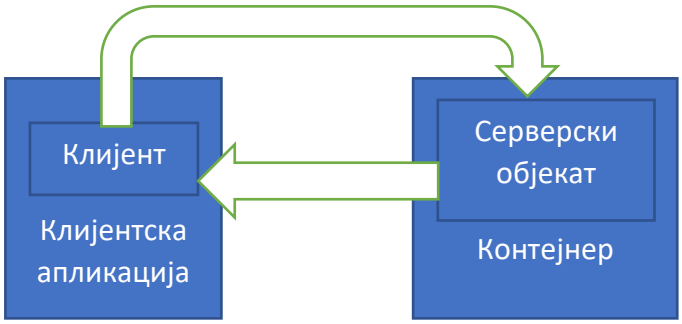
Спецификација *EJB* дефинише модел серверских компоненти за развој апликација које имају вишеслојну архитектуру са дистрибуираним објектима. Вишеслојна архитектура је клијент-сервер архитектура где су презентација, пословна логика и управљање подацима издвојени у засебне подсистеме. Дистрибуирани објекти су објекти који су дистрибуирани на различитим адресним просторима у оквиру различитих процеса једног рачунара, или на више рачунара у рачунарској мрежи, који међусобно комуницирају.

2.3.1 Основни концепти технологије дистрибуираних објеката

Концепт дистрибуираних објеката полази од идеје да неки објекат (инстанца класе) са својим методама и атрибутима може постојати на једном рачунару, а да други програми, односно објекти, који се извршавају на другим рачунарима могу да га користе. Употреба тог објекта значи могућност позивања метода и приступа његовим атрибутима.

Корисници серверског објекта (клијенти) би требало да му што једноставније приступају. Позив методе серверског објекта подразумева извршавање те методе на рачунару на коме се објекат налази. То у ствари значи да се клијентски програм извршава на више рачунара, иницијално на оном на коме је покренут, али и на свим рачунарима на којима се налазе серверски објекти које он користи.

Оваква комуникација се може посматрати као комуникација између два објекта: клијентски објекат је део клијентске апликације, док је серверски објекат део серверске апликације. Серверска апликација се често назива ”контејнер” за објекте, јер је њена основна функција да обезбеди мрежне и друге сервисе неопходне за функционисање серверских објеката.



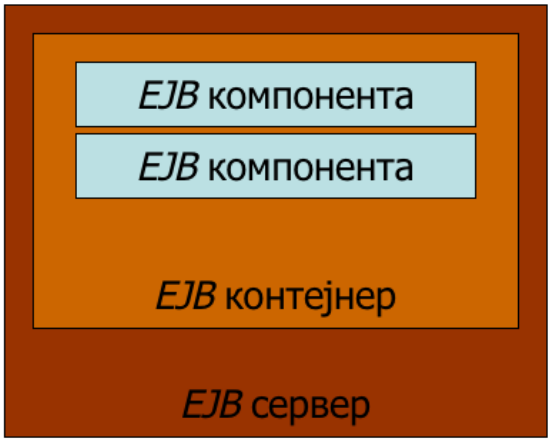
Слика 9: Клијентски и серверски објекати у оквиру својих програма

И клијентски и серверски објекат саставни су део једног програма. Особина тог програма је да се извршава на више рачунара у мрежи.

2.3.2 Основне карактеристике технологије *EJB*

Спецификација *EJB* је прошла кроз неколико верзија: 1.0, 1.1, 2.0, 2.1, 3.0 где је свака од верзија додавала нове могућности. Ова технологија имплементира се искључиво у језику *Java*.

Спецификација *EJB* не бави се клијентском страном, већ само серверском. *EJB* дефинише окружење у коме се налазе и функционишу *EJB* компоненте. Ово окружење састоји се од *EJB* сервера и *EJB* контејнера. Функције сервера и контејнера нису строго раздвојене.



Слика 10: Међусобни однос *EJB* сервера, контејнера и компоненти

Задаци *EJB* контејнера су да обезбеди:

- Подршку трансакцијама
- Сигурносне механизме
- Управљање животним циклусом и ресурсима
- Приступ са удаљених клијената
- Подршку конкурентним захтевима

Аутор *EJB* компоненте не мора у потпуности познавати начин функционисања *EJB* сервера и контејнера. Сви *EJB* сервери, који се држе спецификације, радиће са правилно написаним *EJB* компонентама.

2.3.3 Врсте *EJB* компоненти

EJB компоненте су подељене у неколико група, о којима ће у наставку бити више речи:

- Зрна сесије (енг. *Session Beans*)
  - Зрна сесије која чувају стање (енг. *Stateless Session Beans*)
  - Зрна сесије која не чувају стање (енг. *Stateful Session Beans*)
- Зрна ентитета (енг. *Entity Beans*)
  - *Bean-managed persistence*
  - *Container-managed persistence*
- Ентитети (енг. *Entities*)
- *Message-driven beans*

Зрна сесије су *EJB* компоненте које се извршавају за потребе тачно једног клијента. Животни век ових компоненти везан је за животни век клијента који их користи. Када клијент упути захтев за одређеном компонентом *EJB,* сервер ће креирати нову инстанцу компоненте и доделити је само том клијенту. Основна намена оваквих компоненти је управљање подацима.

Компоненте зрна сесије су подељене на две групе:

- Компоненте које чувају своје стање између вишеструких позива клијента који их користи. Можемо их посматрати као објекте који имају своје атрибуте чија се вредност чува између вишеструких позива њихових метода.
- Компоненте које не чувају своје стање између вишеструких позива. Ове компоненте омогућавају позивање њихових метода, али извршавање тих метода не утиче на стање компоненте, позиви су међусобно независни.

Да бисмо дефинисали неку компоненту као компоненту која чува своје стање или не, довољно је испред дефиниције класе навести анотације: *@Stateful* или *@Stateless*.

Зрна ентитета подржавају истовремени приступ од стране више клијената. Намењени су, пре свега, за представљање података. Може се рећи да су намењени за имплементацију модела података апликације. Животни век ових компоненти везан је за животни век података које престављају.

Зрна ентитета се деле у две подврсте у зависности од тога како се обезбеђује њихова трајност:

- *Bean-managed persistence* – компонента сама имплементира операције које обезбеђују трајно чување података које она садржи.
- *Container-managed persistence* – операције које обезбеђују чување података, које компонента садржи, имплементира *EJB* контејнер, који сам брине о њиховом иницирању у одговарајућим тренуцима.

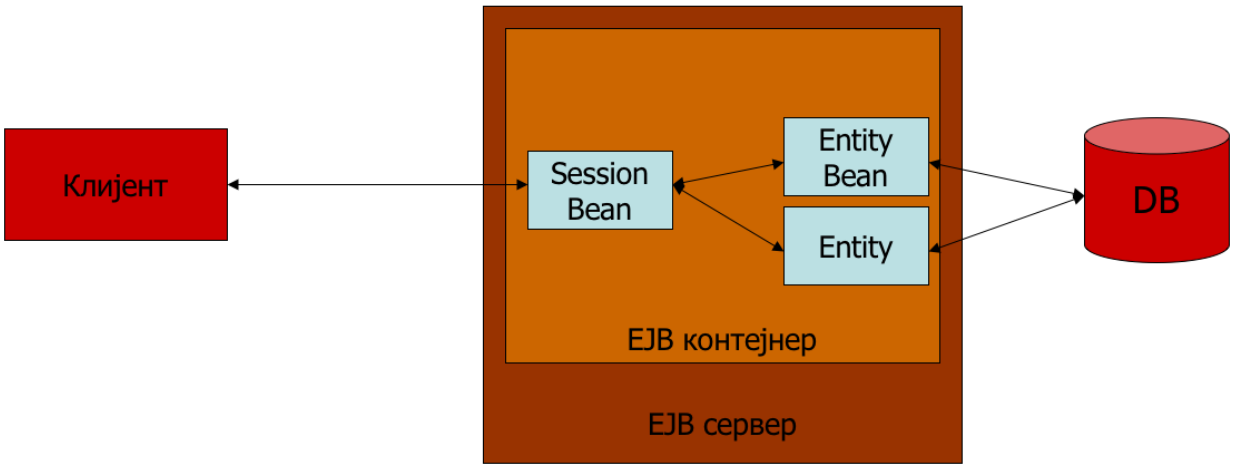
С обзиром на особине зрна сесија и зрна ентитета, препорука је да клијенти директно комуницирају само са зрнима сесије, које имплементирају потребне операције за управљање подацима. Подаци којима те операције рукују представљени су одговарајућим зрнима ентитета, које за потребе трајног чувања свог стања користе базу података.

До верзије *EJB* 2.1 ове компоненте су биле једине које су подржавале перзистенцију, од верзије *EJB* 3.0 постоје због компатибилности са старијим програмима.

Ентитети су намењени за чување података. Као и код зрна ентитета, животни век им је везан за животни век података које представљају.

Ове компоненте представљене су као *POJO* (*Plain Old Java Object*) објекти, а примарни кључ, начин његовог формирања итд. се дефинише уз помоћ анотација. Ове компоненте нису видљиве ”споља”, тј. може им се приступити само преко зрна сесије.

***Message-driven beans* компоненте** реагују на појаву порука *JMS (Java Message Service)*, тачније задужене су за пријем и обраду оваквих порука које стижу од клијента. Користе се за асинхрону комуникацију између клијента и *EJB* компоненте, па не поседују интерфејс који користе клијенти и представљају најједноставније компоненте са становишта развоја. Потребно је имплементирати само једну методу *onMessage*, која ће бити позвана од старне контејнера када је потребно обрадити пристиглу поруку.



Слика 11: Архитектура *EJB* система

2.3.4 Структура *EJB* компоненте

Писање *EJB* компоненте обухвата дефинисање следећих елемената:

- *Bean* класа – имплементира одговарајућу компоненту у ужем смислу, тј. имплементира интерфејсе *SessionBean* или *EntityBean*.
- *Home* интерфејс – интерфејс који дефинише методе за креирање, уклањање и проналажење компоненте у свом контејнеру.
- *Remote* интерфејс – интерфејс који дефинише методе које су доступне клијентима.
- Локални интерфејс – дефинише методе које су доступне локалним клијентима, односно клијентима у истој виртуелној машини (друге компоненте унутар *EJB* контејнера).
- Локални *home* интерфејс – дефинише методе за креирање, уклањање и проналажење компоненте у свом контејнеру, али ове методе су доступне само клијентима у оквиру истог контејнера где је и сама компонента.

Дакле, клијент комуницира са компонентом искључиво преко њених *home* и *remote* интерфејса, док у случају да једна *EJB* компонента позива другу, она се понаша као клијент те компоненте, проналази је и користи преко мреже као и други клијенти.

Локални и *remote* интерфејси не морају поседовати исти скуп метода. Тачније, комуникација са локалним клијентима може се разликовати од комуникације са удаљеним клијентима.

Циљ актуелне 3.0 спецификације је поједностављење развоја *EJB* компоненти. Ово поједностављење постигнуто је дефинисањем новог модела компоненти, међутим *EJB* 3.0 сервери морају у потпуности подржавати рад са старим компонентама.

