UNIVERZITET „UNION“
RAČUNARSKI FAKULTET
Knez Mihailova 6/VI
11000 BEOGRAD

Broj:

Datum:
30.12.2010.

UNIVERZITET UNION
RAČUNARSKI FAKULTET
BEOGRAD
Programiranje

DIPLOMSKI RAD

Kandidat: **Igor Ilijašević**
Broj indeksa: 27/05

Tema rada: **Realizacija Webmail aplikacije upotrebom RichFaces i Hibernate okruženja**

Mentor rada: **docent dr Milan Vidaković**

Beograd, 30.12.2010.

ZADATAK DIPLOMSKOG-MASTER RADA

Zadatak ovog rada je upoznavanje sa tehnologijama za razvoj *web* aplikacija, koje uključuju JSF (Java Server Faces), Facelets, AJAX (Asynchronous JavaScript And XML), RichFaces, JPA (Java Persistence API), Hibernate i JavaMail API, kao i implementacija Webmail aplikacije upotrebom navedenih tehnologija. Specifikacija realizovanog sistema predstavljena je UML dijagramima. Implementacija sistema izvršena je u programskom jeziku Java, u okviru Eclipse programskog okruženja.

SADRŽAJ

1. UVOD 5
1.1. JSF 5
1.1.1. Osnovne karakteristike 5
1.1.2. Model komponenti korisničkog interfejsa 6
1.1.2.1. Klase komponenti korisničkog interfejsa 6
1.1.2.2. *Rendering model* 7
1.1.2.3. Osluškivači događaja 7
1.1.2.4. Konvertori 8
1.1.2.5. Validatori 8
1.1.3. Biblioteke JSF tag-ova 8
1.1.4. *Backing bean*-ovi 9
1.1.5. Navigaciona pravila 10
1.1.6. Konfiguracioni dokument 11
1.1.7. Faze JSF aplikacije 12
1.2. Facelets 13
1.3. Ajax 14
1.4. RichFaces 14
1.4.1. Tok razvoja 14
1.4.2. Osnovne karakteristike 15
1.4.3. *Skinnability* 16
1.4.4. Osnovne komponente 17
1.4.4.1. <a4j:commandButton> i <a4j:commandLink> 17
1.4.4.2. <a4j:support> 17
1.4.4.3. <rich:dataTable> 18
1.4.4.4. <rich:fileUpload> 18
1.4.4.5. <rich:modalPanel> 19
1.4.4.6. <rich:tree> 19
1.5. JPA 20
1.5.1. Objektno-relaciono mapiranje 20
1.5.1.1. Osnovne anotacije 21
1.5.1.2. Mapiranje veza 21
1.5.1.3. Mapiranje primarnih ključeva 22
1.5.2. Rukovanje objektnim modelom 22
1.5.3. JPQL-a (Java Persistence Query Language) 23
1.6. JavaMail API 24
1.6.1. Arhitektura API-ja 24
1.6.2. *Internet Mail* 26
1.6.3. JavaMail *framework* 27
2. SPECIFIKACIJA SISTEMA 28
2.1. Dijagram slučajeva korišćenja 28
2.2. Dijagrami klasa 41
2.3. Dijagrami sekvence 57
2.3. Model baze podataka 59

3. IMPLEMENTACIJA 61
3.1. Implementacija korisničkog interfejsa 61
3.2. Implementacija na serverskoj strani 66
3.2.1. *Entity* klase 66
3.2.2. *Managed bean*-ovi 67
3.2.3. Klase za upravljanje JavaMail API-jem 71
3.3. Izgled Webmail aplikacije 73
4. ZAKLJUČAK 77
LITERATURA 79

1. UVOD

U ovom poglavlju će biti opisane tehnologije koje su korišćene u toku razvoja Webmail aplikacije. Pošto je Hibernate implementacija JPA tehnologije o njemu neće biti detaljnijeg objašnjenja.

1.1. JSF

JSF (JavaServer Faces)[1] je *framework* namenjen *web* aplikacijama baziranim na Java-tehnologiji. JSF pojednostavljuje razvoj korisničkog interfejsa sofisticiranih *web* aplikacija definisanjem modela komponenti korisničkog interfejsa, koji je povezan sa dobro razrađenim ciklusom obrade zahteva.

1.1.1. Osnovne karakteristike

JSF predstavlja apstrakciju postojećeg *framework*-a, koji za izgradnju web aplikacija, koje se baziraju na MVC2 (Model-View-Controller) arhitekturi, koriste Java Servlet API i JSP (JavaServer Pages). Ključne komponente tog pristupa su:

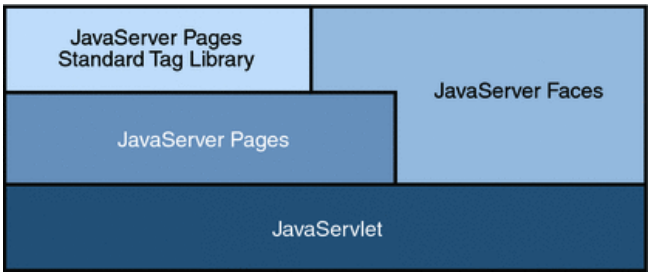
- JSP stranice – tekstualni dokumenti koji sadrže dva tipa teksta – statički sadržaj, koji se izražava u bilo kom tekst-baziranom formatu, poput HTML-a ili XML-a, i dinamički sadržaj, koji se konstruiše JSP elementima. U MVC2 arhitekturi, JSP stranice predstavljaju prezentacioni sloj (*View*).
- Servleti – Java klase koje nasleđuju *HttpServlet* klasu definisanu unutar Java Servlet API-ja, i redefinišu metode koje prihvataju objekat zahteva klijenta i konstruišu odgovor. Unutar MVC2 arhitekture, servlet predstavlja kontroler (*Controller*) koji analizira zahtev i po potrebi manipuliše modelom, izvršava poslovnu logiku i odlučuje šta će biti prikazano na prezentacionom sloju.
- JavaBeans – Java klase koje enkapsuliraju relevantne podatke unutar jednog serijalizabilnog objekta. U MVC2 arhitekturi, JavaBeans predstavljaju model podataka (*Model*).

I pored očiglednih prednosti razdvajanja prezentacije, kontrolera i podataka u JSP aplikacijama, ovim pristupom nije moguće mapirati HTTP zahteve na obrađivače događaja specifične za pojedine prezentacione komponente, niti upravljati komponentama čuvajući na serverskoj strani njihovo stanje.

JSF, sa arhitekturom zasnovanom na komponentama, pruža sofisticiranije razdvajanje između ponašanja i prezentacije datih komponenti, sakrivajući veliki deo kompleksnosti *View* sloja MVC2 arhitekture. Uopšteno gledano, osnovne komponente JSF tehnologije čine:

- API za reprezentaciju komponenti korisničkog interfejsa i rukovanje njihovim stanjem, rukovanje događajima, validaciju na serverskoj strani, konverziju podataka, definisanje navigacije između *web* stranica, te podršku internacionalizaciji.
- Dve biblioteke specifičnih JSP tag-ova za predstavu komponenti korisničkog interfejsa na JSP stranici i njihovo povezivanje sa objektima na strani servera.

U JSF aplikaciji, JSP stranice i dalje predstavljaju *View* sloj MVC2 arhitekture, ali je razvoj stranica olakšan upotrebom JSP tagova za predstavu JSF komponenti. Doduše, iako JSF uključuje dve biblioteke JSP tagova za predstavu komponenti na JSP stranici, JSP nije obavezna prezentaciona tehnologija. Pogledajmo sliku 1.1., koja prikazuje Java Web tehnologije, kao i veze između njih. Može se primetiti da JavaServlet tehnologija predstavlja temelj za sve ostale, i da je JSF smešten tačno iznad JavaServlet tehnologije.



Slika 1.1. – Java Web tehnologije

Ovakvim raslojavanjem API-ja omogućeno je nekoliko važnih slučajeva korišćenja JSF aplikacija, poput mogućnosti korišćenja druge prezentacione tehnologije, kreiranje sopstvenih komponenti direktnom upotrebom postojećih, kao i mogućnost generisanja izlaza za više tipova klijenata (*browser*-a), poput WML klijenata (Wireless Markup Language).

Sa arhitekturom zasnovanom na komponentama, uz podršku objektno-orijentisanom razvoju, JSF svakako predstavalja veliki korak napred u pojednostavljivanju i ubrzavanju razvoja *web* aplikacija.

1.1.2. Model komponenti korisničkog interfejsa

JSF komponente korisničkog interfejsa (u daljem tekstu UI komponente) su konfigurabilni i ponovo upotrebljivi elementi od kojih je sastavljen interfejs JSF aplikacija. U okviru arhitekture *framework*-a sadržani su:

- set *UIComponent* klasa za specifikaciju stanja i ponašanja UI komponenti
- *rendering* model koji definiše prikazivanje UI komponenti na različite načine
- *event-listeners* model kojim se definiše rukovanje događajima na UI komponentama
- *conversion* model kojim se definiše kako se konverter podataka registruje za UI komponentu
- *validation* model kojim se definiše kako se validator podataka registruje za UI komponentu

1.1.2.1. Klase komponenti korisničkog interfejsa

U okviru JSF framework-a se nalazi set klasa i pridruženih interfejsa koji u potpunosti opisuju funkcionalnost UI komponenti, poput čuvanja stanja komponente, održavanja referenci ka drugim objektima i rukovanje događajima. Klase komponenti su nasledive, što omogućava razvoj sopstvenih komponenti.

Sve ove klase nasleđuju klasu *UIComponentBase*, koja definiše podrazumevano stanje i ponašanje UI komponente. Pored nasleđivanja date klase, klase UI komponenti implementiraju i jedan ili više interfejsa koji definišu ponašanje seta komponenti čije klase implementiraju dati interfejs. Ovi "interfejsi ponašanja" (*behavioral interfaces*) su:

- *ActionSource* – naznačava da komponenta može izazvati događaj (*event*)
- *ActionSource2* – jednak *ActionSource*-u, uz mogućnost korišćenja objedinjenog jezika izraza (*Unified Expression Language*) pri referenciranju metoda koje obrađuju događaj
- *EditableValueHolde* – nasleđuje *ValueHolder* i specificira dodatna svojstva editabilnih komponenti
- *NamingContainer* – naznačava da sve podređene komponente moraju imati jedinstveni identifikator

- *StateHolder* – naznačava da komponenta poseduje stanje koje mora biti sačuvano između zahteva
- *ValueHolder* – naznačava da komponenta čuva lokalnu vrednost

1.1.2.2. *Rendering model*

UI komponenta predstavlja samo atribute, ponašanja i događaje vezane za datu komponentu, ali ne i način na koji će biti prikazana klijentu, te je prikaz komponente definisan na drugi način – rendererima. Ovaj pristup omogućava da se ponašanje komponente definiše jednom, i razvije više renderera, od kojih svaki definiše različit način prikaza (renderovanja) istom ili različitim klijentima. S druge strane, ovim pristupom je takođe omogućeno da se pri razvoju stranice promeni izgled jedne komponente izborom *tag*-a sa odgovarajućom kombinacijom komponente i renderera.

Render kit definiše kako se klase UI komponenti mapiraju na *tag*-ove komponenti koji su prikladni za pojedine klijente. *Render kit* sadrži set *Renderer* klasa, od kojih svaka definiše jedan način prikaza odgovarajuće komponente za koju je definisan. Unutar JSF *framework*-a, definisan je standardni HTML *render kit*, za prikaz JSF UI komponenti na HTML klijentu. Detaljnije o toj biblioteci će biti reči nešto kasnije.

1.1.2.3. Osluškivači događaja

Deo JSF *framework*-a zadužen za obradu događaja je sličan onom koji se može naći kod *Swing* komponenti, po tom što sadrži razdvojene klase događaja i interfejse osluškivača koje aplikacija može da koristi u rukovanju događajima koje je izazvala UI komponenta.

Event objekat identifikuje komponentu koja je izazvala događaj, i uz to čuva podatke o samom događaju. Da bi bila obaveštena o događaju koji se desio na određenoj komponenti, aplikacija mora sadržati implementaciju *Listener* klase, i mora datu klasu registrovati kao osluškivač za datu komponentu. Po aktiviranju komponente, poput klika dugmetom, kreira se događaj i poziva se metoda koja je registrovana kao osluškivač.

JSF podržava tri vrste događaja:

- *Action event* – nastaje aktiviranjem komponente koja implementira *ActionSource* interfejs, poput dugmadi i hiperlinkova.
- *Value-change event* – nastaje promenom vrednosti UI komponente, poput selektovanja *check box*-a.
- *Data-model event* – nastaje selekcijom reda *UIData* komponente.

Postoje dve pristupa obradi događaja unutar aplikacije. Prvi podrazumeva implementaciju *Listener* klase kao osluškivača, i registrovanje datog osluškivača ugnježdavanjem *valueChangeListener tag*-a unutar *tag*-a koji reprezentuje komponentu. Pogledajmo primer ovog pristupa:

```
<h:commandButton id="exampleButton" value="Click">
     <f:actionListener>
          type="somePackage.exampleListener"
     </f:actionListener>
</h:commandButton>
```

Drugi pristup podrazumeva implementaciju metode unutar *backing bean*-a, i referenciranje date metode atributom *tag*-a kojim je komponenta predstavljena. U nastavku je dat primer ovog pristupa:

```
<h:commandButton id="exampleButton" value="Click"
     action="#{exampleBean.exampleMethod}"/>
```

1.1.2.4. Konvertori

Pri povezivanju UI komponente sa atributom *backing bean*-a, aplikacija može tretirati podatke iz komponente na dva načina:

- kao tipove podataka, poput *long* ili *int*¸ u okviru *backing bean*-a
- kao prezentacione podatke, kada su oni predstavljeni korisniku na način koji on može pročitati, ili zapisati, i to je najčešće *String*

JSF framework automatski konvertuje ove podatke pri prelasku iz jednog oblika u drugi, kada je tip podataka atributa *bean*-a podržan od strane komponente sa kojom je povezan. Na primer, ukoliko *UISelectBoolean* komponenta bude povezana sa atributom *bean*-a tipa *java.lang.Boolean*, *framework* će automatski konvertovati komponentine podatke tipa *String* u *Boolean*.

Ponekad postoji potreba za konverzijom podataka iz komponente u neki tip podataka koji nije standardan, pa samim tim ni podržan automatskom konverzijom od strane *framework*-a. JSF stoga dozvoljava registrovanje specifičnih konvertora za pojedinačne komponente. Moguće je koristiti neke od konvertora koji već postoje u *framework*-u, ili kreirati sopstveni konverter.

Pri kreiranju sopstvenog konvertora, potrebno je implementirati *Converter* interfejs, definisan unutar JSF framework-a. Potom, konverter je potrebno registrovati unutar konfiguracionog dokumenta aplikacije. Na kraju, potrebno je i ugnezditi konverter *tag* unutar *tag*-a komponente čije podatke želimo da konvertujemo. Pogledajmo primer:

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

1.1.2.5. Validatori

JSF *framework* poseduje mehanizam validacije vrednosti koje su unesene na ulaznim komponentama od strane korisnika aplikacije. Kao i za konverziju, framework već poseduje set standardnih i najčešće korišćenih validatora, poput validatora minimalne i maksimalne dozvoljene vrednosti unesenog numeričkog podatka.

Takođe je moguće definisati i sopstveni validator. Potrebno je implementirati *Validator* interfejs, definisan u okviru *framework*-a, kao i registrovati dati validator u konfiguracionom dokumentu aplikacije. Na kraju, potrebno je i ugnezditi validator *tag* unutar *tag*-a komponente čije podatke želimo da validiramo. Pogledajmo primer:

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

Pored implementacije interfejsa, validator je moguće kreirati i implementacijom metode unutar *backing bean*-a.

1.1.3. Biblioteke JSF tag-ova

Kao što je ranije naglašeno, unutar JSF *framework*-a definisane su dve biblioteke *tag*-ova, biblioteka osnovnih JSF *tag*-ova (*JSF Core tags Library*), i biblioteka HTML *tag*-ova (*JSF HTML Tag Libraries*).

Da bi se sadržaj ovih biblioteka koristio na stranici, potrebno ih je deklarisati unutar stranice. Pogledajmo deklaracije biblioteke osnovnih JSF *tag*-ova i biblioteke HTML *tag*-ova, respektivno:

```
<%@taglib uri="http://java.sun.com/jsf/core"
    prefix="f"
%>
<%@taglib uri="http://java.sun.com/jsf/html"
    prefix="h"
%>
```

Biblioteka osnovnih JSF *tag*-ova sadrži *tag*-ove koji su zaduženi za kategorizaciju raznih elementa korisničkog interfejsa na formi, konverziju i validaciju, upravljanje osluškivačima događaja za određenu komponentu, itd. *Tag*-ovi iz biblioteke osnovnih JSF komponenti mogu služiti za proširenje funkcionalnosti komponente deklarisane njima nadređenim *tag*-om iz biblioteke HTML tagova, poput narednih:

- *actionListener*, *valueChangeListener* – za komponentu deklarisanu nadređenim *tag*-om definišu osluškivač akcija ili osluškivač promene vrednosti, respektivno
- *validateRange*, *validateLongRange*, *validateDoubleRange* – vrše validaciju vrednosti komponente deklarisane nadređenim *tag*-om
- *convertDateTime*, *convertNumber* – vrše konverzije vrednosti komponente deklarisane nadređenim *tag*-om u datum i broj, respektivno

Doduše, postoje i tagovi koji nisu zavisni od drugih komponenti:

- *view*, *subview* – kontejnerski *tag*-ovi za ostale *tag*-ove na stranici
- *loadBundle* – učitava resursne dokumente, koji najčešće služe za internacionalizaciju, smeštanjem tekstualnih poruka na različitim jezicima unutar svog sadržaja.

Što se tiče *tag*-ova unutar biblioteke HTML *tag*-ova, pomoću njih se vrši prikaz komponenti korisničkog interfejsa na stranici HTML klijenta (*browser*-a). Pogledajmo neke od njih:

- *inputHidden*, *inputSecret*, *inputText*, *inputTextarea* – ulazni *tag*-ovi
- *outputFormat*, *outputLabel*, *outputLink*, *outputText* – izlazni *tag*-ovi
- *selectBooleanCheckbox*, *selectManyCheckbox*, *selectManyListbox*, *selectManyMenu*, *selectOneRadio*, *selectOneListbox*, *selectOneMenu* - selekcioni *tag*-ovi
- *commandButton*, *commandLink* – komandni *tag*-ovi
- *dataTable*, *column* – *tag*-ovi za prikaz tabela

1.1.4. *Backing bean*-ovi

Tipična JSF aplikacija uključuje i određeni broj *backing bean*-ova, klasa koji predstavljaju JSF *managed bean*-ove, i čiji atributi i metode služe za povezivanje sa komponentama na JSP stranici. JSF *managed bean*-ovi su JavaBeans komponente, koje se deklarišu i konfigurišu u konfiguracionom dokumentu aplikacije.

Backing bean klasa poseduje set atributa koji se mogu povezati sa vrednošću UI komponente, sa instancom same UI komponente, ili sa instancom određenog osluškivača, konvertora ili validatora. *Bean* poseduje i set metoda koje obavljaju procesiranja za UI komponente. Ove metode najčešće vrše validaciju podataka unetih na ulaznim UI komponentama, rukuju događajem koji se desio na komponenti, ili vrše procesiranje u cilju utvrđivanja na koju stranicu je potrebno navigirati korisnika.

Backing bean mora biti deklarisan i konfigurisan u konfiguracionom dokumentu aplikacije kao *managed-bean*, čime se postiže opisivanje jednostavnih *bean*-ova i kompleksnih stabala *bean*-ova, inicalizacija početnih vrednosti atributa *bean*-ova, smeštanje *bean*-a u odgovarajući opseg, kao i davanje pristupa *bean*-u iz objedinjenog jezika izraza u

okviru JSP stranica. Opseg (*scope*) *managed bean*-a može imati jednu od sledeće četiri vrednosti:

- *none* – opseg *bean*-a nije definisan
- *request* – *bean* je dostupan do završetka obrade aktuelnog zahteva
- *session* – *bean* je dostupan tokom trajanja sesije klijenta; po isteku ili invalidaciji sesije, *bean* više nije dostupan
- *application* – *bean* je dostupan sve vreme rada aplikacije

Pogledajmo primer deklaracije managed bean-a:

```
<managed-bean>
     <managed-bean-name>
          messageHandlerBean
     </managed-bean-name>
     <managed-bean-class>
          managedbeans.MessageHandlerBean
     </managed-bean-class>
     <managed-bean-scope>
          request
     </managed-bean-scope>
</managed-bean>
```

U navedenom primeru, bean je deklarisan svojim imenom, kao i navođenjem implementirajuće klase. Takođe je naveden i opseg pod kojim je moguće naći dati bean, u ovom slučaju *request* opseg.

Pogledajmo sada primere povezivanja vrednosti UI komponente sa atributom *backing bean-a*, te povezivanja osluškivača događaja UI komponente sa metodom backing bean-a na JSP stranici, respektivno.

```
<h:inputText id="exampleInput"
     value="#{exampleBean.value}">
</h:inputText>
...
<h:commandLink id="exampleLink"
     actionListener="#{exampleBean.listenerMethod}">
</h:commandLink>
```

U prvom primeru, UI komponenta predstavljena *inputText tag*-om povezana je sa atributom *value exampleBean*-a. U drugom primeru, za osluškivač događaja *commandLink* komponente proglašena je metoda *listenerMethod exampleBean*-a.

1.1.5. Navigaciona pravila

Navigacija u JSF aplikaciji određena je skupom pravila koja definišu na koju stranicu će korisnik biti upućen nakon klika na dugme ili hiperlink. Unutar jednog pravila definiše se polazna stranica, te više mogućih navigacionih slučajeva (*navigation case*), u okviru kojih je definisana stranica na koju će korisnik biti upućen i uslov prelaska na datu stranicu. Ovi uslovi se opisuju navođenjem *String* vrednosti (*outcome*) koja predstavlja povratnu vrednost metode koja vrši procesiranje s ciljem utvrđivanja na koju stranicu će se korisnik navigirati. Drugi način je da se navede naziv same metode koja vrši procesiranje (*action*). Napokon, moguće je koristiti oba uslova istovremeno.

Navigaciona pravila definišu se unutar konfiguracionog dokumenta aplikacije. Pogledajmo primer navigacionog pravila:

```
<navigation-rule>
     <from-view-id>/firstPage.jsp</from-view-id>
     <navigation-case>
          <from-action> #{exampleBean.process}
          </from-action>
          <from-outcome>success</from-outcome>
          <to-view-id>/secondPage.jsp
          </to-view-id>
          <redirect />
     </navigation-case>
</navigation-rule>
```

U navedenom primeru, navigacija će se izvršiti sa stranice *firstPage.jsp* na stranicu *secondPage.jsp*, ukoliko nakon klika na komandnu komponentu na stranici *firstPage.jsp* koja kao osluškivač ima definisanu metodu *exampleBean.process*, ova metoda vrati *String* vrednosti *success*.

1.1.6. Konfiguracioni dokument

Konfiguracioni dokument je XML dokument, najčešće naziva **faces-config.xml**, unutar kojeg se deklarišu i konfigurišu preostale komponente JSF aplikacije, poput *backing bean*-ova, validatora, konvertora, i navigacionih pravila. Pogledajmo primer konfiguracionog dokumenta.

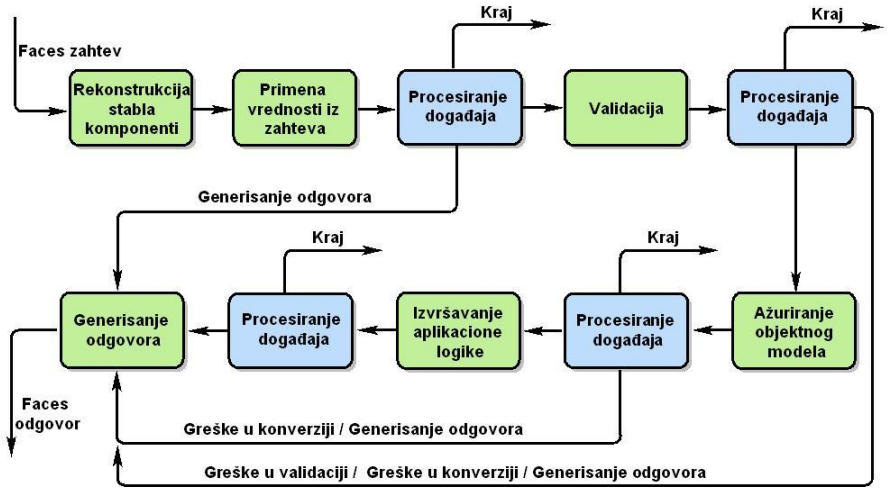
```
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
"http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd"
version="1.2">
     <managed-bean>
          <managed-bean-name>
               messageHandlerBean
          </managed-bean-name>
          <managed-bean-class>
               managedbeans.MessageHandlerBean
          </managed-bean-class>
          <managed-bean-scope>
               request
          </managed-bean-scope>
     </managed-bean>
     <managed-bean>
          <managed-bean-name>
               userSettingsBean
          </managed-bean-name>
          <managed-bean-class>
               managedbeans.UserSettingsBean
          </managed-bean-class>
          <managed-bean-scope>
               session
          </managed-bean-scope>
     </managed-bean>

     <validator>
          <validator-id>
```

```
            emailAddressValidator
        </validator-id>
        <validator-class>
            validators.EmailAddressValidator
        </validator-class>
    </validator>

    <converter>
        <converter-id>
            customConverter
        </converter-id>
        <converter-class>
            converters.CustomConverter
        </converter-class>
    </converter>

    <navigation-rule>
        <display-name>signIn</display-name>
        <from-view-id>/login.jsp</from-view-id>
        <navigation-case>
            <from-outcome>success</from-outcome>
            <to-view-id>/mail.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>

    </faces-config>
```

U datom primeru deklarisana su dva *managed bean*-a, jedan validator, i jedan konverter, respektivno. Na kraju, unutar konfiguracionog dokumenta definisan je i primer navigacionog pravila.

1.1.7. Faze JSF aplikacije

Svaki zahtev koji pristigne od strane klijenta prolazi kroz ciklus JSF faza obrade zahteva. Prihvatanjem zahteva, upravljanjem fazama obrade zahteva i eventualnom inicijalizacijom resursa upravlja instanca *FacesServlet*-a, kojeg poseduje svaka JSF aplikacija.



Dijagram 1.1. – Faze JSF aplikacije

Na dijagramu 1.1. prikazan je ciklus JSF faza obrade zahteva. Postoji šest faza kroz koje zahtev prolazi:

1. Rekonstrukcija stabla komponenti (View Restoration)
2. Primena vrednosti iz zahteva (Applying Request Values)
3. Validacija ulaznih podataka (User Input Validation)
4. Ažuriranje objektnog modela (Update Object Model)
5. Izvršavanje logike aplikacije (Application Logic Execution)
6. Generisanje odgovora za klijente (Rendering Response)

1.2. Facelets

Facelets[5] je open source web framework and view handler (deklarativni jezik) za JSF. Napravljen je sa ciljem da nadogradi nedostatke u JSP tehnologiji(brže izvršavanja, brze kreiranje u toku izvršenja, proširivanje mogućnosti postojećih komponenti i serverskih objekata, brzi prikaz, provera EL (Expresion Language) u toku kompajliranja, dodavanje nestandardne EL implementacije i omogući rad sa XHTML fajlovima. Zahteva validan XML format za rad i podržava sve JSF UI komponente.

Dodaje dodatnu biblioteku tagova koja omogućuje dodatne opcije u vezi sa manipulisanjem XHTML strana (npr. Template strane, define i include metode itd..). Takođe omogućava kreiranje dodatnih biblioteka za korisničke potrebe.

