UNIVERZITET METROPOLITAN BEOGRAD

PROJEKTNI ZADATAK

NAZIV PROJEKTA	Razvoj veb distribuirnog IS "World Wide Wings"
TIM	Razvojni tim studenata treće godine FIT-a
RUKOVODILAC PROJEKTA	Vladimir Milićević
ČLANOVI PROJEKTNOG TIMA	Student 3. godine FIT-a Miloš Savić
POLJE PROJEKTA	CS - računarske nauke; CS230-Distribuirani sistemi
MESTO IZVOĐENJA PROJEKTA	RU2, Univerzitet Metropolitan u Beogradu
FIRMA ZA KOJU SE PROJEKAT RADI	Projekat je samostalna ideja
CILJ PROJEKTA	Razvoj funkcionalnog veb distribuiranog IS
IZLAZ KOJI SE DOBIJA PROJEKTOM	Funkcionalno i testirano softversko rešenje za podršku avionske kompanije
AKTIVNOSTI TOKOM REALIZACIJE PROJEKTA	Ideja - rezultat naručenog posla ili dogovora projektnog tima;s Izrada projektne dokumentacije; Projektovanje rešenja; Kodiranje rešenja; Testiranje i implementacija.
TEHNOLOGIJE I ALATI	JDK 8, Java EE 7, JSF okvir, JPA API, NetBeans IDE
HARDVERSKI RESURDI	Lični računar - za izradu projekta i testiranje.
VREMESKI PLAN REALIZACIJE PROJEKTA	10.06. - 10.07.2018.
MESTO I DATUM IZRADE PROJEKTNOG ZADATKA	U Beogradu, 13.06.2018.

Korisnički zahtevi - funkcionalni zahtevi

Funkcionalni zahtevi aplikacije predstavljaju spisak zahteva koji se odnose na funkcionalnosti i mogućnosti aplikacije. Oni su definisani i obrazloženi sledećim elementima:

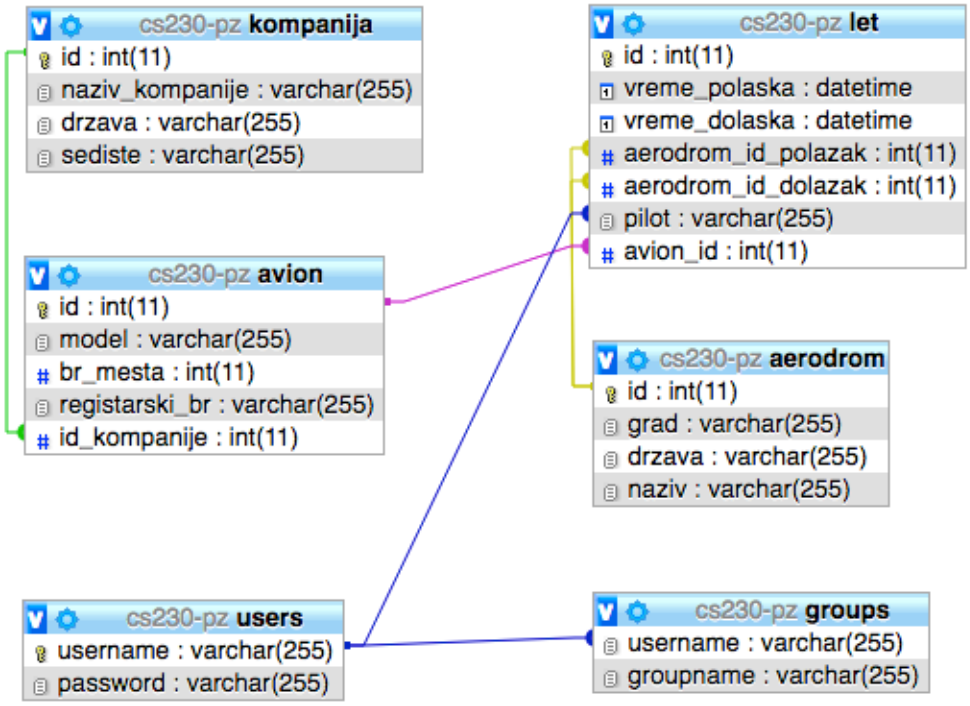
- Korisnici – Sistem će posedovati dva tipa korisnika, administratora i običnog korisnika. Korisnici se prijavljuju preko forme za login.
- Baza podataka – Administrator ima mogućnost da doda podatke o avionima, o aerodromima i slično, da pregleda sve podatke, da ih izmeni, kao i da ih ukloni ukoliko za to postoji potreba. Dok obični korisnik može samo da pregleda podatke.
- Korisnički Interfejs – Mora biti implementiran korisnički interfejs na klijent strani za interakciju korisnika sa aplikacijom. Za svaku operaciju koju korisnik izvodi nad bazom podataka veb aplikacije mora biti realizovana adekvatna JSF stranica sa odgovarajućim korisničkim interfejsom;
- Efikasno upravljanje bazom podataka - Upravljanje bazom podataka mora biti realizovano po **ORM** principima. Biće korišćen ***RDMS sistem MySQL*** u sinergiji sa ***JPA API*** za ***ORM***.

Korisnički zahtevi - nefunkcionalni zahtevi

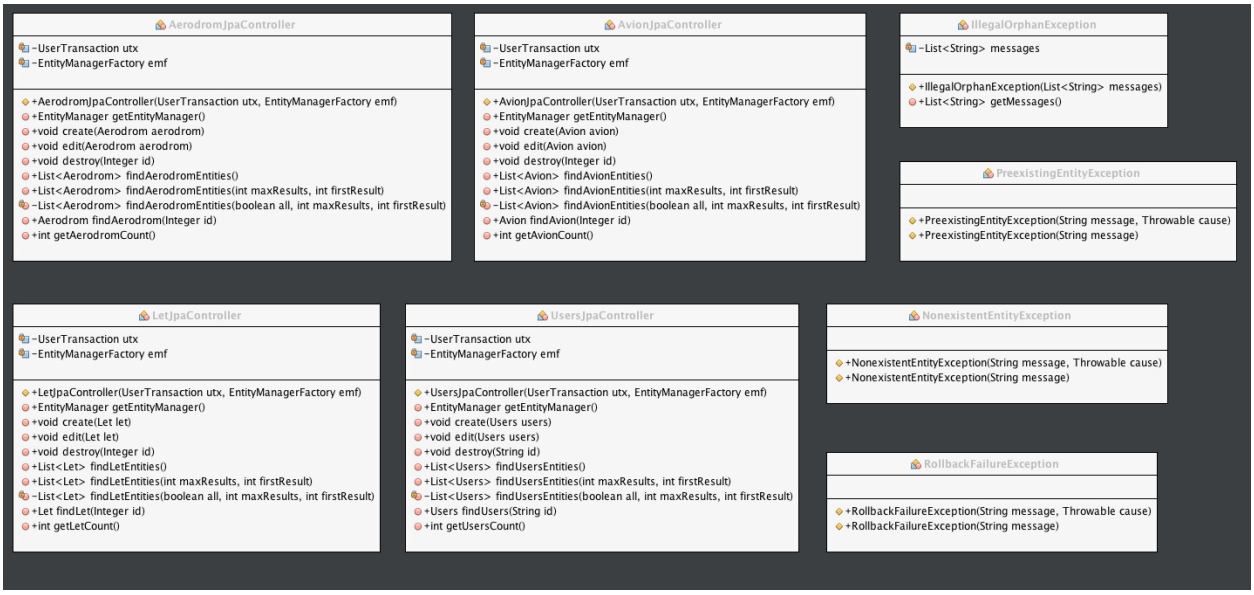
Nefunkcionalni zahtevi predstavljaju zahteve koji se tiču načina implementacije aplikacije. U tom svetlu je neophodno opisati sledeće elemente:

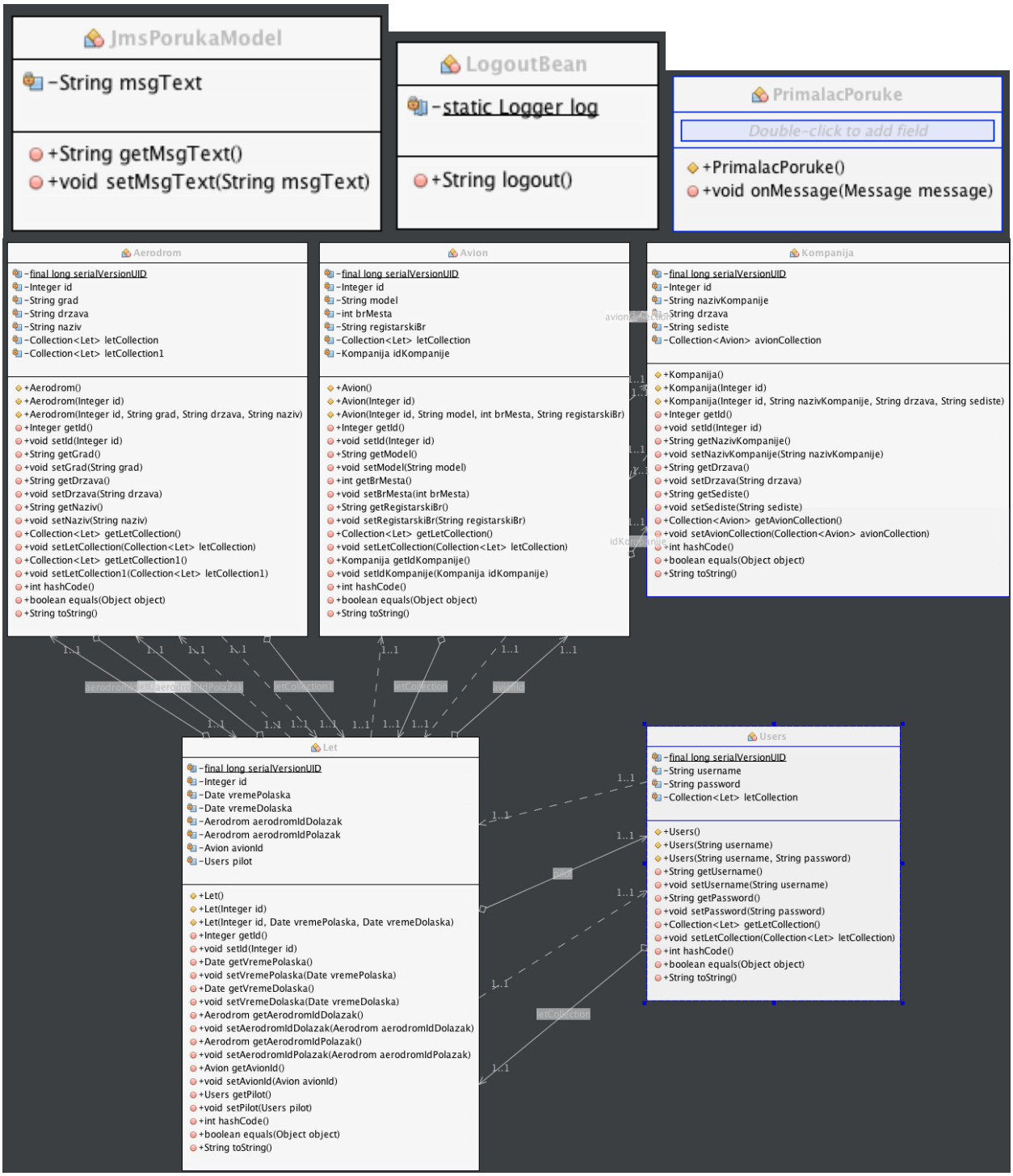
- Performanse – Izborom poslednje verzije Java EE 7 platforme i automatizovanih operacija koje izvode savremena razvojna okruženja značajno je unapređeno polje performansi distribuiranih veb sistema;
- Bezbednost – Izborom poslednje verzije Java EE 7 platforme i automatizovanih operacija koje izvode savremena razvojna okruženja značajno je unapređeno polje bezbednosti distribuiranih veb sistema. Ovo je veoma osetljivo polje kojem bi trebalo posvetiti veliku pažnju tokom razvoja aplikacije. Aplikacija će koristiti Realm opciju GlassFish servera za bezbednost.
- Održivost – Bitno je razviti program korišćenjem dobrih programerskih praksi, kako bi kasnije bilo lakše realizovati dodavanje novih funkcionalnosti ili ispravljanje eventualnih grešaka na koje su testeri ili korisnici ukazali, a koje mogu da nastanu tokom procesa razvoja aplikacije.

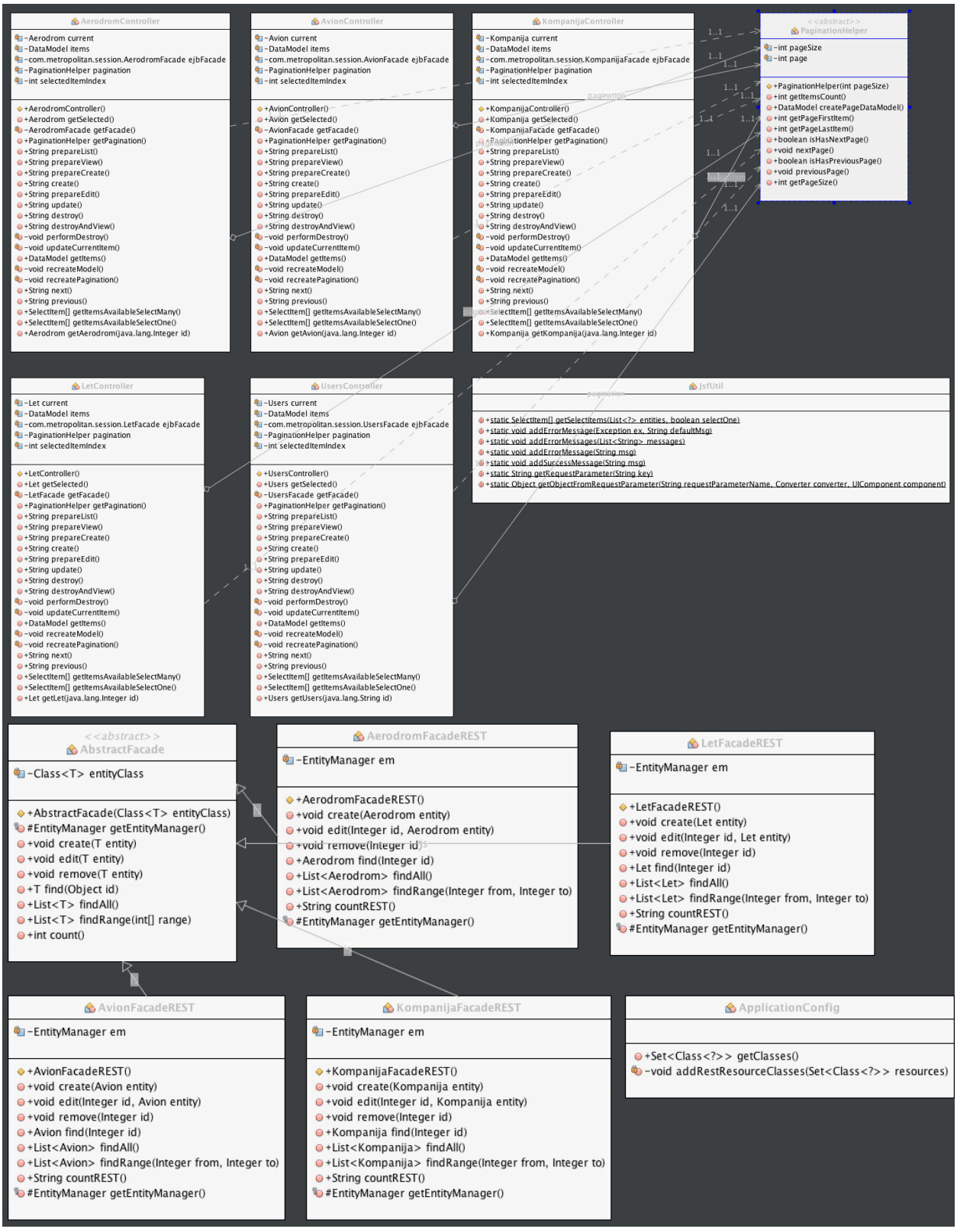
Dijagrami - dijagram klasa i šema baze podataka