Зрна сесије су поједностављена на следеће начине:

- Није потребно писати *remote* и *home* интерфејсе. Ово значајно поједностављује развој компоненти, међутим, методе које су доступне клијентима и даље треба описати у засебном интерфејсу (интерфејс пословне логике, енг. *Business interface*) за који не важе неки посебни услови, тј. сваки *Java* интерфејс може бити интерфејс пословне логике.
- Није потребно имплементирати интерфејс компоненте. Зрна сесије су вршила интеракцију са контејнером путем метода прописаних интерфејсом *SessionBean* који позива контејнер у одговарајућим тренуцима. Уколико ти позиви нису били потребни, методе су се свакако морале имплементирати јер су прописане интерфејсом. Сада било која метода у класи може служити за обраду контејнерових позива.
- Коришћење анотација. Назнаке *EJB* контејнеру о намени појединих елемената класе, која представља *EJB* 3.0 зрно сесије, формулишу се помоћу анотација.

Зрна ентитета нису доживела већу промену са новом спецификацијом. Поред тога, дефинисан је потпуно нови програмски модел за писање перзистентних класа, класа које своје стање могу трајно сачувати у бази података. И ова спецификација, названа *JPA (Java Persistence API)* омогућава писање *POJO* објеката који у великој мери преузимају функционалност зрна ентитета уз поједностављен развој. Овај нови тип компоненте се уобичајено зове ентитет и пише се као *POJO* објекат доупуњен анотацијама које описују начин складиштења објеката у бази података.

Пример: Уколико имамо *Java* класу са одређеним скупом метода за управљање корисницима система, као што су: проналажење корисника на основу корисничког имена и лозинке, додавање новог корисника или ажурирање и брисање постојећег, да бисмо је означили као зрно сесије, неопходно је испред дефиниције саме класе навести одговарајућу анотацију *@Stateful* или *@Stateless*. Такође, класа мора да имплементира одређени интерфејс пословне логике, у овом примеру конкретно за управљање корисницима, који ће дефинисати које методе зрна сесије су доступне клијентском делу апликације. Клијентски део апликације овај интерфејс укључује употребом анотације *@EJB* чиме се омогућава позив методе из зрна сесије.

Пример интерфејса пословне логике за управљање корисницима:

```
public interface UsersService {
	User findUser(String username, String password);
	void addUser(User user);
	User deleteUser(Long userId);
	void updateUser(User user);
}
```

Пример зрна сесије са дефинисаним методама за управљање корисницима:

```
@Stateless
public class UsersServiceImpl implements UsersService {

	@Override
	public User findUser(String username, String password) {
		User user;
		// implementacija logike za dovlačenje korisnika iz baze
		return user;
	}
	// implementacija ostalih metoda interfejsa
}
```

Пример коришћења интерфејса пословне логике у оквиру клијентске апликације (Зрно подршке *JSF* имплементације):

```
@ManagedBean
@SessionScoped
public class LoginController {

	@EJB
	private UsersService service;
	public String login() {
		String action = null;
		this.user = service.findUser(username, password);
		// ostatak logike vezano za logovanje korisnika
	}
}
```

2.4 JPA (Java Persistence API)

Спецификација *JPA* омогућава објектно-релационо мапирање у *Java* апликацијама које доста упрошћава управљање подацима. Као што је већ поменуто, развијен је уз спецификацију *EJB* 3.0, међутим, коришћење му није ограничено искључиво на *EJB* компоненте.

2.4.1 Објектно-релационо мапирање

Објектно-релационо мапирање је најчешћи начин перзистирања *Java* објеката. Перзистенција подразумева складиштење *Java* објеката у релациону базу података. Тачније, оно представља превођење објеката у табеле и обрнуто. *Java* класа служи за дефинисање шеме базе података и приликом чувања подаци из објеката се снимају у табеле, док се приликом читања података подаци из табела стављају у објекте који су за то направљени.

Сва мапирања се реализују анотацијама. Анотације представљају специјалну форму мета података који стоји уз класе, методе и атрибуте класа. Анотације језика *Java* су рефлексивне, тј. уграђују се у датотеку *.class* дефинисану од стране *Java* преводиоца и могу бити задржане у оквирима *Java* виртуелне машине како би биле доступне у време извршавања.

Java објекат, који је мапиран на табелу у бази података, назива се ентитет (енг. *Entity*) и уколико се експлицитно не наведе, тај објекат се мапира на табелу истог имена, са колонама истог имена као атрибути у класи. *EJB* контејнер може самостално креирати табелу у бази података уколико не пронађе одговарајућу.

2.4.2 Ентитети

Као што је већ поменуто, ентитети се пишу као *POJO* објекти, односно објекти који садрже атрибуте и одговарајуће методе приступа, тј. методе *get* и *set*. Не може им се приступити изван *EJB* контејнера, већ се приступ врши искључиво преко зрна сесије.

Анотацијом *@Entity*, из пакета *javax.persistence.Entity*, испред дефиниције класе се наводи да ће се наредна класа користити за мапирање са истоименом табелом у бази података. Уколико се имена разликују, могуће је додати анотацију *@Table*, такође испред имена класе и у оквиру атрибута *name* ове анотације конкретно навести име табеле на коју се врши мапирање. Обавезно је у класи навести атрибут који ће представљати примарни кључ, означити га анотацијом *@Id* из пакета *javax.persistence.Id* и тако извршити његово мапирање. Примарни кључ је кандидат за кључ који јединствено идентификује сваку врсту у табели.

Пример:

```
@Entity
@Table(name = "StudentGroup")
public class Group {

@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;

...
```

Менаџер ентитета (енг. *Entity manager*) као основни задатак има управљање ентитетима, односно да оствари комуникацију са базом података, а скуп свих тих ентитета назива се *Persistence context*. Тачније, менаџер ентитета представља интерфејс који дефинише методе које *Persistence context* мора да имплеметира.

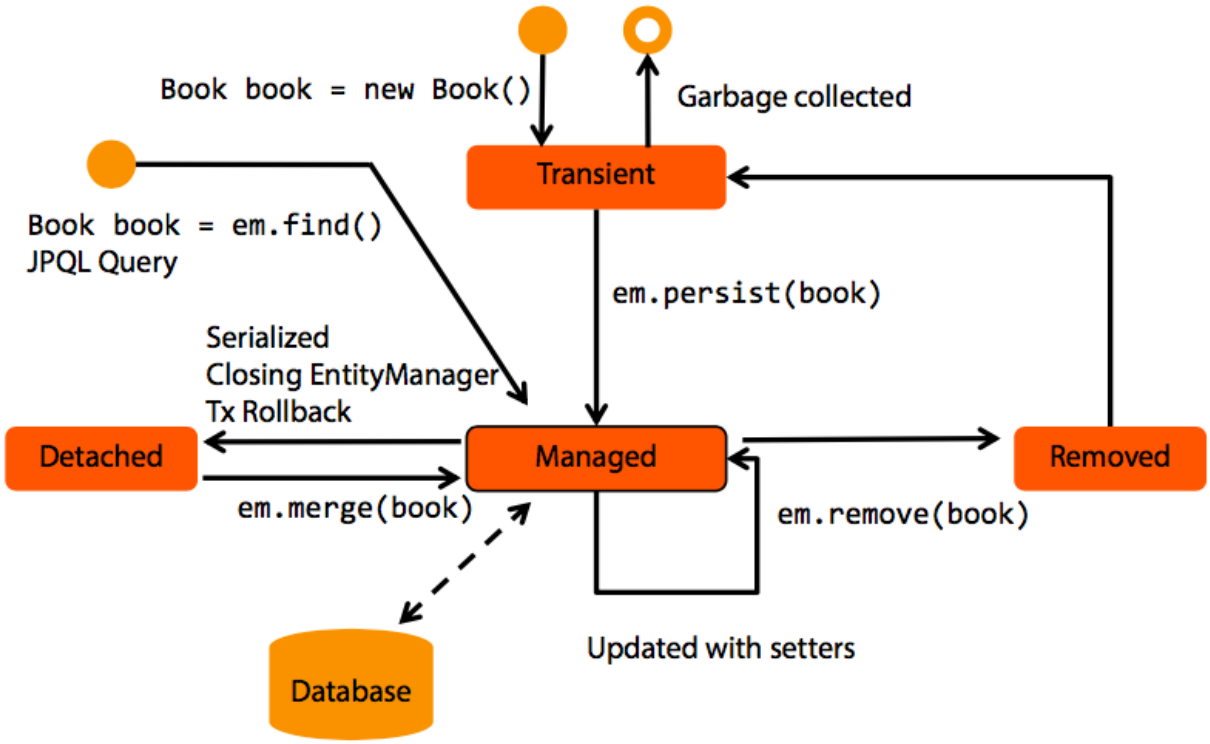
Ентитет пролази кроз неколико стања током рада са подацима које он представља:

- **Нови ентитет (енг. *New* или *Transient*) -** Ентитет који је креиран али још увек није сачуван у бази.
- **Синхронизовани ентитет (енг. *Managed*)** – Све промене над атрибутима се синхронизују са базом. Методе менаџера ентитета које пребацују ентитет у ово стање су: *persist*(), *merge*() и *find*().
- **Несинхронизовани ентитет (енг. *Detached*)** – Стање ентитета није синхронизовано са базом, ентитет прелази у ово стање након завршетка трансакције.
- **Обрисан ентитет (енг. *Removed*)** – У ово стање ентитет прелази након позива методе *remove*(), али ће наставити да постоји у меморији као *Java* објекат.

Основне операције које можемо извршити над неким ентитетом преко менаџера ентитета су:

- **Креирање новог ентитета** се изводи тако што се прво креира *Java* објекат са одговарајућим вредностима атрибута и након тога позива метода менаџера ентитета *persist*().
- **Ажурирање вредности ентитета** се може извести на два начина. Након позива методе *find*() менаџера ентитета и уколико је објекат пронађен у бази, довољно је позвати методе за измену вредности атрибута (енг. *Setter* методе) и метода менаџера ентитета *update*() ће се извршити аутоматски. Други начин ажурирања вредности јесте уз помоћ методе *merge*() менаџера ентитета и, након тога, поново позивањем метода за измену вредности атрибута објекта.
- **Брисање ентитета** се изводи позивањем методе *remove*() над одговарајућим ентитетом. Пре самог брисања неопходно је позвати методу *find*() како би се добио одговарајући објекат из базе. Уколико већ имамо ентитет који није синхронизован са базом, неопходно је прво позвати методу *merge*() над ентитетом.

- **Претрага ентитета** изводи се позивањем методе *find*() којој се као аргумент прослеђује вредност примарног кључа траженог објекта. Уколико одговарајући објекат није пронађен, ова метода враћа вредност *null*.



Слика 12: Животни циклус ентитета [6]

2.4.3 Основне анотације

Као што је већ поменуто, приликом креирања сваког етитета морају постојати анотације ***@Entity*** над класом и ***@Id*** над атрибутом који представља примарни кључ.

Уколико не желимо подразумевано мапирање класа са табелама, односно уколико се догоди да се имена класе и табеле у бази података не поклапају, могуће је користити анотацију ***@Table*** и у оквиру атрибута *name* навести конкретно име табеле. Исто је могуће урадити са колонама, односно атрибутима ентитета коришћењем анотације ***@Column***.

Приликом дефинисања примарног кључа најчешће се дефинише и сам генератор примарног кључа анотацијом ***@GeneratedValue(strategy=GenerationType.AUTO)***. Атрибутом *strategy* се одређује како се дефинише примарни кључ. Анотација ***@Transient*** се користи над атрибутом који није перзистентан, односно не мапира се ни на једну колону у бази података.

