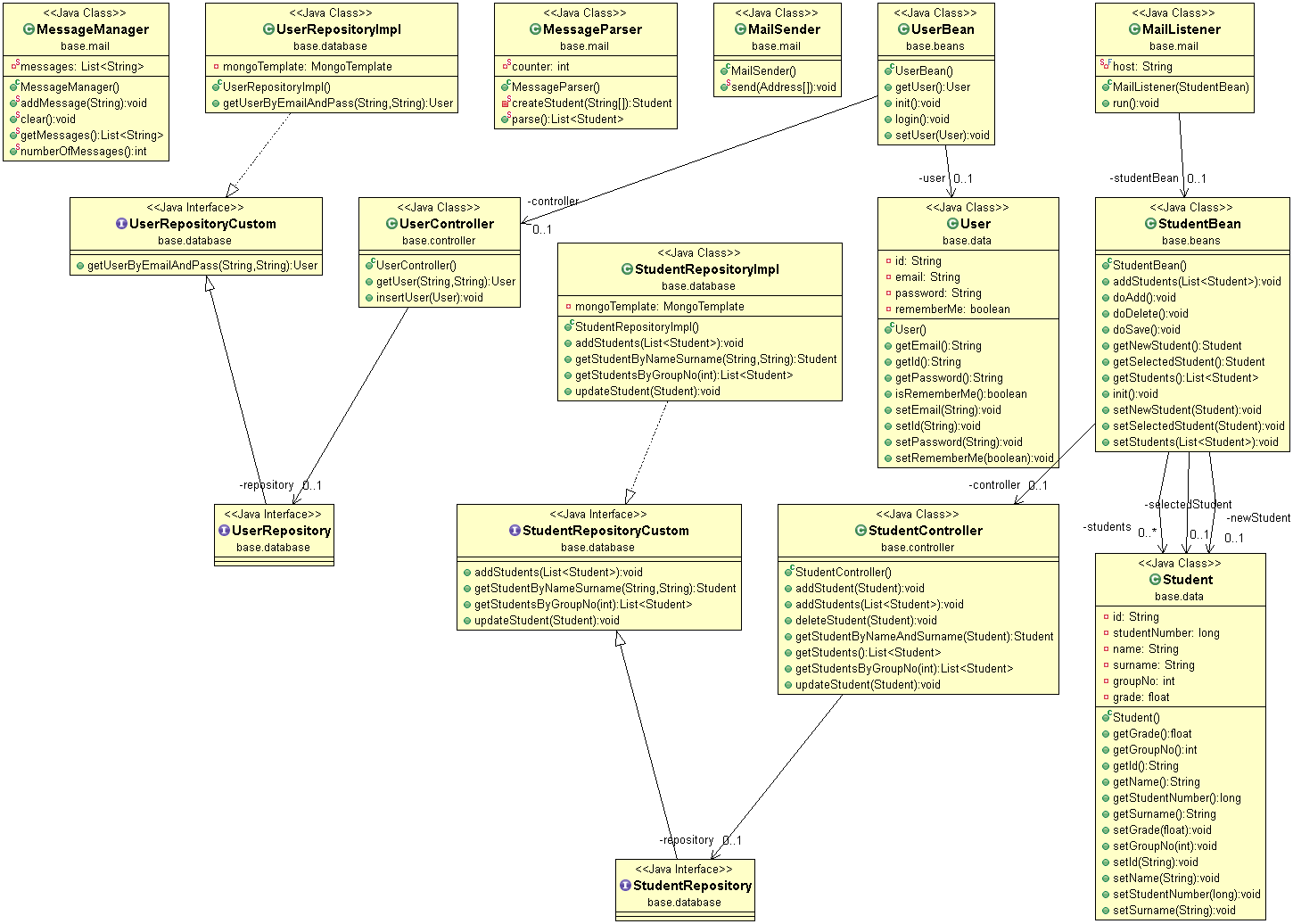
# 1) TASARIM DÖKÜMANI

## 1.1) UML Gösterimi

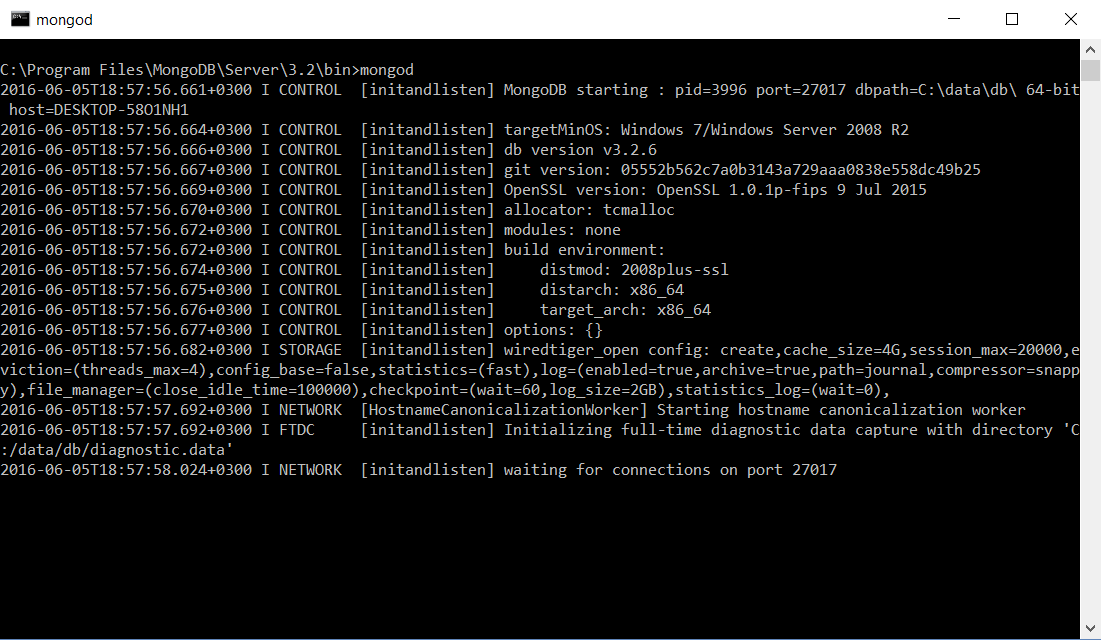


## 1.2) Açıklamalar

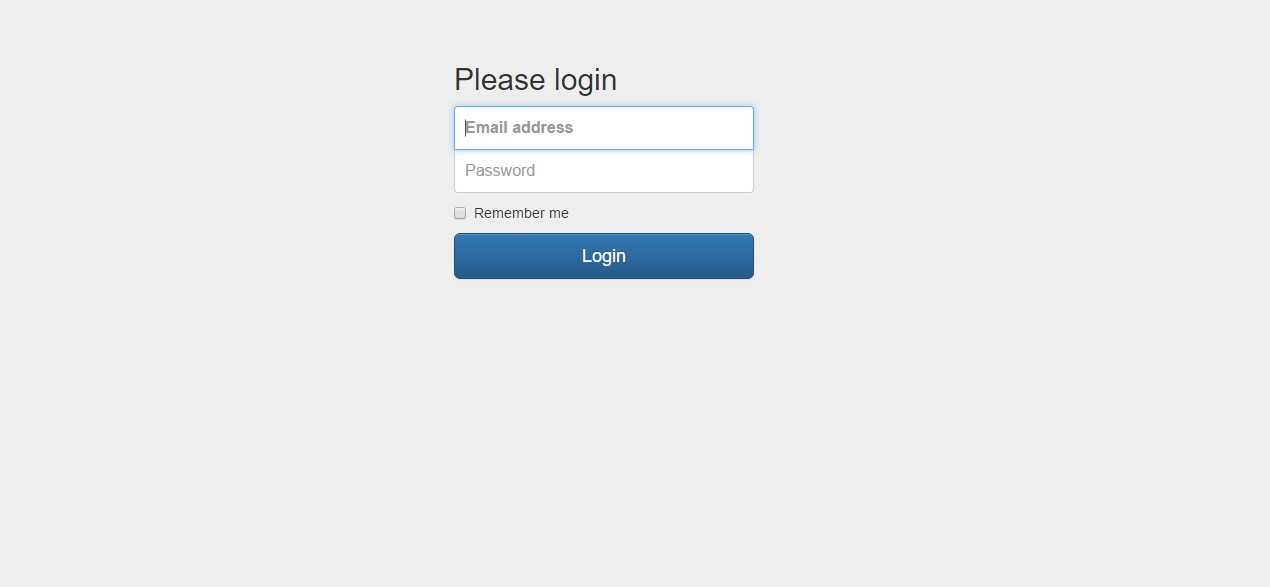
* UserRepository, UserRepositoryCustom ve UserRepositoryImpl sınıfları kısaca kullanıcı veritabanına custom functionality, CRUD özellikleri ve bunların kombine edilmiş halini sunmak için tasarlanmışlardır. Burada genel olarak email ve şifreye göre sorgu yapma özelliği bulunmaktadır.
* StudentRepository, StudentRepositoryCustom ve StudentRepositoryImpl sınıfları kısaca kullanıcı veritabanına custom functionality, CRUD özellikleri ve bunların kombine edilmiş halini sunmak için tasarlanmışlardır. Burada genel olarak grup numarasına göre sorgu yapma, isim ve soyisme göre sorgu yapma, öğrenci koleksiyonu ekleme ve öğrenci güncellemesi yapma özellikleri bulunmaktadır.