Inicijalno, Facelets je bio dostupan kao alternativni deklarativni jezik za JSF 1.1 i JSF 1.2 kod kojih je JSP osnovni deklarativni jezik. Od JSF 2.0 Facelets je unapređen u glavni, a JSP u sekundarni deklarativni jezik.

1.3. Ajax

Ajax (Asynchronous JavaScript And XML)[2] ne predstavlja sam po sebi novu tehnologiju, već pristup zajedničkom korišćenju postojećih standarda, koji uključuju:

- HTML (ili XHTML) i CSS (Cascading Style Sheets) za prezentaciju podataka na klijentskoj strani
- JavaScript kao skript jezik koji dodaje funkcionalnost *web* stranicama
- DOM (Document Object Model) za pristup i manipulaciju HTML dokumentima
- XML za razmenu podataka
- XSLT za manipulaciju podacima
- *XMLHttpRequest* objekat za asinhronu komunikaciju između klijentske i serverske strane

Uprkos činjenici da Ajax u svom nazivu uključuje JavaScript, postoje i drugi skript jezici kojima se može implementirati Ajax aplikacija, poput VBScript-a (VisualBasic Sript). Takođe, XML nije obavezan za razmenu podataka – često se koristi i JSON (JavaScript Object Notation), a moguće je koristiti i druge standarde, pa čak i običan tekst.

Tradicionalni pristup funkcionisanju *web* aplikacije podrazumeva da se celokupan sadržaj stranice ponovo učitava sa servera nakon svakog korisničkog zahteva. Upotrebom Ajax-a u *web* aplikaciji, razmena podataka između *web* stranice i servera može da se obavlja asinhrono, u pozadini. Na ovaj način se nakon Ajax zahteva ne učitava celokupan sadržaj stranice, već samo onaj njen deo koji je od interesa, pri čemu se ne menja izgled i ponašanje postojeće stranice. Pored toga, CSS-ovi i skriptovi unutar konkretne stranice se učitavaju samo jednom. U korisničkim aplikacijama, posebno pri razmeni malih količina podataka sa serverom, korišćenje Ajax-a u velikoj meri doprinosi povećanju funkcionalnosti, interaktivnosti i brzine aplikacije.

Međutim, pored mnogobrojnih prednosti, upotreba Ajax-a unosi i određene probleme u *web* aplikacije. Kompleksnost aplikacija raste, jer je prezentaciona logika potrebna kako u klijentskim HTML stranicama, tako i u logici na serverskoj strani da bi se ispravno generisao XML sadržaj potreban na HTML stranicama. Ajax aplikacije je stoga teško i debagirati, jer procesna logika postoji i na klijentskoj i na serverskoj strani. Pored svega toga, dinamički sadržaj nastao nakon Ajax zahteva se ne registruje u istoriji *browser*-a, te će pritiskom na "Back" dugme u okviru *browser*-a korisnik biti vraćen na prethodno posećenu stranicu, a ne na prethodno stanje stranice. Takođe, *browser* mora imati ugrađenu podršku za JavaScript ili Ajax da bi se Ajax aplikacija uopšte izvršavala unutar *browser*-a. Određeni *browser*-i čak i sa ugrađenom Ajax podrškom imaju problema sa interpretacijom dinamičkog sadržaja.

1.4. RichFaces

RichFaces[3] je *open-source framework* čija je osnovna namena proširivanje postojećeg skupa korisničkih komponenti i ugradnja Ajax podrške u JSF *framework*. Upotreba RichFaces-a omogućava razvoj JSF aplikacija sa Ajax sposobnostima bez korišćenja JavaScript-a, kao i razvoj komponentama bogatog korisničkog interfejsa sa lako promenljivim karakteristikama upotrebom *skin*-ova.

1.4.1. Tok razvoja

Da bismo bolje razumeli način na koji RichFaces *framework* funkcioniše danas, pogledajmo najpre pozadinu njegovog razvoja.

RichFaces je izgrađen na Ajax4jsf *framework*-u, koji nastao sa ciljem da objedini funkcionalnosti JSF-a i Ajax aplikacija. Exadel, koji stoji iza prve samostalne verzije Ajax4jsf

framework-a, imao je kao osnovnu ideju ugradnju Ajax sposobnosti u JSF aplikacije, i ta ideja je realizovana JSF komponentama koje su rukovale Ajax pozivima automatski, bez potrebe da se piše JavaScript kod i koristi *XMLHttpRequest* objekat. U tom periodu, RichFaces se paralelno razvijao kao samostalan *framework* čija je osnovna svrha bila uvećanje broja UI komponenti definisanih unutar JSF-a. Između ova dva *framework*-a postojala je suštinska razlika – Ajax4jsf omogućavao je podršku za Ajax na celoj stranici (*page-wide Ajax support*), a ne samo za pojedinačne komponente (*component-centric Ajax approach*) kakav je bio slučaj kod RichFaces-a.

U međuvremenu, oba *framework*-a su dospela pod okrilje JBoss-a, koji je odlučio da objedini ova dva *framework*-a u jedan, danas nazvan samo RichFaces. Ipak, današnji RichFaces je u mnogočemu baziran na Ajax4jsf *framework*-u, pre svega na *page-wide Ajax support* pristupu. Usto, komponente koje su proistekle iz Ajax4jsf-a su i dalje reprezentovane na JSF stranicama *tag*-ovima sa *"a4j"* prefiksom.

1.4.2. Osnovne karakteristike

Velika prednost RichFaces framework-a u odnosu na niz idejno sličnih *framework*-a je činjenica da je u potpunosti integrisan u JSF životni ciklus. Dok drugi *framework*-ovi najčešće omogućavaju pristup *backing bean*-ovima, RichFaces pored toga nudi i pristup osluškivačima događaja, te pobuđuje konvertore i validatore na serverskoj strani za vreme obrade zahteva.

Ugradnjom u JSF životni ciklus, RichFaces komponentama je omogućeno definisanje događaja koji će izazvati Ajax zahtev i osvežiti druge komponente te stranice koje treba da budu sinhronizovane sa JSF stablom komponenti nakon promena nastalih usled Ajax zahteva. Ovo se postiže navođenjem svih komponenti kojima je osvežavanje potrebno u posebnom atributu *tag*-a koji na JSF stranici reprezentuje RichFaces komponentu koja je poslala zahtev. Taj atribut je *reRenderer*. Pogledajmo primer upotrebe ovog atributa:

```
<a4j:commandButton value="Submit"
     actionListener="#{exampleBean.listener}"
     reRender="component1,component2"
</a4j:commandButton>
```

U datom primeru, nakon izvršavanja metode definisane *actionListener* atributom, osvežiće se prikaz komponenti *component1* i *component2* na JSF stranici.

RichFaces koristi pristup baziran na formama za slanje Ajax zahteva. Ovo znači da sve komponente RichFaces biblioteke koje mogu da pošalju Ajax zahtev, u oviru *XMLHttpRequest* objekta u zahtevu šalju i ostale podatke sa najbliže JSF forme, u koju je ugnježdena komponenta koja je izazvala zahtev. Prosleđeni podaci sadrže vrednosti ulaznih komponenti sa forme, ili sporedne informacije poput podataka o stanju komponenti. Ukoliko postoji potreba da se izbegne ovakvo ponašanje, potrebno je vrednost *ajaxSingle* atributa komponente koja šalje zahtev postaviti na "*true*". Na ovaj način, u okviru zahteva biće prosleđena samo vrednost komponente koja zahtev i prosledila, i izbeći će se prosleđivanje sadržaja celokupne forme. Izuzetak je komponenta *support*, čija je svrha da omogući Ajax funkcionalnost JSF i RichFaces komponentama koje je nemaju ugrađenu. Pogledajmo stoga primer korišćenja atributa *ajaxSingle* u okviru *support* komponente.

```
<h:form>
     <h:inputText value="#{person.firstName}">
          <a4j:support event="onkeyup"
               actionListener="#{exampleBean.listener}"
               ajaxSingle="true"/>
```

```
    </h:inputText>
    <h:inputText value="#{person.lastName}"/>
    ...
</h:form>
```

U datom primeru, pri obradi događaja definisanog *event* atributom *support* komponente, sa zahtevom će se prosleđivati i vrednost komponente u koju je support komponenta i ugnježdena. Zbog vrednosti atributa *ajaxSingle*, vrednosti drugih komponenti unutar forme neće biti prosleđivane.

Upotrebom *ajaxSingle* atributa redukuje se količina podataka koja se šalje sa zahtevom. Obrada zahteva se dalje može unaprediti upotrebom *immediate* atributa, kojom se naglašava da osluškivač vezan za komponentu treba da bude izvršen odmah, za vreme JSF faze primene vrednosti iz zahteva, umesto da se čeka do faze izvršavanja aplikacione logike. Ovim se omogućava da vrednosti modela podataka ipak budu podešeni pre faze validacije, ukoliko postoji problem sa uspešnim prolaskom kroz JSF fazu validacije.

Atribut *bypassUpdates* omogućava da se pri obradi zahteva potpuno zaobiđe JSF faza ažuriranja objektnog modela, i on može biti koristan u slučajevima kada je potrebno pomoću validatora proveriti unesenu vrednost bez osvežavanja modela podataka datom vrednošću.

Iako je više puta naglašeno da RichFaces komponente rukuju Ajax zahtevima i *XMLHttpRequest* objektom automatski, i prema tome nema potrebe za pisanjem JavaScript koda, ovime nije isključena mogućnost upotrebe JavaScript-a na stranicama. Naprotiv, RichFaces komponente su obogaćene setom atributa koje omogućuju pobudu JavaScript koda, u odgovarajućim trenucima. Pogledajmo nekoliko važnijih atributa:

- *onsubmit* – atribut koji omogućava pobudu JavaScript koda pre slanja Ajax zahteva, te ukoliko je povratna vrednost dobijena izvršavanjem datog koda *false*, Ajax zahtev neće biti prosleđen
- *onclick* – veoma sličan *onsubmit* atributu, ali odnosi se samo na klikabilne komponente – *commandLink* i *commandButton*
- *onbeforedomupdate* – omogućava pobudu JavaScript koda nakon povratka odgovora na Ajax zahtev, i pre početka osvežavanja DOM stabla komponenti unutar stranice
- *oncomplete* – omogućava pobudu JavaScript koda nakon povratka odgovora na Ajax zahtev, i nakon završetka osvežavanja DOM stabla komponenti unutar stranice
- *data* – atribut koji omogućava prihvatanje dodatnih podataka sa servera za vreme trajanja Ajax zahteva

Pored spomenuti atributa, RichFaces nudi i određeni broj funkcija koje mogu biti korišćene u JavaScript-u, olakšavajući pritom pristup i rukovanje komponentama.

1.4.3. *Skinnability*

Skinnability svojstvo RichFaces *framework*-a ima za cilj da unapredi dizajniranje korisničkog interfejsa, umanjujući potrebu za definisanjem velikog broja elemenata unutar CSS dokumenta. Jedan skin definiše paletu boja i niz drugih parametara koji se primenjuju na celokupan korisnički interfejs odjednom, čime se izbegava ponavljanje parametara koji imaju istu vrednost u CSS dokumentu. Jednom promenom skina, sinhronizovano se menja izgled svih komponenti interfejsa bez potrebe za pojedinačnim izmenama u CSS-u.

No skinovi ne zamenjuju u potpunosti CSS standard, i sasvim izvesno ga ne isključuju iz upotrebe u *web* aplikacijama baziranim na RichFaces tehnologiji. *Skinnability* svojstvo se može posmatrati kao proširenje CSS-a na visokom nivou, i ne postoji prepreka da ono bude korišćeno u paru sa regularnim CSS izrazima.

RichFaces *framework* nudi set predefinisanih skinova, i izbor skina se definiše unutar *deployment descriptor*-a pojedinačnih aplikacija. Pored toga, *framework* nudi i mehanizam kreiranja sopstvenih skinova.

1.4.4. Osnovne komponente

Unutar RichFaces *framework*-a definisane su dve biblioteke komponenti, Core Ajax biblioteka i UI biblioteka.

Core Ajax biblioteka sadrži set komponenti koje unose Ajax funkcionalnost u stranice bez potrebe za pisanjem JavaScript koda. Korišćenjem komponenti ove biblioteke moguće je čak dodati Ajax funkcionalnost komponentama na postojećim stranica, bez potrebe da one budu zamenjene.

UI biblioteka nudi čitav niz komponenti korisničkog interfejsa – menije, kontrole za unos i prikaz podataka kao što su stablo, tabele sa slajderima, filterima podataka, kombinovanim sadržajima (tekst i slike), razne vrste panela, separatora, *tab*-ova, *toolbar*-ova, podršku za *drag-and-drop*, kalendar itd. Sve ove komponente su podesivog izgleda i osobina i podržavaju skinove.

Pogledajmo stoga primere nekoliko komponenti ovih biblioteka, uz objašnjenje njihovih najvažnijih atributa.

1.4.4.1. <a4j:commandButton> i <a4j:commandLink>

Ove komponente predstavljaju osnovne komandne komponente RichFaces *framework*-a, i navedene su zajedno zbog velikog broja sličnosti u funkcionalnosti i setu ponuđenih atributa. Intuitivno je jasno da *commandButton* predstavlja dugme, dok *commandLink* predstavlja hiperlink na stranici, i osnovna karakteristika obe komponente je da po kliku na njih biva generisan Ajax zahtev, koji prosleđuje vrednosti sa forme kojoj pripadaju na server. Pogledajmo osnovne atribute ovih komponenti:

- *action* – povezuje komponentu sa metodom *backing bean*-a u aplikaciji koja će biti pozvana ako je komponenta aktivirana na stranici
- *actionListener* – povezuje komponentu sa metodom *backing bean*-a koja prima kao parametar ActionEvent objekat, a povratna vrednost joj je tipa *void*
- *rendered* – *boolean* vrednost koja označava da li komponenta biti prikazana
- *reRenderer* – identifikatori komponenti čiji će prikaz biti osvežen po završetku obrade zahteva

1.4.4.2. <a4j:support>

Ugnježdavanje *support* komponente omogućava bilo kojoj drugoj RichFaces i JSF komponenti da pošalje Ajax zahtev, uz definisanje događaja koji će zahtev izazvati, i liste komponenti koje će biti osvežene nakon završetka obrade zahteva. Ključni atributi ove komponente su:

- *action* – povezuje komponentu sa metodom *backing bean*-a u aplikaciji koja će biti pozvana ako je komponenta aktivirana na stranici
- *actionListener* – povezuje komponentu sa metodom *backing bean*-a koja prima kao parametar ActionEvent objekat, a povratna vrednost joj je tipa *void*
- *event* – naziv JavaScript događaja (npr. *onclick*) nad roditeljskom komponentom, koji će izazvati prosleđivanje zahteva
- *reRenderer* – identifikatori komponenti čiji će prikaz biti osvežen po završetku obrade zahteva

1.4.4.3. <rich:dataTable>

Komponentom *dataTable* se realizuje tabelarno predstavljanje podataka koji mogu biti sadržani u kolekciji objekata u *backing bean*-u. Atributi objekata se mapiraju na pojedinačne kolone u tabeli. Komponenta sadrži veliki broj atributa kojima se definiše obrada događaja nad tabelom u JavaScript-u, podešava izgled tabele, i omogućava sortiranje podataka u tabeli. Ključni atributi ove komponente su:

- *rendered* – *boolean* vrednost koja označava da li komponenta biti prikazana
- *columns* – broj kolona u tabeli
- *rows* – broj redova u tabeli, ukoliko je vrednost ovog atributa nula, biće prikazani svi preostali redovi
- *value* – vrednost komponente, odnosno kolekcija objekata koji će biti predstavljeni redovima u tabeli
- *var* – promenljiva pod *request* opsegom pomoću koje se, u toku iteracije, pristupa objektu čiji se podaci predstavljaju u trenutnom redu tabele

Na slici 1.2. prikazana je *dataTable* komponenta.

United States Capitals

Capitals and States Table			
State Flag	Capital Name	State Name	TimeZone
	Montgomery	Alabama	GMT-6
	Juneau	Alaska	GMT-9
	Phoenix	Arizona	GMT-7
	Little Rock	Arkansas	GMT-6
	Sacramento	California	GMT-8
State Flag	Capital Name	State Name	TimeZone
Capitals and States Table			

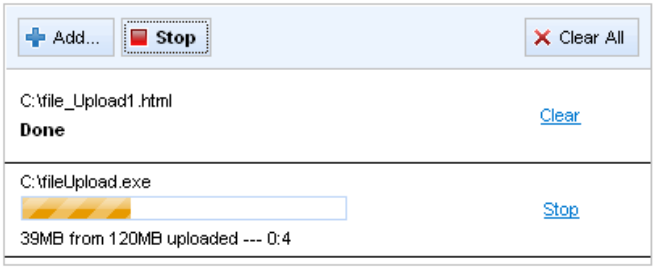
Slika 1.2. – *dataTable* komponenta

1.4.4.4. <rich:fileUpload>

Komponenta *fileUpload* omogućava slanje datoteka na server koristeći Ajax zahtev. Ključni atributi ove komponente su:

- *acceptedTypes* – dozvoljeni tipovi datoteka
- *fileUploadListener* – povezuje komponentu sa metodom *backing bean*-a koja vrši prihvatanje i obradu datoteka
- *immediateUpload* – ako je vrednost ovog atributa "true", slanje se vrši odmah po izboru datoteke, u protivnom, nakon izbora datoteka, korisnik aktivira slanje pritiskom na dugme "Upload"
- *maxFilesQuantity* – definiše maksimalan broj datoteka za slanje na server

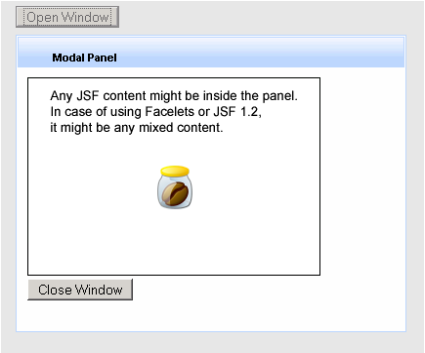
Na slici 1.3. prikazana je *fileUpload* komponenta.



Slika 1.3. – *fileUpload* komponenta

1.4.4.5. <rich:modalPanel>

Komponenta *modalPanel* implementira modalni prozor, i dok je ovaj prozor aktivan, sve operacije u glavnom prozoru aplikacije su zaključane.



Slika 1.4. – *modalPanel* komponenta

Otvaranje i zatvaranje ovog prozora se obavlja kroz JavaScript kod. Sadržaj modalnog prozora mogu činiti bilo koje RichFaces i JSF komponente. Na slici 1.4. prikazana je modalPanel komponenta.

1.4.4.6. <rich:tree>

Komponenta *tree* omogućava hijerarhijsku prezentaciju podataka kroz strukturu stabla. Pored ugrađene Ajax podrške, komponenta nudi i podršku za *drag-and-drop*. Ključni atributi ove komponente su:

- *ajaxSubmitSelection* – *boolean* vrednost koja određuje da li će se na server slati Ajax zahtev po selekciji čvora u stablu
- *nodeSelectListener* – povezuje komponentu sa metodom *backing bean*-a koja vrši obradu selekcije čvora u stablu
- *reRender* – identifikatori komponenti čiji će prikaz biti osvežen po završetku obrade zahteva
- *switchType* – vrednost koja opisuje na koji način će se vršiti preuzimanje podataka o podčvorovima selektovanog čvora. Moguće vrednosti su "client", "server" i "ajax"
- *value* – vrednost komponente, odnosno kolekcija objekata koja će biti predstavljena čvorovima stabla
- *var* – promenljiva pod *request* opsegom pomoću koje se, u toku iteracije, pristupa objektu čiji se podaci predstavljaju u trenutnom čvoru stabla

Na slici 1.5. prikazana je *tree* komponenta.

Slika 1.5. – *tree* komponenta

1.5. JPA

JPA (Java Persistence API)[1] omogućava objektno-relaciono mapiranje u Java aplikacijama koje u velikoj meri olakšava upravljanje relacionim podacima. Iako je razvijen uz EJB 3.0 (Enterprise JavaBeans), korišćenje mu nije ograničeno na upotrebu isključivo uz EJB softverske komponente. Moguće ga je koristiti direktno u *web* aplikacijama, pa čak i van Java EE platforme, na primer u Java SE aplikacijama.

Uopšteno gledano, upravljanje perzistencijom podataka uz upotrebu JPA-a čine tri ključna koraka:

- opisivanje objektno-relacionog mapiranja anotacijama
- rukovanje objektnim modelom upotrebom JPA-a
- upotreba JPQL-a (Java Persistence Query Language)

1.5.1. Objektno-relaciono mapiranje

JPA pruža mehanizam za upravljanje objektno-relacionim mapiranjem sa Java objekata na trajno sačuvane podatke u bazi podataka i obrnuto. Sva mapiranja se realizuju **anotacijama**. Anotacije predstavljaju specijalnu formu meta-podataka, koji stoje uz klase, metode i atribute klasa. Java anotacije su refleksivne, na taj način da bivaju ugrađene u .class datoteku generisanu od strane Java prevodioca, i mogu biti zadržane u okvirima Java virtuelne mašine kako bi bile dostupne u vreme izvršavanja.

Java objekat koji je mapiran na tabelu u bazi podataka naziva se *entity*, i njegovi atributi su mapirani na kolone odgovarajuće tabele. Pogledajmo osnovne karakteristike *entity* klasa:

- *entity* klasa mora biti anotirana *javax.persistence.Entity* anotacijom
- *entity* klasa obavezna je definisati prazan konstruktor, pri čemu je dozvoljena definicija i drugih konstruktora
- implementiranje *Serializable* interfejsa nije obavezno, ali ukoliko ga *entity* klasa implementira, moguće je razmenjivati *entity* objekte između svih slojeva aplikacije bez pisanja posebnih DTO (Data Transfer Object) klasa
- ni *entity* klasa, niti njeni atributi i metode ne smeju biti deklarisani kao *final*
- dozvoljeno je nasleđivanje drugih klasa
- perzistentni atributi moraju biti deklarisani kao privatni atributi klase, i njima se pristupa samo putem *get* i *set* metoda

Kako se jedan *entity* najčešće mapira na jednu tabelu u bazi podataka, tako se atributi entity klase najčešće mapiraju na odgovarajuće kolone date tabele. Ova mapiranja se takođe realizuju anotacijama, te iako JPA definiše set anotacija upotrebom kojih je moguće relacioni model u potpunosti predstaviti objektnim modelom, mi ćemo se upoznati samo sa osnovnim anotacijama, mapiranjem veza između *entity*-ja i mapiranjem primarnih ključeva.

1.5.1.1. Osnovne anotacije

Kako je već ranije naglašeno, entity klasa mora biti anotirana *Entity* anotacijom. Uz to, mapiranje na odgovarajuću tabelu u bazi podataka ostvaruje se *Table* anotacijom. Pogledajmo primer anotacija u kom je *Entity* klasa *Book* mapirana na tabelu u bazi podataka imena *BOOKS*. Ime tabele navodi se upotrebom anotacionog atributa *name*.

```
@Entity
@Table(name = "BOOKS")
public class Book implements Serializable {...}
```

Što se tiče kolona u tabeli, one se mapiraju na atribute klase ili njima odgovarajuće *get* metode upotrebom *Column* anotacija. Pogledajmo primer atributa *bookName* koji je mapiran na kolonu *BOOK_NAME*. Ime kolone navodi se upotrebom anotacionog atributa *name*.

```
@Column(name = "BOOK_NAME")
private String bookName;
```

Pored *Column* anotacije, važna je i *Transient* anotacija, kojom se anotira atribut koji nije perzistentan, odnosno ne mapira se ni na jednu kolonu u bazi podataka.

Važno je napomenuti da pored anotacionog atributa *name* koji je korišćen u primerima, unutar spomenutih anotacija postoji niz drugih anotacionih atributa kojima je moguće znatno preciznije opisati mapiranje sa objektnog na relacioni model, poput oznake da li kolona može poprimiti *null* vrednost, da li joj je vrednost u datoj koloni jednistvena itd.

1.5.1.2. Mapiranje veza

Entity klasa može da referencira proizvoljan broj drugih *entity*-ja, i veza između njih se uspostavlja, kao i bilo koja druga veza između dva objekta u Java programskom jeziku, deklaracijom jednog *entity*-ja kao atributa drugog.

Na osnovu smera ovih veza, može se uspostaviti sledeća podela veza:

- bidirekciona veza – oba *entity*-ja sadrže atribut koji referencira *entity* sa suprotne strane veze
- unidirekciona veza – samo jedan od dva *entity*-ja u vezi referencira *entity* s suprotne strane veze

S druge strane, kardinalitet veze između dva *entity*-ja može biti:

- 1:1
- 1:n
- m:n

Za opisivanje veze kardinaliteta 1:1 pomoću anotacija, uz atribut kojim se referencira *entity* s druge strane veze koristi se anotacija *OneToOne*. Pored ove, može se koristiti i anotacija *JoinColumn*, kojom se pobliže opisuje kolona koja predstavlja spoljni ključ u tabeli, na koju se dati atribut mapira. Ukoliko se spoljni ključ sastoji od više kolona, koristi se anotacija *JoinColumns*.

Za opisivanje veze kardinaliteta 1:n pomoću anotacija, 1-strana koristi anotaciju *OneToMany*, a n-strana anotaciju *ManyToOne*. Uz to, *entity* sa n-strane više nije

reprezentovan jedinstvenom instancom unutar *entity*-ja na 1-strani, nego **kolekcijom** instanci. Takođe, n-strana može koristiti i *JoinColumn* ili *JoinColumns* anotacije.

I na kraju, veza kardinaliteta m:n se opisuje anotacijom *ManyToMany* na obe strane veze. Oba *entity*-ija su ovaj put reprezentovana kolekcijom instanci na suprotnoj strani, i bilo koja strana može koristiti anotacije *JoinColumn* ili *JoinColumns*.

1.5.1.3. Mapiranje primarnih ključeva

Svaka *entity* klasa obavezno sadrži atribut koji opisuje jedinstveni identifikator, odnosno mapira primarni ključ u tabeli baze podataka, koji omogućava klijentima da lociraju konkretnu instancu *entity* klase. Ovaj ključ, u skladu sa osobinama relacionih baza podataka, može biti prosti i kompozitni.

Prosti ključevi su u objektnom modelu atributi entity klase anotirani *Id* anotacijom.

S druge strane, mapiranje kompozitnih ključeva se obavlja na nešto kompleksniji način – potrebno je, naime, definisati posebnu klasu za primarni ključ, čiji će atributi mapirati pojedinačne kolone koje čine kompozitni ključ odgovarajuće tabele upotrebom *Column*, *JoinColumn* i *JoinColumns* anotacija. Ova klasa poseduje sledeće karakteristike:

- modifikator pristupa date klase mora biti *public*
- klasa mora implementirati *Serializable* interfejs
- klasa je obavezna definisati javni podrazumevani konstruktor
- unutar klase moraju biti implementirane *hashCode()* i *equals(Object other)* metode

Datu klasu koja odgovara kompozitnom ključu potrebno je deklarisati kao atribut *entity* klase, i obeležiti *EmbeddedId* anotacijom.

1.5.2. Rukovanje objektnim modelom

Kao rezultat objektno-relacionog mapiranja nastaje objektni model u vidu određenog broja *entity* klasa koje su mapirane na odgovarajuće tabele u bazi podataka. Upotreba *framework*-a poput Hibernate-a podrazumevala je opis ovog modela unutar konfiguracionih dokumenata. Iako JPA izbegava ovaj pristup, ipak je potrebno definisati barem jedan konfiguracioni dokument. Ovaj dokument, naziva **persistence.xml**, služi za opisivanje perzistencione jedinice (*persistence unit*), koja predstavlja jednu grupu perzistencionih klasa (*entity*-ja) i parametara mapiranja. Jedna aplikacija nije ograničena na samo jednu perzistencionu jedinicu.