2.4.4 Кардиналност између објеката

Ентитет може да референцира произвољан број других ентитета и веза између њих се успоставља као било која друга веза између два *Java* објекта, декларацијом једног ентитета као атрибут другог. Кардиналитет везе између два ентитета може бити:

- **1:1 (један према један)** дефинише се анотацијом *@OneToOne* над атрибутом којим се референцира ентитет.
- **1:n (један према више)** дефинише се анотацијама *@OneToMany* (1-страна) и *@ManyToOne* (n-страна). Ентитет са n-стране више није репрезентован јединственом инстанцом унутар ентитета на 1-страни, већ колекцијом инстанци (листа, низ,…).
- **m:n (више према више)** дефинише се анотацијом *@ManyToMany* на обе стране везе. Оба ентитета су у овом случају репрезентована колекцијом инстанци на супротној страни.

У сваком од претходно наведених случајева могуће је користити анотације за описивање спољних кључева у табелама на коју се дати атрибут мапира. Подразумевано понашање је да анотације *@OneToOne* и *@OneToMany* користе мапирање на основу колоне која представља спољни кључ у табели, док анотације *@ManyToOne* и *@ManyToMany* користе ново-креирану табелу за мапирање. Ова табела садржи две колоне које представљају спољне кључеве оба ентитета.

Уколико је веза између ентитета једносмерна, у другом ентитету не наводимо атрибут који представља први ентитет.

Претходне анотације коришћене за описивање кардиналности међу ентитетима могу садржати додатне атрибуте који дефинишу на који начин се довлаче мапирани ентитети приликом претраге или шта се дешава са њима приликом брисања, ажурирања итд. На пример, атрибут *cascade=CascadeType.ALL* омогућава да се приликом трајног чувања ентитета аутоматски памте и сви њему придружени ентитети. Атрибут *Fetch* дефинише на који начин се довлаче придружени ентитети:

- *FetchType.EAGER* – Уколико је више ентитета повезано различитим везама, када позовемо први, у меморији ће се одмах учитати и остали повезани ентитети.
- *FetchType.LAZY* – Учитава се само први ентитет приликом позива, без обзира да ли садржи повезане ентитете. Уколико нам је потребан неки од повезаних ентитета, неопходно је експлицитно га позвати уз помоћ одговарајуће *get* методе.

Пример:

```
@Entity
@Table(name = "Lesson")
public class Lesson {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long id;

	@ManyToOne
	@JoinColumn(name = "subjectId")
	private Subject subject;

	@OneToMany(fetch = FetchType.EAGER)
	@JoinTable(name = "LessonTeacher",
		joinColumns = @JoinColumn(name = "lessonId"),
		inverseJoinColumns = @JoinColumn(name = "teacherId"))
	private Set<User> teachers;
```

2.4.5 Наслеђивање

С обзиром да у базама података не постоји могућност наслеђивања међу табелама, а програмски језик *Java* подржава наслеђивање, *JPA* има три различите стратегије за реализацију наслеђивања:

- *single-table-per-class* – Атрибути свих ентитета су смештени у једну табелу која садржи додатну колону именовану *DTYPE* уз помоћ које се чува инфрмација који атрибут припада ком ентитету. *DTYPE* је типа *VARCHAR* и садржи име ентитета.
- *joined-subclass* – свака потkласа и наткласа имају засебну табелу па се ради спајање табела по потреби.
- *table-per-concrete-class* – Свака потkласа и наткласа имају своје табеле, с тим што табеле које представљају потkласе имају предефинисане атрибуте из наткласе.

Подразумевано мапирање код наслеђивања класа је *single-table-per-class*, а анотација која се користи за дефинисање методе наслеђивања је:

@Inheritance (strategy=SINGLE_TABLE)

Ова антоција наводи се испред имена класе која треба бити наслеђена. Такође поред ње неопходно је укључити и анотацију *@MappedSuperclass* која означава класу чија се подаци за мапирање примењују на ентитете који је наслеђују.

На пример, претпоставимо да имамо три веома сличне табеле, са неколико колона истог назива и типа, рецимо табеле за чување врста термина посебно за професоре, групе и учионице где свака од табела садржи колоне: идентификатор (примарни кључ), тип

термина, ознаку за дан у недељи, редни број термина и референцу на семестар, док једину разлику међу табелама представља колона за референцу на професора, односно групу или учионицу. Како бисмо избегли вишеструко дефинисање истих атрибута међу различитим ентитетима, довољно је креирати једну наткласу са дефинисаним заједничким атрибутима, која ће бити наслеђена од стране три поткласе које ће садржати дефинисане само атрибуте специфичне за дати ентитет.

2.4.6 JPQL (Java Persistance Query Language)

Језик *JPQL* омогућава извршавање упита над ентитетима. Велика предност овог језика је писање упита који су невезани од конкретног система за управљање базом података који се користи за чување података. Овде не манипулишемо са базом и табелама, већ радимо са објектима и атрибутима.

JPA нуди две врсте упита:

- **Статичке или именоване упите** – Ови упити су непроменљиви. Апликација их једном преводи и извршава више пута. Наводе се у облику анотације *@NamedQuery* приликом дефинисања ентитета и имају два атрибута: (која морају бити дефинисана) *name* и *query*. Овакви упити доприносе организацији кода. Такође, не могу се понављати имена овог типа упита, па је неко уобичајено правило да се додаје префикс који представља име ентитета којем упит припада. Овде је немогућа конкатенација упита.
- **Динамичке упите** – Омогућавају конкатенацију упита и на тај начин се омогућава динамички приступ. С обзиром да приликом креирања динамичких упита не знамо како ће на крају упит изгледати, сваки пут приликом извршавања *JPQL* ће се преводити у *SQL*, стандардни језик за управљање подацима у бази података и због тога именовани упити могу бити бољи по питању перформанси система.

Пример креирања динамичког упита коришћењем менаџера ентитета:

```
TypedQuery<TeacherAvailability> availabilityList = em.createQuery(
        "SELECT a FROM TeacherAvailability a WHERE a.teacher.id = :teacherId
                AND a.semester.id = :semesterId", TeacherAvailability.class);
        availabilityList.setParameter("teacherId", teacherId);
        availabilityList.setParameter("semesterId", semesterId);
```

Упит из претходног примера је динамички из разлога што је могуће излистати врсте термина за различите професоре, као и за различите семестре, и због тога је неопходна конкатенација, односно динамички приступ. У овом случају није коришћена класична конкатенација ниски, већ су параметри дефинисани у оквиру упита употребом знака ":", а затим је методом *setParameter* извршена замена параметра упита одговарајућом вредношћу. За креирање динамичког упита коришћена је метода *createQuery* менаџера ентитета, који је у овом примеру именован *em*.

3 Систем *ArgoTimetabling* за аутоматско распоређивање

Систем *ArgoTimetabling* за аутоматско распоређивање часова за основне школе, средње школе и факултете развијен је на Математичком факултету, Универзитета у Београду и у склопу његовог развоја уједно су дефинисани и формати спецификације и записа распореда часова. Дакле, систем као улаз очекује текстуалну датотеку која садржи спецификацију распореда часова у формату **.tts* (*time-table-specification*), док је резултат рада система *ArgoTimetabling* такође текстуална датотека са распоредом часова у формату **.ttbl* (*timetable*). На основу ове датотеке компонента за приказ распореда генерише документе распореда појединачно за све сале, наставнике, групе,... У наставку рада ће ова два формата бити укратко описана.

3.1 Проблем распоређивања

Проблем распоређивања часова подразумева распоређивање часова са унапред датог списка часова у унапред фиксиране временске термине. Основни концепти система су: термини, наставници, групе, сале, предмети, часови и ограничења. Наставна недеља подељена је на радне дане, а сваки дан подељен је на термине. Иако настава врло често представља рад у две смене, уобичајена је пракса да се распоред часова за сваку од смена креира посебно, тако да концепт смене не постоји.

Наставници држе часове из неког предмета истовремено једној или више група ученика у једном или више везаних термина у некој од адекватних сала. Систем гарантује да ниједан наставник, група ни сала неће имати више часова распоређених у исти термин. Уз то, поштују се задата огранчења. Ограничења могу бити тврда (она која морају бити испуњена) или мека (она која не морају бити испуњена, али је пожељно испунити их што више). Ограничења се односе на расположивост наставника, група, односно сала, на број и облик пауза у току наставе за наставнике и групе, на дужину трајања наставе наставницима и групама и слично.

3.2 Формат спецификације распореда часова (**.tts*)

Спецификација распореда врши се у текстуалној датотеци (кодираној *UTF-8* кодирањем). Датотека је подељена на секције, при чему се називи секција наводе у заградама […]. Садржај секције чине линије, линије су подељене на колоне коришћењем табулатора, а свака колона може додатно бити подељена на појединачне ставке коришћењем запета. Линије у датотеци, које почињу карактерима //, сматрају се коментарима и занемарују се.

3.2.1 Секција [*periods*]

Ова секција садржи списак радних дана и могућих термина у оквиру тих радних дана. Термини означавају редни број часа.

```
[periods]
pon     Ponedeljak      1,2,3,4,5,6
uto     Utorak          1,2,3,4,5,6,7
sre     Sreda           1,2,3,4,5,6,7
cet     Cetvrtak        1,2,3,4,5,6,7
pet     Petak           1,2,3,4,5
```

3.2.2 Секција [*teachers*]

Ова секција служи за навођење списка свих наставника. За сваког наставника наводи се идентификатор, а затим након табулатора, пуно име и презиме. Идентификатори морају бити јединствени, па их због тога апликација аутоматски генерише и нису познати крајњем кориснику.

```
[teachers]
pera    Petar Petrovic
mika    Mika Mikic
laza    Lazar Lazarevic
```

3.2.3 Секција [*groups*]

Ова секција служи за навођење списка свих одељења, тј. група ученика, односно студената. За сваку групу наводи се идентификатор, а затим након табулатора, пуна ознака групе. Идентификатори морају бити јединствени, па их због тога апликација аутоматски генерише и нису познати крајњем кориснику. Сви ученици/студенти у оквиру једне групе имају апсолутно идентичан распоред. Уколико се неко одељење дели на подгрупе које имају различит распоред за неке предмете, свака подгрупа се појединачно наводи, тј. све се посматрају као група за себе.

```
[groups]
2mv     Druga godina, smerovi M, N i V
11a     Odeljenje I1, grupa A
11b     Odeljenje I1, grupa B
```

3.2.4 Секција [*subjects*]

Ова секција служи за навођење свих предмета. За сваки предмет наводи се идентификатор, а затим након табулатора, пуни назив. Идентификатори морају бити јединствени, па их због тога апликација аутоматски генерише и нису познати крајњем кориснику.

[subjects]
mat Matematika
gk.v Gradjevinske konstrukcije (vezbe)
gk.p Gradjevinske konstrukcije (predavanja)

3.2.5 Секција [*rooms*]

Ова секција служи за навођење списка свих сала, тј. учионица. За сваку салу наводи се идентификатор, а затим након табулатора, пуни назив. Идентификатори морају бити јединствени, па их због тога апликација аутоматски генерише и нису познати крајњем кориснику.