* StudentController sınıfı bir servis sınıfıdır. StudentBean ve StudentRepository arasındaki iletişimi sağlar.
* UserController sınıfı bir servis sınıfıdır. UserBean ve UserRepository arasındaki iletişimi sağlar.
* UserBean sınıfı login.xhtml sayfasından alınan verileri saklayan ve login komutunun işlendiği sınıftır. Eğer login başarılı olursa project.xhtml sayfasına yönlendirir.
* StudentBean sınıfı project.xhtml sayfasından alınan verileri saklayan öğrenci eklenmesi, güncellenmesi ve silinmesi işlemlerinin yapıldığı, ayrıca MailListener threadinin tetiklendiği sınıftır.
* MailListener sınıfı imaps protokolü kullanarak admin mailine gelen maillerin alındığı ve parse işlemi için MessageParser sınıfına gönderdikten sonra oluşan yeni öğrencileri StudentBean nesnesi kullanarak veritabanına aktarır.
* MailSender sınıfı smtp protokolü kullanarak maillerin geldiği adreslere otomatik onaylama maili yollayan sınıftır.
* MessageManager sınıfı gelen maillerin içeriklerinin tutulduğu, yönetildiği sınıftır.
* MessageParser sınıfı oluşturulan mesajların parse edilerek Student veri yapılarının oluşturulduğu sınıftır.
* Student ve User sınıfları veritabanına saklanan veri yapılarını tanımlayan sınıflardır.

# 2) KULLANICI DÖKÜMANI VE EKRAN GÖRÜNTÜLERİ

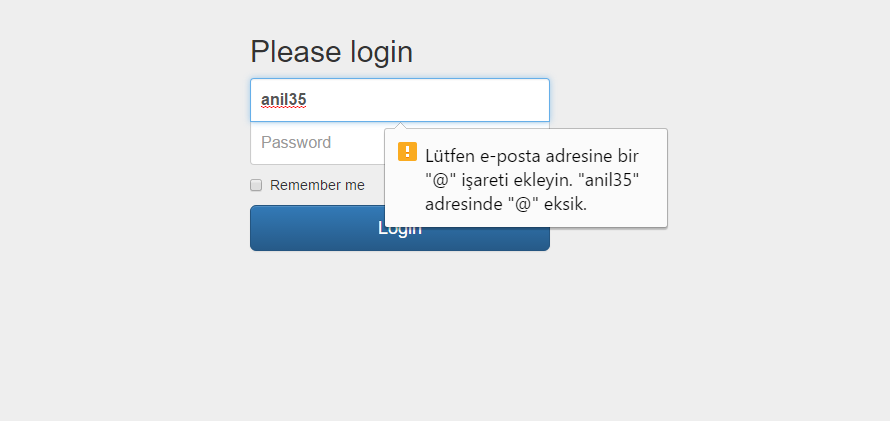
Programın çalıştırılması için ilk olarak MongoDb server ı başlatılmalıdır. Bunun için MongoDb nin kurulu olduğu dosya yolundan komut satırı çalıştırılarak “mongod” komutu verilmesiyle server başlatılır.