Entity-ima rukuje instanca *javax.persistence.EntityManager* interfejsa (u daljem tekstu *entity manager*). Svaka instanca *entity manager*-a povezana je sa perzistencionim kontekstom, koji definiše opseg pod kojim se pojedinačne instance *entity*-ja kreiraju, perzistiraju i uklanjaju. Prema načinu rukovanja, *entity manager*-i se mogu podeliti na:

- Kontejnerom-upravljani *entity manager* (*Container-Managed Entity Manager*) – perzistencioni kontekst instance *entity manager*-a je automatski propagiran od strane kontejnera svim aplikacionim slojevima koji koriste *entity manager*
- Aplikacijom-upravljani *entity manager* (*Application-Managed Entity Manager*) – životnim ciklusom instanci *entity manager*-a upravlja aplikacija. U daljem tekstu, podrazumeva se ovaj način rukovanja

Na osnovu definisane perzistenicone jedinice, aplikacija kreira instancu *EntityManagerFactory* interfejsa, koji predstavlja reprezentaciju objektno-relacionog mapiranja. Dalja komunikacija se sa bazom podataka odvija u sesijama, i svaku sesiju opisuje jedan *EntityManager* objekat, kojeg kreira data instanca *EntityManagerFactory*-ja. *EntityManager* objekat poseduje sve metode potrebne za perzistenciju, izmenu i uklanjanje

konkretnih *entity* objekata, kao i izvršavanje proizvoljnih upita. Konkretni *entity* objekti se pritom mogu naći u jednom od naredna četiri stanja:

- Nove instance (*New*) – još uvek nemaju perzistencioni identitet, i nisu povezane sa perzistencionim kontekstom
- Upravljane instance (*Managed*) – imaju perzistencioni identitet, i povezane su sa perzistencionim kontekstom
- Nezavisne instance (*Detached*) – imaju perzistencioni identitet, i nisu trenutno povezane sa perzistencionim kontekstom
- Uklonjene instance (*Removed*) – imaju perzistencioni identitet, povezane su sa perzistencionim kontekstom, i potrebno ih je ukloniti iz trajno sačuvanih podataka

1.5.3. JPQL-a (Java Persistence Query Language)

JPQL (u daljem tekstu upitni jezik) je jezik kojim se definišu upiti kojim se rukuje *entity*-jima. Velika prednost ovog jezika je mogućnost pisanja upita koji su nezavisni od konkretnog sistema za upravljanje bazom podataka koji se koristi za čuvanje podataka.

Upitni jezik koristi apstraktne perzistencione šeme *entity*-ja, uključujući i veze između *entity*-ja, za svoj model podataka, te definiše operatore i izraze bazirane na tom modelu i koristeći sintaksu sličnu SQL-u omogućava selekciju objekata ili vrednosti iz baze podataka, kao i izmenu ili uklanjanje objekata iz baze podataka.

Selekcioni izraz u upitnom jeziku se može predstaviti BNF sintaksom na sledeći način:

```
Select_izraz ::= select_klauzula from_klauzula
                 [where_klauzula][groupby_klauzula]
                 [having_klauzula][orderby_klauzula]
```

Select klauzula definiše tipove objekata ili vrednosti koje upit treba da vrati.

From klauzula definiše opseg upita, deklarišući jednu ili više identifikacionih varijabli, koje mogu biti referenicrane u *Select* i *Where* klauzalama. Ove identifikacione varijable mogu biti:

- ime *entity*-ja definisano u apstraktnoj šemi
- kolekcija koja sadrži *entity*-je sa suprotne strane veze, u slučaju veze između dva *entity*-ja
- jedan element, *entity* sa suprotne strane veze, u slučaju veze između dva *entity*-ja
- određeni član kolekcije koja sadrži *entity*-je sa suprotne strane veze, u slučaju veze između dva *entity*-ja

Where klauzula je kondicioni izraz koji filtrira objekte ili vrednosti koji su zahvaćeni upitom.

Group by klauzula grupiše rezlutate upita u skladu sa navedenim atributima.

Having klauzula se koristi sa *group by* klauzulom da bi se dalje filtrirali rezultati upita u skladu sa navedenim kondicionim izrazom.

Order by klauzula sortira objekte ili vrednosti zahvaćene upitom.

Izrazi za izmenu ili uklanjanje omogućavaju operacije nad setom entiteta, i imaju sledeću sintaksu:

```
Update_izraz ::= update_klauzula
                 [where_klauzula]
Delete_izraz ::= delete_klauzula
                 [where_klauzula]
```

Update klauzula i *delete* klauzula određuju tip objekata koji trebaju biti izmenjeni ili uklonjeni.

Where klauzula je kondicioni izraz koji filtrira objekte ili vrednosti koji su zahvaćeni izrazom za izmenu ili uklanjanje.

1.6. JavaMail API

JavaMail API[4] pruža set apstraktnih klasa koje definišu objekte uključene u jedan sistem elektronske pošte. Pored apstraktnih klasa, API takođe pruža i određeni broj konkretnih klasa koje nasleđuju apstraktne klase, i pritom implementiraju široko korišćene *Internet mail* protokole, u skladu sa specificiranim standardima. Klase koje definiše JavaMail API mogu biti nasleđene drugim klasama, koje dodaju funkcionalnost implementirajući nove protokole ukoliko za tim postoji potreba.

1.6.1. Arhitektura API-ja

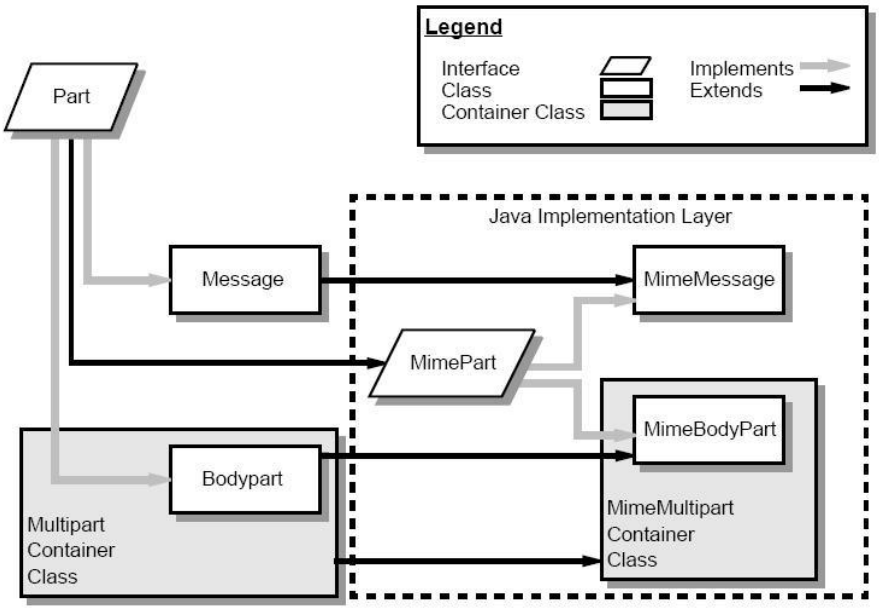
Osnovne komponente arhitekture JavaMail-a smeštene su u dva sloja – apstraktni sloj (*Abstract Layer*) i implementacioni sloj (*Implementation Layer*).

Apstraktni sloj deklariše klase, interfejse i apstraktne metode namenjene podršci funkcijama rukovanja poštom koje podržavaju **svi** sistemi za elektronsku poštu. Elementi API-ja sadržani u apstraktnom sloju namenjeni su nasleđivanju od strane drugih klasa po potrebi, kako bi podržali standardne tipove podataka, te da po potrebi pružaju podršku protokolima pristupa porukama i transporta poruka.

Implementacioni sloj implementira deo apstraktnog sloja koristeći Internet standarde – RFC822 i MIME (Multipurpose Internet Mail Extensions*)*.

Važno je napomenuti da JavaMail koristi JAF (JavaBeans Activation Framework) kako bi enkapsulirao podatke vezane za poruke i rukovao komandama namenjenim za interakciju sa tim podacima.

Opisana slojevita arhitektura omogućava različitim klijentima da koriste iste pozive kroz JavaMail API da bi poslali, primili i trajno sačuvali poruke, koristeći pritom različite tipove podataka iz raznih repozitorijuma poruka i različite protokole za transport poruka. Na slici 1.6. možemo pogledati osnovne klase i interfejse koje čine komponente slojevite arhitekture JavaMail API-ja.

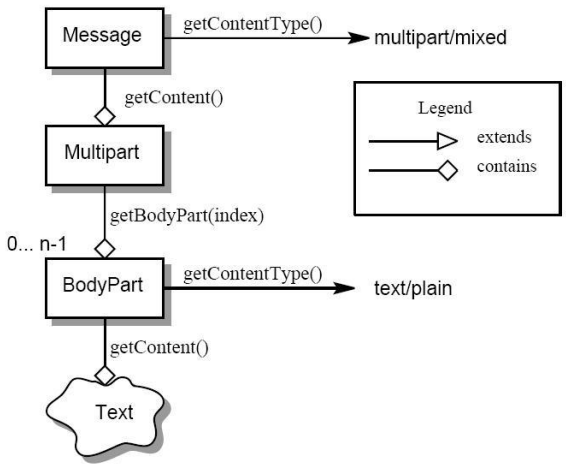


Slika 1.6. – Osnovne klase JavaMail API-ja

Klasa *Message* je apstraktna klasa koja definiše set atributa i tip podataka sadržaja poruke (*Content type*). Dati atributi specificiraju informacije potrebne za ispravno adresiranje poruke, poput atributa koji specificiraju od koga poruka dolazi (*From*) i kome je namenjena (*To*), na koju adresu da se uputi eventualni odgovor na poruku (*ReplyTo*), te i kratak opis sadržaja poruke (*Subject*). Pored toga, atributi klase *Message* u potpunosti definišu strukturu poruke. Konkretni sadržaj poruke je obavijen *DataHandler* objektom. Na ovaj način, objekat klase *Message* nema direktnih saznanja o prirodi i značenju svog sadržaja, čime su razdvojeni struktura i sadržaj poruke.

Time je omogućeno da objekat klase *Message* može da se sastoji od više delova, od kojih svaki sadrži svoj set atributa i svoj sadržaj. Takav objekat klase *Message* u sebi sadrži objekat klase *Multipart*, koja predstavlja kontejner objekata klase *BodyPart*. *BodyPart* klasa sadrži atribute koji detaljnije opisuju konkretni deo poruke, poput tipa podataka sadržaja, no ne uključuju atribute za adresiranje poruke koji su sadržani unutar *Message* klase.

Strukturu poruke koja se sastoji od više delova, predstavljenu UML dijagramom, možemo pogledati na slici 1.7.



Slika 1.7. – Struktura poruke od više delova

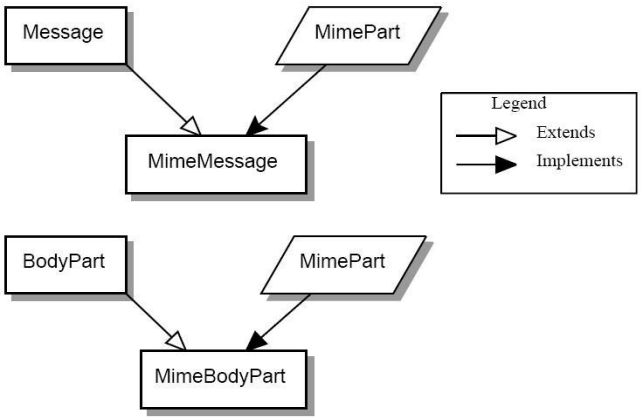
Klase *Message* i *Bodypart* implementiraju interfejs *Part*, koji definiše set standardnih atributa uobičajenih za većinu sistema elektronske pošte, kao i uobičajene *get* i *set* metode za pristup ovim atributima. Pored toga, *Part* definiše i tip podataka sadržaja (*Content type*).

1.6.2. *Internet Mail*

Iako JavaMail specifikacija ne definiše bilo kakve implementacije, API u implementacionom sloju ipak uključuje set klasa koje implementiraju Internet Mail standarde, definisane narednim RFC-ovima:

- RFC822 (*Standard for the Format of Internet Text Messages*) – opisuje strukturu poruka koje se razmenjuju putem Interneta
- RFC2045, RFC2046, RFC2047 – nabrojani MIME RFC-ovi definišu strukturu sadržaja poruke, definišući strukturirane delove poruka, mehanizam za identifikaciju različitih tipova medija i mehanizam za upotrebu različitih *character set*-ova unutar zaglavlja poruke, respektivno

Ove klase nasleđuju klase iz apstraktnog sloja API-ija, i omogućavaju klijentima kreiranje, slanje i prijem poruka u skladu sa nabrojanim standardima. Pogledajmo sliku 1.8., na kojoj je prikazan dijagram klasa implementacionog sloja.



Slika 1.8. – Klase implementacionog sloja

Kao što se može videti, *MimeMessage* klasa nasleđuje klasu *Message* i implementira *MimePart* interfejs. Ova klasa predstavlja implementaciju *e-mail* poruke u skladu sa RFC822 i MIME standardima, koje smo spomenuli nešto ranije. *MimeBodyPart* klasa nasleđuje *BodyPart* apstraktnu klasu i takođe implementira *MimePart* interfejs. *MimeBodyPart* opisuje jedan nezavisni deo poruke. U skladu sa strukturom apstraktnog sloja, implementirana je i klasa *MimeMultipart* kao naslednik klase *Multipart*, koja služi kao kontejner pojedinačnih delova poruke reprezentovanih *MimeBodyPart*-om i koja predstavlja sadržaj MIME poruke.

1.6.3. JavaMail *framework*

JavaMail *framework* je namenjen izvršavanju zadataka koji čine standardan proces rukovanja poštom tipične klijentske aplikacije:

- Kreiranje poruke sa zaglavljem (*header*) i telom (*body*).
- Kreiranje *Session* objekta, koji služi za autentifikaciju korisnika, kontroliše pristup repozitorijumu poruka i transportu.
- Slanje poruke listi primalaca.
- Prijem poruka iz repozitorijuma poruka.

Poruke su pohranjene u repozitorijumu poruka unutar objekata koje opisuje *Folder* klasa JavaMail Api-ja. Folder može sadržati kako poruke, tako i druge foldere, kreirajući pritom strukturu stabla. Unutar *Folder* klase definisane su sve metode potrebne za rukovanje sadržanim porukama, uključujući zahvat, dodavanje i brisanje poruka.

S druge strane, repozitorijum poruka koji sadrži strukturu *Folder* objekata i u njima sadržanih poruka, predstavljen je klasom *Store*. *Store* klasa takođe specificira i protokole pristupa folderima, dobavlja poruke koje su u njima pohranjene, te pruža i metode kojima se uspostavlja veza sa repozitorijumom, zahvataju folderi i zatvara uspostavljena veza. Klijent uglavnom započinje sesiju uspostavljanjem veze sa repozitorijumom reprezentovanim konkretnim *Store* objektom.

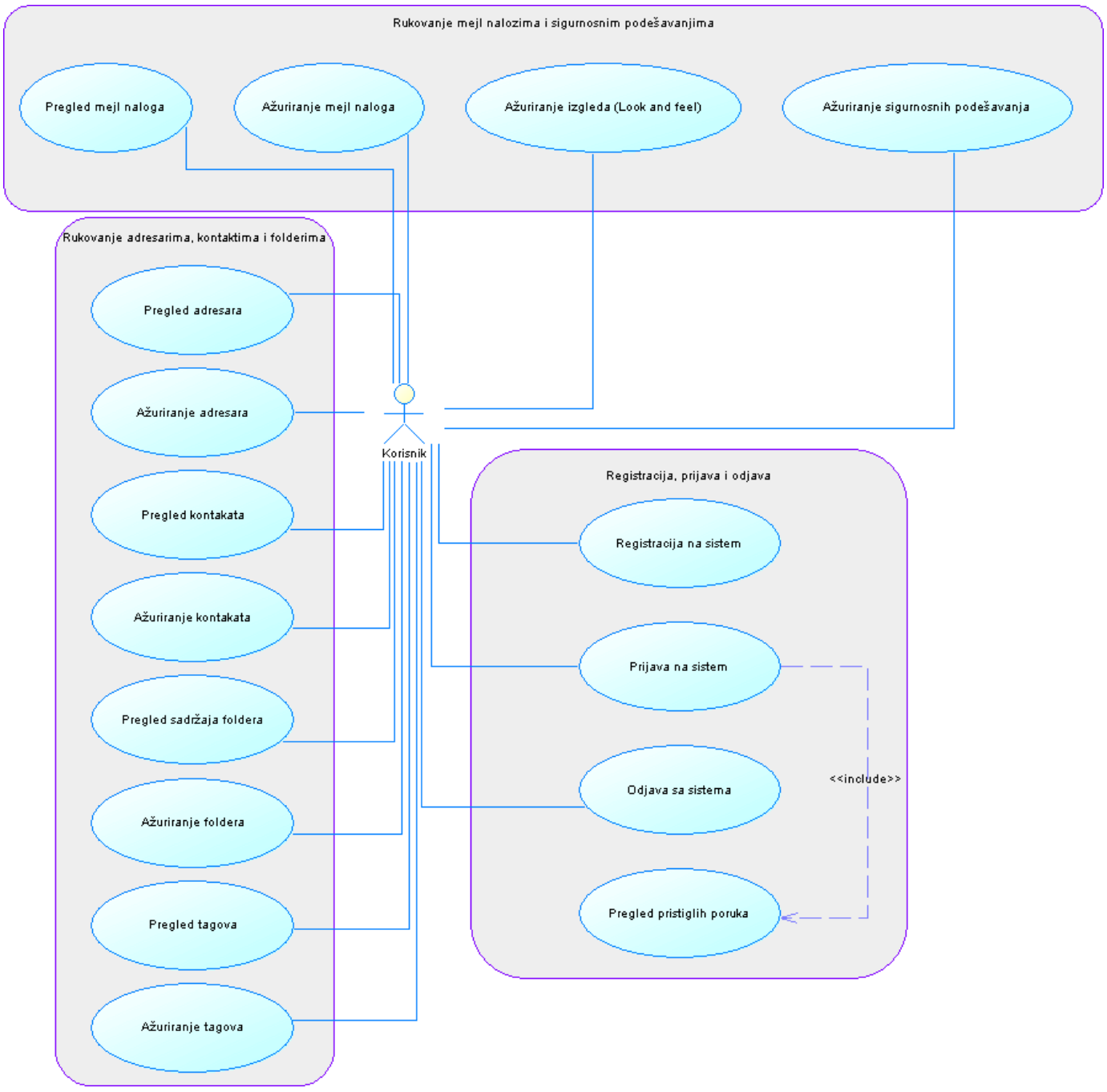
S druge strane, klasa *Transport* modeluje agenta koji upućuje poruku na odredišnu adresu. Da bismo poslali uspešno kreiranu instancu neke klase koja nasleđuje klasu *Message*, potrebno je pobuditi metodu *Transport.send*, koja će prosleđenu poruku uputiti listi primalaca navedenih u zaglavlju poruke.

2. SPECIFIKACIJA SISTEMA

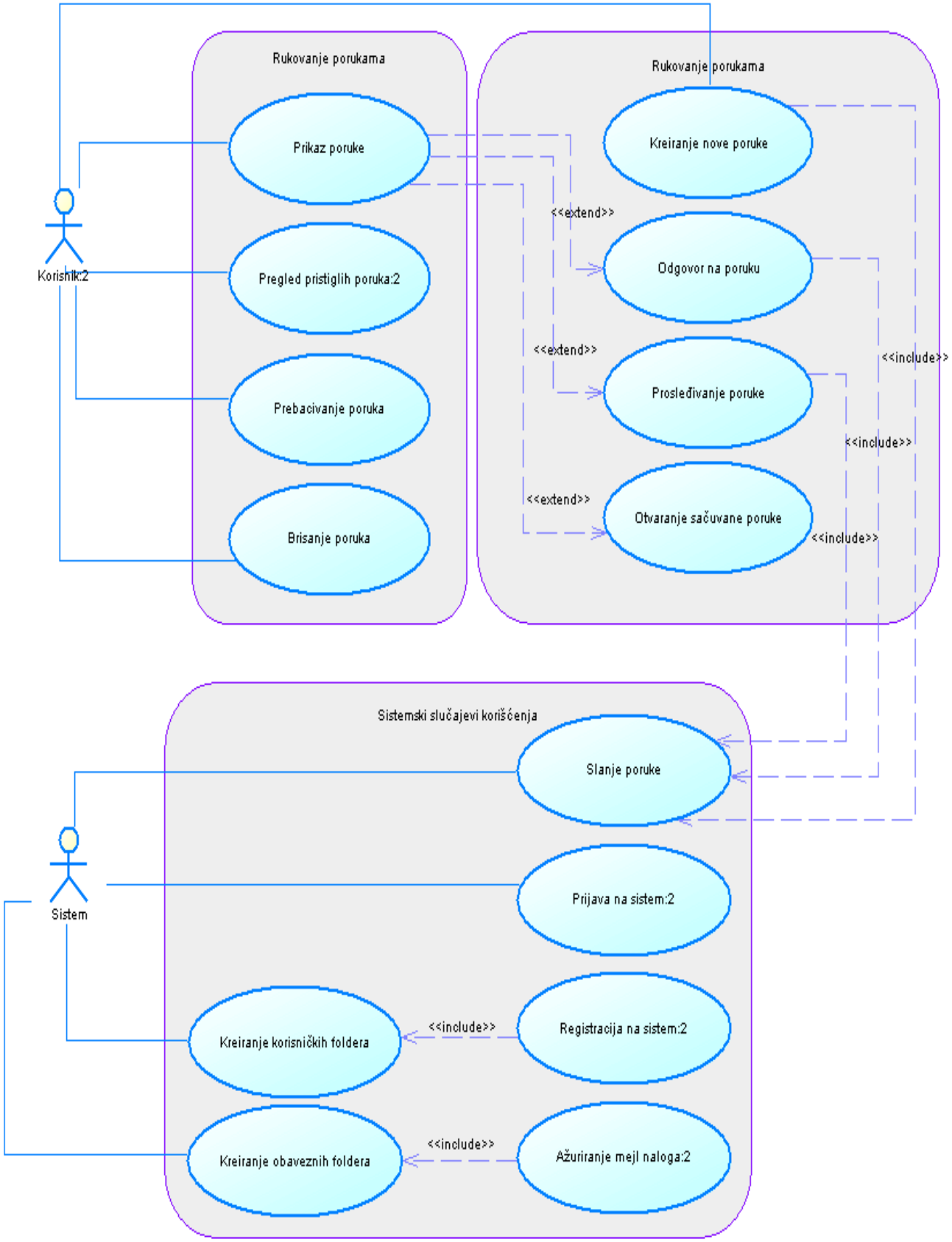
U ovom poglavlju će biti opisana specifikacija Webmail aplikacije pomoću UML dijagrama.

2.1. Dijagram slučajeva korišćenja

Na dijagramima 2.1. i 2.2.. prikazani su slučajevi korišćenja Webmail aplikacije. Svi slučajevi korišćenja su detaljnije opisani u nastavku teksta.



Dijagram 2.1. – Slučajevi korišćenja Web Mail aplikacije



Dijagram 2.2. – Slučajevi korišćenja Web Mail aplikacije

<table>
<tr><th>Naziv</th><th colspan="2">Registracija na sistem</th></tr>
<tr><td>Kratak opis</td><td colspan="2">Registracija korisnika na sistem Webmail aplikacije, unošenjem osnovnih podataka na stranici za registraciju</td></tr>
<tr><td>Preduslovi</td><td colspan="2">Korisnik nije registrovan</td></tr>
<tr><td>Garancije uspeha</td><td colspan="2">Korisnik je upućen na stranicu za prijavu</td></tr>
<tr><td>Garancije neuspeha</td><td colspan="2">Korisnik je vraćen na stranicu za registraciju na sistem</td></tr>
<tr><td>Osnovni uspešni scenario</td><td colspan="2"><table>
<tr><th>Korak</th><th>Akcija</th></tr>
<tr><td>1. (triger)</td><td>Korisnik unosi korisničko ime, lozinku, ponovljenu lozinku, registracioni emajl, ponovljen registracioni emajl, sigurnosno pitanje i sigurnosni odgovor na stranici za prijavu i klikne mišem na dugme Register na stranici za registraciju</td></tr>
<tr><td>2.</td><td>Sistem vrši proveru unesenih podataka, vrši Kreiranje korisničkih foldera i upućuje korisnika na stranicu za prijavu</td></tr>
</table></td></tr>
<tr><td>Tačke proširenja</td><td colspan="2">2.a. Uneseno korisničko ime i registracioni emajl su registrovani u sistemu, lozinka i ponovljena lozinka se ne slažu, registracioni emajl i ponovljeni registracioni emajl se ne slažu, neko od polja je prazno, povratak na korak 1.</td></tr>
</table>

<table>
<tr><th>Naziv</th><th colspan="2">Prijava na sistem</th></tr>
<tr><td>Kratak opis</td><td colspan="2">Prijava korisnika na sistem Webmail aplikacije, unošenjem korisničkog imena i lozinke na stranici za prijavu</td></tr>
<tr><td>Preduslovi</td><td colspan="2">Korisnik nije prijavljen na sistem</td></tr>
<tr><td>Garancije uspeha</td><td colspan="2">Korisnik je upućen na osnovnu stranicu aplikacije</td></tr>
<tr><td>Garancije neuspeha</td><td colspan="2">Korisnik je vraćen na stranicu za prijavu na sistem</td></tr>
<tr><td>Osnovni uspešni scenario</td><td colspan="2"><table>
<tr><th>Korak</th><th>Akcija</th></tr>
<tr><td>1. (triger)</td><td>Korisnik unosi korisničko ime i lozinku na stranici za prijavu i klikne mišem na dugme Login na stranici za prijavu</td></tr>
<tr><td>2.</td><td>Sistem vrši proveru unesenih podataka, upućuje korisnika na osnovnu stranicu aplikacije, i vrši Pregled pristiglih poruka</td></tr>
</table></td></tr>
<tr><td>Tačke proširenja</td><td colspan="2">2.a. Uneseno korisničko ime i lozinka nisu registrovani u sistemu, korisničko ime i lozinka se ne poklapaju sa podacima u sistemu, sistem vraća korisnika na početnu stranicu</td></tr>
</table>

<table>
<tr><th>Naziv</th><th>Odjava sa sistema</th></tr>
<tr><td>Kratak opis</td><td>Odjava korisnika sa sistema Webmail aplikacije</td></tr>
<tr><td>Preduslovi</td><td>Korisnik je prijavljen</td></tr>
<tr><td>Garancije uspeha</td><td>Korisnik je vraćen na stranicu za prijavu na sistem</td></tr>
<tr><td>Garancije neuspeha</td><td>Nema</td></tr>
<tr><td>Osnovni uspešni scenario</td><td>
<table>
<tr><th>Korak</th><th>Akcija</th></tr>
<tr><td>1. (triger)</td><td>Korisnik klikne mišem na link za odjavu sa sistema, u glavnom meniju aplikacije</td></tr>
<tr><td>2.</td><td>Sistem odjavljuje korisnika, i upućuje ga na index stranu</td></tr>
</table>
</td></tr>
<tr><td>Tačke proširenja</td><td>Nema</td></tr>
</table>

<table>
<tr><th>Naziv</th><th>Ažuriranje adresara</th></tr>
<tr><td>Kratak opis</td><td>Ažuriranje adresara podrazumeva dodavanje novog adresara, izmenu i brisanje postojećeg adresara</td></tr>
<tr><td>Preduslovi</td><td>U slučaju izmene i brisanja, adresar je selektovan</td></tr>
<tr><td>Garancije uspeha</td><td>U stablu adresara je vidljiva promena nastala ažuriranjem</td></tr>
<tr><td>Garancije neuspeha</td><td>Prikazano obaveštenje o grešci pri ažuriranju</td></tr>
<tr><td>Osnovni uspešni scenario</td><td>
<table>
<tr><th>Korak</th><th>Akcija</th></tr>
<tr><td>1. (triger)</td><td>Korisnik klikne mišem dugme New Address Book na stranici</td></tr>
<tr><td>2.</td><td>Sistem prikazuje formu za unos podataka o adresaru</td></tr>
<tr><td>3.</td><td>Korisnik unosi podatke i klikne mišem na dugme Save</td></tr>
<tr><td>4</td><td>Sistem kreira novi adresar</td></tr>
</table>
</td></tr>
<tr><td>Tačke proširenja</td><td>3.a. Korisnik klikne mišem na dugme Cancel, korak 4. se ne izvršava
4.a. Korisnik nije uneo obavezne podatke za novi adresar ili uneseni podaci ne ispunjavaju osnovne zahteve ispravnosti, povratak na korak 3.</td></tr>
</table>