[rooms]
706 Sala broj 706 – 4. Sprat
U11/21 Ucionica odeljenja I1 I II1

3.2.6 Секција [*lessons*]

Ова секција представља поделу наставе. Основни елемент поделе наставе је наставни час (енг. *Lesson*). У оквиру ове секције има неколико услова који морају бити испуњени:

- Час може да држи један или више наставника (у пракси то је најчешће само један, али је остављена могућност за више наставника).
- Час похађа једна или више група ученика или студената истовремено.
- Час се одржава из тачно једног предмета.
- Час има своје трајање, изражено у броју термина. Специјално, често се дешава да се наставни часови из неког предмета распоређују више пута у току недеље, те се они у оквиру спецификације наводе одвојено.
- Час се одвија у тачно једној од понуђених сала.

Линије у оквиру ове секције имају 5 колона раздвојених табулаторима:

- Идентификатори наставника (одвојених запетом уколико их има више од једног)
- Идентификатори група (одвојених запетом уколико их има више од једне)
- Идентификатор предмета
- Фонд часова за сваки од дана
- Идентификатори сала у којима је могуће одржавати час (одвојених запетом уколико их има више од једне). На крају ће бити изабрана једна.

[lessons]
mika 11a, 11b mat 2, 1, 1 U11/21
pera 11a gk.v 3 U11/21, 706
pera 11b gk.v 3 U11/21, 706
pera, laza 11a, 11b gk.p 2 U11/21

У претходно наведеном примеру, задато је да наставник са идентификатором мика држи час из предмета са идентификатором мат (математика) заједно групама са идентификаторима *11a* и *11b* (тј. целом одељењу I1) три пута недељно: једном двочас који представља два везана термина и два пута недељно по један час који су појединачни термини. Ови часови се одржавају у учионици са идентификатором *U11/21* (матичној учионици одељења I1).

Наставник са идентификатором *pera* држи час из предмета са идентификатором gk.v (вежбе из грађевинских конструкција) посебно групи са идентификатором 11a (тј. првој половини одељења I1) у трајању од три термина, а посебно групи са идентификатором 11b (тј. другој половини одељења I1) опет у трајању од три термина. Ови часови се одржавају било у учионици са идентификатором U11/21, било у учионици са идентификатором 706.

Наставници са идентификаторима pera и laza заједнички држе предмет са идентификатором gk.p (предавања из грађевинских конструкција) истовремено групама са идентификаторима *11a* и *11b* (тј. целом одељењу I1) једном, у трајању од два термина, у учионици са идентификатором U11/21.

Услови итенгритета ове секције захтевају да сваки идентификатор наставника буде раније дефинисан у секцији [teachers], да сваки идентификатор групе буде раније дефинисан у секцији [groups], да сваки идентификатор предмета буде претходно дефинисан у секцији [subjects] и да сваки идентификатор сале буде раније дефинисан у секцији [rooms].

3.2.7 Секција [*availability*]

Сваки термин за наставника, групу, односно салу може бити обичан (већина термина би требало да буде овакво), забрањен, непожељан, обавезан или пожељан.

- **Забрањени термини** – Гарантује се да ниједан час неће бити распоређен у забрањене термине
- **Непожељни термини** – Тежи се да што мање часова буде распоређено у непожељне термине, али се за разлику од забрањених термина не гарантује да часови никада неће бити распоређени у њих
- **Пожељни термини** – Гледа се да се што више часова распореди у ове термине, али се за разлику од обавезних термина, не гарантује да ће часови увек бити распоређени у њих.
- **Обавезни термини** – Гарантује се да ће у обавезним терминима бити распоређени неки часови
- **Обични термини** – Никакви специјални услови нису везани за ове термине (свеједно је да ли се настава распоређује у њих)

У овој секцији наводе се забрањени, непожељни, пожељни и обавезни термини за одређене наставнике, групе или сале. Сви ненаведени термини сматрају се обичним. За наставнике, групе и сале који нису наведени у оквиру ове секције сви термини се сматрају обичним.

Линије у овој секцији садрже 5 колона раздвојених табулаторима:

- Идентификатори наставника, група и сала на које се ово ограничење односи
- Забрањени термини
- Непожељни термини
- Пожељни термини
- Обавезни термини

Уколико се у више линија нађу ограничења везана за истог наставника или групу, сва се ограничења сматрају релевантним, тј. посматра се унија свих наведених ограничења.

```
[availability]
//          zabranjeni          nepozeljni                  pozeljni    obavezni
pera        pon_1, pon_2, pon_3 uto_1, sre_1, cet_1, pet_1
11a, 11b    pet_5                                                       pon_1, uto_1
706         sre_3
```

У наведеном примеру забрањено је да наставник са идентификатором pera ради понедељком прва три часа, а први часови осталих дана су му означени као непожељни. Групе са идентификаторима 11a и 11b имају забрањен петак 5. час и обавезне прве часове понедељком и уторком. Коришћење сале 706 је забрањено средом 3. час.

Услови интегритета ове секције захтевају да сваки идентификатор наставника буде раније дефинисан у секцији [teachers], да сваки идентификатор групе буде раније дефинисан у секцији [groups] и да сваки идентификатор сале буде раније дефинисан у секцији [rooms].

3.2.8 Секција [*num_days*]

У овој секцији наводе се ограничења на број радних дана наставника или група. Ограничења се задају у облику бројева *min*, *max* и *opt*. Гарантује се да ће број радних дана бити у интервалу [*min*, *max*], при чему се тежи да број радних дана буде што ближе вредности *opt*. Спецификација се наводи у 4 колоне. Прва садржи идентификаторе наставника, односно група, друга минимуме, трећа оптимуме и четврта максимуме.

```
[num_days]
//      min     opt     max
pera    2       2       3
mika    2       3       4
11a     5       5       5
```

Услови интегритета захтевају да је $1 \leq min \leq opt \leq max \leq \#_d$ где је $\#_d$ укупан број радних дана задатих у секцији [*periods*]. Такође се сви идентификатори наставника и група морају јавити у одговарајућим секцијама [*teachers*] односно [*groups*].

3.2.9 Секција [*idles*]

Паузе у току наставе настају у ситуацији када наставник или група нема часове у неком термину, а има часове у том дану пре и после тог термина. На пример, ако наставник има у понедељак трећи час, затим нема четврти и пети, па има шести, кажемо да има двочас паузе у том дану.

У овој секцији наводе се ограничења на број и врсту пауза за наставнике, односно групе. Спецификација се задаје у четири колоне раздвојене табулаторима:

- У првој колони се наводе идентификатори наставника и група на које се ограничења односе. Специјални идентификатори !*TEACHERS* и !*GROUPS* служе да се наведе да се ограничење односи на све наставнике, тј. групе за које услови нису задати у посебним редовима (када се ограничења наведу за појединачног наставника или групу, на њега се више не односе општа ограничења).
- У другој колони се наводи максимална дужина паузе у току једног дана (изражена у броју термина).
- У трећој колони се наводи да ли је дозвољено да у току једног дана постоје вишеструке паузе (нпр. има први, нема други, има трећи, нема четврти, има пети час).
- У четвртој колони се наводи број дана у којима постоје паузе.

```
[idles]
//          max   multiple     days
!TEACHERS   2     0            3
!GROUPS     0     0            0
pera        1     1            2
2mnv        3     0            1
```

У претходном примеру наведено је да наставници имају највише двочас паузе. Забрањене су им вишеструке паузе у току дана и највише 3 радна дана могу имати паузе. Групама су забрањене било какве паузе. Специјално, наставнику са идентификатором *pera* дозвољено је да има паузе дужине највише један час, дозвољене су му вишеструке паузе у току једног дана и број дана са паузама му је највише 2. Даље, групи са идентификатором *2mnv* допуштено је да пауза буде највише у једном дану, али је дужина паузе највише 3 часа.

Услови интегритета захтевају да се сви наведени идентификатори јављају у одговарајућим секцијама, као и да не постоје две линије које описују ограничења за истог наставника, односно групу. Вредности у колони *multiple* су логичке (1 – дозвољено, 0 – забрањено).

3.2.10 Секција [*load*]

У овој секцији се задаје број термина у којима наставник, тј. група има наставу у току једног дана (укључујући и часове паузе). Спецификација се задаје у три колоне. У првој колони се наводе идентификатори наставника или група на које се ограничења односе. Специјални идентификатори !*TEACHERS* и !*GROUPS* служе да се наведе да се ограничење односи на све наставнике, тј. групе за које услови нису задати у посебним редовима (када се ограничења наведу за појединачног наставника или групу, на њега се више не односе општа ограничења). У другој колони се наводи најмањи број термина које наставник покрива у току једног дана, а у другој највећи број таквих термина.

```
[load]
//          min   max
!GROUPS     2     7
!TEACHERS   2     6
pera        1     6
```

У овом примеру групе имају између два и седам часова дневно, наставници између два и шест, док специјално, наставник са идентификатором *pera* има између једног и шест часова у току дана.

3.3 Формат записа распореда часова (*.*ttbl*)

Направљени распоред часова се записује у текстуалној датотеци која се састоји од 7 колона које у многоме подсећају на секцију *lessons* у оквиру формата *.*tts*:

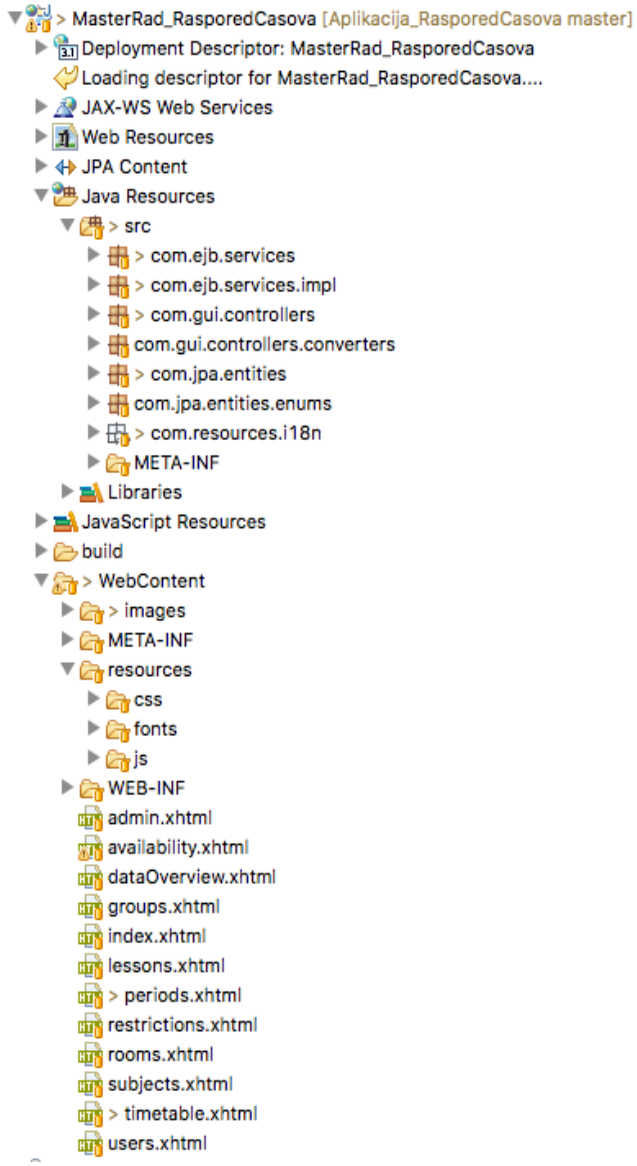
- У првој колони се налазе идентификатори професора који држе одговарајући час.
- У другој идентификатори група које час слушају.
- У трећој идентификатори предмета.
- У четвртој дужина трајања часа.
- У петој сала у којој се час одржава.
- У шестој дан у коме се час одржава.
- У седмој термин у коме час почиње.

```
[timetable]
mika        11a, 11b    mat     2    U11/21    pon    1
mika        11a, 11b    mat     1    U11/21    uto    2
mika        11a, 11b    mat     1    U11/21    sre    4
pera        11a         gk.v    3    706       pon    3
pera        11b         gk.v    3    U11/21    uto    3
pera, laza  11a, 11b    gk.p    2    U11/21    sre    2
```

У примеру групе *11a, 11b* имају понедељак први и други час код наставника *mika* у сали *U11/21* из предмета *mat*. Слично, на пример, Група *11b* има код наставника *pera* уторком трећи, четврти и пети час предмет *gk.v* у сали 706.