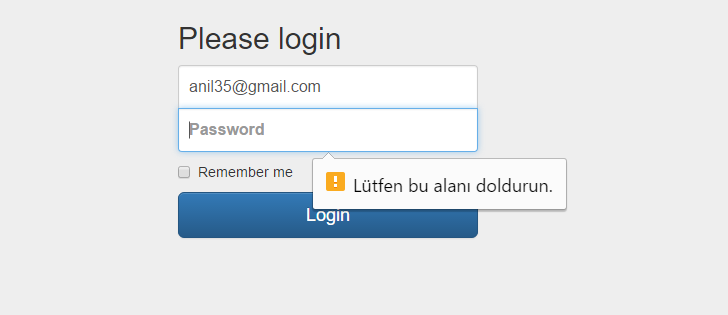


Programın çalıştırılmasıyla birlikte ilk olarak Login ekranı karşımıza gelir.

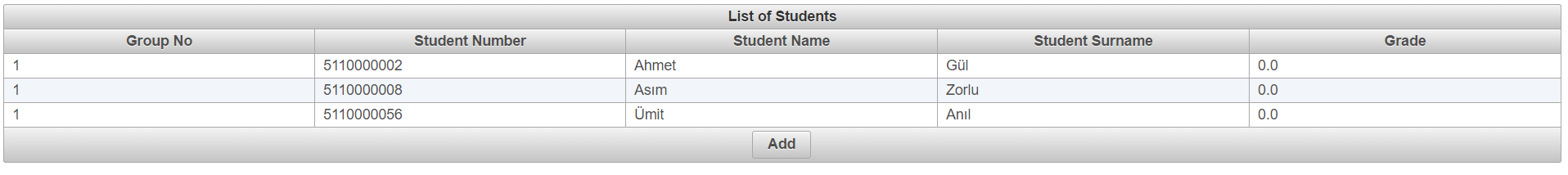


Burada email adresi doğru formatta yazılmazsa veya alanlardan biri boş bırakılırsa hata mesajı gösterilir.