<table>
<tr><th>Naziv</th><th>Ažuriranje kontakata</th></tr>
<tr><td>Kratak opis</td><td>Ažuriranje kontakata podrazumeva dodavanje novog kontakta, izmenu i brisanje postojećeg kontakta</td></tr>
<tr><td>Preduslovi</td><td>U slučaju izmene i brisanja, kontakt je selektovan</td></tr>
<tr><td>Garancije uspeha</td><td>U tabeli kontakata je vidljiva promena nastala ažuriranjem</td></tr>
<tr><td>Garancije neuspeha</td><td>Prikazano obaveštenje o grešci pri ažuriranju</td></tr>
<tr><td>Osnovni uspešni scenario</td><td>
<table>
<tr><th>Korak</th><th>Akcija</th></tr>
<tr><td>1. (triger)</td><td>Korisnik klikne mišem dugme New Contact na stranici</td></tr>
<tr><td>2.</td><td>Sistem prikazuje formu za unos podataka o kontaktu</td></tr>
<tr><td>3</td><td>Korisnik unosi podatke i klikne mišem na dugme Save</td></tr>
<tr><td>4.</td><td>Sistem kreira novi kontakt</td></tr>
</table>
</td></tr>
<tr><td>Tačke proširenja</td><td>3.a. Korisnik klikne mišem na dugme Cancel, korak 4. se ne izvršava
4.a. Korisnik nije uneo obavezne podatke za novi kontakt ili uneseni podaci ne ispunjavaju osnovne zahteve ispravnosti, povratak na korak 3.</td></tr>
</table>

<table>
<tr><th>Naziv</th><th>Ažuriranje tagova</th></tr>
<tr><td>Kratak opis</td><td>Ažuriranje tagova podrazumeva dodavanje novog taga, izmenu i brisanje postojećeg taga</td></tr>
<tr><td>Preduslovi</td><td>U slučaju izmene i brisanja, tag je selektovan</td></tr>
<tr><td>Garancije uspeha</td><td>U tabeli tagova je vidljiva promena nastala ažuriranjem</td></tr>
<tr><td>Garancije neuspeha</td><td>Prikazano obaveštenje o grešci pri ažuriranju</td></tr>
<tr><td>Osnovni uspešni scenario</td><td>
<table>
<tr><th>Korak</th><th>Akcija</th></tr>
<tr><td>1. (triger)</td><td>Korisnik klikne mišem dugme New Tag na stranici</td></tr>
<tr><td>2.</td><td>Sistem prikazuje formu za unos podataka o tagu</td></tr>
<tr><td>3.</td><td>Korisnik unosi podatke i klikne mišem na dugme Save</td></tr>
<tr><td>4.</td><td>Sistem kreira novi tag</td></tr>
</table>
</td></tr>
<tr><td>Tačke proširenja</td><td>3.a. Korisnik klikne mišem na dugme Cancel, korak 4. se ne izvršava
4.a. Korisnik nije uneo obavezne podatke za novi tag ili uneseni podaci ne ispunjavaju osnovne zahteve ispravnosti, povratak na korak 3.</td></tr>
</table>

<table>
<tr><th>Naziv</th><th colspan="2">Ažuriranje foldera</th></tr>
<tr><td>Kratak opis</td><td colspan="2">Ažuriranje foldera podrazumeva dodavanje novog korenskog foldera, dodavanje podfoldera postojećem folderu, izmenu i brisanje postojećeg foldera.</td></tr>
<tr><td>Preduslovi</td><td colspan="2">Za dodavanje podfoldera i brisanje foldera, postojanje barem jednog foldera kreiranog od strane korisnika</td></tr>
<tr><td>Garancije uspeha</td><td colspan="2">U stablu foldera na glavnoj stranici vidljiva je promena nastala ažuriranjem</td></tr>
<tr><td>Garancije neuspeha</td><td colspan="2">Prikazano obaveštenje o grešci pri ažuriranju</td></tr>
<tr><td>Osnovni uspešni scenario</td><td colspan="2"><table>
<tr><th>Korak</th><th>Akcija</th></tr>
<tr><td>1. (triger)</td><td>Korisnik klikne mišem na link New folder na glavnoj stranici</td></tr>
<tr><td>2.</td><td>Sistem prikazuje formu za unos podataka o folderu</td></tr>
<tr><td>3.</td><td>Korisnik unosi podatke i klikne mišem na dugme Create</td></tr>
<tr><td>4.</td><td>Sistem kreira novi folder</td></tr>
</table></td></tr>
<tr><td>Tačke proširenja</td><td colspan="2">3.a Korisnik klikne mišem na dugme Cancel, sistem upućuje korisnika na osnovnu stranicu, korak 4. se ne izvršava
4.a. Korisnik nije uneo obavezne podatke za novi folder ili uneseni podaci ne ispunjavaju osnovne zahteve ispravnosti, povratak na korak 3.</td></tr>
</table>

<table>
<tr><th>Naziv</th><th colspan="2">Ažuriranje emajl naloga</th></tr>
<tr><td>Kratak opis</td><td colspan="2">Ažuriranje emajl naloga podrazumeva dodavanje novog emajl naloga, izmenu i brisanje postojećeg emajl naloga.</td></tr>
<tr><td>Preduslovi</td><td colspan="2">U slučaju izmene i brisanja, emajl nalog je selektovan</td></tr>
<tr><td>Garancije uspeha</td><td colspan="2">U tabeli emajl naloga je vidljiva promena nastala ažuriranjem</td></tr>
<tr><td>Garancije neuspeha</td><td colspan="2">Prikazano obaveštenje o grešci pri ažuriranju</td></tr>
<tr><td>Osnovni uspešni scenario</td><td colspan="2"><table>
<tr><th>Korak</th><th>Akcija</th></tr>
<tr><td>1. (triger)</td><td>Korisnik klikne mišem dugme Add new account na stranici</td></tr>
<tr><td>2.</td><td>Sistem prikazuje formu za unos podataka o emajl nalogu</td></tr>
<tr><td>3.</td><td>Korisnik unosi podatke i klikne mišem na dugme Save</td></tr>
<tr><td>4.</td><td>Sistem kreira novi emajl nalog</td></tr>
</table></td></tr>
<tr><td>Tačke proširenja</td><td colspan="2">3.a. Korisnik klikne mišem na dugme Cancel, korak 4. se ne izvršava
4.a. Korisnik nije uneo obavezne podatke za novi emajl nalog ili uneseni podaci ne ispunjavaju osnovne</td></tr>
</table>

	zahteve ispravnosti, povratak na korak 3.

Naziv	**Ažuriranje izgleda (look and feel)**
Kratak opis	Ažuriranje izgleda podrazumeva izmenu opcija vezanih za izgled aplikacije.
Preduslovi	Nema
Garancije uspeha	Izmena je izvršena i vidljiva na strani aplikacije
Garancije neuspeha	Nema
Osnovni uspešni scenario	**Korak** / **Akcija** 1. (triger) — Korisnik klikne mišem na opciju *Options* unutar menija *Tools* 2. — Sistem prikazuje formu za odabir opcija o izgledu aplikacije 3. — Korisnik unosi podatke i klikne mišem na dugme *Save* 4. — Sistem ponovo iscrtava glavnu stranu aplikacije u skladu sa izmenama
Tačke proširenja	3.a. Korisnik klikne mišem na dugme Cancel, korak 4. se ne izvršava

Naziv	**Ažuriranje sigurnosnih podešavanja**
Kratak opis	Ažuriranje sigurnosnih podešavanja podrazumeva izmenu opcija vezanih za izgled aplikacije.
Preduslovi	Nema
Garancije uspeha	Izmena je izvršena
Garancije neuspeha	Nema
Osnovni uspešni scenario	**Korak** / **Akcija** 1. (triger) — Korisnik klikne mišem na opciju *Security options* 2. — Sistem prikazuje formu za unos sigurnosnih podataka 3. — Korisnik unosi podatke i klikne mišem na dugme *Save* 4. — Sistem vrši izmeni sigurnosnih podataka
Tačke proširenja	3.a. Korisnik klikne mišem na dugme Cancel, korak 4. se ne izvršava 4.a. Korisnik nije uneo obavezne podatke ili uneseni podaci ne ispunjavaju osnovne zahteve ispravnosti, povratak na korak 3.

Naziv	**Pregled sadržaja foldera**	
Kratak opis	Prikazivanje poruka koje su pohranjene u folderu koji je korisnik selektovao	
Preduslovi	Nema	
Garancije uspeha	Prikazan sadržaj foldera u tabeli poruka na osnovnoj stranici	
Garancije neuspeha	Nema	
Osnovni uspešni scenario	**Korak**	**Akcija**
	1. (triger)	Korisnik klikne mišem na folder prikazan u stablu foldera na osnovnoj stranici
	2.	Sistem u tabeli poruka na glavnoj stranici prikazuje poruke koje su sadržane u selektovanom folderu
Tačke proširenja	Nema	

Naziv	**Pregled pristiglih poruka**	
Kratak opis	Pregledanje novih poruka za prijavljenog korisnika	
Preduslovi	Nema	
Garancije uspeha	Prikazane nove poruke u tabeli sa porukama na osnovnoj stranici	
Garancije neuspeha	Prikazano obaveštenje o grešci pri prijemu novih poruka	
Osnovni uspešni scenario	**Korak**	**Akcija**
	1. (triger)	Korisnik klikne mišem na meni *Get mail* u glavnom meniju is odabere željeni nalog ili odabere preuzimanje poruka za sve naloge, na osnovnoj stranici
	2.	Sistem vrši prijem poruka i smešta ih u *Inbox* folder odgovarajućeg naloga
	3.	Sistem prikazuje nove poruke
Tačke proširenja	Nema	

<table>
<tr><th>Naziv</th><th colspan="2">Prikaz poruke</th></tr>
<tr><td>Kratak opis</td><td colspan="2">Pregledanje kompletnog sadržaja poruke, uključujući zaglavlje, tekst poruke i priložene attachment-e</td></tr>
<tr><td>Preduslovi</td><td colspan="2">Zaglavlje poruke je prikazano u tabeli sa porukama na osnovnoj stranici i panelu sa osnovnim podacima poruke</td></tr>
<tr><td>Garancije uspeha</td><td colspan="2">Sadržaj poruke je prikazan</td></tr>
<tr><td>Garancije neuspeha</td><td colspan="2">Nema</td></tr>
<tr><td>Osnovni uspešni scenario</td><td colspan="2"><table><tr><th>Korak</th><th>Akcija</th></tr><tr><td>1. (triger)</td><td>Korisnik klikne mišem na red u tabeli sa porukama na osnovnoj stranici</td></tr><tr><td>2.</td><td>Sistem prikazuje sadržaj poruke</td></tr></table></td></tr>
<tr><td>Tačke proširenja</td><td colspan="2">3. Korisnik klikne mišem na link kojim je predstavljen attachment sadržan u poruci
4. Browser prikazuje dijalog za preuzimanje attachment-a
5. Korisnik klikne mišem na Download dugme na dijalogu i preuzima attachment</td></tr>
</table>

<table>
<tr><th>Naziv</th><th colspan="2">Prebacivanje poruka</th></tr>
<tr><td>Kratak opis</td><td colspan="2">Prebacivanje jedne ili više poruka, iz foldera u kom se trenutno nalaze, u željeni folder</td></tr>
<tr><td>Preduslovi</td><td colspan="2">U aktuelnom folderu postoje poruke, selektovana je jedna poruka, i postoji barem jedan korisnički folder unutar istog naloga u koji se poruke mogu prebaciti</td></tr>
<tr><td>Garancije uspeha</td><td colspan="2">Poruke su premeštene iz aktuelnog foldera, uz odgovarajuće obaveštenje</td></tr>
<tr><td>Garancije neuspeha</td><td colspan="2">Prikazano obaveštenje o grešci pri prebacivanju poruka</td></tr>
<tr><td>Osnovni uspešni scenario</td><td colspan="2"><table><tr><th>Korak</th><th>Akcija</th></tr><tr><td>1. (triger)</td><td>Korisnik selektuje poruku u aktuelnom folderu i pomoću drag-and-drop je premesti u drugi folder</td></tr><tr><td>2.</td><td>Sistem vrši prebacivanje poruka u selektovani folder</td></tr></table></td></tr>
<tr><td>Tačke proširenja</td><td colspan="2">Nema</td></tr>
</table>

<table>
<tr><th>Naziv</th><th colspan="2">Brisanje poruka</th></tr>
<tr><td>Kratak opis</td><td colspan="2">U zavisnosti od podešavanja izvršiće se obeležavanje poruke da je obrisana, trajno brisanje poruke ili prebacivanje u folder Trash ukoliko je poruka smeštena u nekom drugom folderu</td></tr>
<tr><td>Preduslovi</td><td colspan="2">U aktuelnom folderu postoje poruke, selektovana je jedna poruka</td></tr>
<tr><td>Garancije uspeha</td><td colspan="2">Poruka je prebačena ili obrisana</td></tr>
<tr><td>Garancije neuspeha</td><td colspan="2">Prikazano obaveštenje o grešci pri brisanju poruke</td></tr>
<tr><td>Osnovni uspešni scenario</td><td>
<table>
<tr><th>Korak</th><th>Akcija</th></tr>
<tr><td>1. (triger)</td><td>Korisnik selektuje poruku u aktuelnom folderu i klikne na dugme Delete</td></tr>
<tr><td>2.</td><td>Sistem trajno briše poruku</td></tr>
</table>
</td></tr>
<tr><td>Tačke proširenja</td><td colspan="2">2.a. Podešavanje podrazumeva markiranje poruke, sistem markira poruku, korak 3. se ne izvršava
2.b. . Podešavanje podrazumeva premeštanje poruke u Trash folder, aktuelni folder nije folder Trash, sistem prebacuje selektovane poruke u folder Trash, korak 3. se ne izvršava u suprotnom sistem briše poruku iz Trash foldera</td></tr>
</table>

<table>
<tr><th>Naziv</th><th colspan="2">Kreiranje nove poruke</th></tr>
<tr><td>Kratak opis</td><td colspan="2">Pisanje sadržaja nove poruke, dodavanje attachment-a i slanje poruke</td></tr>
<tr><td>Preduslovi</td><td colspan="2">Nema</td></tr>
<tr><td>Garancije uspeha</td><td colspan="2">U zavisnosti od podešavanja poruka je smeštena u Sent folder</td></tr>
<tr><td>Garancije neuspeha</td><td colspan="2">Prikazano obaveštenje o grešci pri slanju poruke</td></tr>
<tr><td>Osnovni uspešni scenario</td><td>
<table>
<tr><th>Korak</th><th>Akcija</th></tr>
<tr><td>1. (triger)</td><td>Korisnik klikne na link Write na glavnoj strani</td></tr>
<tr><td>2.</td><td>Sistem prikazuje formu za kreiranje nove poruke</td></tr>
<tr><td>3.</td><td>Korisnik navodi primaoce poruke, unosi tekst poruke, dodaje attachment-e</td></tr>
<tr><td>4.</td><td>Korisnik klikne na dugme Send</td></tr>
<tr><td>5.</td><td>Sistem vrši Slanje poruke</td></tr>
</table>
</td></tr>
<tr><td>Tačke proširenja</td><td colspan="2">4. Korisnik klikne na dugme Cancel, korak 5. se ne izvršava</td></tr>
</table>

<table>
<tr><th>Naziv</th><th colspan="2">Odgovor na poruku</th></tr>
<tr><td>Kratak opis</td><td colspan="2">Pisanje odgovora na postojeću poruku, eventualno dodavanje attachment-a i slanje poruke</td></tr>
<tr><td>Preduslovi</td><td colspan="2">Prikazan je sadržaj postojeće poruke na osnovnoj stranici, poruka je selektovana</td></tr>
<tr><td>Garancije uspeha</td><td colspan="2">U zavisnosti od podešavanja poruka je smeštena u Sent folder</td></tr>
<tr><td>Garancije neuspeha</td><td colspan="2">Prikazano obaveštenje o grešci pri slanju poruke</td></tr>
<tr><td>Osnovni uspešni scenario</td><td colspan="2"><table>
<tr><th>Korak</th><th>Akcija</th></tr>
<tr><td>1. (triger)</td><td>Korisnik klikne na dugme Reply na osnovnoj stranici</td></tr>
<tr><td>2.</td><td>Sistem prikazuje formu za kreiranje poruke, sa unesenim primaocem i opisom poruke (Subject), te unesenim tekstom originalne poruke</td></tr>
<tr><td>3.</td><td>Korisnik eventualno dopunjava trenutni sadržaj poruke i/ili dodaje attachment-e</td></tr>
<tr><td>4.</td><td>Korisnik klikne na dugme Send message</td></tr>
<tr><td>5.</td><td>Sistem vrši Slanje poruke</td></tr>
</table></td></tr>
<tr><td>Tačke proširenja</td><td colspan="2">4.a. Korisnik klikne na dugme Cancel na dijalogu, korak 5. se ne izvršava</td></tr>
</table>

<table>
<tr><th>Naziv</th><th colspan="2">Prosleđivanje poruke</th></tr>
<tr><td>Kratak opis</td><td colspan="2">Prosleđivanje postojeće poruke, eventualno dodavanje attachment-a i slanje poruke</td></tr>
<tr><td>Preduslovi</td><td colspan="2">Prikazan je sadržaj postojeće poruke na osnovnoj stranici, poruka je selektovana</td></tr>
<tr><td>Garancije uspeha</td><td colspan="2">U zavisnosti od podešavanja poruka je smeštena u Sent folder</td></tr>
<tr><td>Garancije neuspeha</td><td colspan="2">Prikazano obaveštenje o grešci pri slanju poruke</td></tr>
<tr><td>Osnovni uspešni scenario</td><td colspan="2"><table>
<tr><th>Korak</th><th>Akcija</th></tr>
<tr><td>1. (triger)</td><td>Korisnik klikne na dugme Forward na osnovnoj stranici</td></tr>
<tr><td>2.</td><td>Sistem prikazuje formu za kreiranje poruke, sa unesenim opisom poruke (Subject), unesenim tekstom, i attachment-ima originalne poruke</td></tr>
<tr><td>3.</td><td>Korisnik navodi primaoce poruke, eventualno dopunjava trenutni sadržaj poruke i/ili dodaje attachment-e</td></tr>
<tr><td>4.</td><td>Korisnik klikne na dugme Send</td></tr>
<tr><td>5.</td><td>Sistem vrši Slanje poruke</td></tr>
</table></td></tr>
<tr><td>Tačke proširenja</td><td colspan="2">4.a. Korisnik klikne na dugme Cancel na dijalogu, korak 5. se ne izvršava</td></tr>
</table>

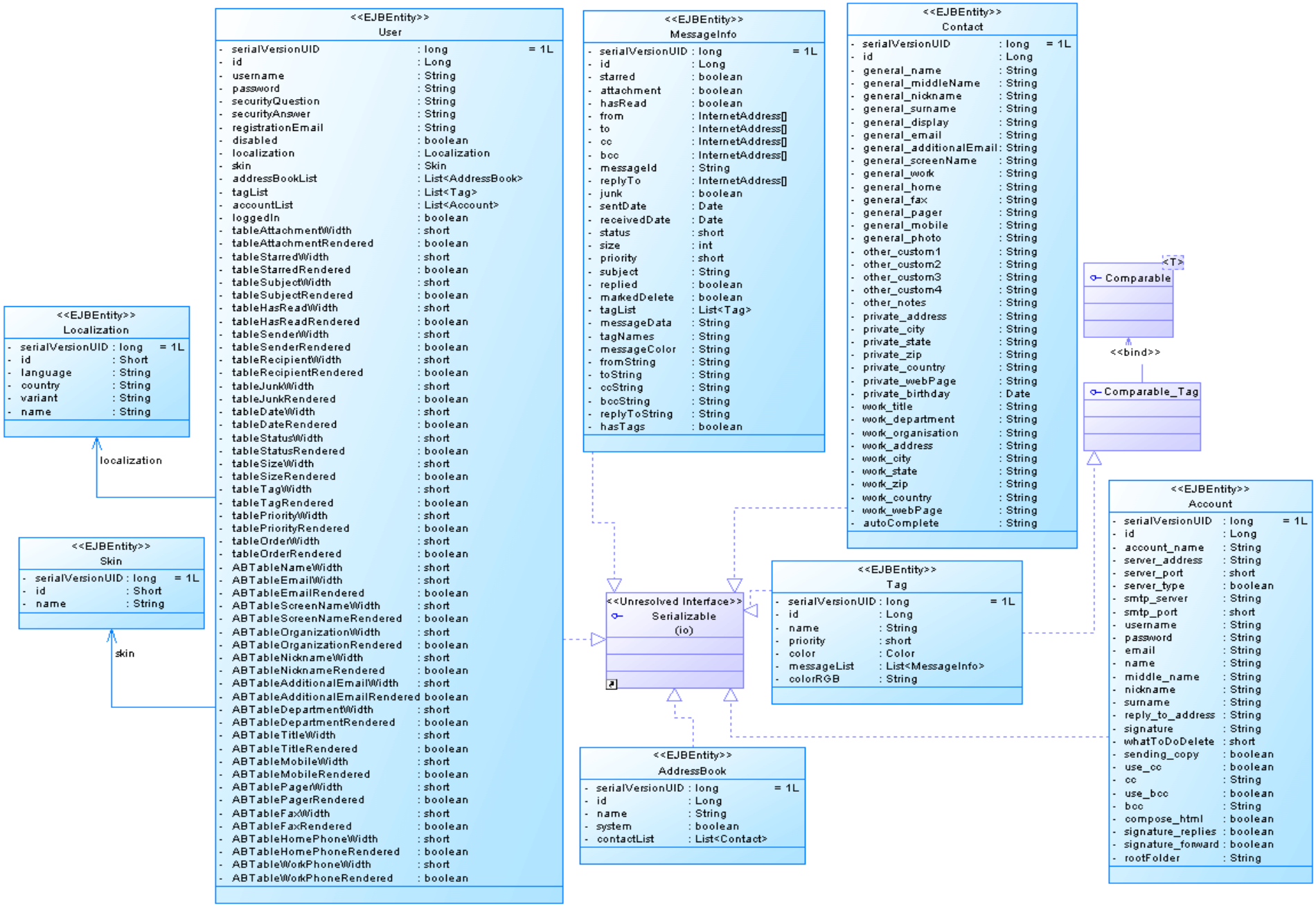
Naziv	**Slanje poruke**	
Kratak opis	Prosleđivanje poruka na adrese koje je korisnik naveo	
Preduslovi	Korisnik je naveo barem jednu adresu primaoca	
Garancije uspeha	Poruka uspešno poslana	
Garancije neuspeha	Poruka nije poslana	
Osnovni uspešni scenario	**Korak**	**Akcija**
	1. (triger)	Korisnik klikne na dugme *Send* na dijalogu za kreiranje poruke
	2.	Sistem proverava ispravnost adresa primaoca
	3.	Sistem prosleđuje poruku primaocima
Tačke proširenja	2.a. Sistem prikazuje obaveštenje o neispravnoj adresi primaoca, korak 3. se ne izvršava	

Naziv	**Kreiranje obaveznih foldera**	
Kratak opis	Kreiranje obaveznih foldera, *Inbox*, *Sent* i *Trash*, koje mora posedovati svaki emajl nalog. Ovi folderi ne mogu se obrisati niti preimenovati	
Preduslovi	Emajl nalog se kreira, i obavezni folderi nisu kreirani	
Garancije uspeha	Obavezni folderi uspešno kreirani	
Garancije neuspeha	Obavezni folderi nisu kreirani	
Osnovni uspešni scenario	**Korak**	**Akcija**
	1. (triger)	Korisnik klikne na *Save* dugme na stranici za kreiranje emajl naloga
	2.	Sistem proverava da li su za emajl nalog već kreirani obavezni folderi
	3.	Sistem kreira obavezne foldere
Tačke proširenja	3.a. Obavezni folderi postoje, stoga se ne kreiraju ponovo	

<table>
<tr><th>Naziv</th><th colspan="2">Kreiranje korisničkih foldera</th></tr>
<tr><td>Kratak opis</td><td colspan="2">Kreiranje korisničkih foldera, accounts i contact_images, koje mora posedovati svaki korisnički nalog. Ovi folderi ne mogu se obrisati niti preimenovati</td></tr>
<tr><td>Preduslovi</td><td colspan="2">Korisnik se prvi put prijavljuje na sistem i korisnički folderi nisu kreirani</td></tr>
<tr><td>Garancije uspeha</td><td colspan="2">Korisnički folderi uspešno kreirani</td></tr>
<tr><td>Garancije neuspeha</td><td colspan="2">Korisnički folderi nisu kreirani</td></tr>
<tr><td>Osnovni uspešni scenario</td><td colspan="2"><table>
<tr><th>Korak</th><th>Akcija</th></tr>
<tr><td>1. (triger)</td><td>Korisnik klikne na Register dugme na stranici za registraciju</td></tr>
<tr><td>2.</td><td>Sistem proverava da li su za korisnički nalog već kreirani folderi</td></tr>
<tr><td>3.</td><td>Sistem kreira korisničke foldere</td></tr>
</table></td></tr>
<tr><td>Tačke proširenja</td><td colspan="2">3.a. Korisnički folderi postoje, stoga se ne kreiraju ponovo</td></tr>
</table>

2.2. Dijagrami klasa

U nastavku su predstavljeni dijagrami klasa Webmail aplikacije. Klase su na dijagramima grupisane u logičke celine – klase koje implementiraju *entity*-je, klase koje implementiraju *managed bean*-ove i pomoćne klase. U nastavku je dato i kraće objašnjenje uloge klasa koje su vidljive na dijagramima.



Dijagram 2.3. – Dijagram *entity* klasa

Na dijagramu 2.3. su predstavljene *entity* klase, realizovane u Webmail aplikaciji.

Klasa *User* opisuje korisnički nalog Webmail aplikacije.

Klasa *Localization* opisuje podešavanja vezana za lokalizaciju i internacionalizaciju koja je podržana na sistemu.

Klasa *Skin* opisuje moguće teme (skin) za prikaz aplikacije.