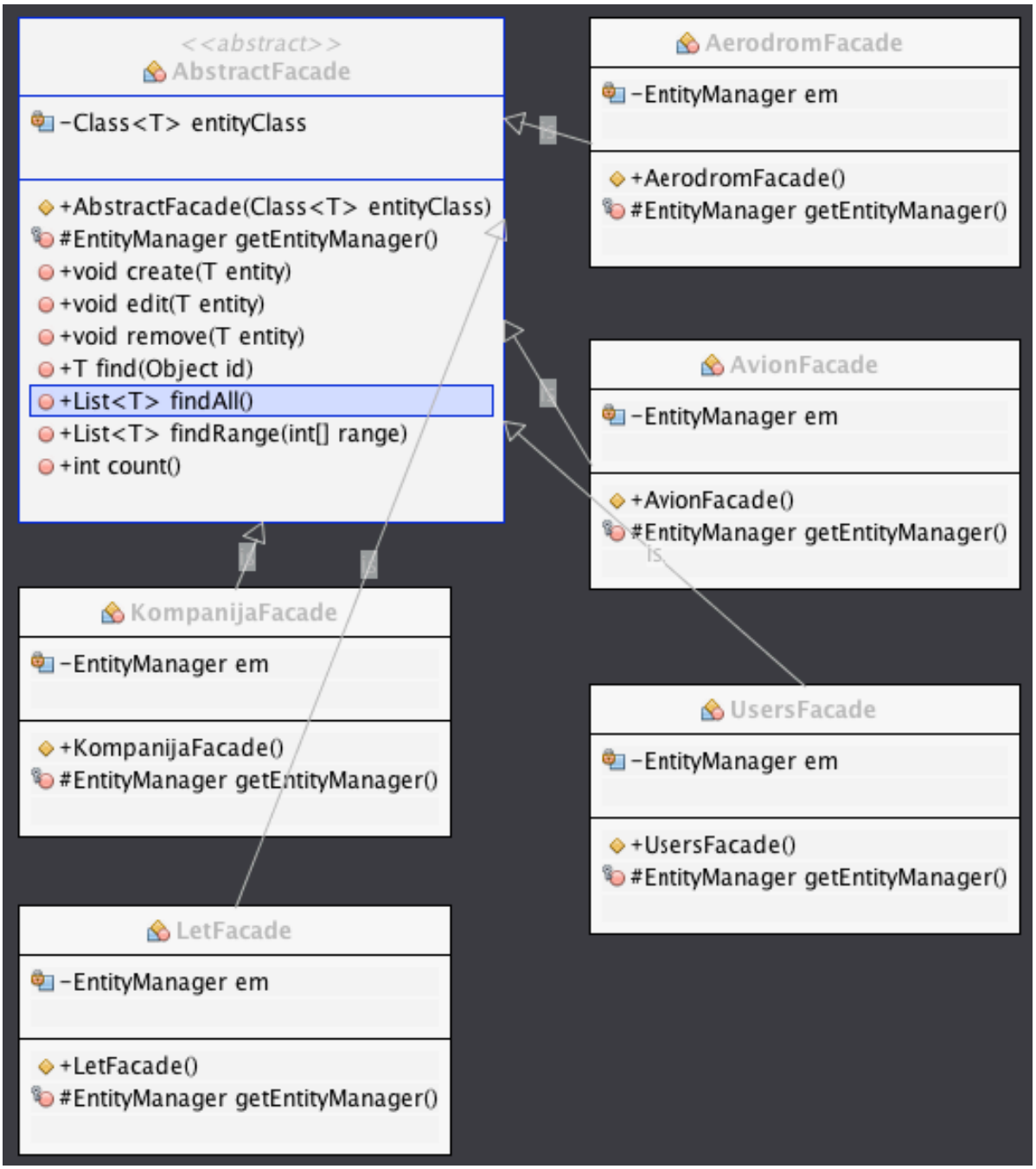


Slika 1. Šema baze podataka




JmsPorukaModel
-String msgText
+String getMsgText()
+void setMsgText(String msgText)
LogoutBean
-static Logger log
+String logout()
PrimalacPoruke
Double-click to add field
+PrimalacPoruke()
+void onMessage(Message message)
Aerodrom
-final long serialVersionUID
-Integer id
-String grad
-String drzava
-String naziv
-Collection<Let> letCollection
-Collection<Let> letCollection1
+Aerodrom()
+Aerodrom(Integer id)
+Aerodrom(Integer id, String grad, String drzava, String naziv)
+Integer getId()
+void setId(Integer id)
+String getGrad()
+void setGrad(String grad)
+String getDrzava()
+void setDrzava(String drzava)
+String getNaziv()
+void setNaziv(String naziv)
+Collection<Let> getLetCollection()
+void setLetCollection(Collection<Let> letCollection)
+Collection<Let> getLetCollection1()
+void setLetCollection1(Collection<Let> letCollection1)
+int hashCode()
+boolean equals(Object object)
+String toString()
Avion
-final long serialVersionUID
-Integer id
-String model
-int brMesta
-String registarskiBr
-Collection<Let> letCollection
-Kompanija idKompanije
+Avion()
+Avion(Integer id)
+Avion(Integer id, String model, int brMesta, String registarskiBr)
+Integer getId()
+void setId(Integer id)
+String getModel()
+void setModel(String model)
+int getBrMesta()
+void setBrMesta(int brMesta)
+String getRegistarskiBr()
+void setRegistarskiBr(String registarskiBr)
+Collection<Let> getLetCollection()
+void setLetCollection(Collection<Let> letCollection)
+Kompanija getIdKompanije()
+void setIdKompanije(Kompanija idKompanije)
+int hashCode()
+boolean equals(Object object)
+String toString()
Kompanija
-final long serialVersionUID
-Integer id
-String nazivKompanije
-String drzava
-String sediste
-Collection<Avion> avionCollection
+Kompanija()
+Kompanija(Integer id)
+Kompanija(Integer id, String nazivKompanije, String drzava, String sediste)
+Integer getId()
+void setId(Integer id)
+String getNazivKompanije()
+void setNazivKompanije(String nazivKompanije)
+String getDrzava()
+void setDrzava(String drzava)
+String getSediste()
+void setSediste(String sediste)
+Collection<Avion> getAvionCollection()
+void setAvionCollection(Collection<Avion> avionCollection)
+int hashCode()
+boolean equals(Object object)
+String toString()
Let
-final long serialVersionUID
-Integer id
-Date vremePolaska
-Date vremeDolaska
-Aerodrom aerodromIdDolazak
-Aerodrom aerodromIdPolazak
-Avion avionId
-Users pilot
+Let()
+Let(Integer id)
+Let(Integer id, Date vremePolaska, Date vremeDolaska)
+Integer getId()
+void setId(Integer id)
+Date getVremePolaska()
+void setVremePolaska(Date vremePolaska)
+Date getVremeDolaska()
+void setVremeDolaska(Date vremeDolaska)
+Aerodrom getAerodromIdDolazak()
+void setAerodromIdDolazak(Aerodrom aerodromIdDolazak)
+Aerodrom getAerodromIdPolazak()
+void setAerodromIdPolazak(Aerodrom aerodromIdPolazak)
+Avion getAvionId()
+void setAvionId(Avion avionId)
+Users getPilot()
+void setPilot(Users pilot)
+int hashCode()
+boolean equals(Object object)
+String toString()
Users
-final long serialVersionUID
-String username
-String password
-Collection<Let> letCollection
+Users()
+Users(String username)
+Users(String username, String password)
+String getUsername()
+void setUsername(String username)
+String getPassword()
+void setPassword(String password)
+Collection<Let> getLetCollection()
+void setLetCollection(Collection<Let> letCollection)
+int hashCode()
+boolean equals(Object object)
+String toString()
1..1
pilot
letCollection