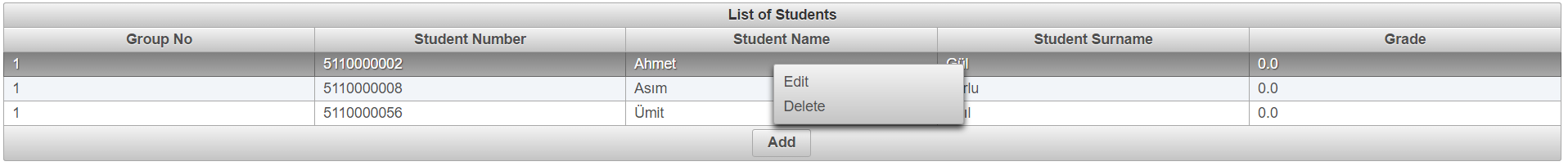




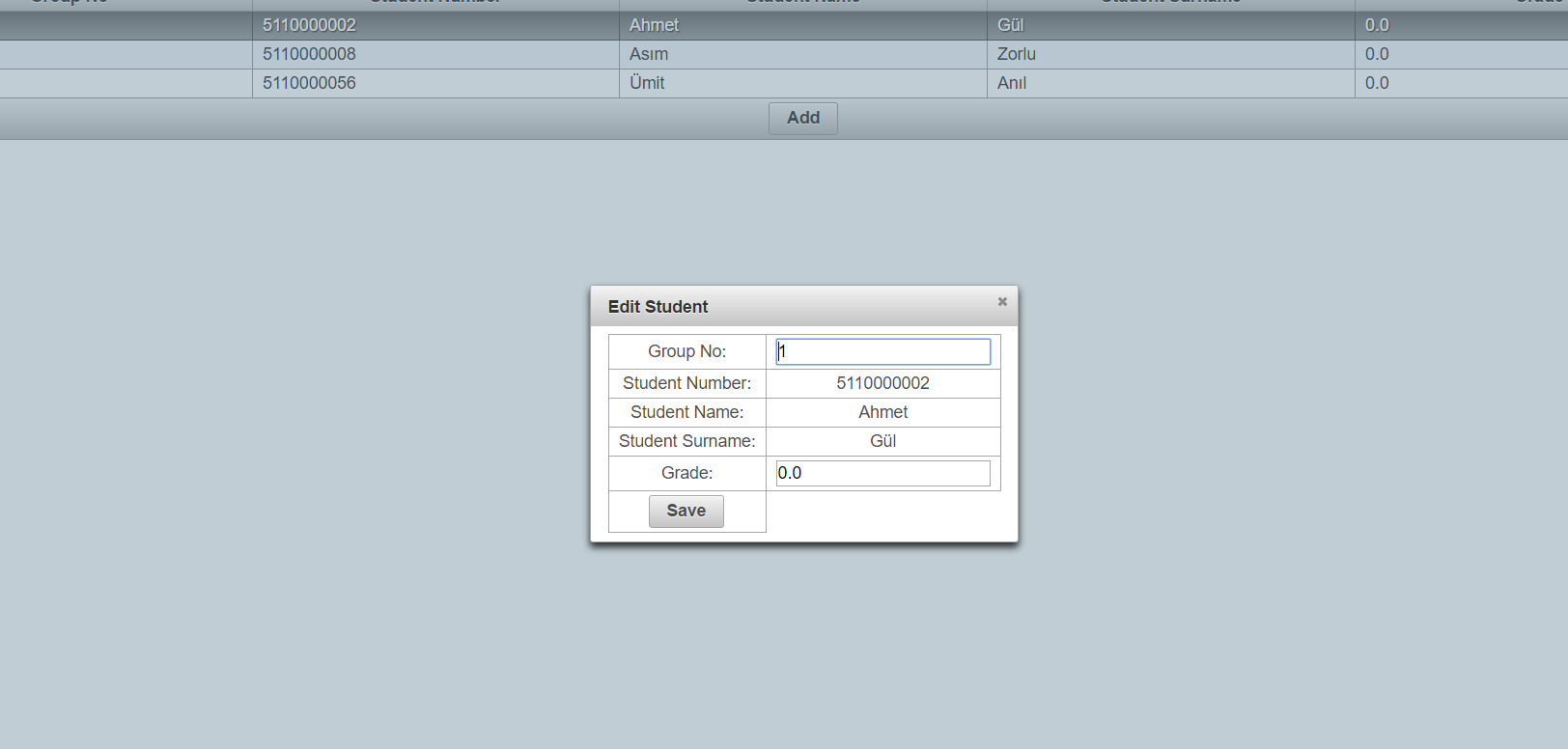
Doğru bir şekilde giriş yapıldığında karşımıza öğrenciler listesi gelir.



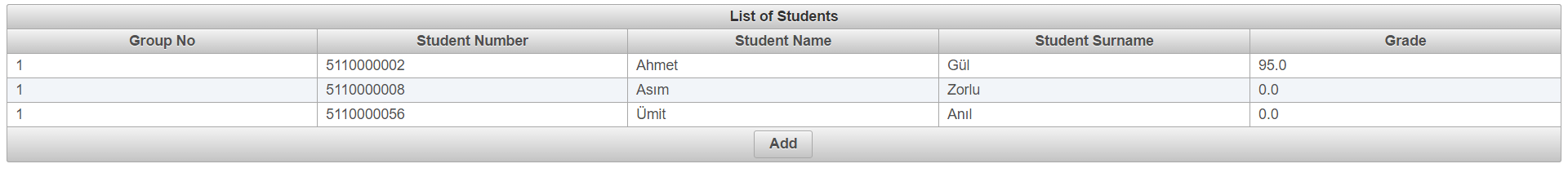
Burada interaction ları kullanmak için örneğin silme ve güncelleme işlemleri istenilen öğrenciye sağ tıklanarak çıkan context menu den seçilir.