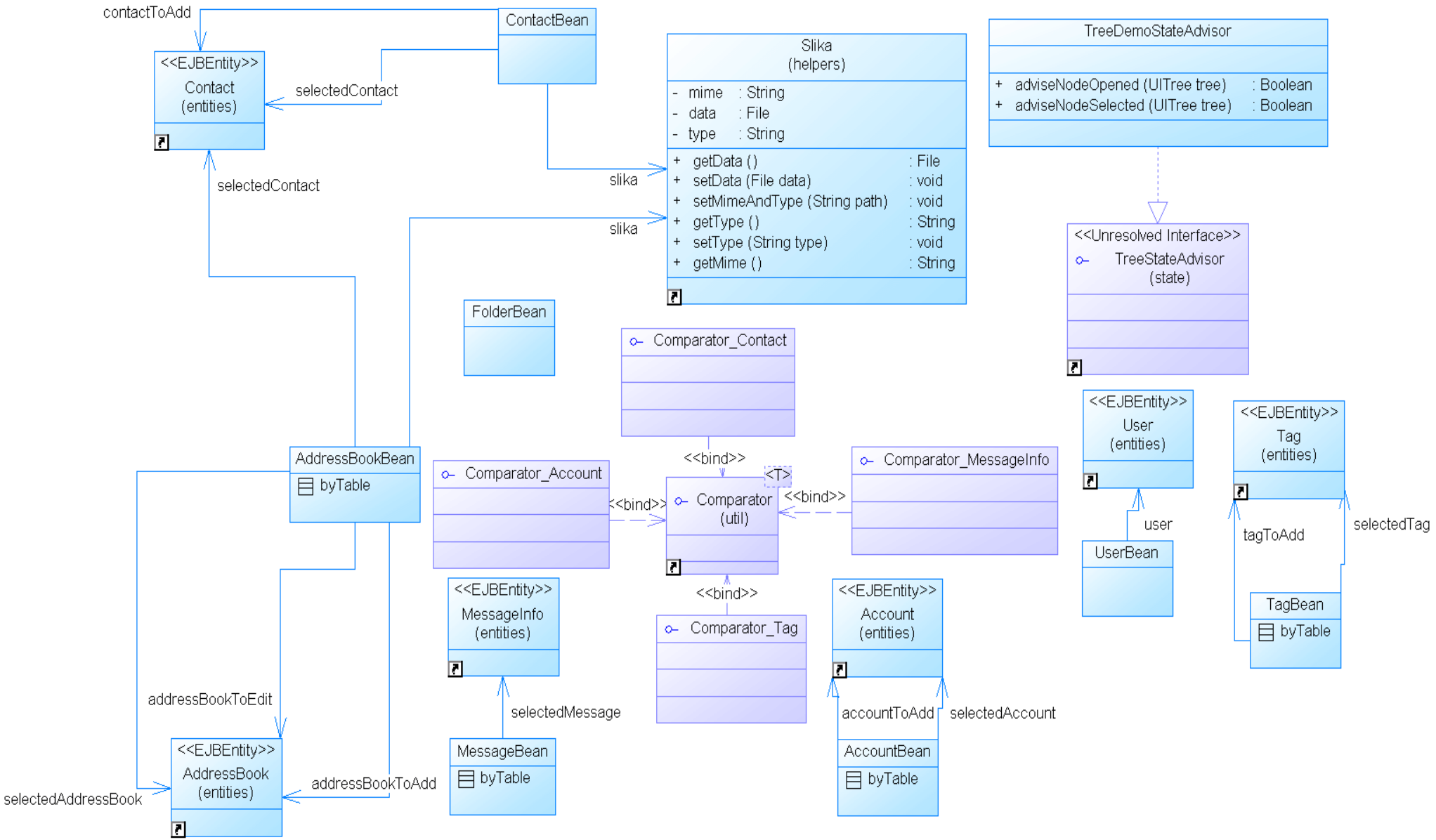
Klasa *AddressBook* opisuje korisnički adresar. Unutar klase *AddressBook* definisana je i lista tipa *Contact* koja sadrži sve kontakte u datom adresaru.

Klasa *Tag* opisuje tag (oznaku) kojim se može obeležiti poruka. Unutar klase *Tag* definisana je i lista tipa *MessagingInfo* koja sadrži sve poruke koje su obeležene datim tagom.

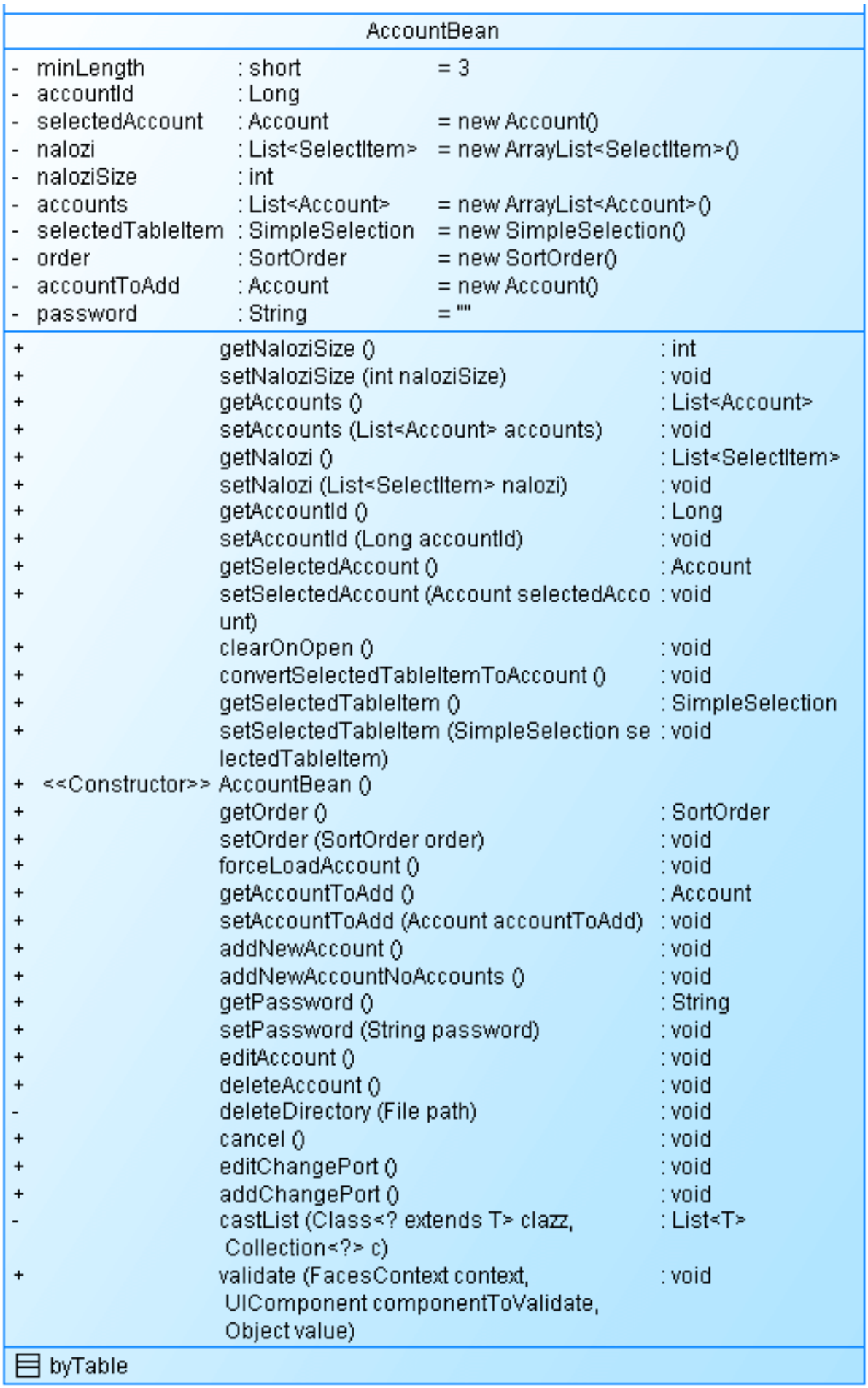
Klasa *MessageInfo* opisuje važne informacije vezane za svaku poruku (datum kad je primljena, lokacija na fajl sistemu gde se nalaze fajlovi same poruke…).

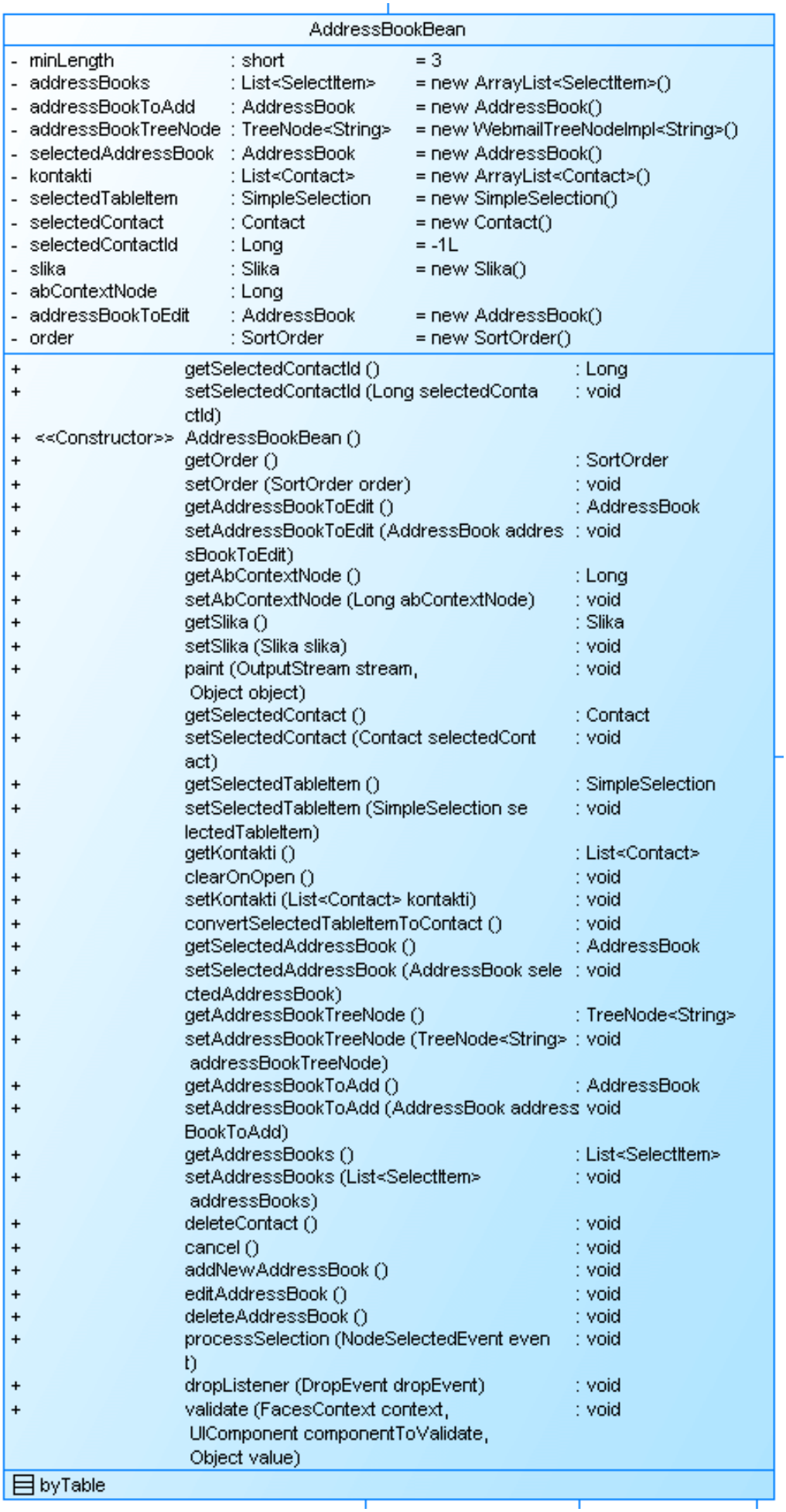
Klasa *Account* opisuje emajl nalog i osnovne podatke vezane za emajl naloge (adresa servera, pristupni port, korisničko ime i šifra…). Svaki nalog obavezno mora biti opisan portom, adresom, korisničkim imenom i šifrom.

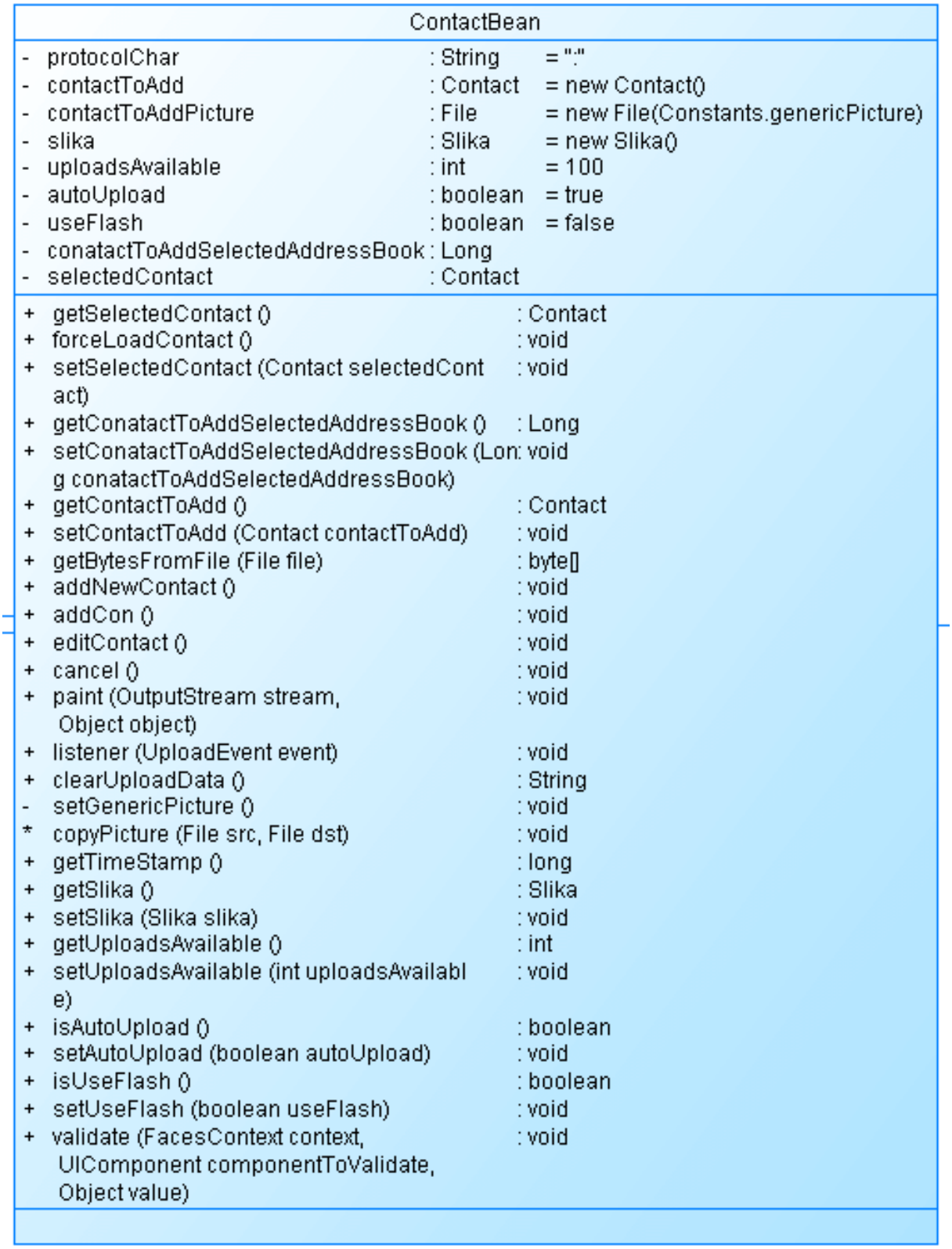
Klasa *Contact* opisuje kontakt i osnovne podatke vezane za dati kontakt. Svaki kontakt obavezno mora biti opisan e-mail adresom, dok su ostali podaci opcioni. Unutar klase *Contact* definisan je i *User* kome dati kontakt pripada.



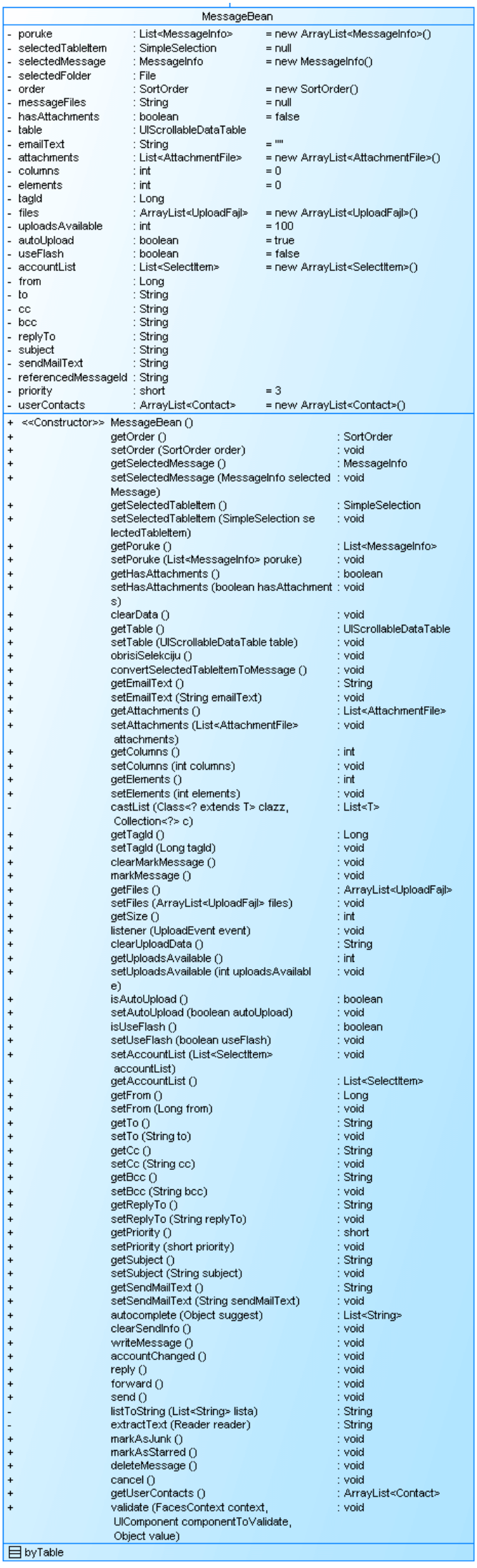
Dijagram 2.4. – Dijagram *managed bean* klasa

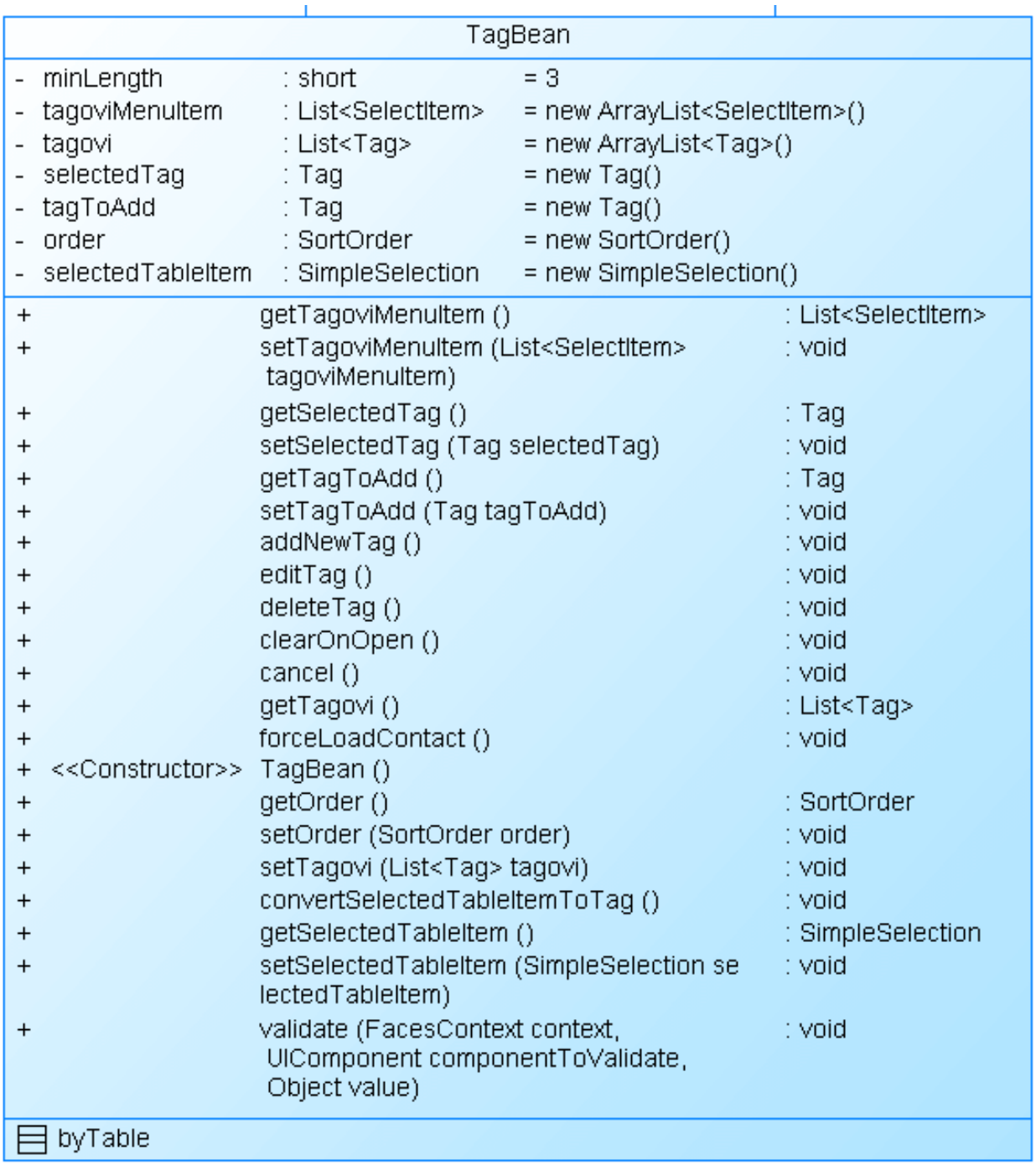

AccountBean
- minLength : short = 3
- accountId : Long
- selectedAccount : Account = new Account()
- nalozi : List<SelectItem> = new ArrayList<SelectItem>()
- naloziSize : int
- accounts : List<Account> = new ArrayList<Account>()
- selectedTableItem : SimpleSelection = new SimpleSelection()
- order : SortOrder = new SortOrder()
- accountToAdd : Account = new Account()
- password : String = ""
+ getNaloziSize () : int
+ setNaloziSize (int naloziSize) : void
+ getAccounts () : List<Account>
+ setAccounts (List<Account> accounts) : void
+ getNalozi () : List<SelectItem>
+ setNalozi (List<SelectItem> nalozi) : void
+ getAccountId () : Long
+ setAccountId (Long accountId) : void
+ getSelectedAccount () : Account
+ setSelectedAccount (Account selectedAcco : void
unt)
+ clearOnOpen () : void
+ convertSelectedTableItemToAccount () : void
+ getSelectedTableItem () : SimpleSelection
+ setSelectedTableItem (SimpleSelection se : void
lectedTableItem)
+ <<Constructor>> AccountBean ()
+ getOrder () : SortOrder
+ setOrder (SortOrder order) : void
+ forceLoadAccount () : void
+ getAccountToAdd () : Account
+ setAccountToAdd (Account accountToAdd) : void
+ addNewAccount () : void
+ addNewAccountNoAccounts () : void
+ getPassword () : String
+ setPassword (String password) : void
+ editAccount () : void
+ deleteAccount () : void
- deleteDirectory (File path) : void
+ cancel () : void
+ editChangePort () : void
+ addChangePort () : void
- castList (Class<? extends T> clazz, : List<T>
Collection<?> c)
+ validate (FacesContext context, : void
UIComponent componentToValidate,
Object value)
byTable

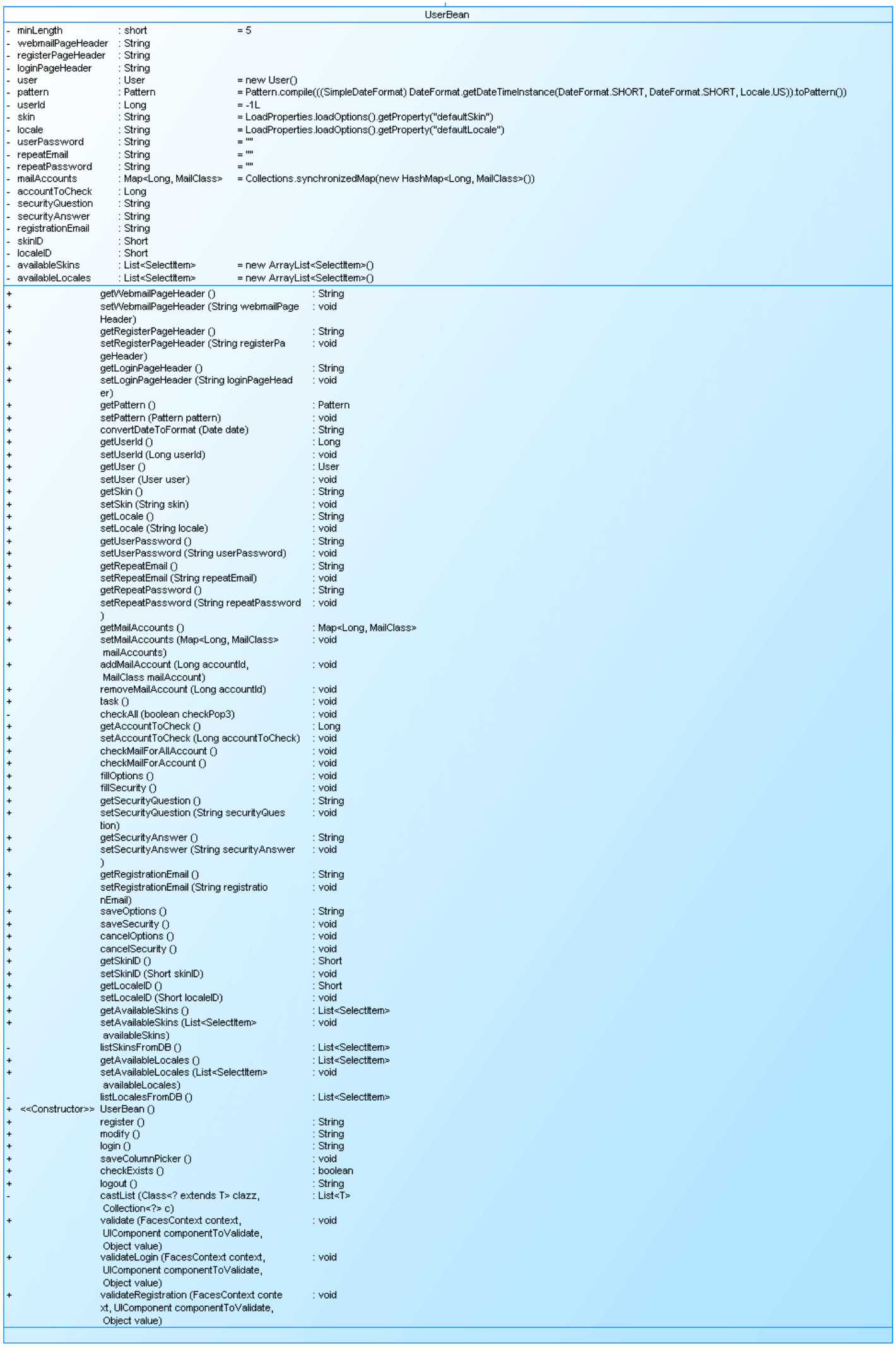

AddressBookBean
- minLength : short = 3
- addressBooks : List<SelectItem> = new ArrayList<SelectItem>()
- addressBookToAdd : AddressBook = new AddressBook()
- addressBookTreeNode : TreeNode<String> = new WebmailTreeNodeImpl<String>()
- selectedAddressBook : AddressBook = new AddressBook()
- kontakti : List<Contact> = new ArrayList<Contact>()
- selectedTableItem : SimpleSelection = new SimpleSelection()
- selectedContact : Contact = new Contact()
- selectedContactId : Long = -1L
- slika : Slika = new Slika()
- abContextNode : Long
- addressBookToEdit : AddressBook = new AddressBook()
- order : SortOrder = new SortOrder()
+ getSelectedContactId () : Long
+ setSelectedContactId (Long selectedContactId) : void
+ <<Constructor>> AddressBookBean ()
+ getOrder () : SortOrder
+ setOrder (SortOrder order) : void
+ getAddressBookToEdit () : AddressBook
+ setAddressBookToEdit (AddressBook addressBookToEdit) : void
+ getAbContextNode () : Long
+ setAbContextNode (Long abContextNode) : void
+ getSlika () : Slika
+ setSlika (Slika slika) : void
+ paint (OutputStream stream, Object object) : void
+ getSelectedContact () : Contact
+ setSelectedContact (Contact selectedContact) : void
+ getSelectedTableItem () : SimpleSelection
+ setSelectedTableItem (SimpleSelection selectedTableItem) : void
+ getKontakti () : List<Contact>
+ clearOnOpen () : void
+ setKontakti (List<Contact> kontakti) : void
+ convertSelectedTableItemToContact () : void
+ getSelectedAddressBook () : AddressBook
+ setSelectedAddressBook (AddressBook selectedAddressBook) : void
+ getAddressBookTreeNode () : TreeNode<String>
+ setAddressBookTreeNode (TreeNode<String> addressBookTreeNode) : void
+ getAddressBookToAdd () : AddressBook
+ setAddressBookToAdd (AddressBook addressBookToAdd) void
+ getAddressBooks () : List<SelectItem>
+ setAddressBooks (List<SelectItem> addressBooks) : void
+ deleteContact () : void
+ cancel () : void
+ addNewAddressBook () : void
+ editAddressBook () : void
+ deleteAddressBook () : void
+ processSelection (NodeSelectedEvent event) : void
+ dropListener (DropEvent dropEvent) : void
+ validate (FacesContext context, UIComponent componentToValidate, Object value) : void
byTable