4 Апликација за креирање распореда часова

Апликација за креирање рапореда часова служи за прикупљање и складиштење неопходних података (на пример, поделе наставе, жеља наставног особља итд.) од стране корисника апликације, а затим врши приказивање готовог распореда и проверу свих услова његове коректности након самог креирања, које се одвија у позадини од стране независног готовог програма.



Слика 13: Структура пројекта

Пројекат је развијен у развојном окружењу *Eclipse Neon*. Користи се *Java jdk1.8.0_111* као и *WildFly 10.x* за апликативни сервер.
Коришћен је *Git* систем за контролисање верзија, као и *GitHub* платформа за складиштење кода.

Пакет *com.ejb.services* садржи скуп интерфејса пословне логике чије су методе имплементиране у зрнима сесије у оквиру пакета *com.ejb.services.impl*. Методе декларисане у интерфејсу се користе од стране клијента.

Пакет *com.jpa.entities* садржи скуп свих ентитета дефинисаних у бази података, док пакет *com.gui.controllers* обухвата логику зрна подршке *JSF* апликације.

Приликом имплементације добра је пракса да се све лабеле и поруке дефинишу на једном месту. Ово је посебно корисно уколико апликација подрзумева коришћење више језика. За ову сврху коришћен је пакет *com.resources.i18n*.

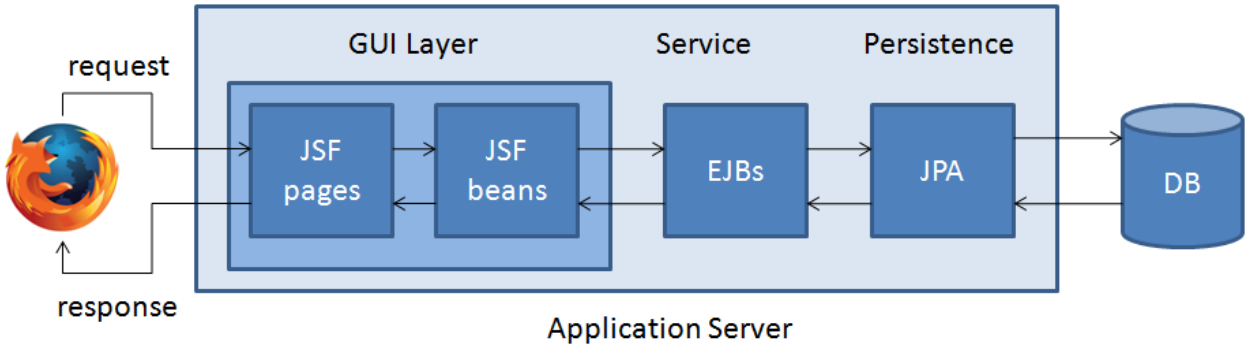
За управљање изгледном компоненти корисничког интерфејса коришћена је библиотека *Bootstrap*.

4.1 Архитектура система

Коришћењем технологија описаних у претходним поглављима реализована је апликација за креирање распореда часова у складу са модел-поглед-контролер архитектуром.

Презентациони слој апликације чине *JSF* странице, на којима су као основни елементи интерфејса коришћене *JSF* и *RichFaces* компоненте. Реализоване странице омогућавају кориснику основне функционалности система као што су излиставање, унос, измена, брисање и уопште управљање подацима који се тичу смерова, професора, учионица, група, предмета, термина и семестара. Поред тога, на располагању су и функционалности уношења, ажурирања и брисања различитих рестрикција које се односе засебно на професоре, групе, учионице или саме термине.

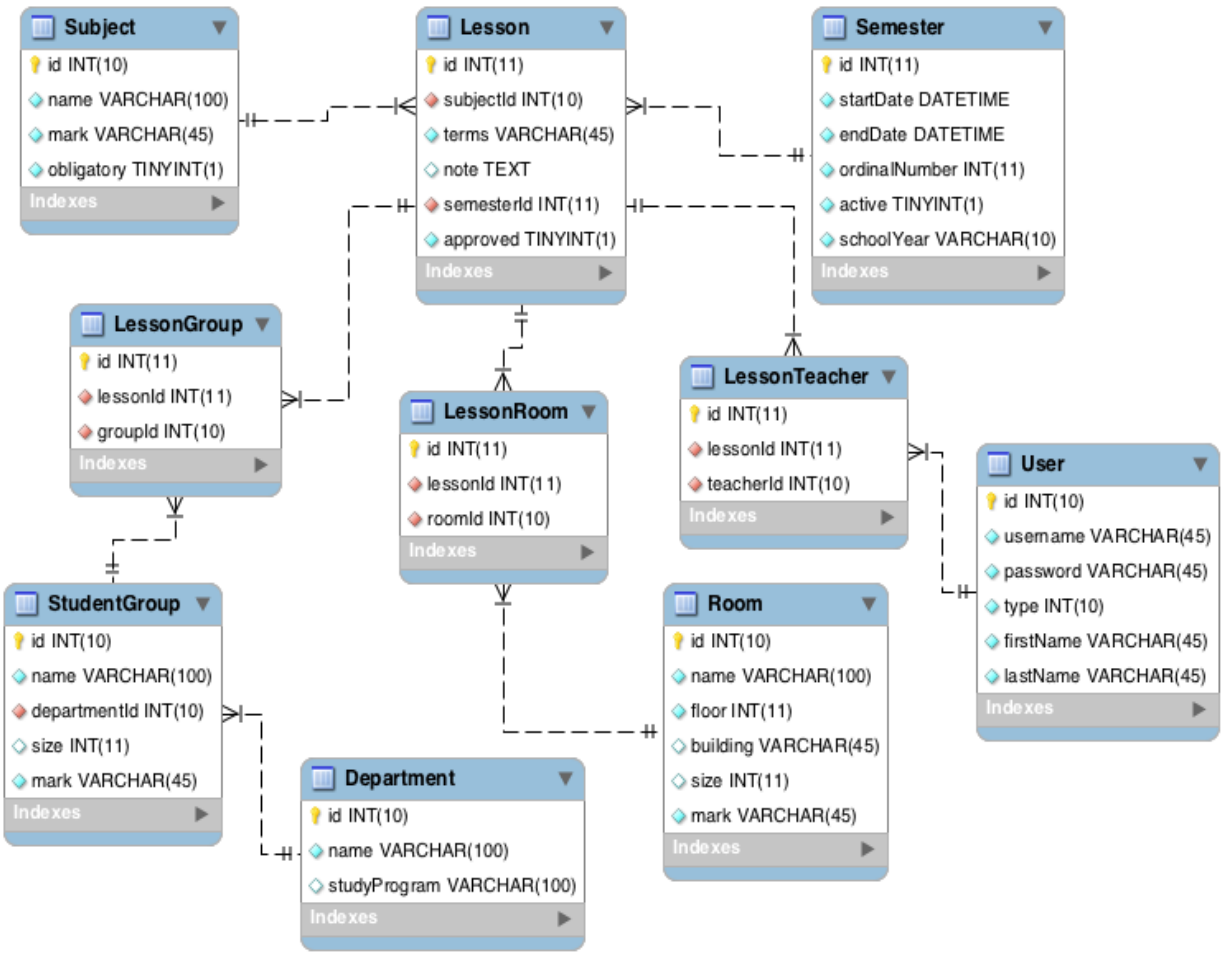
Ajax захтеве, које компонента корисничког интерфејса прослеђује серверској страни, прихвата и обрађује контролерски слој апликације, односно зрна подршке. Сам слој података представљен је групом ентитета, а за сваки од њих дефинисано је засебно зрно сесије које имплементира одговарајући интерфејс пословне логике и садржи методе за унос, излиставање, брисање или ажурирање података у бази.



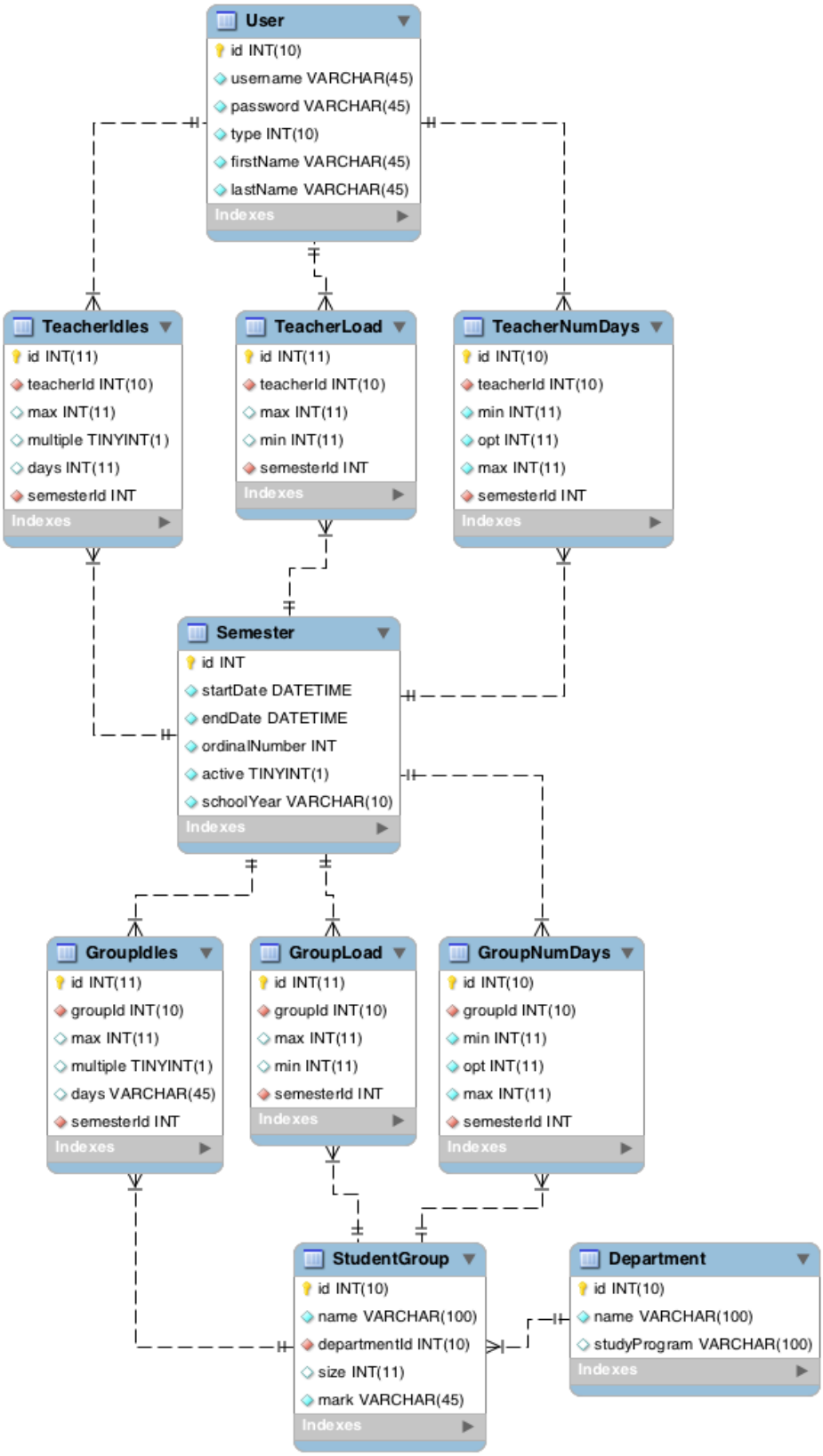
Слика 14: Архитектура система [10]