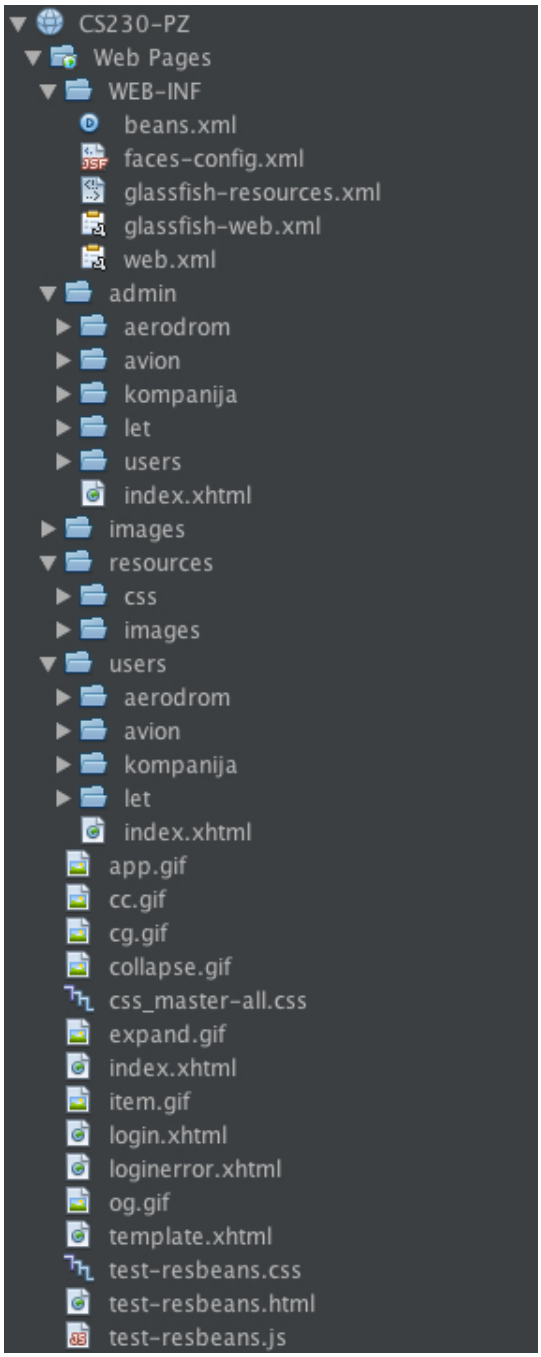

AerodromController
-Aerodrom current
-DataModel items
-com.metropolitan.session.AerodromFacade ejbFacade
-PaginationHelper pagination
-int selectedItemIndex
+AerodromController()
+Aerodrom getSelected()
-AerodromFacade getFacade()
+PaginationHelper getPagination()
+String prepareList()
+String prepareView()
+String prepareCreate()
+String create()
+String prepareEdit()
+String update()
+String destroy()
+String destroyAndView()
-void performDestroy()
-void updateCurrentItem()
+DataModel getItems()
-void recreateModel()
-void recreatePagination()
+String next()
+String previous()
+SelectItem[] getItemsAvailableSelectMany()
+SelectItem[] getItemsAvailableSelectOne()
+Aerodrom getAerodrom(java.lang.Integer id)
AvionController
-Avion current
-DataModel items
-com.metropolitan.session.AvionFacade ejbFacade
-PaginationHelper pagination
-int selectedItemIndex
+AvionController()
+Avion getSelected()
-AvionFacade getFacade()
+PaginationHelper getPagination()
+String prepareList()
+String prepareView()
+String prepareCreate()
+String create()
+String prepareEdit()
+String update()
+String destroy()
+String destroyAndView()
-void performDestroy()
-void updateCurrentItem()
+DataModel getItems()
-void recreateModel()
-void recreatePagination()
+String next()
+String previous()
+SelectItem[] getItemsAvailableSelectMany()
+SelectItem[] getItemsAvailableSelectOne()
+Avion getAvion(java.lang.Integer id)
KompanijaController
-Kompanija current
-DataModel items
-com.metropolitan.session.KompanijaFacade ejbFacade
-PaginationHelper pagination
-int selectedItemIndex
+KompanijaController()
+Kompanija getSelected()
-KompanijaFacade getFacade()
+PaginationHelper getPagination()
+String prepareList()
+String prepareView()
+String prepareCreate()
+String create()
+String prepareEdit()
+String update()
+String destroy()
+String destroyAndView()
-void performDestroy()
-void updateCurrentItem()
+DataModel getItems()
-void recreateModel()
-void recreatePagination()
+String next()
+String previous()
+SelectItem[] getItemsAvailableSelectMany()
+SelectItem[] getItemsAvailableSelectOne()
+Kompanija getKompanija(java.lang.Integer id)
<<abstract>>
PaginationHelper
-int pageSize
-int page
+PaginationHelper(int pageSize)
+int getItemsCount()
+DataModel createPageDataModel()
+int getPageFirstItem()
+int getPageLastItem()
+boolean isHasNextPage()
+void nextPage()
+boolean isHasPreviousPage()
+void previousPage()
+int getPageSize()
LetController
-Let current
-DataModel items
-com.metropolitan.session.LetFacade ejbFacade
-PaginationHelper pagination
-int selectedItemIndex
+LetController()
+Let getSelected()
-LetFacade getFacade()
+PaginationHelper getPagination()
+String prepareList()
+String prepareView()
+String prepareCreate()
+String create()
+String prepareEdit()
+String update()
+String destroy()
+String destroyAndView()
-void performDestroy()
-void updateCurrentItem()
+DataModel getItems()
-void recreateModel()
-void recreatePagination()
+String next()
+String previous()
+SelectItem[] getItemsAvailableSelectMany()
+SelectItem[] getItemsAvailableSelectOne()
+Let getLet(java.lang.Integer id)
UsersController
-Users current
-DataModel items
-com.metropolitan.session.UsersFacade ejbFacade
-PaginationHelper pagination
-int selectedItemIndex
+UsersController()
+Users getSelected()
-UsersFacade getFacade()
+PaginationHelper getPagination()
+String prepareList()
+String prepareView()
+String prepareCreate()
+String create()
+String prepareEdit()
+String update()
+String destroy()
+String destroyAndView()
-void performDestroy()
-void updateCurrentItem()
+DataModel getItems()
-void recreateModel()
-void recreatePagination()
+String next()
+String previous()
+SelectItem[] getItemsAvailableSelectMany()
+SelectItem[] getItemsAvailableSelectOne()
+Users getUsers(java.lang.String id)
JsfUtil
+static SelectItem[] getSelectItems(List<?> entities, boolean selectOne)
+static void addErrorMessage(Exception ex, String defaultMsg)
+static void addErrorMessages(List<String> messages)
+static void addErrorMessage(String msg)
+static void addSuccessMessage(String msg)
+static String getRequestParameter(String key)
+static Object getObjectFromRequestParameter(String requestParameterName, Converter converter, UIComponent component)
<<abstract>>
AbstractFacade
-Class<T> entityClass
+AbstractFacade(Class<T> entityClass)
#EntityManager getEntityManager()
+void create(T entity)
+void edit(T entity)
+void remove(T entity)
+T find(Object id)
+List<T> findAll()
+List<T> findRange(int[] range)
+int count()
AerodromFacadeREST
-EntityManager em
+AerodromFacadeREST()
+void create(Aerodrom entity)
+void edit(Integer id, Aerodrom entity)
+void remove(Integer id)
+Aerodrom find(Integer id)
+List<Aerodrom> findAll()
+List<Aerodrom> findRange(Integer from, Integer to)
+String countREST()
#EntityManager getEntityManager()
LetFacadeREST
-EntityManager em
+LetFacadeREST()
+void create(Let entity)
+void edit(Integer id, Let entity)
+void remove(Integer id)
+Let find(Integer id)
+List<Let> findAll()
+List<Let> findRange(Integer from, Integer to)
+String countREST()
#EntityManager getEntityManager()
AvionFacadeREST
-EntityManager em
+AvionFacadeREST()
+void create(Avion entity)
+void edit(Integer id, Avion entity)
+void remove(Integer id)
+Avion find(Integer id)
+List<Avion> findAll()
+List<Avion> findRange(Integer from, Integer to)
+String countREST()
#EntityManager getEntityManager()
KompanijaFacadeREST
-EntityManager em
+KompanijaFacadeREST()
+void create(Kompanija entity)
+void edit(Integer id, Kompanija entity)
+void remove(Integer id)
+Kompanija find(Integer id)
+List<Kompanija> findAll()
+List<Kompanija> findRange(Integer from, Integer to)
+String countREST()
#EntityManager getEntityManager()
ApplicationConfig
+Set<Class<?>> getClasses()
-void addRestResourceClasses(Set<Class<?>> resources)