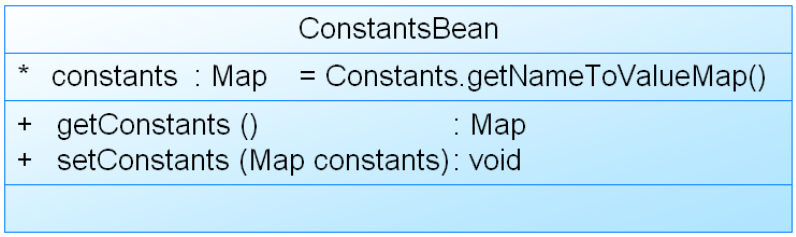

ContactBean
- protocolChar : String = ":"
- contactToAdd : Contact = new Contact()
- contactToAddPicture : File = new File(Constants.genericPicture)
- slika : Slika = new Slika()
- uploadsAvailable : int = 100
- autoUpload : boolean = true
- useFlash : boolean = false
- conatactToAddSelectedAddressBook : Long
- selectedContact : Contact
+ getSelectedContact () : Contact
+ forceLoadContact () : void
+ setSelectedContact (Contact selectedCont act) : void
+ getConatactToAddSelectedAddressBook () : Long
+ setConatactToAddSelectedAddressBook (Lon g conatactToAddSelectedAddressBook) : void
+ getContactToAdd () : Contact
+ setContactToAdd (Contact contactToAdd) : void
+ getBytesFromFile (File file) : byte[]
+ addNewContact () : void
+ addCon () : void
+ editContact () : void
+ cancel () : void
+ paint (OutputStream stream, Object object) : void
+ listener (UploadEvent event) : void
+ clearUploadData () : String
- setGenericPicture () : void
* copyPicture (File src, File dst) : void
+ getTimeStamp () : long
+ getSlika () : Slika
+ setSlika (Slika slika) : void
+ getUploadsAvailable () : int
+ setUploadsAvailable (int uploadsAvailabl e) : void
+ isAutoUpload () : boolean
+ setAutoUpload (boolean autoUpload) : void
+ isUseFlash () : boolean
+ setUseFlash (boolean useFlash) : void
+ validate (FacesContext context, UIComponent componentToValidate, Object value) : void

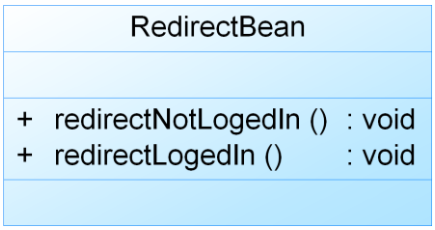
FolderBean		
- minLength	: short	= 3
- folderTreeNode	: TreeNode<String>	= new WebmailTreeNodeImpl<String>()
- selectedFolder	: String	
- accountContextNode	: Long	
- folderContextNode	: String	
- folderToAdd	: String	
- folderToEdit	: String	
+ getFolderToEdit ()		: String
+ setFolderToEdit (String folderToEdit)		: void
+ getFolderToAdd ()		: String
+ setFolderToAdd (String folderToAdd)		: void
+ getAccountContextNode ()		: Long
+ setAccountContextNode (Long accountContextNode)		: void
+ getFolderContextNode ()		: String
+ setFolderContextNode (String folderContextNode)		: void
+ getSelectedFolder ()		: String
+ setSelectedFolder (String selectedFolder)		: void
+ getFolderTreeNode ()		: TreeNode<String>
- recursivlyFindChildrenForTree (File currentFolder, boolean accountType, Long accountId, boolean root)		: TreeNode<String>
+ setFolderTreeNode (TreeNode<String> folderTreeNode)		: void
+ sendFolderToTrash (String folderToMoveTo)		: void
+ deleteFolder ()		: void
- deleteDirectory (File path)		: void
+ addNewFolder ()		: void
+ editFolder ()		: void
+ cancel ()		: void
+ processSelection (NodeSelectedEvent event)		: void
+ dropListener (DropEvent dropEvent)		: void
- castList (Class<? extends T> clazz, Collection<?> c)		: List<T>
+ validate (FacesContext context, UIComponent componentToValidate, Object value)		: void


MessageBean
- poruke : List<MessageInfo> = new ArrayList<MessageInfo>()
- selectedTableItem : SimpleSelection = null
- selectedMessage : MessageInfo = new MessageInfo()
- selectedFolder : File
- order : SortOrder = new SortOrder()
- messageFiles : String = null
- hasAttachments : boolean = false
- table : UIScrollableDataTable
- emailText : String = ""
- attachments : List<AttachmentFile> = new ArrayList<AttachmentFile>()
- columns : int = 0
- elements : int = 0
- tagId : Long
- files : ArrayList<UploadFajl> = new ArrayList<UploadFajl>()
- uploadsAvailable : int = 100
- autoUpload : boolean = true
- useFlash : boolean = false
- accountList : List<SelectItem> = new ArrayList<SelectItem>()
- from : Long
- to : String
- cc : String
- bcc : String
- replyTo : String
- subject : String
- sendMailText : String
- referencedMessageId : String
- priority : short = 3
- userContacts : ArrayList<Contact> = new ArrayList<Contact>()
+ <<Constructor>> MessageBean ()
+ getOrder () : SortOrder
+ setOrder (SortOrder order) : void
+ getSelectedMessage () : MessageInfo
+ setSelectedMessage (MessageInfo selectedMessage) : void
+ getSelectedTableItem () : SimpleSelection
+ setSelectedTableItem (SimpleSelection selectedTableItem) : void
+ getPoruke () : List<MessageInfo>
+ setPoruke (List<MessageInfo> poruke) : void
+ getHasAttachments () : boolean
+ setHasAttachments (boolean hasAttachments) : void
+ clearData () : void
+ getTable () : UIScrollableDataTable
+ setTable (UIScrollableDataTable table) : void
+ obrisiSelekciju () : void
+ convertSelectedTableItemToMessage () : void
+ getEmailText () : String
+ setEmailText (String emailText) : void
+ getAttachments () : List<AttachmentFile>
+ setAttachments (List<AttachmentFile> attachments) : void
+ getColumns () : int
+ setColumns (int columns) : void
+ getElements () : int
+ setElements (int elements) : void
- castList (Class<? extends T> clazz, Collection<?> c) : List<T>
+ getTagId () : Long
+ setTagId (Long tagId) : void
+ clearMarkMessage () : void
+ markMessage () : void
+ getFiles () : ArrayList<UploadFajl>
+ setFiles (ArrayList<UploadFajl> files) : void
+ getSize () : int
+ listener (UploadEvent event) : void
+ clearUploadData () : String
+ getUploadsAvailable () : int
+ setUploadsAvailable (int uploadsAvailable) : void
+ isAutoUpload () : boolean
+ setAutoUpload (boolean autoUpload) : void
+ isUseFlash () : boolean
+ setUseFlash (boolean useFlash) : void
+ setAccountList (List<SelectItem> accountList) : void
+ getAccountList () : List<SelectItem>
+ getFrom () : Long
+ setFrom (Long from) : void
+ getTo () : String
+ setTo (String to) : void
+ getCc () : String
+ setCc (String cc) : void
+ getBcc () : String
+ setBcc (String bcc) : void
+ getReplyTo () : String
+ setReplyTo (String replyTo) : void
+ getPriority () : short
+ setPriority (short priority) : void
+ getSubject () : String
+ setSubject (String subject) : void
+ getSendMailText () : String
+ setSendMailText (String sendMailText) : void
+ autocomplete (Object suggest) : List<String>
+ clearSendInfo () : void
+ writeMessage () : void
+ accountChanged () : void
+ reply () : void
+ forward () : void
+ send () : void
- listToString (List<String> lista) : String
- extractText (Reader reader) : String
+ markAsJunk () : void
+ markAsStarred () : void
+ deleteMessage () : void
+ cancel () : void
+ getUserContacts () : ArrayList<Contact>
+ validate (FacesContext context, UIComponent componentToValidate, Object value) : void
byTable


TagBean
- minLength : short = 3
- tagoviMenuItem : List<SelectItem> = new ArrayList<SelectItem>()
- tagovi : List<Tag> = new ArrayList<Tag>()
- selectedTag : Tag = new Tag()
- tagToAdd : Tag = new Tag()
- order : SortOrder = new SortOrder()
- selectedTableItem : SimpleSelection = new SimpleSelection()
+ getTagoviMenuItem () : List<SelectItem>
+ setTagoviMenuItem (List<SelectItem> tagoviMenuItem) : void
+ getSelectedTag () : Tag
+ setSelectedTag (Tag selectedTag) : void
+ getTagToAdd () : Tag
+ setTagToAdd (Tag tagToAdd) : void
+ addNewTag () : void
+ editTag () : void
+ deleteTag () : void
+ clearOnOpen () : void
+ cancel () : void
+ getTagovi () : List<Tag>
+ forceLoadContact () : void
+ <<Constructor>> TagBean ()
+ getOrder () : SortOrder
+ setOrder (SortOrder order) : void
+ setTagovi (List<Tag> tagovi) : void
+ convertSelectedTableItemToTag () : void
+ getSelectedTableItem () : SimpleSelection
+ setSelectedTableItem (SimpleSelection selectedTableItem) : void
+ validate (FacesContext context, UIComponent componentToValidate, Object value) : void
byTable


UserBean
- minLength : short = 5
- webmailPageHeader : String
- registerPageHeader : String
- loginPageHeader : String
- user : User = new User()
- pattern : Pattern = Pattern.compile(((SimpleDateFormat) DateFormat.getDateTimeInstance(DateFormat.SHORT, DateFormat.SHORT, Locale.US)).toPattern())
- userId : Long = -1L
- skin : String = LoadProperties.loadOptions().getProperty("defaultSkin")
- locale : String = LoadProperties.loadOptions().getProperty("defaultLocale")
- userPassword : String = ""
- repeatEmail : String = ""
- repeatPassword : String = ""
- mailAccounts : Map<Long, MailClass> = Collections.synchronizedMap(new HashMap<Long, MailClass>())
- accountToCheck : Long
- securityQuestion : String
- securityAnswer : String
- registrationEmail : String
- skinID : Short
- localeID : Short
- availableSkins : List<SelectItem> = new ArrayList<SelectItem>()
- availableLocales : List<SelectItem> = new ArrayList<SelectItem>()
+ getWebmailPageHeader () : String
+ setWebmailPageHeader (String webmailPageHeader) : void
+ getRegisterPageHeader () : String
+ setRegisterPageHeader (String registerPageHeader) : void
+ getLoginPageHeader () : String
+ setLoginPageHeader (String loginPageHeader) : void
+ getPattern () : Pattern
+ setPattern (Pattern pattern) : void
+ convertDateToFormat (Date date) : String
+ getUserId () : Long
+ setUserId (Long userId) : void
+ getUser () : User
+ setUser (User user) : void
+ getSkin () : String
+ setSkin (String skin) : void
+ getLocale () : String
+ setLocale (String locale) : void
+ getUserPassword () : String
+ setUserPassword (String userPassword) : void
+ getRepeatEmail () : String
+ setRepeatEmail (String repeatEmail) : void
+ getRepeatPassword () : String
+ setRepeatPassword (String repeatPassword) : void
+ getMailAccounts () : Map<Long, MailClass>
+ setMailAccounts (Map<Long, MailClass> mailAccounts) : void
+ addMailAccount (Long accountId, MailClass mailAccount) : void
+ removeMailAccount (Long accountId) : void
+ task () : void
- checkAll (boolean checkPop3) : void
+ getAccountToCheck () : Long
+ setAccountToCheck (Long accountToCheck) : void
+ checkMailForAllAccount () : void
+ checkMailForAccount () : void
+ fillOptions () : void
+ fillSecurity () : void
+ getSecurityQuestion () : String
+ setSecurityQuestion (String securityQuestion) : void
+ getSecurityAnswer () : String
+ setSecurityAnswer (String securityAnswer) : void
+ getRegistrationEmail () : String
+ setRegistrationEmail (String registrationEmail) : void
+ saveOptions () : String
+ saveSecurity () : void
+ cancelOptions () : void
+ cancelSecurity () : void
+ getSkinID () : Short
+ setSkinID (Short skinID) : void
+ getLocaleID () : Short
+ setLocaleID (Short localeID) : void
+ getAvailableSkins () : List<SelectItem>
+ setAvailableSkins (List<SelectItem> availableSkins) : void
- listSkinsFromDB () : List<SelectItem>
+ getAvailableLocales () : List<SelectItem>
+ setAvailableLocales (List<SelectItem> availableLocales) : void
- listLocalesFromDB () : List<SelectItem>
+ <<Constructor>> UserBean ()
+ register () : String
+ modify () : String
+ login () : String
+ saveColumnPicker () : void
+ checkExists () : boolean
+ logout () : String
- castList (Class<? extends T> clazz, Collection<?> c) : List<T>
+ validate (FacesContext context, UIComponent componentToValidate, Object value) : void
+ validateLogin (FacesContext context, UIComponent componentToValidate, Object value) : void
+ validateRegistration (FacesContext context, UIComponent componentToValidate, Object value) : void


ConstantsBean
* constants : Map = Constants.getNameToValueMap()
+ getConstants () : Map
+ setConstants (Map constants) : void


RedirectBean
+ redirectNotLogedIn () : void
+ redirectLogedIn () : void

Na dijagramu 2.4. su prikazane klase koje predstavljaju *managed bean*-ove. Atributi ovih klasa najčešće služe za povezivanje sa komponentama korisničkog interfejsa na JSF stranama ili sadrže dinamičku vrednost koja će biti prikazana na stranici, bila ona predstavljena prostim tipom podataka, *entity* objektom ili kolekcijom *entity* objekata. S druge strane, metode ovih klasa najčešće predstavljaju obrađivače događaja koje na JSF stranici inicira korisnik, pri čemu se unutar ovih klasa koriste metode za komunikaciju sa bazom podataka, ukoliko se za tim ukaže potreba. Na dijagramu nisu predstavljeni atributi i metode klasa, radi preglednosti, te su predstavljeni na odvojenim dijagramima.

Klasa *ConstantsBean* predstavlja *application scope managed bean* koji sadrži sve konstante aplikacije. Zbog problema vezanog za JSF i Facelet način upravljanja sa konstantama ova klasa je implementirana kao *managed bean* sa metodama za preuzimanje mape konstanti.

Klasa *RedirectBean* predstavlja *request scope managed bean* koja slično kao i *ConstantsBean* je implementirana sa ciljem rešavanja problema vezanog za Facelet implementaciju *JSTL (Java Standard Tag Library)*. Kao što i samo ime klase kaže implementirane su dve metode koje služe preusmeravanju korisnika u slučaju nedozvoljenog pristupa određenim stranama.

Klasa *AccountBean* kao i sve naredne klase ovog sloja aplikacije se nalazi pod *session* opsegom i implementira metode vezane za ažuriranje, prikaz i brisanje emajl naloga i metodu *validate()* za proveru ispravnosti podataka unesenih na dijalogu za dodavanje ili izmenu nekog od naloga od strane korisnika. Takođe, *AccountBean* sadrži informacije o selektovanom nalogu u tabeli naloga i *sortOrder* vezan za korisnički odabir sortiranja tabele.

Klasa *AddressBookBean* implementira metode vezane za ažuriranje, prikaz i brisanje korisničkih adresara i kontakata unutar njih, prikaz selektovanog kontakta i njegovih detalja, metode za upravljanje premeštanjem kontakata unutar postojećih adresara i metodu *validate()* za proveru ispravnosti podataka unesenih na dijalogu za dodavanje ili izmenu nekog od adresara od strane korisnika. Takođe, *AddressBookBean* sadrži informacije o selektovanom adresaru u stablu adresara i selektovanom kontaktu u tabeli kontakata i *sortOrder* vezan za korisnički odabir sortiranja tabele.

Klasa *ContactBean* implementira metode vezane za ažuriranje i brisanje kontakata, metode vezane za dodavanje slike kontaktu i metodu *validate()* za proveru ispravnosti podataka unesenih na dijalogu za dodavanje ili izmenu nekog od kontakta od strane korisnika.

Klasa *FolderBean* implementira metode vezane za ažuriranje, prikaz i brisanje foldera, metode za prikaz pohranjenih poruka u selektovanom folderu, metode za upravljanje premeštanjem poruka i foldera unutar hijerarhije foldera i metodu *validate()* za proveru ispravnosti podataka unesenih na dijalogu za dodavanje ili izmenu nekog od foldera od strane korisnika. Takođe, *FolderBean* sadrži informacije o selektovanom folderu u stablu foldera.

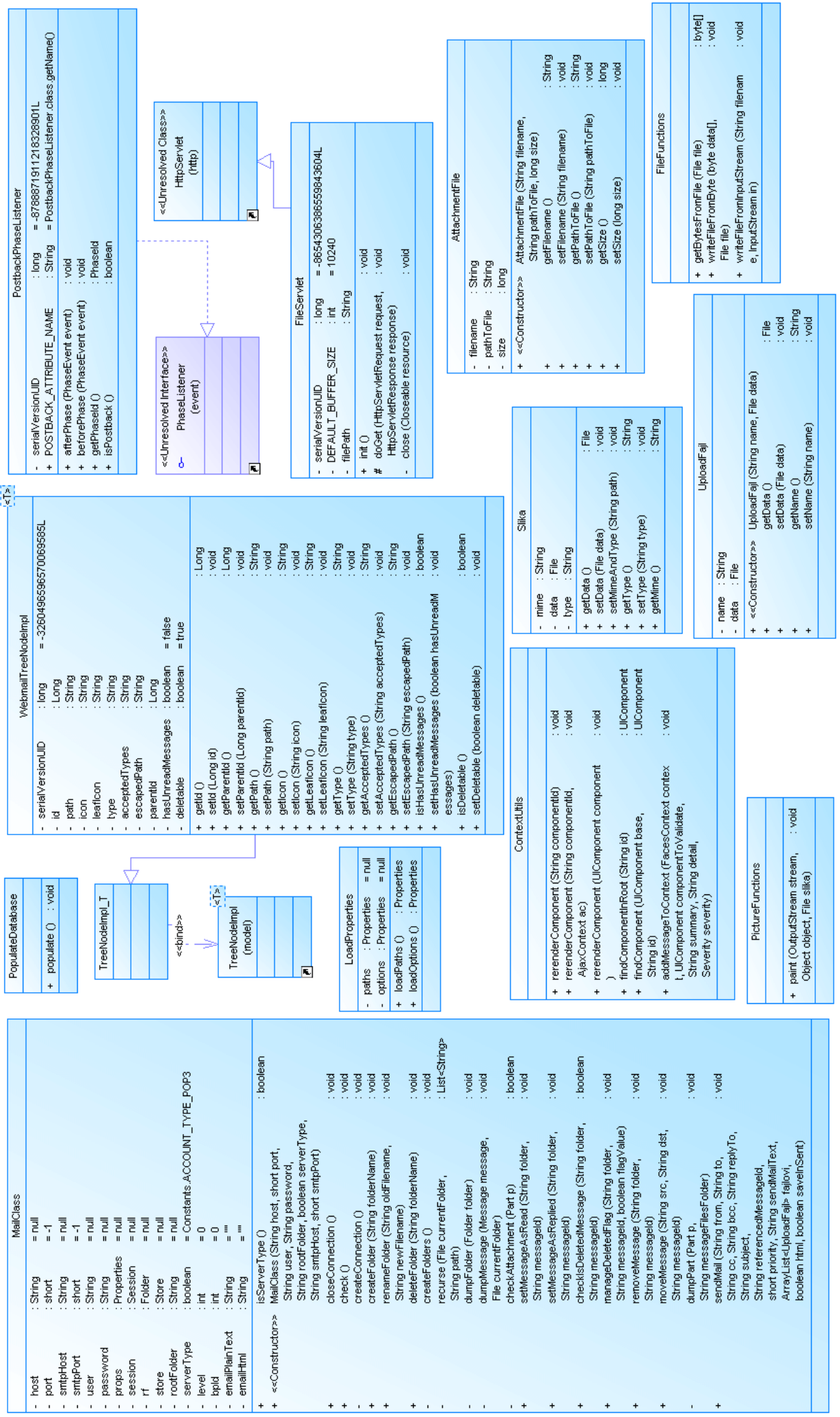
Klasa *MessageBean* implementira metode vezane za upravljanje porukama, slanje poruka pomoću klase *MailClass*, atribute koji su povezani sa komponentama na stranici za kreiranje nove poruke i metodu *validate()* za proveru ispravnosti podataka unesenih na dijalogu za slanje poruke od strane korisnika. Takođe, *MessageBean* sadrži informacije o selektovanoj poruci u tabeli poruka i *sortOrder* vezan za korisnički odabir sortiranja tabele.

Klasa *TagBean* implementira metode vezane ažuriranje, prikaz i brisanje tagova, obeležavanje poruka tagovima i metodu *validate()* za proveru ispravnosti podataka unesenih na dijalogu za dodavanje ili izmenu vrednosti taga od strane korisnika. Takođe, *TagBean* sadrži informacije o selektovanom tagu u tabeli tagova i *sortOrder* vezan za korisnički odabir sortiranja tabele.

Klasa *UserBean* sadrži atribute i metode ove klase koje služe za čuvanje stanja određenih komponenti tokom čitave sesije jednog korisnika, kao i pomoćne metode vezane za preuzimanje pošte sa korisničkih emajl naloga u određenom intervalu ili korisničkim pozivom date funkcije, metode konverzije datuma u lokokalizovanu instancu, metode čuvanja promena

koje vezanih za izgled tabela unutar aplikacije kao i metode vezane za izmenu korisničkih podataka i registraciju ili proveru validnosti unesenih podataka na strani za prijavu. *UserBean* čuva referencu na *entity* objekat trenutno ulogovanog korisnika.

Klasa *TreeDemoStateAdvisor* je implementirana zbog nedostatka Richfaces implementacije *tree* komponente koja ne omogućava jednostavan način označavanja koji koreni stabla trebaju da budu otvoreni pri samom kreiranju tog stabla.



Dijagram 2.5. – Dijagram *pomoćnih* klasa

Za kraj, na dijagramu 2.5. su prikazane klase čije funkcionalnosti čine zasebne logičke celine.

MailClass implementira metode koje pomoću JavaMail API-ja komuniciraju sa emajl serverom radi preuzimanja, premeštanja, slanja ili obeležavanja pošte, kreiranja, premeštanja i brisanja foldera i metoda vezanih za proveru stanja poruke na serveru. Takođe to je klasa koja vrši prebacivanje informacija iz *javax.mail.Message* objekata na *entity* objekte koji se perzistiraju u bazi podataka i koja smešta odgovarajući sadržaj poruke na fajl sistem.

PopulateDatabase je klasa čiji jedini cilj predstavlja inicijalno punjenje baze podataka sa osnovnim podacima.

WebmailTreeNodeImpl je klasa koja proširuje osnovu klasu *TreeNodeImpl_T* sa metodama i atributima čiji je cilj da omogući rad sa stablima unutar aplikacije i pojednostavi način pronalaženja i upravljanja samim stablom i granama stabla u slučaju drag-and-drop akcije određenih komponenti na datom stablu.

LoadProperties je klasa sa metodama vezanim za učitavanje *Properties* fajlova koji sadrže osnovne podatke vezane za lokalizaciju lokalizacija ili podešavanje sistema.

PictureFunctions je klasa sa metodama vezanim za iscrtavanje slike vezane za kontakt na dijalozima na kojima je to potrebno.

Slika je klasa sa atributima i metodama kojoj je cilj da pojednostavi određivanje *MimeType* slike, bez potrebe korišćenja kompleksnih metoda, koja će biti prikazana unutar programa.

UploadFile je klasa sa koja služi za prihvatanje fajlova koje korisnik prosledi aplikaciji kao *attachment*-e određene poruke.

FileFunctions je klasa sa koja sadrži metode za smeštanje i preuzimanje fajlova na fajl sistemu.

AttachmentFile je klasa sa koja sadrži atribute i metode koji omogućavaju preuzimanje fajlova od aplikacije.

FileServlet je klasa koja implementira *Servlet* metode sa ciljem da se omogući preuzimanje fajlova od aplikacije, upravo ova klasa koristi objekte tipa *AttachmentFile* radi lakšeg preuzimanja fajlova od aplikacije.

PostbackPhaseListener je klasa sa koja sadrži atribute i metode koji omogućavaju lakši rad sa stablima unutar aplikacije.

ContextUtils je klasa sa koja sadrži metode koji omogućavaju pronalaženje i ponovno iscrtavanje strane ili komponente na strani u slučaju da je to potrebno ili zadato od strane neke metode.

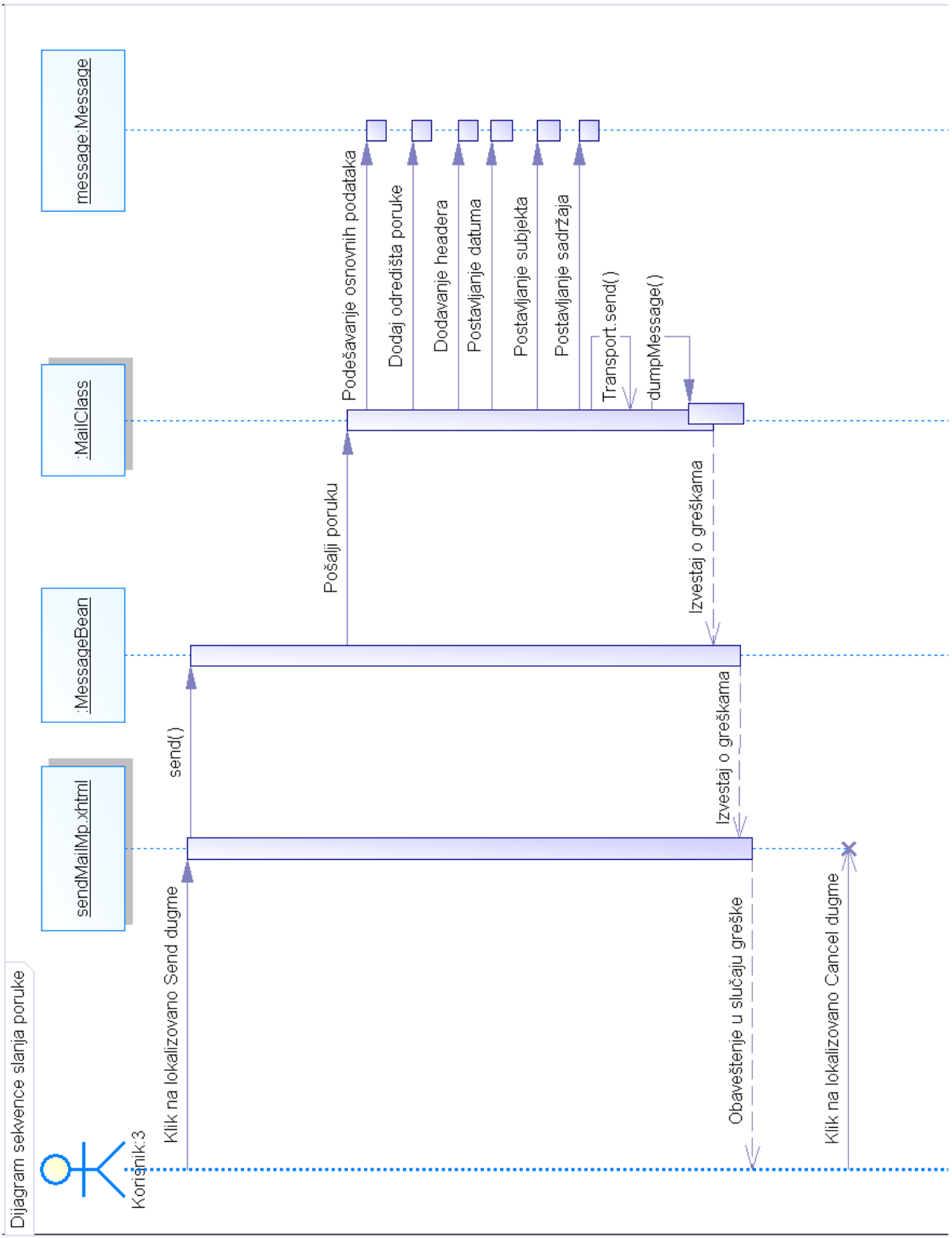
U okviru *PersistenceUtil* klase se kreira statička instanca *EntityManagerFactory* klase, na osnovu koje metoda ove klase kreira *EntityManager* objekte koji se preuzimaju pomoću *getEntiyManager* metode.