4.2 Дијаграми класа

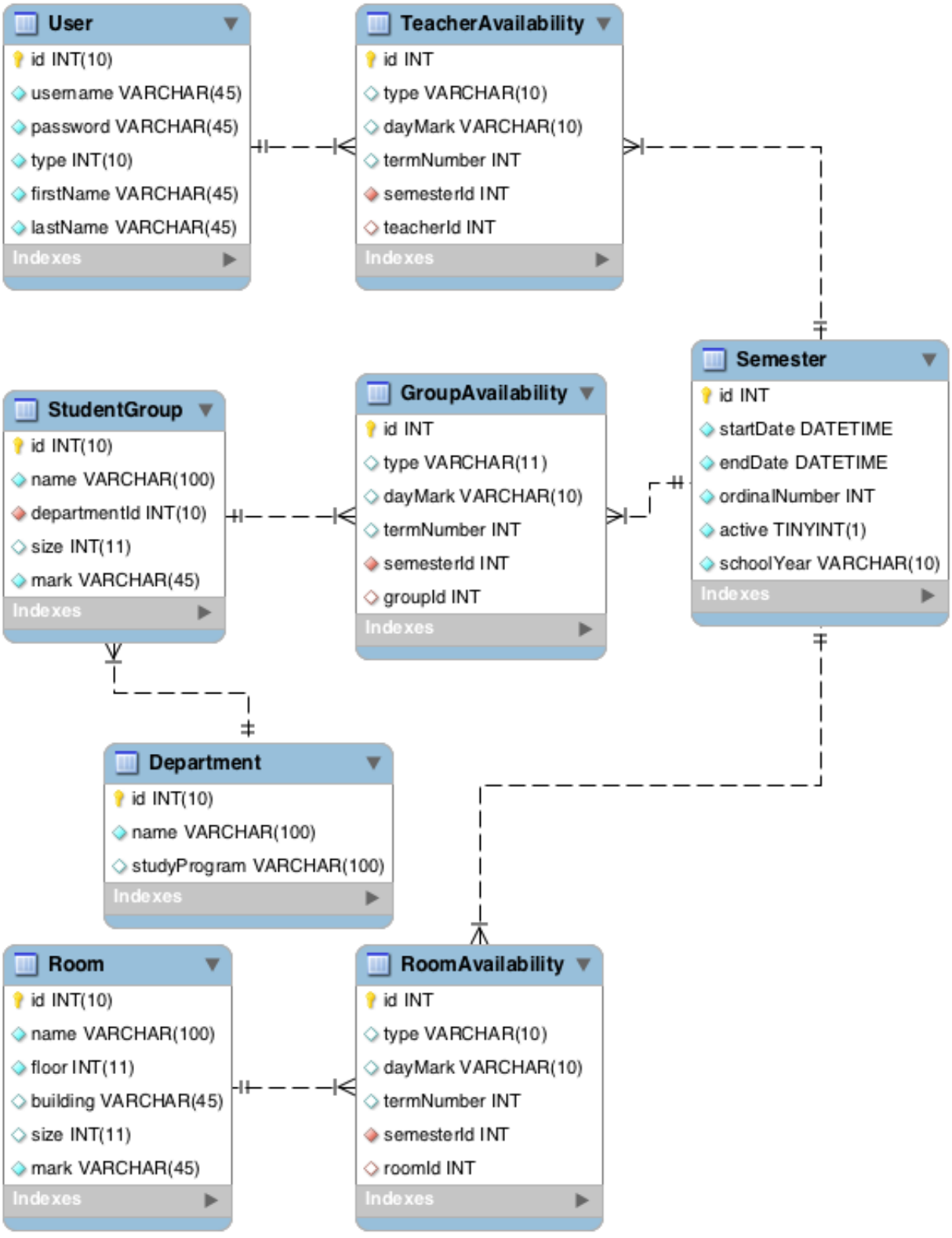
У наставку су представљени дијаграми класа апликације за креирање распореда часова. Класе су на дијаграмима груписане у целине, како би читаоцу био јаснији однос ентитета у бази података:



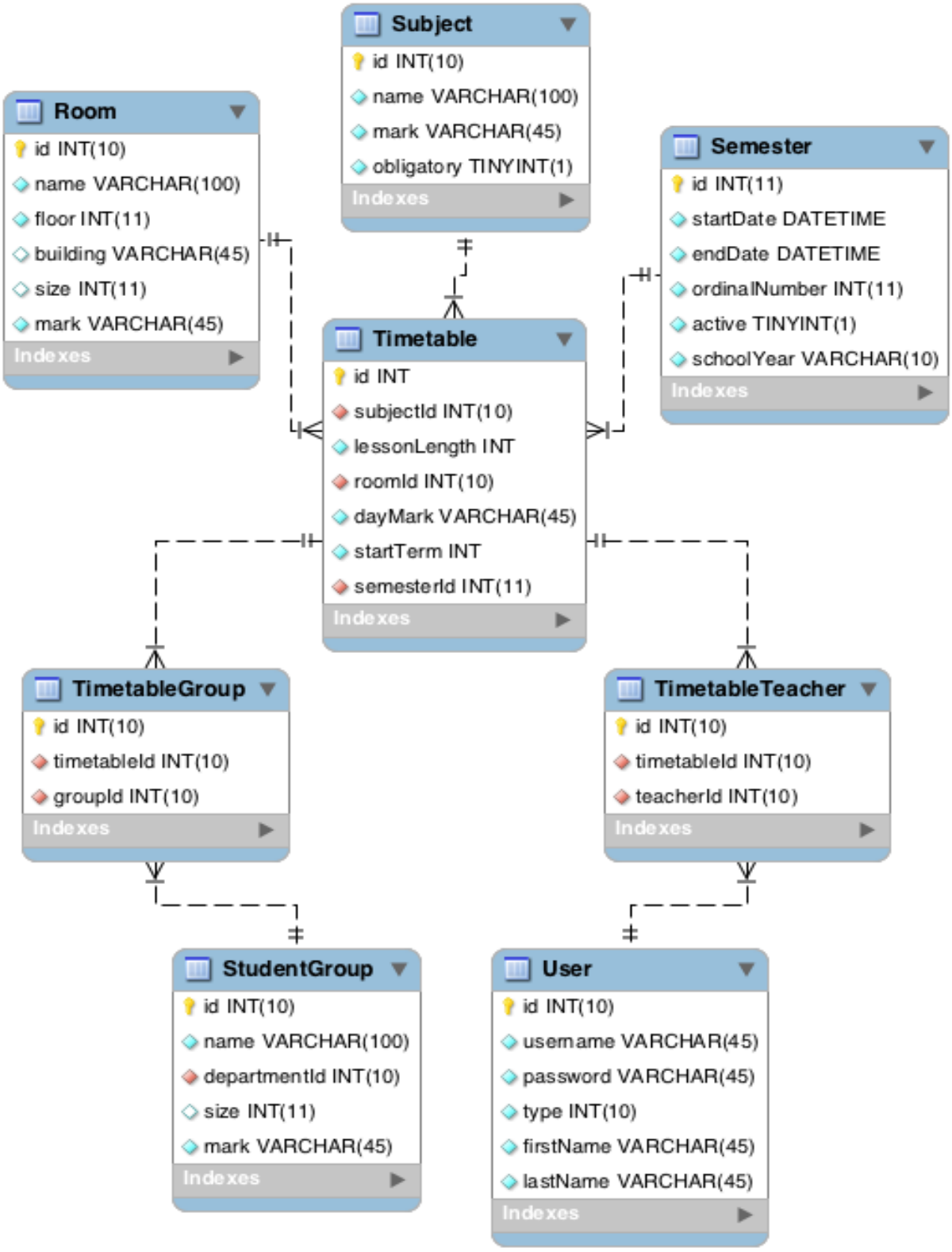
Слика 15: Дијаграм класа ентитета који учествују у дефинисању поделе наставе



Слика 16: Дијаграм класа ентитета који учествују у дефинисању рестрикција



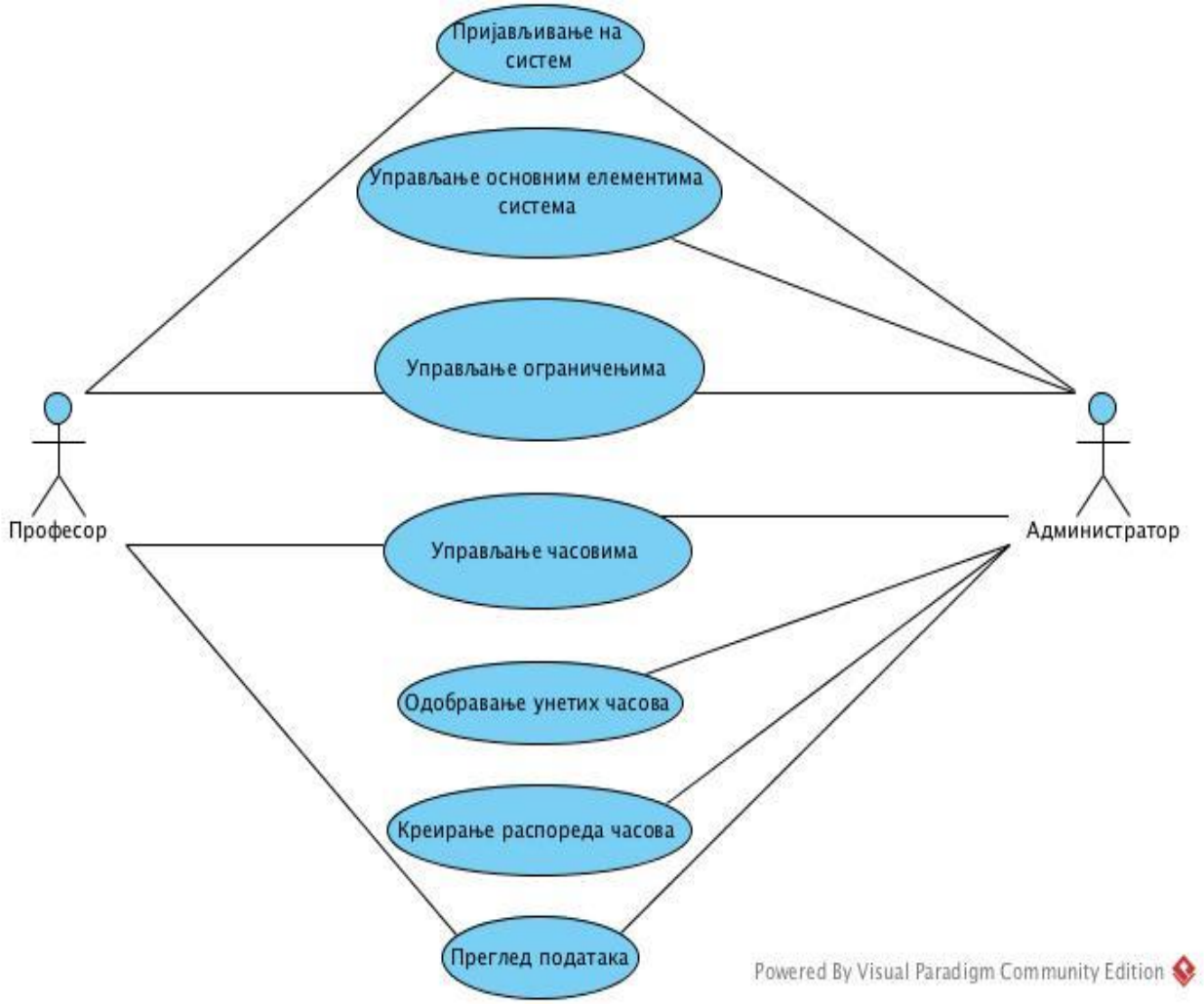
Слика 17: Дијаграм класа ентитета који учествују у дефинисању врста термина