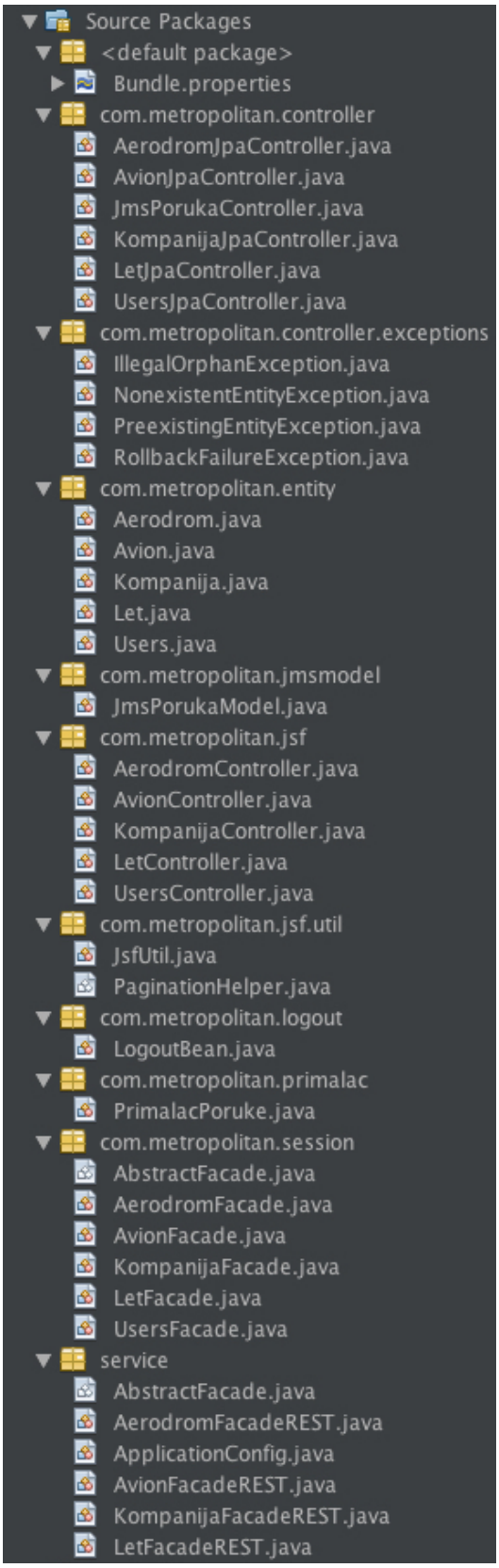


Slika 2. Dijagrami klasa

Implementacija aplikacije - hijerarhija projekta "World Wide Wings"

Hijerarhija pokazuje sve foldere sa pripadajućim datotekama, klasama i paketima, zavisnostima koji učestvuju u razvoju konkretne Java EE 7 veb aplikacije. Navedeno je prikazano sledećom slikom.

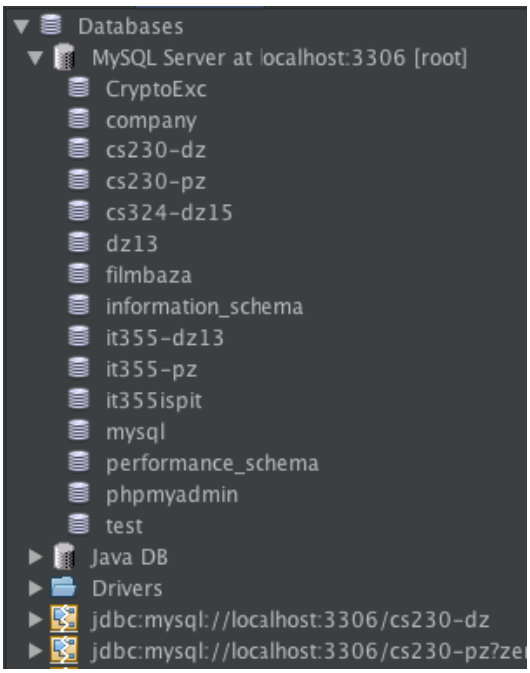




Slika 3. Hijerarhija projekta

Implementacija aplikacije - baza podataka

Ako se u razvojnom okruženju, za konkretan projekat, pažnja usmeri na tab Services, moguće je uočiti sekciju za bazu podataka koja pripada aplikaciji koja se razvija (sledeća slika). Na ovoj lokaciji je moguće pregledati sve relevantne informacije o bati podataka: naziv, skup tabela, tip baze podataka i odgovarajući drajver i tako dalje.



Slika 4. Podaci o bazi podataka

Tabele baze podataka softverskog rešenja, koje je predmet razvoja, kreirane su uz pomoć phpMyAdmin-a. Upiti su priloženi sledećim listingom:

```
CREATE TABLE `aerodrom` (

  `id` int(11) NOT NULL,

  `grad` varchar(255) NOT NULL,

  `drzava` varchar(255) NOT NULL,

  `naziv` varchar(255) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
CREATE TABLE `avion` (
  `id` int(11) NOT NULL,
  `model` varchar(255) NOT NULL,
  `br_mesta` int(11) NOT NULL,
  `registarski_br` varchar(255) NOT NULL,
  `id_kompanije` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `groups` (
  `username` varchar(255) NOT NULL,
  `groupname` varchar(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `kompanija` (
  `id` int(11) NOT NULL,
  `naziv_kompanije` varchar(255) NOT NULL,
  `drzava` varchar(255) NOT NULL,
  `sediste` varchar(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `let` (
  `id` int(11) NOT NULL,
  `vreme_polaska` datetime NOT NULL,
```

```
  `vreme_dolaska` datetime NOT NULL,
  `aerodrom_id_polazak` int(11) NOT NULL,
  `aerodrom_id_dolazak` int(11) NOT NULL,
  `pilot` varchar(255) DEFAULT NULL,
  `avion_id` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `users` (
  `username` varchar(255) NOT NULL,
  `password` varchar(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Implementacija aplikacije - JAVA klase - JPA entiteti

Sledi programski kod:

```
package com.metropolitan.entity;


import java.io.Serializable;
import java.util.Collection;
import javax.persistence.Basic;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
```

```
import javax.persistence.Id;

import javax.persistence.NamedQueries;

import javax.persistence.NamedQuery;

import javax.persistence.OneToMany;

import javax.persistence.Table;