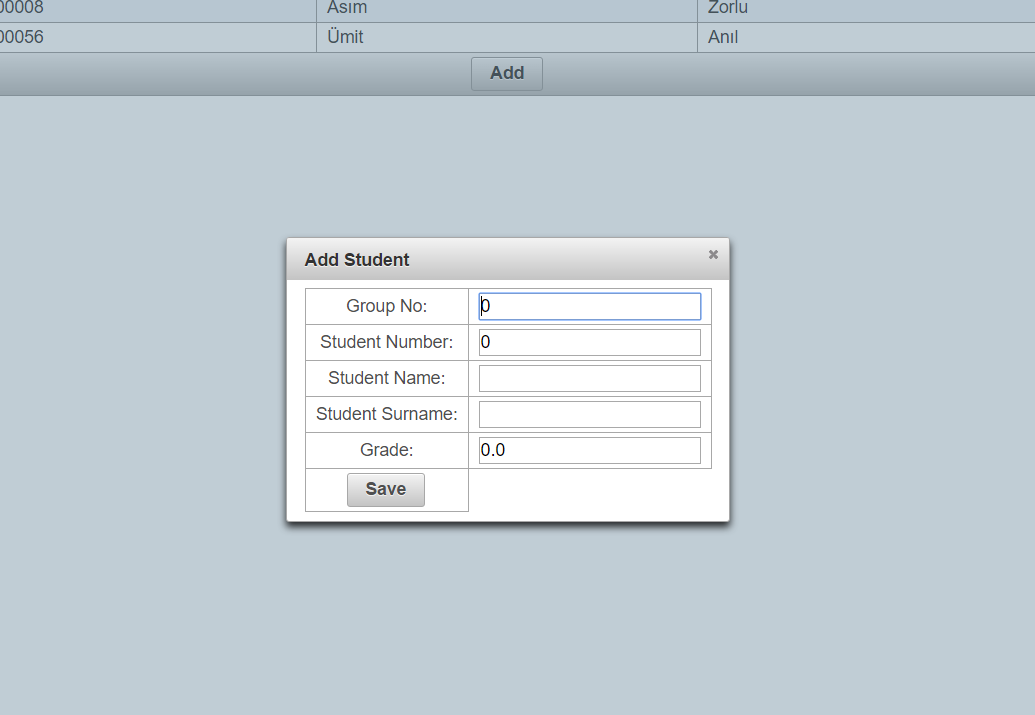
Örnek olarak Edit komutu çalıştırılır.



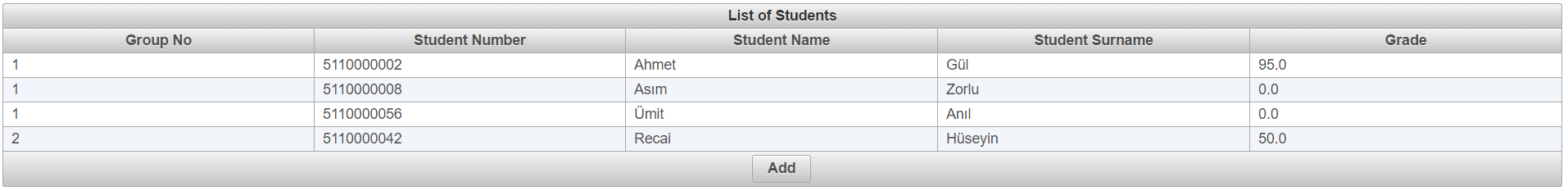
Karşımıza dialog ekranı gelir. Burada edit etmek istediğimiz öğrencinin grup numarası ve notunu değiştirebiliriz. Daha sonra Save butonuyla listeye güncellenmiş halini ekleyebiliriz.



Add komutunu kullanarak listeye sıfırdan yeni bir öğrenci ekleyebiliriz. Bu butona bastığımızda ilk olarak karşımıza dialog ekranı gelir.



Bu ekranda grup numarası, öğrenci numarası, öğrenci adı, öğrenci soyadı ve notu seçeneklerini girerek Save butonuyla listemize yeni bir öğrenci eklemiş oluruz.



# 3) KAYNAK KOD

package base.beans;

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.Executors;

import java.util.concurrent.ScheduledExecutorService;

import java.util.concurrent.TimeUnit;

import javax.annotation.PostConstruct;

import javax.faces.bean.SessionScoped;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import base.controller.StudentController;

import base.data.Student;

import base.mail.MailListener;

/\*\*

\* This class is responsible for holding the given input from project.xhtml view and transmit those

\* data to controller.