Слика 18: Дијаграм класа ентитета који учествују у складиштењу информација о креираном распореду часова

4.3 Случајеви употребе

Пројекат је подељен у седам великих случајева употребе, а неки од њих садрже више мањих. На наредној слици приказан је дијаграм основних случајева употребе:



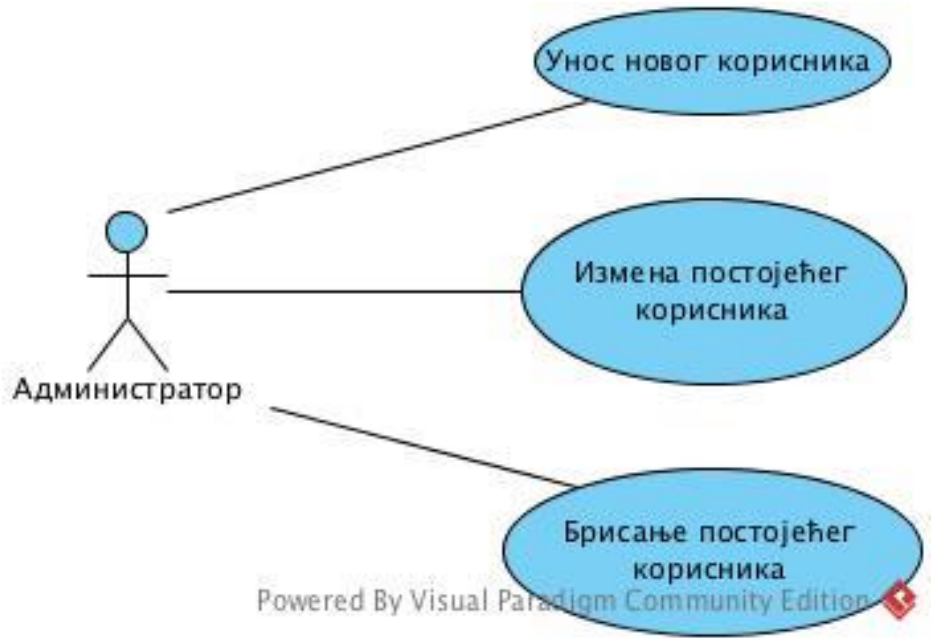
Слика 19: Дијаграм основних случајева употребе

Списак свих случајева употребе:

1. **Пријављивање корисника на систем**
    - Корисник може бити пријављен као професор или администратор.
2. **Управљање основним елеменатима система** (администратор)
    - Унос, измена и брисање корисника. Измена корисничког имена није могућа. Брисањем корисника бришу се и сва његова ограничења.
    - Унос, измена и брисање група. Брисањем групе бришу се и сва њена ограничења.
    - Унос, измена и брисање учионица. Брисањем учионице бришу се и сва њена ограничења.
    - Унос, измена и брисање предмета.
    - Унос и брисање семестара. Активација и деактивација семестара. У сваком тренутку може бити само један или ниједан семестар активан.
    - Унос и брисање смерова.
    - Унос, измена и брисање термина.
3. **Управљање ограничењима** (администратор и професор)
    - Унос, измена и брисање ограничења на број радних дана – у овом случају администратор је у могућности да управља ограничењима како за групе тако и за професоре, док професор може управљати само ограничењима која се тичу искључиво њега самог.
    - Унос, измена и брисање ограничења на број пауза у току наставе – исти услови важе као у претходном случају.
    - Унос, измена и брисање ограничења на број темина у току дана – исти услови важе као у претходна два случаја.
    - Унос, измена и брисање врста термина – у овом случају такође администратор може контролисати унос како за професоре, тако и за групе или учионице. Професор може уносити, мењати или брисати искључиво своје термине.
4. **Управљање часовима** (администратор и професор)
    - Унос, измену и брисање часова могу обављати и администратор и професор.
5. **Одобравање унетих часова** (администратор)
    - Нопходно је да администратор одобри часове унете од стране професора како би били узети у обзир приликом креирања распореда часова.
6. **Креирање распореда часова** (администратор)
    - Искључиво администратор може креирати распоред часова.
7. **Преглед података** (администратор и професор)
    - Сваки корисник система може прегледати, извршити сортирање или претрагу основних елемената система. Администратор може прегледати све рестрикције које се тичу професора, група или учионица, док професор може видети само оне рестрикције које се тичу њега самог. Такође преглед распореда часова за одређени семестар, уколико је креиран, могу прегледати сви корисници система.

У наставку ће неки од случајева употребе бити детаљније објашњени. Прво ће се наводити дијаграм са појединим случајевима употребе, а затим ће сви случајеви употребе са датог дијаграма бити објашњени.

4.3.1 Управљање корисницима



Слика 20: Случајеви употребе – Управљање корисницима

Опис: Регистровани корисник типа администратор врши унос новог или измену (брисање) већ постојећег корисника.

Учесници: Администратор

Предуслов: Корисник је успешно улогован на систем и има привилегије администратора. Уколико се врши унос новог корисника, неопходно је да корисничко име буде јединствено.

Главни ток:

1. Администратор се логује на систем уносећи корисничко име и лозинку.
2. Администратор попуњава сва поља на форми за упис новог корисника.
3. Администратор проналази унетог корисника у листи свих корисника.
4. Администратор врши измену података изабраног корисника.
5. Администратор пронализи измењеног корисника у листи корисника.
6. Администратор брише одабраног корисника.

Алтернативни ток:

1. Погрешно корисничко име или лозинка. Повратак на корак 1 главног тока.
2. Изабрано постојеће корисничко име. Повратак на корак 2 главног тока.
3. Непопуњена сва поља форме. Повратак на корак 2 или корак 4 гавног тока.

4.3.2 Управљање ограничењима

Слика 21: Случајеви употребе – Управљање ограничењима

Опис: Регистровани корисник врши унос новог, односно измену (брисање) постојећег ограничења.

Учесници: Регистровани корисник

Предуслов: Корисник је успешно улогован на систем и има привилегије професора или администратора. Уколико је регистровани корисник професор омогућено је управљање само личним ограничењима. Уколико је администратор може управљати свим ограничењима.

Главни ток:

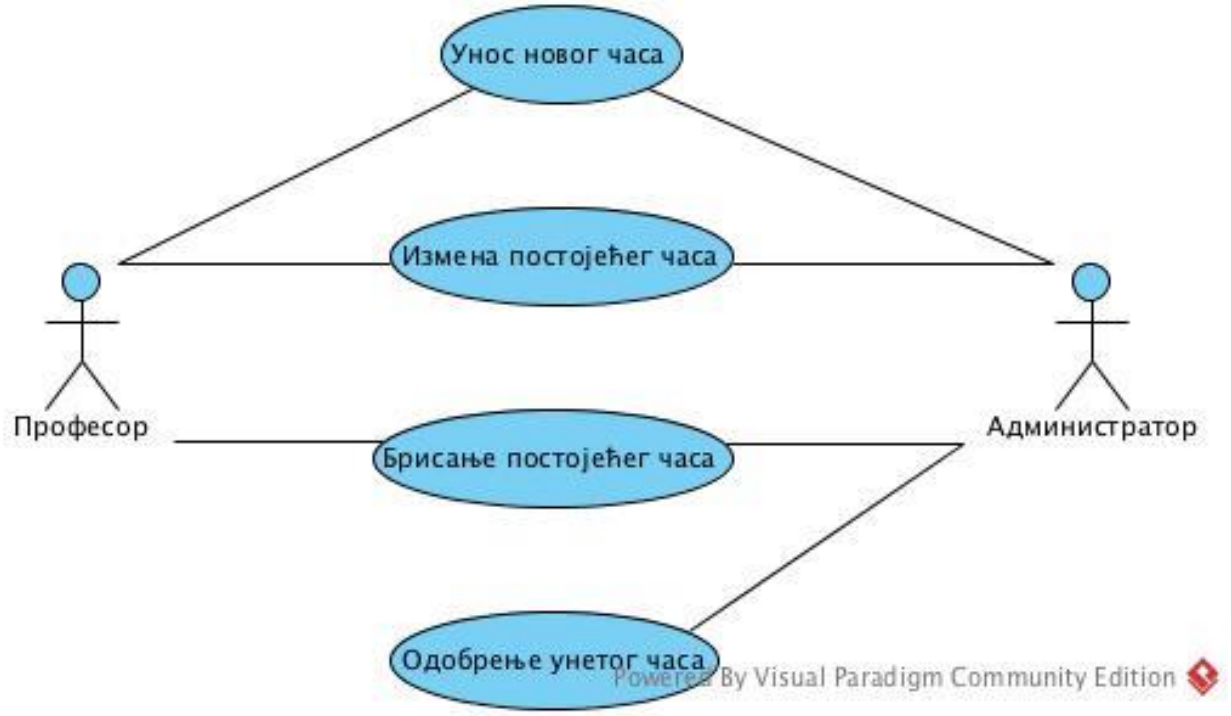
1. Корисник се логује на систем уносећи корисничко име и лозинку.
2. Регистровани корисник попуњава сва поља на форми за унос новог ограничења.
3. Регистровани корисник проналази унето ограничење у листи ограничења одговарајућег типа.

4. Регистровани корисник врши измену вредности изабраног ограничења.
5. Регистровани корисник пронализи измењено ограничење у листи ограничења одговарајућег типа.
6. Регистровани корисник брише одабрано ограничење.

Алтернативни ток:

1. Погрешно корисничко име или лозинка. Повратак на корак 1 главног тока.
2. Непопуњена сва поља форме. Повратак на корак 2 или корак 4 гавног тока.