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

import javax.xml.bind.annotation.XmlRootElement;

import javax.xml.bind.annotation.XmlTransient;


/**

 *

 * @author Milos

 */

@Entity

@Table(name = "aerodrom")

@XmlRootElement

@NamedQueries({

    @NamedQuery(name = "Aerodrom.findAll", query = "SELECT a FROM Aerodrom a")

    , @NamedQuery(name = "Aerodrom.findById", query = "SELECT a FROM Aerodrom a
WHERE a.id = :id")

    , @NamedQuery(name = "Aerodrom.findByGrad", query = "SELECT a FROM Aerodrom a
WHERE a.grad = :grad")

    , @NamedQuery(name = "Aerodrom.findByDrzava", query = "SELECT a FROM Aerodrom a
WHERE a.drzava = :drzava")
```

```
    , @NamedQuery(name = "Aerodrom.findByNaziv", query = "SELECT a FROM Aerodrom a
WHERE a.naziv = :naziv")})

public class Aerodrom implements Serializable {



    private static final long serialVersionUID = 1L;

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    @Basic(optional = false)

    @Column(name = "id")

    private Integer id;

    @Basic(optional = false)

    @NotNull

    @Size(min = 1, max = 255)

    @Column(name = "grad")

    private String grad;

    @Basic(optional = false)

    @NotNull

    @Size(min = 1, max = 255)

    @Column(name = "drzava")

    private String drzava;

    @Basic(optional = false)

    @NotNull

    @Size(min = 1, max = 255)

    @Column(name = "naziv")

    private String naziv;
```

```
@OneToMany(cascade = CascadeType.ALL, mappedBy = "aerodromIdDolazak")
private Collection<Let> letCollection;
@OneToMany(cascade = CascadeType.ALL, mappedBy = "aerodromIdPolazak")
private Collection<Let> letCollection1;

public Aerodrom() {
}

public Aerodrom(Integer id) {
    this.id = id;
}

public Aerodrom(Integer id, String grad, String drzava, String naziv) {
    this.id = id;
    this.grad = grad;
    this.drzava = drzava;
    this.naziv = naziv;
}

public Integer getId() {
    return id;
}

public void setId(Integer id) {
```

```
    this.id = id;
}

public String getGrad() {
    return grad;
}

public void setGrad(String grad) {
    this.grad = grad;
}

public String getDrzava() {
    return drzava;
}

public void setDrzava(String drzava) {
    this.drzava = drzava;
}

public String getNaziv() {
    return naziv;
}

public void setNaziv(String naziv) {
```

```
    this.naziv = naziv;
}

@XmlTransient
public Collection<Let> getLetCollection() {
    return letCollection;
}

public void setLetCollection(Collection<Let> letCollection) {
    this.letCollection = letCollection;
}

@XmlTransient
public Collection<Let> getLetCollection1() {
    return letCollection1;
}

public void setLetCollection1(Collection<Let> letCollection1) {
    this.letCollection1 = letCollection1;
}

@Override
public int hashCode() {
    int hash = 0;
```

```
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set
        if (!(object instanceof Aerodrom)) {
            return false;
        }
        Aerodrom other = (Aerodrom) object;
        if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return naziv;
    }
}
```

Ostatak klasa nije prikazan zbog obimnosti.

Implementacija aplikacije - JAVA klase - Kontroleri JPA entiteta

```
package com.metropolitan.controller;


import com.metropolitan.controller.exceptions.IllegalOrphanException;
import com.metropolitan.controller.exceptions.NonexistentEntityException;
import com.metropolitan.controller.exceptions.RollbackFailureException;
import com.metropolitan.entity.Aerodrom;
import java.io.Serializable;
import javax.persistence.Query;
import javax.persistence.EntityNotFoundException;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import com.metropolitan.entity.Let;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.transaction.UserTransaction;


/**
 *
 * @author Milos
 */
```

```
public class AerodromJpaController implements Serializable {

    public AerodromJpaController(UserTransaction utx, EntityManagerFactory emf) {
        this.utx = utx;
        this.emf = emf;
    }
    private UserTransaction utx = null;
    private EntityManagerFactory emf = null;

    public EntityManager getEntityManager() {
        return emf.createEntityManager();
    }

    public void create(Aerodrom aerodrom) throws RollbackFailureException, Exception {
        if (aerodrom.getLetCollection() == null) {
            aerodrom.setLetCollection(new ArrayList<Let>());
        }
        if (aerodrom.getLetCollection1() == null) {
            aerodrom.setLetCollection1(new ArrayList<Let>());
        }
        EntityManager em = null;
        try {
            utx.begin();
            em = getEntityManager();
```

```
        Collection<Let> attachedLetCollection = new ArrayList<Let>();

        for (Let letCollectionLetToAttach : aerodrom.getLetCollection()) {

            letCollectionLetToAttach = em.getReference(letCollectionLetToAttach.getClass(),
letCollectionLetToAttach.getId());

            attachedLetCollection.add(letCollectionLetToAttach);

        }

        aerodrom.setLetCollection(attachedLetCollection);

        Collection<Let> attachedLetCollection1 = new ArrayList<Let>();

        for (Let letCollection1LetToAttach : aerodrom.getLetCollection1()) {

            letCollection1LetToAttach = em.getReference(letCollection1LetToAttach.getClass(),
letCollection1LetToAttach.getId());

            attachedLetCollection1.add(letCollection1LetToAttach);

        }

        aerodrom.setLetCollection1(attachedLetCollection1);

        em.persist(aerodrom);

        for (Let letCollectionLet : aerodrom.getLetCollection()) {

            Aerodrom oldAerodromIdDolazakOfLetCollectionLet =
letCollectionLet.getAerodromIdDolazak();

            letCollectionLet.setAerodromIdDolazak(aerodrom);

            letCollectionLet = em.merge(letCollectionLet);

            if (oldAerodromIdDolazakOfLetCollectionLet != null) {


oldAerodromIdDolazakOfLetCollectionLet.getLetCollection().remove(letCollectionLet);

                oldAerodromIdDolazakOfLetCollectionLet =
em.merge(oldAerodromIdDolazakOfLetCollectionLet);

            }

        }
```

```
            for (Let letCollection1Let : aerodrom.getLetCollection1()) {

                Aerodrom oldAerodromIdPolazakOfLetCollection1Let =
letCollection1Let.getAerodromIdPolazak();

                letCollection1Let.setAerodromIdPolazak(aerodrom);

                letCollection1Let = em.merge(letCollection1Let);

                if (oldAerodromIdPolazakOfLetCollection1Let != null) {


oldAerodromIdPolazakOfLetCollection1Let.getLetCollection1().remove(letCollection1Let);

                    oldAerodromIdPolazakOfLetCollection1Let =
em.merge(oldAerodromIdPolazakOfLetCollection1Let);

                }

            }

            utx.commit();

        } catch (Exception ex) {

            try {

                utx.rollback();

            } catch (Exception re) {

                throw new RollbackFailureException("An error occurred attempting to roll back the
transaction.", re);

            }

            throw ex;

        } finally {

            if (em != null) {

                em.close();

            }

        }
```

```
    }


    public void edit(Aerodrom aerodrom) throws IllegalOrphanException,
NonexistentEntityException, RollbackFailureException, Exception {

        EntityManager em = null;

        try {

            utx.begin();

            em = getEntityManager();

            Aerodrom persistentAerodrom = em.find(Aerodrom.class, aerodrom.getId());

            Collection<Let> letCollectionOld = persistentAerodrom.getLetCollection();

            Collection<Let> letCollectionNew = aerodrom.getLetCollection();

            Collection<Let> letCollection1Old = persistentAerodrom.getLetCollection1();

            Collection<Let> letCollection1New = aerodrom.getLetCollection1();

            List<String> illegalOrphanMessages = null;

            for (Let letCollectionOldLet : letCollectionOld) {

                if (!letCollectionNew.contains(letCollectionOldLet)) {

                    if (illegalOrphanMessages == null) {

                        illegalOrphanMessages = new ArrayList<String>();