\*

\* @author anıl öztürk

\* @author ahmet gül

\* @author asım zorlu

\*/

@Component

@SessionScoped

public class StudentBean {

@Autowired

private StudentController controller;

private List<Student> students;

private Student selectedStudent;

private Student newStudent = new Student();

public void addStudents(final List<Student> newStudents) {

this.controller.addStudents(newStudents);

this.students.addAll(newStudents);

}

public void doAdd() {

// System.out.println("Add");

this.controller.addStudent(this.newStudent);

this.students.add(this.newStudent);

this.newStudent = new Student();

}

public void doDelete() {

// System.out.println("Delete");

this.controller.deleteStudent(this.selectedStudent);

this.students.remove(this.selectedStudent);

this.selectedStudent = null;

}

public void doSave() {

// System.out.println("Save");

this.controller.updateStudent(this.selectedStudent);

this.students.set(this.students.indexOf(this.selectedStudent), this.selectedStudent);

this.selectedStudent = null;

}

public Student getNewStudent() {

return this.newStudent;

}

public Student getSelectedStudent() {

return this.selectedStudent;

}

public List<Student> getStudents() {

return this.students;

}

/\*\*

\* This method will be call just once to trigger the mail listener thread and load table from

\* database.

\*

\*/

@PostConstruct

public void init() {

// we start to listen mail at here, every 2 minutes it will run.

final ScheduledExecutorService exec = Executors.newSingleThreadScheduledExecutor();

exec.scheduleAtFixedRate(new MailListener(this), 0, 2, TimeUnit.MINUTES);

this.students = new ArrayList<Student>();

this.students.addAll(this.controller.getStudents());

// System.out.println("Students have been initialized.");

}

public void setNewStudent(final Student newStudent) {

this.newStudent = newStudent;

}

public void setSelectedStudent(final Student selectedStudent) {

this.selectedStudent = selectedStudent;

}

public void setStudents(final List<Student> students) {

this.students = students;

}

}

package base.beans;

import java.io.IOException;

import javax.annotation.PostConstruct;

import javax.faces.application.FacesMessage;

import javax.faces.bean.SessionScoped;

import javax.faces.context.FacesContext;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import base.controller.UserController;

import base.data.User;

import base.util.DbUtil;

import base.util.PageUtil;

/\*\*

\* This class is responsible for holding the given input from login.xhtml view and transmit those

\* data to controller.

\*

\* @author anıl öztürk

\* @author ahmet gül

\* @author asım zorlu

\*/

@Component

@SessionScoped

public class UserBean {

@Autowired

private UserController controller;

private User user;

public User getUser() {

return this.user;

}

/\*\*

\* This method will be call just once and initialize the User database with admin.

\*

\*/

@PostConstruct

public void init() {

this.user = new User();

final User admin = new User();

admin.setEmail(DbUtil.USER);

admin.setPassword(DbUtil.PASS);

this.controller.insertUser(admin);

}

/\*\*

\* This method is responsible for check the user which return from database related to given

\* information and if succeed redirect to project page.

\*

\*/

public void login() {

final User returnedUser =

this.controller.getUser(this.user.getEmail(), this.user.getPassword());

if (returnedUser == null) {

FacesContext.getCurrentInstance().addMessage("loginForm:loginButton",

new FacesMessage(PageUtil.LOGIN\_ERROR));

} else {

try {

FacesContext.getCurrentInstance().getExternalContext().redirect(PageUtil.PROJECT\_PAGE);

} catch (final IOException e) {

e.printStackTrace();

}

}

}

public void setUser(final User user) {

this.user = user;

}

}

package base.controller;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import base.data.Student;

import base.database.StudentRepository;

/\*\*

\* This class is responsible for communication with student repository.