4.3.3 Управљање часовима и одобравање истих



Слика 22: Случајеви употребе – Управљање часовима

Опис: Регистровани корисник врши унос новог, односно измену (брисање) постојећег часа. Уколико регистровани корисник има привилегије администратора омогућено му је одобравање часова унетих од стране професора.

Учесници: Регистровани корисник

Предуслов: Корисник је успешно улогован на систем и има привилегије професора или администратора.

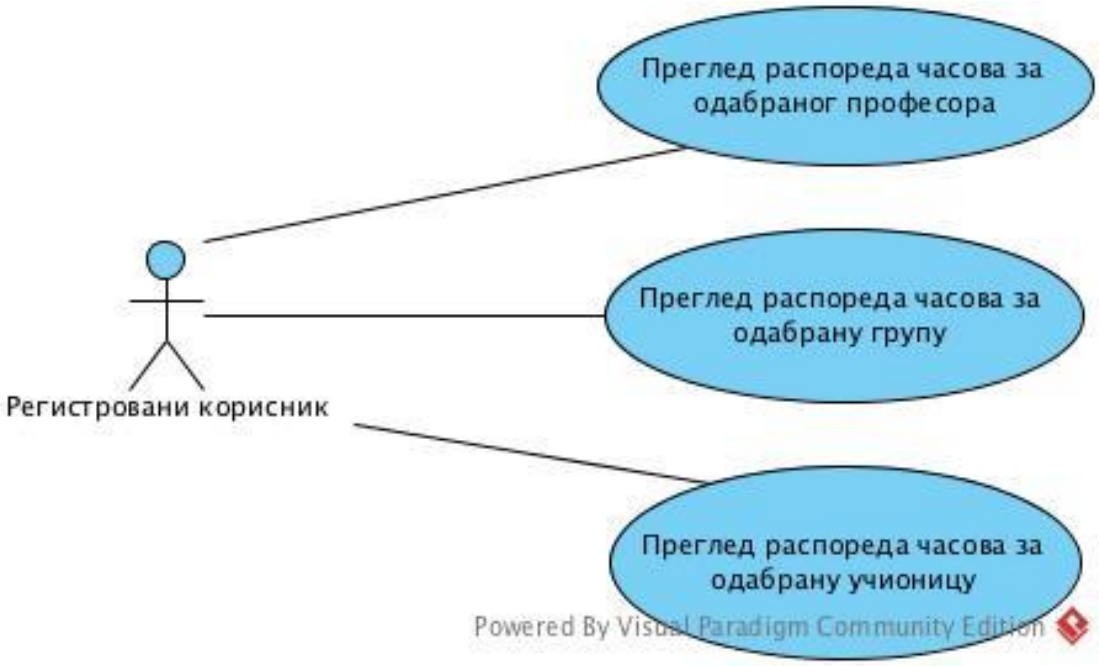
Главни ток:

1. Корисник се логује на систем уносећи корисничко име и лозинку.
2. Регистровани корисник попуњава сва поља на форми за унос новог часа. Уколико регистровани корисник има привилегије администратора, унети час ће аутоматски бити одобрен. У супротном неопходно је одобрење администратора како би час био обухваћен приликом креирања распореда.
3. Регистровани корисник проналази унети час у листи часова на основу одабраног предмета.
4. Регистровани корисник врши измену вредности изабраног часа.
5. Регистровани корисник пронализи измењени час у листи часова на основу одабраног предмета.
6. Регистровани корисник брише одабрани час.
7. Администратор врши одобрења часова који су унети од стране професора.

Алтернативни ток:

1. Погрешно корисничко име или лозинка. Повратак на корак 1 главног тока.
2. Непопуњена сва поља форме. Повратак на корак 2, односно корак 4 гавног тока.
3. Уколико час није одобрен, неопходан је поновни унос или измена истог. Повратак на корак 2, односно корак 4 главног тока.

4.3.4 Преглед распореда часова



Слика 23: Случајеви употребе – Преглед распореда часова

Опис: Регистровани корисник врши преглед распореда часова на основу различитих критеријума.

Учесници: Регистровани корисник

Предуслов: Корисник је успешно регистрован на систем и има привилегије професора или администратора.

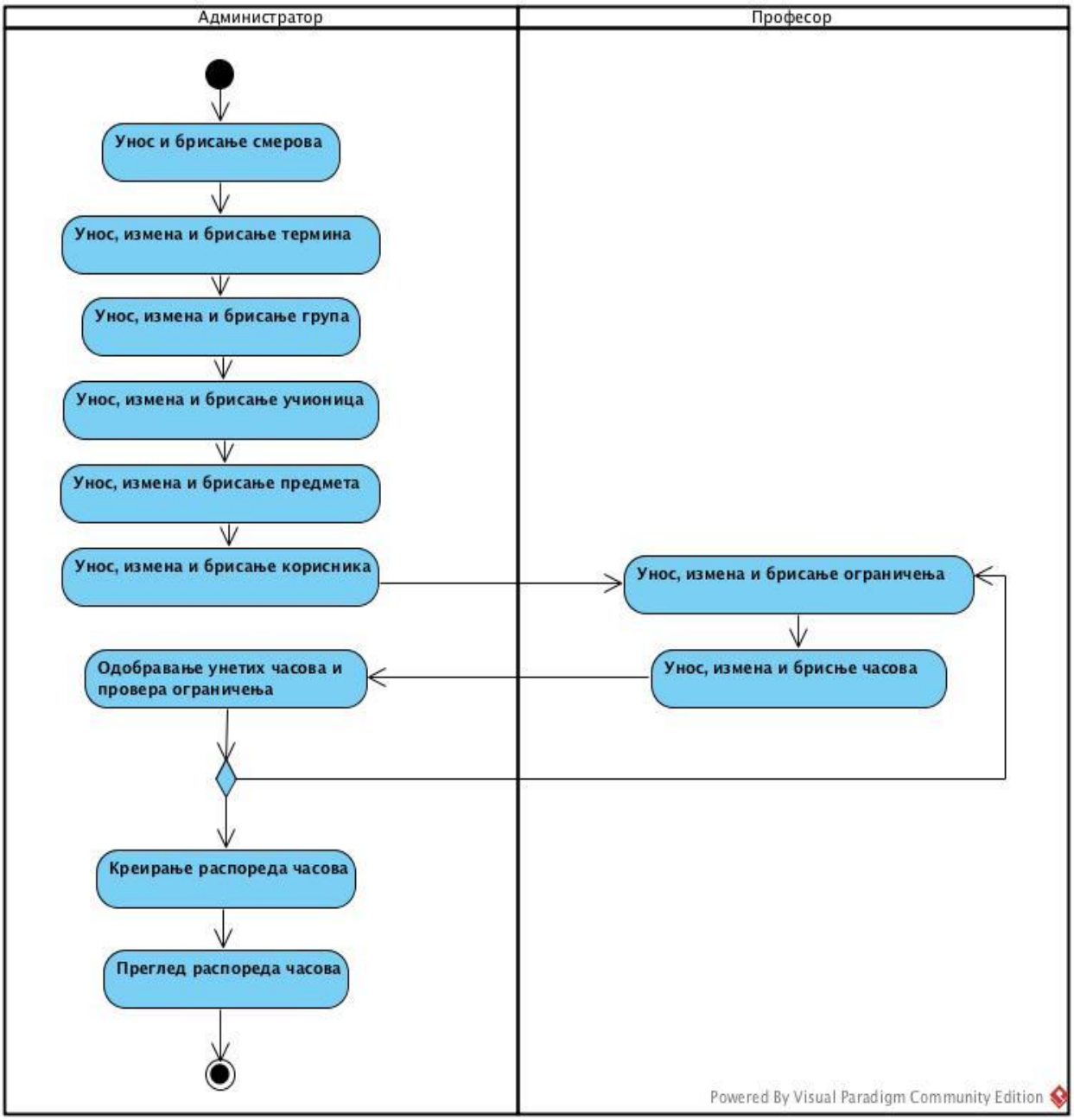
Главни ток:

1. Корисник се логује на систем уносећи корисничко име и лозинку.
2. Регистровани корисник врши одабир критеријума на основу којих ће бити излистан распоред часова. Могућ је одабир семестра, где ће у супротном бити излистан распоред за тренутно активни семестар, као и одабир корисника, групе или учионице за које треба излистати распоред.

Алтернативни ток:

1. Погрешно корисничко име или лозинка. Повратак на корак 1 главног тока.

4.4 Дијаграм активности



Слика 24: Дијаграм активности уноса и прегледа елемената система

5 Закључак

У овом раду анализиране су технологије коришћене за израду корисничког интерфејса софтвера за аутоматско распоређивање часова, чија је имплементација такође детаљно представљена.

Кориснички интерфејс омогућава кориснику уношење свих релевантних информација потребних за креирање распореда часова. Ове информације прослеђују се систему *ArgoTimetabling* који као резултат враћа датотеку која се од стране компоненте за приказ распореда парсира и генерише документе распореда појединачно за све професоре, групе и учионице.

Развој апликације је у великој мери олакшан коришћењем технологија описаним у овом раду. Комбинација оквира *JSF* и *RichFaces* нам пружа велики избор компоненти па у сваком тренутку можемо наћи решење у некој од њих приликом развоја сложених и модерних апликација. Такође, спецификације *EJB* и *JPA* су се показале као врло једноставне за коришћење и манипулисање подацима у бази података.

Иако је у апликацији коришћен мањи део укупне функционалности које неке од споменутих технологија нуде, апликација представља добар основ за њихово учење и разумевање.

6 Литература

1. Бранко Милосављевић, Милан Видаковић. Јава и интернет програмирање. Универзитет у Новом Саду, Факултет техничких наука. Нови Сад, 2010.
2. JSF 2.0 specification [На мрежи, датум последњег приступа 15.08.2017.] https://jcp.org/aboutJava/communityprocess/final/jsr314/index.html
3. JSF TutorialsPoint [На мрежи, датум последњег приступа 15.05.2017.] https://www.tutorialspoint.com/jsf/
4. RichFaces официјелна веб-страница http://richfaces.jboss.org [На мрежи, датум последњег приступа 15.05.2017.]
5. RichFaces Showcase [На мрежи, датум последњег приступа 15.05.2017.] http://showcase.richfaces.org
6. PluralSight JPA courses [На мрежи, датум последњег приступа 15.05.2017.] https://app.pluralsight.com/library/courses/java-persistence-api-21
7. Balaram, MVC1 vs. MVC2 [На мрежи, датум последњег приступа 15.05.2017.] http://completejavaj2ee.blogspot.rs/2012/03/mvc-1-vs-mvc2.html
8. The Java EE 5 Tutorial [На мрежи, датум последњег приступа 15.05.2017.] http://docs.oracle.com/javaee/5/tutorial/doc/bnara.html
9. Проф. Др. Бошко Николић. Електротехнички факултет, Универзитета у Београду. Програмирање интернет апликација. [На мрежи, датум последњег приступа 16.05.2017.] http://rti.etf.bg.ac.rs/rti/ir4pia/
10. theJavaGeek.com, JSF-EJB-JPA application [На мрежи, датум последњег приступа 15.05.2017.] http://www.thejavageek.com/2015/01/09/creating-jsf-ejb-jpa-application-using-eclipse-wildfly/