2.3. Dijagrami sekvence

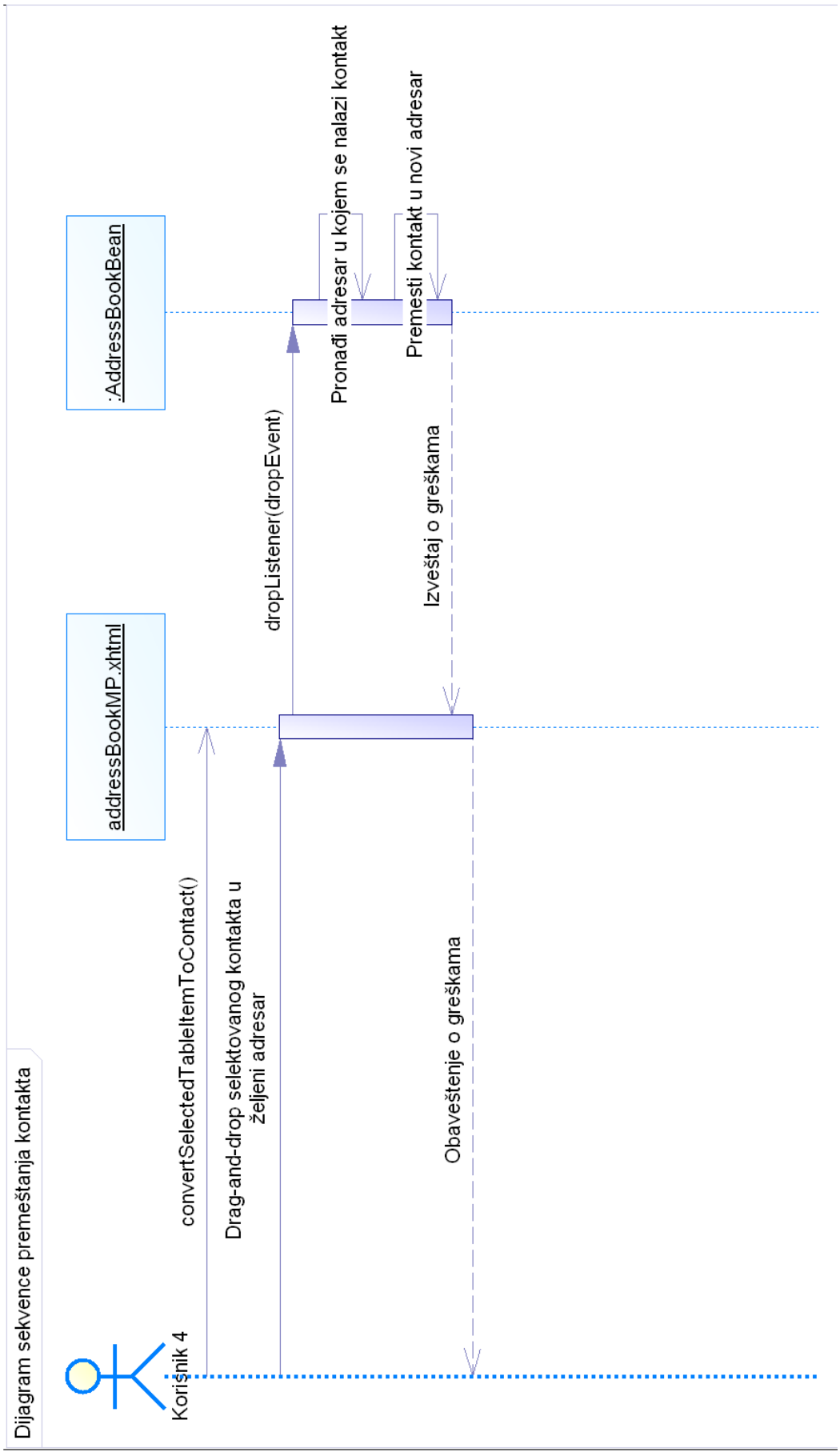
U nastavku su prezentovani dijagrami sekvence, koji imaju za cilj da pobliže objasne tok obrade zahteva u okviru aplikacije.

Na dijagramu 2.6. prezentovan je dijagram sekvence slanja poruke. Po kliku na dugme "*Send"* na JSF stranici, poziva se *send* metoda u okviru *MessageBean*-a koja preuzima vrednosti komponenti na samoj JSF strani i posle uspešne validacije poziva metodu *sendMessage MailClass* klase. Unutar *MailClass* klase se kreira objekat *javax.mail.Message* koji predstavlja novu poruku, i njegovi atributi dobijaju one vrednosti koje je korisnik uneo na stranici i koje su prosleđene iz *MessageBean*. Nakon toga pomoću JavaMail API-ja se šalje sama poruka i u zavisnosti podešavanja sama poruka koja je poslata se smešta u *Sent* folder i za kraj, metoda *dumpMessage* perzistira poruku u bazi podataka. U slučaju greške u bilo kojem koraku obaveštava se korisnik o tipu greške.

Dijagram 2.7. predstavlja dijagram sekvence premeštanja kontakta iz jednog adresara u drugi. Pošto se selektuje kontakt i pošto sistem izvrši konverziju selektovanog reda tabele i po završetku drag-and-drop akcije na željeni adresar, aktivira se *listener* klase *AddressBookBean* koji na osnovu *drag-and-drop* događaja određuje adresar u koji se premešta kontakt i na osnovu selektovanog kontakta određuje njegov trenutni adresar. Potom metoda vrši premeštanje i perzistira promene u bazi podataka. Konačno metoda poziva *RerenderComponent* metodu klase *ContextUtils* i zatražuje ponovno iscrtavanje tabele kontakata. U slučaju greške u bilo kojem koraku obaveštava se korisnik o tipu greške.



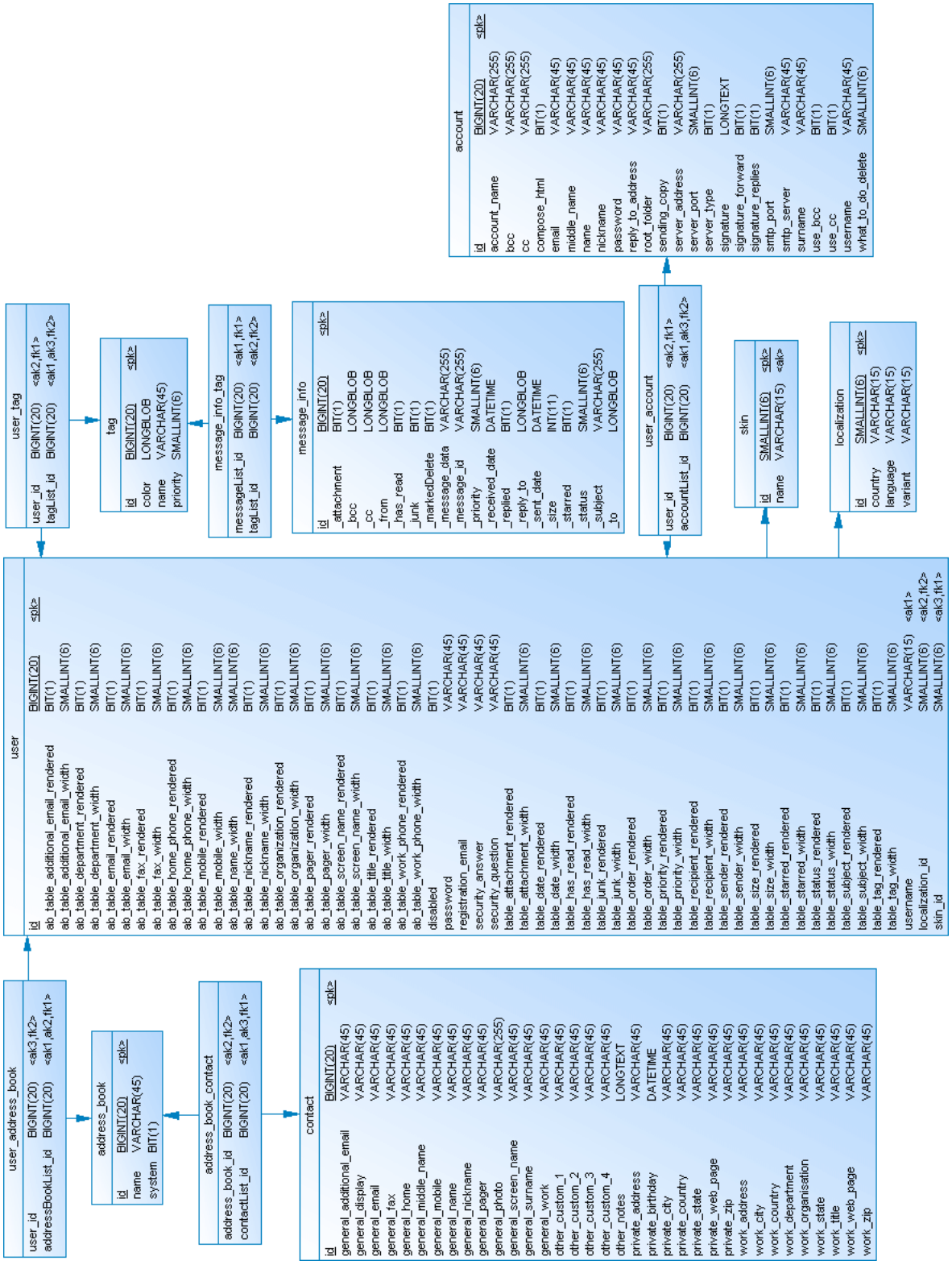
Dijagram 2.6. – Dijagram sekvence slanja poruke



Dijagram 2.7. – Dijagram sekvence premeštanja kontakta

2.3. Model baze podataka

Na dijagramu 2.8. prezentovan je dijagram fizičkog modela baze podataka koji neće biti detaljno opisan pošto predstavlja fizičku implementaciju *entity* objekata aplikacije koji se kreiraju pomoću *Java Persistence Api*.



Dijagram 2.8. – Model baze podataka

3. IMPLEMENTACIJA

Webmail aplikacija predstavlja *web* aplikaciju baziranu na JSF tehnologiji, i njena je implementacija izvršena na programskom jeziku Java, verzija 1.6.0_21, u okviru Eclipse Helios SR1 okruženja. Za grafički izgled aplikacije je iskorišćen Richfaces verzija 3.3.3. Kao servletski kontejner upotrebljen je Apache Tomcat 6.0.29. Za perzistenciju podataka upotrebljen je MySQL server, verzija 5.1. i Hibernate API verzija 3.5.5. Kao *mail server* upotrebljen je hMailServer 5.3.3-B1879.

3.1. Implementacija korisničkog interfejsa

Prezentacioni sloj aplikacije je realizovan pomoću JSF stranica, u okviru kojih su kao komponente korisničkog interfejsa uglavnom korišćene komponente RichFaces i Ajax biblioteke, kao i JSF UI komponente. Izuzimajući index i redirect stranice ukupno su realizovane tri stranice – stranica za prijavu, stranica za registraciju i osnovna stranica aplikacije (iako postoji više *.xhtml fajlova oni ne predstavljaju nove stranice već se u toku rada programa *insertuju* u glavnu stranicu pomoću facelet metode `<ui:include src="" />`.

Osnovnu stranicu korisničkog interfejsa reprezentuje stranica *webmail.xhtml*, i na ovu stranicu je korisnik upućen nakon uspešne prijave na sistem. Na ovoj stranici korisniku je ponuđen meni sa kojeg može da upravlja kontaktima, tagovima, adresarima, korisničkim podacima, zahteva prijem novih poruka sa *mail* server-a, kreira novu poruku, kao i da se odjavi sa sistema. Realizacija glavnog menija prilazna sledećim kodom.

```
<a4j:form id="menuBar" ajaxSubmit="true">
        <rich:toolBar id="mailToolbar" itemSeparator="line">
                <rich:dropSupport acceptedTypes="none">
                </rich:dropSupport>
                <rich:toolBarGroup>
                        <rich:dropDownMenu id="naloziDropDownMenu" event="onmousedown"
styleClass="popravka">
                                <f:facet name="label">
                                        <h:panelGroup>
                                                <h:graphicImage
value="/images/getMail.png" style="width:32px;height:32px;" />
                                                <h:outputText
value="#{locale.menu_toolbar_get_mail}" />
                                        </h:panelGroup>
                                </f:facet>
                                <rich:menuItem
value="#{locale.menu_toolbar_get_all_new_messages}"
action="#{userBean.checkMailForAllAccount}" ajaxSingle="true"
                                        submitMode="ajax">
                                </rich:menuItem>
                                <rich:menuSeparator />
                                <c:forEach var="item" items="#{accountBean.nalozi}">
                                        <rich:menuItem styleClass="popravka"
value="#{item.label}" action="#{userBean.checkMailForAccount}" ajaxSingle="true"
submitMode="ajax">
                                                <f:facet name="icon">
                                                        <h:panelGroup>
                                                                <h:graphicImage
value="/images/mail.png" style="width:16px;height:16px;" />
                                                        </h:panelGroup>
                                                </f:facet>
                                                <a4j:actionparam name="accountToCheck"
assignTo="#{userBean.accountToCheck}" value="#{item.value}" />
                                        </rich:menuItem>
                                </c:forEach>
                        </rich:dropDownMenu>
```

```
                    <a4j:commandLink action="#{messageBean.writeMessage}"
oncomplete="#{rich:component('sendMailMP')}.show()" styleClass="popravka">
                        <h:graphicImage value="/images/write-message.png"
style="display:inline;width:32px;height:32px;border:0;" />
                        <h:outputText value="#{locale.menu_toolbar_write}"
styleClass="rich-toolbar-item" />
                    </a4j:commandLink>

                    <a4j:commandLink action="#{addressBookBean.clearOnOpen}"
oncomplete="#{rich:component('addressBookMP')}.show()" styleClass="popravka">
                        <h:graphicImage value="/images/addrbook.png"
style="display:inline;width:32px;height:32px;border:0;" />
                        <h:outputText
value="#{locale.menu_toolbar_address_book}" styleClass="rich-toolbar-item" />
                    </a4j:commandLink>

            </rich:toolBarGroup>
                    …
        </rich:toolBar>
                                    </a4j:form>
```

Listing 3.1. – Glavni meni

Iako na listingu 3.1. nisu prikazani svi elementi *toolbar*-a ostavljena su dva primera realizacije menija. To su pomoću *dropDownMenu* i enkapsulirani unutar *commandLink* komponente. Može zaključiti da se komandni linkovi glavnog menija nalaze u toolbar komponenti RichFaces biblioteke, i na taj način zaokruženi kao odvojena celina interfejsa osnovne stranice.

Pored glavnog menija, na osnovnoj stranici je korisniku ponuđeno i da selekcijom odgovarajućeg foldera prikaže poruke koje su u njemu sadržane. Za prezentaciju foldera je iskorišćena *tree* komponenta RichFaces biblioteke, koja nudi hijerarhijsku prezentaciju podataka kroz strukturu stabla.

```
<rich:tree id="folderTree" ajaxKeys="#{null}"
nodeSelectListener="#{folderBean.processSelection}" reRender="emailTable"
        stateAdvisor="#{treeDemoStateAdvisor}" ajaxSubmitSelection="true"
switchType="client" dragIndicator="indicatorUsersPanel"
        value="#{folderBean.folderTreeNode}" var="item" treeNodeVar="folderTreeNode"
dropListener="#{folderBean.dropListener}"
        preserveModel="none" showConnectingLines="false">
        <rich:treeNode id="folderTreeNodeId"
acceptedTypes="#{folderTreeNode.acceptedTypes}" dragType="#{folderTreeNode.type}"
                icon="#{folderTreeNode.icon}" iconLeaf="#{folderTreeNode.leafIcon}">
                <rich:dndParam name="label" type="drag" value="#{item}" />
                <h:outputText value="#{item}"
                        style="color:#{folderTreeNode.hasUnreadMessages == true ? 'Blue'
: org.richfaces.SKIN.tableBackgroundColor}; #{folderTreeNode.hasUnreadMessages == true
? 'font-weight:bold;' : 'font-weight:normal;'} #{folderTreeNode.type == 'account' ?
'font-weight:bold;' : ''}" />
                <rich:contextMenu event="oncontextmenu" attachTo="folderTreeNodeId"
submitMode="ajax">
                        <rich:menuItem ajaxSingle="true" submitMode="ajax"
value="#{locale.tree_folder_new_folder}"
                                oncomplete="#{rich:component('addFolderMP')}.show()"
rendered="#{folderTreeNode.type == 'account'}">
                                <a4j:actionparam name="accountContextNode"
assignTo="#{folderBean.accountContextNode}" value="#{folderTreeNode.id}" />
                        </rich:menuItem>
                        …
                </rich:contextMenu>
        </rich:treeNode>
                                    </rich:tree>
```

Listing 3.2. – Stablo foldera

Na listingu 3.2 se može videti atribut *nodeSelectListener* koji definiše metodu *FolderBean managed bean*-a koja predstavlja osluškivač događaja za datu *tree* komponentu. Uočljivo je i da se u okviru stabla razlikuje više tipova čvorova (*treeNode*). U okviru *tree* komponente naveden je atribut *nodeFace*, pomoću kojeg je naznačeno da se čvorovi međusobno razlikuju (da li je sistemski folder ili ne, da li se može menjati ili ne…). na osnovu vrednosti definisane *type* atributom promenljive *folderTreeNode*, koja predstavlja pojedinačne instance *Folder*-a, Nakon toga, potrebno je samo za moguće vrednosti atributa *type* instance *Folder*-a, definisati po jednu *treeNode* komponentu i podesiti njen atribut *type* na datu vrednost. Na ovaj način je moguće programski podesiti tip foldera, i stablo će na osnovu date vrednosti tipa foldera, izvršiti prikaz konkretne instance odgovarajućim čvorom.

Pored toga, unutar čvorova definisana je *contextMenu* komponenta, unutar koje su definisane stavke menija, upotrebom *menuItem* komponenti. Čvorovi koji nisu tipa "systemFolder" i "account" predstavljaju foldere koje je korisnik kreirao, i na ovaj način je korisniku desnim klikom miša omogućen pristup akcijama kreiranja podfoldera za selektovani folder (ista akcija je omogućena i u slučaju da čvorovi jesu tog tipa), i brisanja selektovanog foldera. U slučaju izbora neke od spomenutih akcija, korisniku se prikazuje odgovarajući modalni panel, čime se zaključava sadržaj osnovne stranice, do izvršavanja akcije, ili zatvaranja modalnog panela. Modalni panel je iskorišćen i za prikazivanje forme za unos podataka o novom folderu.

```
<rich:modalPanel id="addFolderMP" autosized="true" resizeable="false">
                        <rich:dropSupport acceptedTypes="none">
                        </rich:dropSupport>
                        <f:facet name="header">
                                <h:outputText value="#{locale.add_folder_header}" />
                        </f:facet>
                        <a4j:form ajaxSubmit="true">
                                <h:panelGrid columns="3">
                                        <h:outputLabel for="folderToAdd_name"
value="#{locale.add_folder_name}" />
                                        <h:inputText id="folderToAdd_name"
value="#{folderBean.folderToAdd}" required="true" maxlength="16" size="16"

       requiredMessage="#{locale.error_cant_be_empty}"
validator="#{folderBean.validate}" />
                                        <mytags:validationMessage
field="folderToAdd_name" />
                                </h:panelGrid>
                                <h:panelGrid columns="2" width="100%">
                                        <a4j:commandButton
action="#{folderBean.addNewFolder}" value="#{locale.create}" style="width: 100%" />
                                        <a4j:commandButton action="#{folderBean.cancel}"
value="#{locale.cancel}" style="width: 100%" immediate="true" />
                                </h:panelGrid>
                        </a4j:form>
                                        </rich:modalPanel>
```

Listing 3.3. – Modalni panel za pravljenje novog foldera

Kao što se može primetiti na listingu 3.3. korisniku se prikazuje jednostavni modalni panel sa jednim poljem za unos podataka i labelom vezanom za to polje. U slučaju greške pri unosu pojaviće se i slikovni znak koji će korisnika obavestiti o grešci pomoću tooltipa. Ukoliko je korisnik uneo validne podatke izvršiće se *create* metoda *FolderBean* klase. Potom u slučaju uspešnog dodavanja biti ponovo iscrtana komponenta *folderTree*, kako bi korisniku promena bila vidljiva.

Pored navedenih komponenti koje čine interfejs osnovne stranice, važno je spomenuti i panel koji prikazuje tabelu poruka koje su sadržane u selektovanom folderu. Jedan red tabele označava zaglavlje jedne poruke, i za njega je definisan osluškivač događaja, koji korisniku prikazuje panel unutar kojeg je prikazan potpun sadržaj poruke.

```
<rich:scrollableDataTable binding="#{messageBean.table}" id="emailTable" var="message"
value="#{messageBean.poruke}" width="100%"
        height="300px" selectionMode="single"
selection="#{messageBean.selectedTableItem}" sortMode="single"
sortOrder="#{messageBean.order}">
        <a4j:support reRender="messagePane,subject,tagoviDropDown"
action="#{messageBean.convertSelectedTableItemToMessage}"
                event="onselectionchange" />
        <rich:column rendered="#{userBean.user.tableAttachmentRendered}"
width="#{userBean.user.tableAttachmentWidth}px" id="attachment">
                <f:facet name="header">
                        <a4j:outputPanel layout="block">
                                <h:graphicImage value="/images/attachments.png"
style="width:16px;height:16px;" />
                        </a4j:outputPanel>
                </f:facet>
                <a4j:outputPanel style="cursor:pointer;width:100%;height:20px"
layout="block">
                        <rich:dragSupport dragIndicator="indicatorEmailTable"
dragType="msg#{accountBean.accountId}" dragValue="#{message}">
                                <rich:dndParam name="label" value="#{message.subject}"
/>
                        </rich:dragSupport>
                        <h:graphicImage rendered="#{message.attachment}"
value="/images/attachments.png" style="width:16px;height:16px;" />
                </a4j:outputPanel>
        </rich:column>
…
                                </rich:scrollableDataTable>
```

Listing 3.4. – Tabela poruka

Kao što se može uočiti na listingu 3.4., atributom *value* komponente *scrollableDataTable* referencirana je lista poruka u *messageBean* klasi, koja sadrži poruke koje treba da se izlistaju u tabeli. Iako je unutar tabele definisano više kolona, u datom listingu je prikazana samo jedna. Koloni *attachment* dodata je Ajax funkcionalnosti upotrebom *a4j:outputPanel* komponente unutar koje se nalazi h:graphicImage komponenta.

Sama tabela ima dodatu Ajax funkcionalnost pomoću *a4:support* komponente koja prilikom promene selekcije u tabeli (*onselecionchange*) poziva metodu *convertSelectedTableItemToMessage* klase *messgaeBean*.

Klikom na red tabele, korisniku se ispod tabele poruka, prikazuje panel koji uključuje osnovne informacije o poruci (*headers*), kompletan sadržaj selektovane poruke, kao i linkove koji omogućuju preuzimanje eventualnih *attachment*-a. Pored toga, na panelu su vidljiva i dodatna dugmad – za odgovor na selektovanu poruku (*Reply*), za prosleđivanje selektovane poruke (*Forward*), za obeležavanje poruke kao značajne poruke (*Mark as starred*), za obeležavanje poruke kao otpada (*Mark as junk*) i za brisanje poruke (*Delete*). Klik na bilo koje od prva dva dugmeta otvara novu stranicu u zasebnom modalnom panelu, *sendMailMP.xhtm*, a isti modalni panel se otvara i klikom na *Write* link u glavnom meniju i na kojoj korisnik kreira novu poruku. U sledećem delu je prikazan deo koda modalnog panela za slanje pošte.

```
<rich:modalPanel id="sendMailMP" autosized="true" resizeable="false"
domElementAttachment="parent">
        <rich:dropSupport acceptedTypes="none">
        </rich:dropSupport>
        <f:facet name="header">
                <h:outputText value="#{locale.send_mail_header}" />
        </f:facet>
        <a4j:form ajaxSubmit="true">
                                                <rich:layout>
                                        …
```

```
<rich:toolBar itemSeparator="line">
        <rich:toolBarGroup>
                <a4j:commandLink type="submit" action="#{messageBean.send}"
styleClass="popravka">
                        <h:graphicImage value="/images/addAccount.png"
style="display:inline;width:32px;height:32px;border:0;" />
                        <h:outputText value="#{locale.send_mail_Send}" styleClass="rich-
toolbar-item" />
                </a4j:commandLink>
                                        </rich:toolBarGroup>
                                                …
<rich:panel>
        <h:outputLabel for="selektujNalogSaKojegSeSalje"
value="#{locale.send_mail_From}" />
        <h:selectOneMenu id="selektujNalogSaKojegSeSalje" value="#{messageBean.from}"
style="width:60%;">
        <a4j:support action="#{messageBean.accountChanged}" event="onchange" />
        <f:selectItems value="#{messageBean.accountList}" />
                                    </h:selectOneMenu>
                                                …
<rich:panel>
        <rich:editor width="700" height="300" viewMode="visual" theme="advanced"
value="#{messageBean.sendMailText}">
                <f:param name="plugins" value="fullpage,advlink,advimage" />
                <f:param name="theme_advanced_buttons3_add" value="fullpage" />
                <f:param name="relative_urls" value="false" />
                <f:param name="remove_script_host" value="false" />
        </rich:editor>
                                        </rich:panel>
                                                …
<rich:dataGrid id="attachmentList" columns="1" elements="3"
value="#{messageBean.files}" var="file" rowKeyVar="row">
        <rich:panel>
                <h:panelGrid columns="1">
                        <h:outputText value="#{file.name}" />
                </h:panelGrid>
        </rich:panel>
        <f:facet name="footer">
                <rich:datascroller></rich:datascroller>
        </f:facet>
                                    </rich:dataGrid>
                                                …
            <ui:include src="/pages/modalPanels/addAttachmentsMP.xhtml" />
                                                …
<a4j:form ajaxSubmit="true">
                                <h:panelGrid>
                                        <rich:fileUpload
fileUploadListener="#{messageBean.listener}"
maxFilesQuantity="#{messageBean.uploadsAvailable}" id="attachmentUpload"

        immediateUpload="#{messageBean.autoUpload}"
allowFlash="#{messageBean.useFlash}">
                                                <a4j:support event="onuploadcomplete"
reRender="attachmentList" />
                                        </rich:fileUpload>
                                </h:panelGrid>
                                <h:panelGrid columns="2">
                                        <h:selectBooleanCheckbox id="autoUpload"
value="#{messageBean.autoUpload}">
                                                <a4j:support event="onchange"
reRender="attachmentUpload" />
                                        </h:selectBooleanCheckbox>
                                        <h:outputLabel for="autoUpload"
value="#{locale.add_attachment_Auto_upload}" />
                                        <h:selectBooleanCheckbox id="useFlash"
value="#{messageBean.useFlash}">
                                                <a4j:support event="onchange"
reRender="attachmentUpload" />
                                        </h:selectBooleanCheckbox>
                                        <h:outputLabel for="useFlash"
value="#{locale.add_attachment_Use_flash}" />
```

```
</h:panelGrid>
                                        </a4j:form>
```

Listing 3.5. – Unos nove poruke

Na listingu 3.5. koji prikazuje modalni panel za unos nove poruke vidljive su sledeće celine: toolbar sa osnovnim komandama *Send, Add attachment, Cancel*, centralni deo za unos podataka o primaocima poruke i *dropDownMenu* za izbor naloga sa kojeg se šalje poruka, integrisani editor za pisanje teksta poruke, *selectOneMenu* za odabir važnosti poruke, *dataGrid* za prikaz dodatih *attachmenta* i navedene komponente su povezane sa odgovarajućim atributima *MessageBean* klase. Ovim je omogućeno ne samo da vrednosti ovih komponenti budu preuzete datim atributima klase nakon klika na dugme za slanje, nego i da vrednosti komponenti budu podešene iz *managed bean*-a, za čim postoji potreba ukoliko korisnik želi da odgovori na primljenu poruku, ili prosledi poruku.