                    }

                    illegalOrphanMessages.add("You must retain Let " + letCollectionOldLet + " since
its aerodromIdDolazak field is not nullable.");

                }

            }

            for (Let letCollection1OldLet : letCollection1Old) {

                if (!letCollection1New.contains(letCollection1OldLet)) {
```

```
                if (illegalOrphanMessages == null) {
                    illegalOrphanMessages = new ArrayList<String>();
                }
                illegalOrphanMessages.add("You must retain Let " + letCollection1OldLet + " since its aerodromIdPolazak field is not nullable.");
            }
        }
        if (illegalOrphanMessages != null) {
            throw new IllegalOrphanException(illegalOrphanMessages);
        }
        Collection<Let> attachedLetCollectionNew = new ArrayList<Let>();
        for (Let letCollectionNewLetToAttach : letCollectionNew) {
            letCollectionNewLetToAttach = em.getReference(letCollectionNewLetToAttach.getClass(), letCollectionNewLetToAttach.getId());
            attachedLetCollectionNew.add(letCollectionNewLetToAttach);
        }
        letCollectionNew = attachedLetCollectionNew;
        aerodrom.setLetCollection(letCollectionNew);
        Collection<Let> attachedLetCollection1New = new ArrayList<Let>();
        for (Let letCollection1NewLetToAttach : letCollection1New) {
            letCollection1NewLetToAttach = em.getReference(letCollection1NewLetToAttach.getClass(), letCollection1NewLetToAttach.getId());
            attachedLetCollection1New.add(letCollection1NewLetToAttach);
        }
        letCollection1New = attachedLetCollection1New;
```

```
            aerodrom.setLetCollection1(letCollection1New);

            aerodrom = em.merge(aerodrom);

            for (Let letCollectionNewLet : letCollectionNew) {

                if (!letCollectionOld.contains(letCollectionNewLet)) {

                    Aerodrom oldAerodromIdDolazakOfLetCollectionNewLet =
letCollectionNewLet.getAerodromIdDolazak();

                    letCollectionNewLet.setAerodromIdDolazak(aerodrom);

                    letCollectionNewLet = em.merge(letCollectionNewLet);

                    if (oldAerodromIdDolazakOfLetCollectionNewLet != null &&
!oldAerodromIdDolazakOfLetCollectionNewLet.equals(aerodrom)) {


oldAerodromIdDolazakOfLetCollectionNewLet.getLetCollection().remove(letCollectionNewLet);

                        oldAerodromIdDolazakOfLetCollectionNewLet =
em.merge(oldAerodromIdDolazakOfLetCollectionNewLet);

                    }

                }

            }

            for (Let letCollection1NewLet : letCollection1New) {

                if (!letCollection1Old.contains(letCollection1NewLet)) {

                    Aerodrom oldAerodromIdPolazakOfLetCollection1NewLet =
letCollection1NewLet.getAerodromIdPolazak();

                    letCollection1NewLet.setAerodromIdPolazak(aerodrom);

                    letCollection1NewLet = em.merge(letCollection1NewLet);

                    if (oldAerodromIdPolazakOfLetCollection1NewLet != null &&
!oldAerodromIdPolazakOfLetCollection1NewLet.equals(aerodrom)) {


oldAerodromIdPolazakOfLetCollection1NewLet.getLetCollection1().remove(letCollection1NewL
et);
```

```
                    oldAerodromIdPolazakOfLetCollection1NewLet =
em.merge(oldAerodromIdPolazakOfLetCollection1NewLet);

                }

            }

        }

        utx.commit();

    } catch (Exception ex) {

        try {

            utx.rollback();

        } catch (Exception re) {

            throw new RollbackFailureException("An error occurred attempting to roll back the
transaction.", re);

        }

        String msg = ex.getLocalizedMessage();

        if (msg == null || msg.length() == 0) {

            Integer id = aerodrom.getId();

            if (findAerodrom(id) == null) {

                throw new NonexistentEntityException("The aerodrom with id " + id + " no longer
exists.");

            }

        }

        throw ex;

    } finally {

        if (em != null) {

            em.close();

        }
```

```
        }

    }


    public void destroy(Integer id) throws IllegalOrphanException, NonexistentEntityException,
RollbackFailureException, Exception {

        EntityManager em = null;

        try {

            utx.begin();

            em = getEntityManager();

            Aerodrom aerodrom;

            try {

                aerodrom = em.getReference(Aerodrom.class, id);

                aerodrom.getId();

            } catch (EntityNotFoundException enfe) {

                throw new NonexistentEntityException("The aerodrom with id " + id + " no longer
exists.", enfe);

            }

            List<String> illegalOrphanMessages = null;

            Collection<Let> letCollectionOrphanCheck = aerodrom.getLetCollection();

            for (Let letCollectionOrphanCheckLet : letCollectionOrphanCheck) {

                if (illegalOrphanMessages == null) {

                    illegalOrphanMessages = new ArrayList<String>();

                }

                illegalOrphanMessages.add("This Aerodrom (" + aerodrom + ") cannot be destroyed
since the Let " + letCollectionOrphanCheckLet + " in its letCollection field has a non-nullable
aerodromIdDolazak field.");
```

```
            }

            Collection<Let> letCollection1OrphanCheck = aerodrom.getLetCollection1();

            for (Let letCollection1OrphanCheckLet : letCollection1OrphanCheck) {

                if (illegalOrphanMessages == null) {

                    illegalOrphanMessages = new ArrayList<String>();

                }

                illegalOrphanMessages.add("This Aerodrom (" + aerodrom + ") cannot be destroyed since the Let " + letCollection1OrphanCheckLet + " in its letCollection1 field has a non-nullable aerodromIdPolazak field.");

            }

            if (illegalOrphanMessages != null) {

                throw new IllegalOrphanException(illegalOrphanMessages);

            }

            em.remove(aerodrom);

            utx.commit();

        } catch (Exception ex) {

            try {

                utx.rollback();

            } catch (Exception re) {

                throw new RollbackFailureException("An error occurred attempting to roll back the transaction.", re);

            }

            throw ex;

        } finally {

            if (em != null) {

                em.close();
```

```
        }
    }
}

public List<Aerodrom> findAerodromEntities() {
    return findAerodromEntities(true, -1, -1);
}

public List<Aerodrom> findAerodromEntities(int maxResults, int firstResult) {
    return findAerodromEntities(false, maxResults, firstResult);
}

private List<Aerodrom> findAerodromEntities(boolean all, int maxResults, int firstResult) {
    EntityManager em = getEntityManager();
    try {
        CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
        cq.select(cq.from(Aerodrom.class));
        Query q = em.createQuery(cq);
        if (!all) {
            q.setMaxResults(maxResults);
            q.setFirstResult(firstResult);
        }
        return q.getResultList();
    } finally {
```

```
        em.close();
    }
}

public Aerodrom findAerodrom(Integer id) {
    EntityManager em = getEntityManager();
    try {
        return em.find(Aerodrom.class, id);
    } finally {
        em.close();
    }
}

public int getAerodromCount() {
    EntityManager em = getEntityManager();
    try {
        CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
        Root<Aerodrom> rt = cq.from(Aerodrom.class);
        cq.select(em.getCriteriaBuilder().count(rt));
        Query q = em.createQuery(cq);
        return ((Long) q.getSingleResult()).intValue();
    } finally {
        em.close();
    }
```

```
    }

}
```

Ostatak klasa nije prikazan zbog obimnosti.

Implementacija aplikacije - JSF stranice

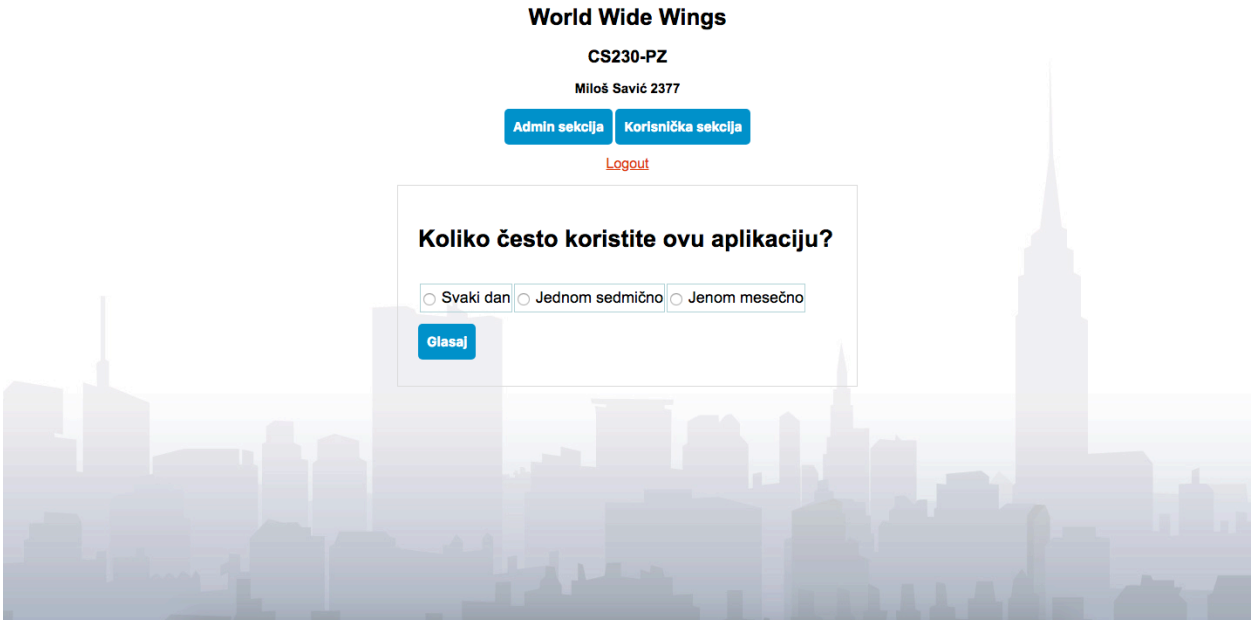
Sledi listing JSF stranice Create.xhtml iz foldera WEB-INF/admin/aerodrom:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">

    <ui:composition template="/template.xhtml">
        <ui:define name="title">
            <h:outputText value=""></h:outputText>
        </ui:define>
        <ui:define name="body">
            <h:panelGroup id="messagePanel" layout="block">
                <h:messages errorStyle="color: red" infoStyle="color: green" layout="table"/>
            </h:panelGroup>
            <h:form>
                <center><h1>Novi aerodrom</h1></center>
                <h:panelGrid columns="2" styleClass="forma">
                    <h:outputLabel value="#{bundle.CreateAerodromLabel_grad}" for="grad" />
                    <h:inputText id="grad" value="#{aerodromController.selected.grad}"
title="#{bundle.CreateAerodromTitle_grad}" required="true"
requiredMessage="#{bundle.CreateAerodromRequiredMessage_grad}"/>
                    <h:outputLabel value="Država: " for="drzava" />
                    <h:inputText id="drzava" value="#{aerodromController.selected.drzava}"
title="#{bundle.CreateAerodromTitle_drzava}" required="true"
requiredMessage="#{bundle.CreateAerodromRequiredMessage_drzava}"/>
                    <h:outputLabel value="#{bundle.CreateAerodromLabel_naziv}" for="naziv"
/>
                    <h:inputText id="naziv" value="#{aerodromController.selected.naziv}"
title="#{bundle.CreateAerodromTitle_naziv}" required="true"
requiredMessage="#{bundle.CreateAerodromRequiredMessage_naziv}"/>
                </h:panelGrid>
                <br />
                <h:panelGrid columns="3">
                <h:commandButton action="#{aerodromController.create}" value="Sačuvaj" />
```

```
            <h:commandButton action="#{aerodromController.prepareList}"
value="Prikaži sve aerodrome" immediate="true"/>
            <h:button outcome="/admin/index"
value="#{bundle.CreateAerodromIndexLink}"/>
            </h:panelGrid>
            <br />
            <br />
        </h:form>
    </ui:define>
</ui:composition>

</html>
```

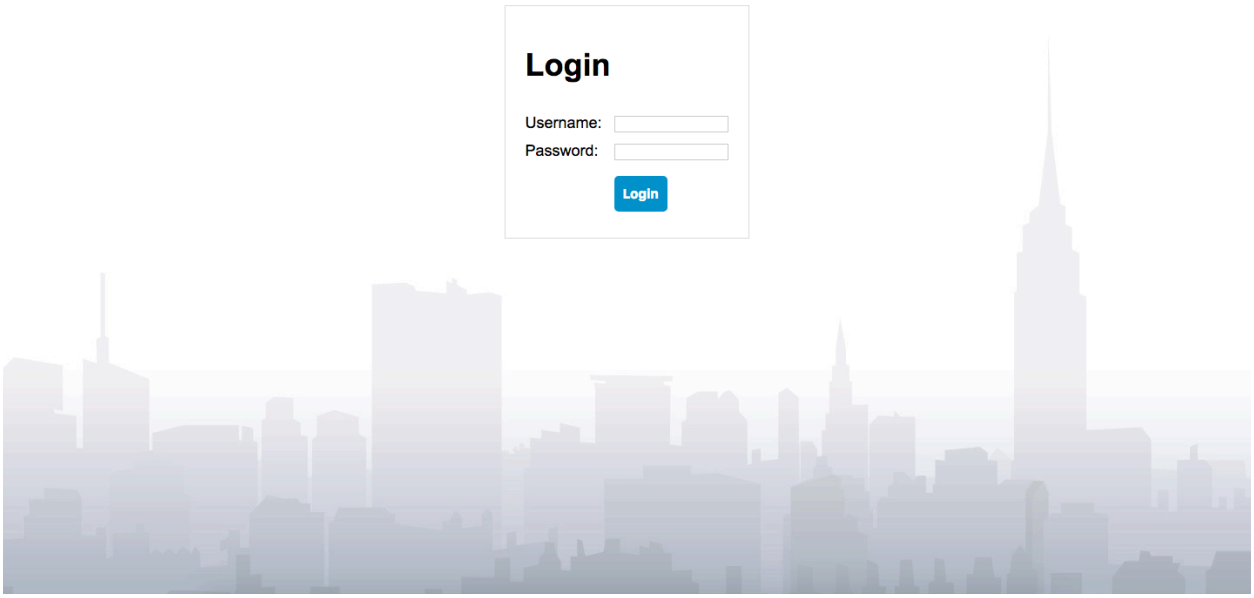
Ostatak listinga nije priložen zbog obimnosti.

Dokumentovanje funkcionalnosti aplikacije



Slika 5. Početna stranica

Početna



Slika 6. Login stranica

List

1..3/3

Id	Grad	Drzava	Naziv	
1	Huangmei	China	Menomonie	View Edit Destroy
2	Khandagayty	Russia	High Crossing	View Edit Destroy
4	Beograd	Srbija	Nikola Tesla	View Edit Destroy

Kreiraj novi aerodrom

JSON prikaz

Index

Slika 7. Lista aerodroma

Slika 8. Pojedinačni prikaz aerodroma

Slika 9. Izmena aerodroma

Slika 10. Kreiranje novog aerodroma


```
[{"drzava":"China","grad":"Huangmei","id":1,"naziv":"Menomonie"
},{"drzava":"Russia","grad":"Khandagayty","id":2,"naziv":"High
Crossing"},
{"drzava":"Srbija","grad":"Beograd","id":4,"naziv":"Nikola
Tesla"}]
```


Slika 11. JSON prikaz

Admin login parametri: username: admin; password: admin