Pored navedenih komponenti za unos, modalni panel omogućava da klikom na dugme *Add attachement* se otvori dodatni modalni panel koji će omogućiti dodavanje *attachmenta* u trenutnu poruku. Prijem i obradu poslanih datoteka obavlja metoda *listener* klase *MessageBean*, i po završetku obrade, ugnježdena *support* komponenta će ponovo iscrtati *attachmentList*, na kom su prikazani dotad *upload*-ovane datoteke, kao i link za njihovo eventualno uklanjanje.

3.2. Implementacija na serverskoj strani

U okviru implementacije na serverskoj strani, razmotrićemo *entity* klase, *managed bean*-ove i klase koje implementiraju JavaMail API.

3.2.1. *Entity* klase

Entity klase predstavljaju model podataka aplikacije, i one omogućuju mapiranje relacionih podataka u bazi podataka na objektni model podataka, upotrebom anotacija. Uvid u implementaciju ovog sloja aplikacije dat je *entity* klasom *Tag*.

```
@Entity
@Table(name = "tag")
@NamedQueries({ @NamedQuery(name = "Tag.findAll", query = "SELECT u FROM Tag u"),
                @NamedQuery(name = "Tag.findById", query = "SELECT u FROM Tag u WHERE
u.id = :id"),
                @NamedQuery(name = "Tag.findByName", query = "SELECT u FROM Tag u WHERE
u.name = :name") })
public final class Tag implements Serializable, Comparable<Tag> {
        private static final long serialVersionUID = 1L;
        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        @Basic(optional = false)
        @Column(name = "id", nullable = false)
        private Long id;
        @Column(name = "name", length = 45)
        private String name;
        @Column(name = "priority")
        private short priority;
        @Lob
        @Column(name = "color")
        private Color color;
        @ManyToMany(cascade = CascadeType.ALL, mappedBy = "tagList", fetch =
FetchType.LAZY)
        private List<MessageInfo> messageList;

        @Transient
        private String colorRGB;

        public Tag() {
                super();
```

```
                this.colorRGB = "";
                this.name = "";
                this.priority = 50;
                this.color = Color.black;
                this.messageList = new ArrayList<MessageInfo>();
        }

        public Tag(String name, short priority, Color color, List<MessageInfo>
messageList) {
                this();
                this.colorRGB = "";
                this.name = name;
                this.priority = priority;
                this.color = color;
                this.messageList = messageList;
        }

        public Long getId() {
                return id;
        }

        public void setId(Long id) {
                this.id = id;
        }
        …
}
```

Listing 3.6. – Stablo foldera

Iz listinga 3.6. može se uočiti da je identifikator Tag *entity*-ja, *id*, anotiran sa *GeneratedValue* anotacijom, čime je specificirana strategija kojom će vrednost primarnog ključa biti automatski generisana od strane *persistence provider*-a. Identifikatori svih ostalih klasa su takođe obeleženi anotacijom *GeneratedValue*.

Ostali atributi klase *Message* predstavljaju pojedinačne kolone tabele *Messages*, dok atribut *messageList* služi za uspostavljanje veze između *entity* klasa *Tag* i *MessageInfo* i omogućava praćenje poruka za koje je vezan određeni tag. Atribut *collorRGB* je anotiran *Transient* anotacijom, i on služi samo kao atribut koji vraća boju u html kompatibilnom heksadecimalnom obliku.

3.2.2. *Managed bean*-ovi

Managed bean-ovi predstavljaju kontrolorski sloj aplikacije, koji posreduje između prezentacionog sloja i sloja podataka. Metode ovih klasa najčešće vrše obradu događaja koji su se desili na prezentacionom sloju, po potrebi manipulišu slojem podataka, i rezultate procesiranja prikazuju na prezentacionom sloju – na JSF stranicama.

U nastavku je predstavljena jednostavna metoda *convertSelectedTableItemToAccount* klase *AccountBean*.

```
public void convertSelectedTableItemToAccount() {
        EntityManager em = PersistenceUtil.getEntityManager();
        selectedAccount = em.find(Account.class, (accounts.get((Integer)
        getSelectedTableItem().getKeys().next())).getId());
        ContextUtils.rerenderComponent("deleteAccount");
        ContextUtils.rerenderComponent("editAccount");
}

public SimpleSelection getSelectedTableItem() {
        return selectedTableItem;
}
```

Listing 3.7. – Metoda *managed bean-a*

Kao što se može videti na listingu 3.7. metoda *convertSelectedTableItemToAccount* predstavlja osluškivač događaja vezan za selekciju emajl naloga u tabeli naloga. Unutar ove

metode može se videti da se dobavljanje samih podataka iz tabele jednostavno može uzeti prolaskom kroz mapu ključeva objekta tipa *SimpleSelection* koju vraća metoda *getSelectedTableItem*. Po pronalasku samog naloga metoda pomoću klase *ContextUtils* izaziva ponovno iscrtavanje komponenti "deleteAccount" i "editAccount" koje su u ovom slučaju dvadugmeta unutar modalnog panela. Ta dva dugmeta su u osnovnom stanju onesposobljena jer nije selektovan nalog te tek pošto se selektuje nalog metoda označava *AjaxContext*-u da treba da iscrta te dve komponente koje će zbog provere koja je definisana za njih biti osposobljene za rad:

```
<a4j:commandLink id="deleteAccount"
  …                              disabled="#{accountBean.selectedAccount.id == null}">
```

Da bi se pokazao i objasnio rad *autoComplete* mogućnosti vezanih za unos emajl adrese u slučaju da postoji ta emajl adresa prikazana je metoda *autocomplete MessageBean* klase.

```
public List<String> autocomplete(Object suggest) {
        EntityManager em = PersistenceUtil.getEntityManager();
        String pref = (String) suggest;
        List<String> result = new ArrayList<String>();
        User user = em.find(User.class, ((UserBean)
        FacesContext.getCurrentInstance().getExternalContext().getSessionMap().get("use
        rBean")).getUserId());
        List<String> contacts = new ArrayList<String>();
        for (AddressBook ab : user.getAddressBookList()) {
                for (Contact contact : ab.getContactList()){
        if(contact.getAutoComplete().toLowerCase().contains(pref.toLowerCase().trim()))
{
        contacts.add(contact.getAutoComplete());
                        }
                }
        }
        Collections.sort(contacts);
        result.addAll(contacts);
        return result;
}
```

Listing 3.8. – *Autocomplete* metoda

Metoda predstavljena kodom na listingu 3.8. je osluškivač vezan za *inputText* komponenti modalnog panela za slanje pošte. Ova metoda jednostavno prolazi kroz listu adresara i svakog kontakta unutar tog adresara trenutnog korisnika i u slučaju da se *emajl* ili neki od identifikatora kontakta slažu sa trenutno unesenim tekstom to stavlja u listu i vraća korisniku na izbor.

Za kraj, pogledajmo i metodu koja vrši slanje poruke.

```
public void send() {
        EntityManager em = PersistenceUtil.getEntityManager();
        Account acc = em.find(Account.class, from);
        if (!acc.isCompose_html()) {
                sendMailText = extractText(new StringReader(sendMailText));
        }
        MailClass mc = ((UserBean)
FacesContext.getCurrentInstance().getExternalContext().getSessionMap().get("userBean")
).getMailAccounts().get(acc.getId());
        mc.sendMail(acc.getEmail(), to, cc, bcc, replyTo, subject, referencedMessageId,
priority, sendMailText, files, acc.isCompose_html(),
                        acc.isSending_copy());
        if (referencedMessageId != null && selectedMessage.getId() != null) {
                EntityTransaction et = em.getTransaction();
                et.begin();
                selectedMessage = em.find(MessageInfo.class, selectedMessage.getId());
                selectedMessage.setReplied(true);
```

```
                String[] podeljen = selectedMessage.getMessageData().split("accounts",
2);
                podeljen[1] = podeljen[1].substring(1);
                podeljen[1] =
podeljen[1].substring(podeljen[1].indexOf(File.separatorChar) + 1);
                mc.setMessageAsReplied(podeljen[1].replace(File.separatorChar, '.'),
selectedMessage.getMessageId());
                em.merge(selectedMessage);
                em.flush();
                et.commit();
        }
        clearSendInfo();
        ContextUtils.rerenderComponent("addAttachmentsMP");
        ContextUtils.rerenderComponent("sendMailMP");
        ContextUtils.rerenderComponent("emailTable");
        ContextUtils.rerenderComponent("folderTree");

}
```

Listing 3.9. – Slanje poruke

Metoda predstavljena kodom na listingu 3.9, kao i prethodne, je osluškivač koji je vezan za dugme za slanje poruke, na stranici za kreiranje poruke. Metoda pronalazi odgovarajući emajl nalog sa kojeg se šalje poruka unutar liste svih emajl naloga za datog korisnika, koji vrši slanje poruke pomoću JavaMail API-ja, i potom pomoću atributa izvučenih iz naloga sa kojeg se šalje poruka popunjava potrebne parametre metode *sendMail* i u slučaju da je to odgovor na poruku poziva metodu *setMessageAsReplied*. Po završetku operacija metoda poziva ponovno iscrtavanje komponenti korisničkog interfejsa.

Pored metoda koje predstavljaju osluškivače, *managed bean*-ovi poseduju i atribute koji se povezuju sa vrednošću određene komponente na korisničkom interfejsu, ili sa nekim drugim atributom korisničke komponente, poput atributa *rendered* čija *boolean* vrednost utvrđuje da li će data komponenta biti prikazana ili ne.

Primer koji najbolje opisuje dodelu vrednosti komponenti korisničkog interfejsa je kreiranje poruke. Poruka se može kreirati na četiri različita načina:

- kreiranje nove poruke
- odgovor na poruku
- prosleđivanje poruke

Vrednosti komponenti prikazanih na stranici za kreiranje sadržaja poruke zavise od toga na koji od navedenih načina korisnik želi da kreira poruku. Na primer, ukoliko korisnik želi da kreira novu poruku, izvesno je da očekuje da vrednosti komponenti za unos adrese i teksta poruke budu prazne, no ako korisnik želi da odgovori na selektovanu poruku – vrednost komponente za unos adrese treba da bude adresa kontakta kome korisnik odgovara, i vrednost komponente za unos teksta poruke treba da bude tekst originalne poruke.

Da bi ovo bilo ostvarivo, potrebno je da se nakon izbora načina kreiranja poruke od strane korisnika, unutar instance *MessageBean*-a, koja se čuva pod *session* opsegom, pozove metoda koja opisuje izabrani način. Potom se korisniku prikazuje novi prozor sa stranicom za kreiranje poruke, i komponente ove stranice će u zavisnosti od pozvane metode poprimiti odgovarajuću vrednost. Pogledajmo stoga na listingu 3.10. primer metode odgovora na poruku.

```
public void reply() {
        EntityManager em = PersistenceUtil.getEntityManager();
        Account acc = em.find(Account.class,
                        ((AccountBean)
FacesContext.getCurrentInstance().getExternalContext().getSessionMap().get("accountBea
n")).getAccountId());
        StringBuilder sb = new StringBuilder();
        if (selectedMessage.getReplyTo() != null && selectedMessage.getReplyTo().length
> 0) {
                for (InternetAddress ia : selectedMessage.getReplyTo()) {
                        if (sb.length() > 0) {
```

```
                        sb.append(", ");
                    }
                    sb.append(ia.toString());
                }
            } else {
                for (InternetAddress ia : selectedMessage.getFrom()) {
                    if (sb.length() > 0) {
                        sb.append(", ");
                    }
                    sb.append(ia.toString());
                }
            }
            sendMailText = "On " + new Date() + " " + sb.toString() + " wrote:
<br><blockquote cite=mid:\""
                    + (selectedMessage.getMessageId().substring(1,
selectedMessage.getMessageId().length() - 1)) + "\" type=\"cite\">" + emailText
                    + "</blockquote>";
            from = ((AccountBean)
FacesContext.getCurrentInstance().getExternalContext().getSessionMap().get("accountBea
n")).getAccountId();
            to = sb.toString();
            sb = null;
            if (acc.isUse_cc()) {
                cc = acc.getCc();
            } else {
                cc = "";
            }
            if (acc.isUse_bcc()) {
                bcc = acc.getBcc();
            } else {
                bcc = "";
            }
            replyTo = acc.getReply_to_address();
            referencedMessageId = selectedMessage.getMessageId();
            subject = "RE: " + selectedMessage.getSubject();
            ContextUtils.rerenderComponent("addAttachmentsMP");
            ContextUtils.rerenderComponent("sendMailMP");

}
```

Listing 3.10. – Metoda odgovora na poruku

Atributi *managed bean*-ova mogu predstavljati i kolekciju *entity* objekata, koji su povezani sa vrednošću kontejnerske komponente korisničkog interfejsa, poput *scrollableDataTable* komponente. Primer rukovanja kolekcijama *entity* objekata dat je na listingu 3.11.

```
private List<Account> accounts = new ArrayList<Account>();

public List<Account> getAccounts() {
        EntityManager em = PersistenceUtil.getEntityManager();
        User user = em.find(User.class, ((UserBean)
FacesContext.getCurrentInstance().getExternalContext().getSessionMap().get("userBean")
).getUserId());
        accounts = user.getAccountList();
        return accounts;
}

public void setAccounts(List<Account> accounts) {
        this.accounts = accounts;
}
```

Listing 3.11. – Rukovanje kolekcijama *entity* objekata

Na stranici za rukovanje nalozima, postojeći nalozi su predstavljeni tabelarno upotrebom *scrollableDataTable* komponente, koja je povezana sa atributom *accounts* klase *AccountBean*.

3.2.3. Klase za upravljanje JavaMail API-jem

Klase ovog sloja implementiraju ključnu funkcionalnost Webmail aplikacije, a to je prijem i slanje poruka, te će biti predstavljeno inicijalno uspostavljanje konekcije sa serverom i metode prijema i slanja pošte.

```
private void createConnection() {
        this.props = System.getProperties();
        this.props.setProperty("mail.transport.protocol", "smtp");
        this.props.setProperty("mail.smtp.host", smtpHost);
        this.props.setProperty("mail.smtp.port", String.valueOf(smtpPort));
        this.props.setProperty("mail.smtp.auth", "false");
        this.session = Session.getInstance(this.props, null);
        this.rf = null;
        try {
                if (serverType == Constants.ACCOUNT_TYPE_POP3) {
                        this.store = this.session.getStore("pop3");
                } else {
                        this.store = this.session.getStore("imap");
                }
                this.store.connect(this.host, this.port, this.user, this.password);
                this.rf = this.store.getDefaultFolder();
        } catch (NoSuchProviderException e) {
                e.printStackTrace();
        } catch (MessagingException e) {
                e.printStackTrace();
        } catch (NumberFormatException e) {
                e.printStackTrace();
        } catch (Exception e) {
                e.printStackTrace();
        }
}
```

Listing 3.12. – Kreiranje konekcije ka *mail* serveru

Metoda prikazana na listingu 3.12. omogućuje rad aplikacije sa *mail* serverom i način povezivanja aplikacije sa *mail* serverom.

```
private void dumpFolder(Folder folder) {
                                                …
                if ((folder.getType() & Folder.HOLDS_MESSAGES) != 0) {
                        try {
                                folder.open(Folder.READ_WRITE);
                        } catch (MessagingException ex) {
                                folder.open(Folder.READ_ONLY);
                        }
                        Message[] msgs = folder.getMessages();
                        for (Message message : msgs) {
                                dumpMessage(message, currentFolder);
                        }
```

Listing 3.13. – Metoda preuzimanja poruke sa određenog foldera na *mail* serveru

Prijem poruka je implementiran u okviru metode *dumpFolder*, prikazane na listingu 3.13, klase *MailClass* i prikazuje samo način prijema pošte. Metoda *dumpMessage* (zbog velike količine koda nije moguće jednostavno prikazati metodu *dumpMessage*) vrši obradu i proveru same pošte i *header* i *attachment* delova te poruke. Nakon toga poruka se smešta u bazu podataka.

```
public void sendMail(String from, String to, String cc, String bcc, String replyTo,
String subject, String referencedMessageId, short priority,
               String sendMailText, ArrayList<UploadFajl> fajlovi, boolean html,
boolean saveInSent) {
       createConnection();
       MimeMessage message = new MimeMessage(session);
                                                …
               Transport.send(message);
```

Listing 3.14. – Metoda slanja pošte

S druge strane, slanje poruka je implementirano u okviru *sendMail* (zbog većeg koda na listingu 3.14. prikazana je osnova samog slanja poruke) metode ukoliko je definisan neki od parametara date metode on se smešta u poruku ili u slučaju *saveInSent* označava da se poruka treba smestiti u *Sent* folder u slučaju uspešnog slanja. Ukoliko postoje i *attachment*-i pridodati uz ovu poruku, unutar ove metode vrši se i uključivanje sadržaja datoteke u odgovarajući objekat klase *MimeMessage*.

3.3. Izgled Webmail aplikacije

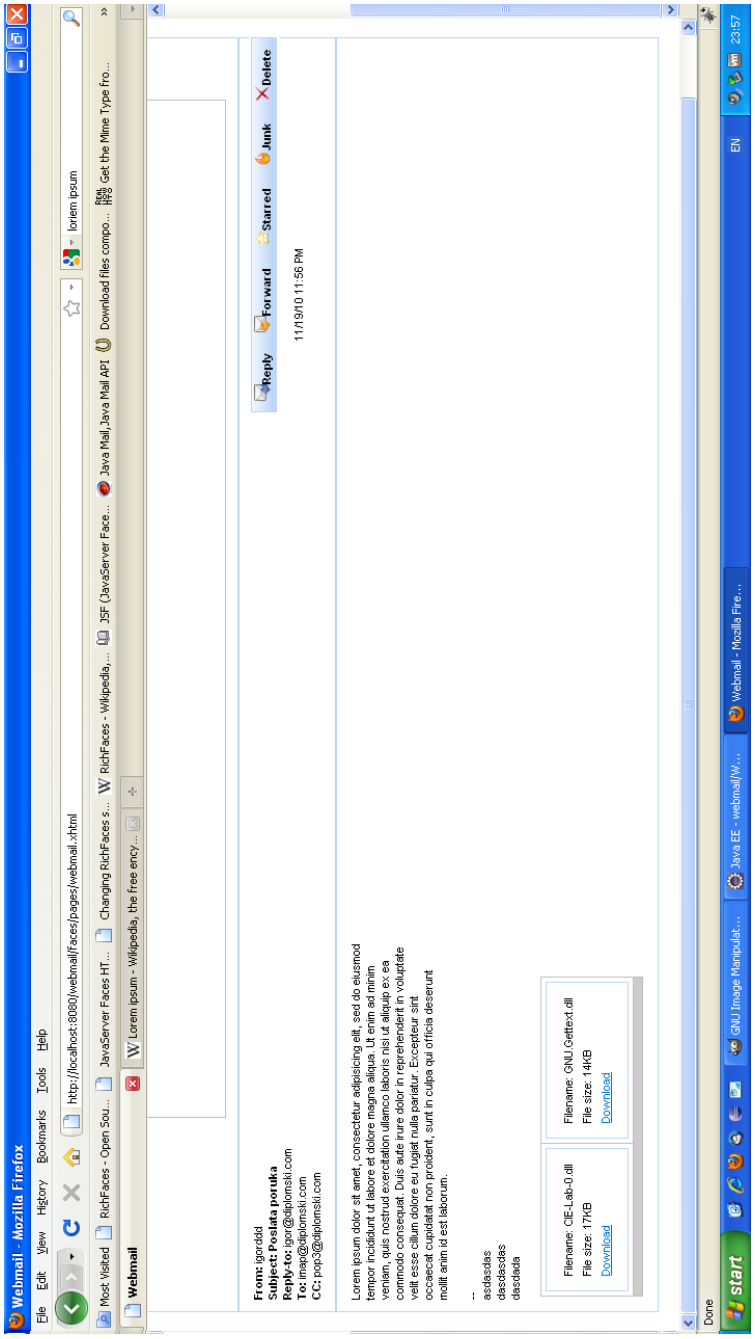
Na slici 3.1. prikazan je izgled dela stranice za prijavu na sistem Webmail aplikacije.

Slika 3.1. – Prijava na sistem

Po unosu korisničkog imena i lozinke, i kliku na dugme "*Login*", korisnik biva upućen na osnovnu stranicu Webmail aplikacije. Sistem automatski prihvata nove poruke sa *mail* servera, i prikazuje ih u tabeli poruka na osnovnoj stranici. Slika 3.2. prikazuje izgled osnovne stranice Webmail aplikacije.

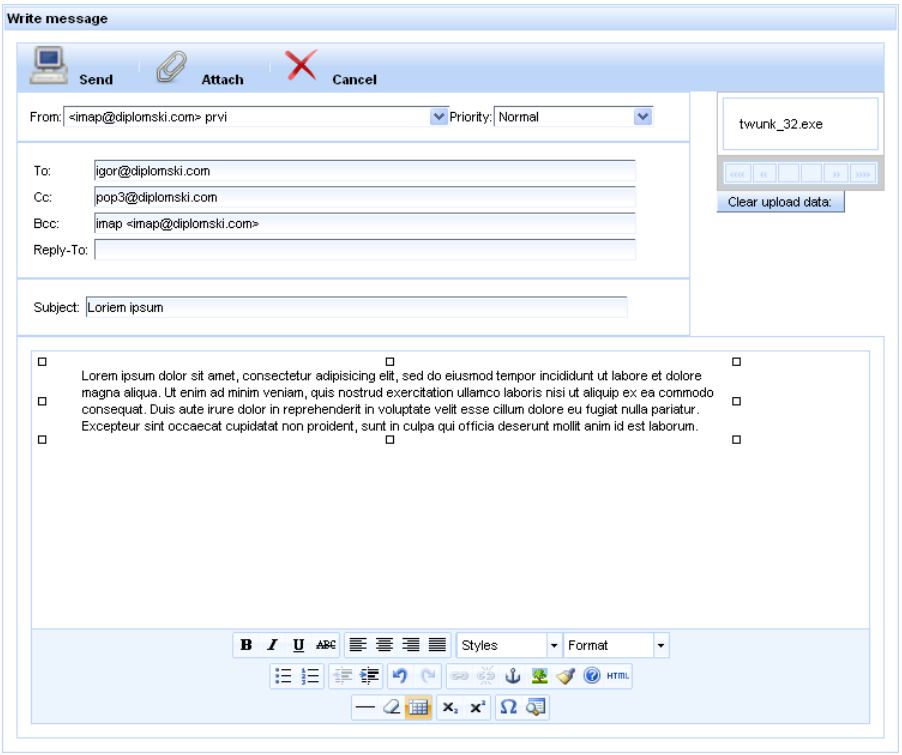
Slika 3.2. – Osnovna stranica

Na slici 3.2. se mogu uočiti ranije opisani elementi korisničkog interfejsa, poput glavnog menija, stabla foldera, kao i tabele sa porukama. Nakon klika na red kojim je reprezentovana odgovarajuća poruka u tabeli poruka, sistem na mestu tabele poruka prikazuje panel sa sadržajem jedne poruke. Ovaj panel prikazan je na slici 3.3.



Slika 3.3. – Sadržaj poruke

Klikom na dugme za odgovor na poruku ("*Reply*"), ili dugme za prosleđivanje poruke ("*Forward*"), otvara se novi prozor, sa stranicom za kreiranje poruke. Isti prozor se otvara i nakon klika na hiperlink "*Write*" koji se nalazi u glavnom meniju aplikacije. Na slici 3.4. prikazana je stranica za kreiranje nove poruke.



Slika 3.4. – Kreiranje nove poruke

Nakon klika na dugme "*Send* " na stranici za kreiranje nove poruke, prikazanoj na slici 3.4., sistem šalje datu poruku na listu primaoca navedenu u polju za unos primaoca na stranici. Pored toga moguće je i prekinuti njeno kreiranje, klikom na dugme "*Cancel*".

4. ZAKLJUČAK

U ovom radu analizirana je Webmail aplikacija, kao i tehnologije koje su korišćene prilikom njene realizacije.

Webmail aplikacija nudi ključne funkcionalnosti jednog *mail* klijenta, omogućujući korisnicima pisanje i slanje tekstualnih poruka uz dodavanje eventualnih *attachment*-a, razmeštanje poruka po folderima, ažuriranje liste kontakata i obeležavanje poruka.

Razvoj Webmail aplikacije je u velikoj meri olakšan upotrebom tehnologija koje su opisane u ovom radu. Iako je u okviru aplikacije korišćen tek mali deo ukupne funkcionalnosti koje nude neke od spomenutih tehnologija, ipak bi bez njihove upotrebe napor i vreme potrebni za postizanje ukupne funkcionalnosti aplikacije bili neuporedivo veći.

Pored toga, niz raznolikih komponenti korisničkog interfejsa koje nude JSF i RichFaces obezbeđuje brz razvoj bogatog korisničkog interfejsa *web* aplikacija, pa je takav slučaj bio i sa interfejsom Webmail aplikacije. Ipak, pored nesumnjivo velikog broja vrlina RichFaces biblioteke, u toku razvoja aplikacije registrovan je i određeni broj problema sa komponentama ove biblioteke. Primera radi, rad sa hijerarhijama pomoću stabala je veoma otežan zbog nemogućnosti jednostavnog određivanja stanja samog stabla pri kreiranju.Takođe jedan od nedostataka je i nemogućnost korišćenja dugmeta ukoliko se želi grafička oznaka, na toolbaru već se morao koristiti commandLink obuhvaćen sa mnogo css elemenata.

Važno je napomenuti da je *JBoss*, koji i stoji iza RichFaces *framework*-a, svestan većine nedostataka unutar *framework*-a, pa tako i ovog, i dati nedostaci se ispravljaju iz verzije u verziju.

LITERATURA

[1] The Java EE 5 Tutorial,
http://java.sun.com/javaee/5/docs/tutorial/doc/

[2] Ajax Tutorial,
http://www.w3schools.com/Ajax/Default.Asp

[3] Jboss RichFaces,
http://www.jboss.org/jbossrichfaces/

[4] JavaMail API,
http://java.sun.com/products/javamail/

[5] Facelets,
https://facelets.dev.java.net/