\*

\* @author anıl öztürk

\* @author ahmet gül

\* @author asım zorlu

\*/

@Service

public class StudentController {

@Autowired

private StudentRepository repository;

public void addStudent(final Student student) {

// System.out.println("Student added.");

this.repository.save(student);

}

public void addStudents(final List<Student> students) {

// System.out.println("Students added.");

this.repository.addStudents(students);

}

public void deleteStudent(final Student student) {

this.repository.delete(student);

}

public Student getStudentByNameAndSurname(final Student student) {

return this.repository.getStudentByNameSurname(student.getName(), student.getSurname());

}

public List<Student> getStudents() {

final List<Student> students = new ArrayList<Student>();

final Iterator<Student> iterator = this.repository.findAll().iterator();

while (iterator.hasNext()) {

final Student student = iterator.next();

students.add(student);

}

return students;

}

public List<Student> getStudentsByGroupNo(final int groupNo) {

return this.repository.getStudentsByGroupNo(groupNo);

}

public void updateStudent(final Student newlyUpdatedStudent) {

this.repository.delete(newlyUpdatedStudent.getId());

this.repository.save(newlyUpdatedStudent);

}

}

package base.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import base.data.User;

import base.database.UserRepository;

/\*\*

\* This class is responsible for communication with user repository.

\*

\* @author anıl öztürk

\* @author ahmet gül

\* @author asım zorlu

\*/

@Service

public class UserController {

@Autowired

private UserRepository repository;

public User getUser(final String email, final String pass) {

return this.repository.getUserByEmailAndPass(email, pass);

}

public void insertUser(final User user) {

this.repository.save(user);

}

}

package base.data;

/\*\*

\* Defines Student data structure.

\*

\* @author anıl öztürk

\* @author ahmet gül

\* @author asım zorlu

\*/

public class Student {

private String id;

private long studentNumber;

private String name;

private String surname;

private int groupNo;

private float grade;

public float getGrade() {

return this.grade;

}

public int getGroupNo() {

return this.groupNo;

}

public String getId() {

return this.id;

}

public String getName() {

return this.name;

}

public long getStudentNumber() {

return this.studentNumber;

}

public String getSurname() {

return this.surname;

}

public void setGrade(final float grade) {

this.grade = grade;

}

public void setGroupNo(final int groupNo) {

this.groupNo = groupNo;

}

public void setId(final String id) {

this.id = id;

}

public void setName(final String name) {

this.name = name;

}

public void setStudentNumber(final long studentNumber) {

this.studentNumber = studentNumber;

}

public void setSurname(final String surname) {

this.surname = surname;

}

}

package base.data;

/\*\*

\* Defines User data structure.

\*

\* @author anıl öztürk

\* @author ahmet gül

\* @author asım zorlu

\*/

public class User {

private String id;

private String email;

private String password;

private boolean rememberMe;

public String getEmail() {

return this.email;

}

public String getId() {

return this.id;

}

public String getPassword() {

return this.password;

}

public boolean isRememberMe() {

return this.rememberMe;

}

public void setEmail(final String email) {

this.email = email;

}

public void setId(final String id) {

this.id = id;

}

public void setPassword(final String password) {

this.password = password;

}

public void setRememberMe(final boolean rememberMe) {

this.rememberMe = rememberMe;

}

}

package base.database;

import org.springframework.data.repository.CrudRepository;

import base.data.Student;

/\*\*

\* Combines the CRUD and custom functionality and makes it available to clients.

\*

\* @author anıl öztürk

\* @author ahmet gül

\* @author asım zorlu

\*/

public interface StudentRepository

extends CrudRepository<Student, String>, StudentRepositoryCustom {

}

package base.database;

import java.util.List;

import base.data.Student;

/\*\*

\* To enrich a repository with custom functionality.

\*

\* @author anıl öztürk

\* @author ahmet gül

\* @author asım zorlu

\*/

public interface StudentRepositoryCustom {

public void addStudents(List<Student> students);

public Student getStudentByNameSurname(String name, String surname);

public List<Student> getStudentsByGroupNo(int groupNo);

public void updateStudent(Student updated);

}

package base.database;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.mongodb.core.MongoTemplate;

import org.springframework.data.mongodb.core.query.Criteria;

import org.springframework.data.mongodb.core.query.Query;

import org.springframework.data.mongodb.core.query.Update;

import base.data.Student;

import base.util.DbUtil;

/\*\*

\* This implementation uses MongoTemplate to achieve the database operations.

\*

\* @author anıl öztürk

\* @author ahmet gül

\* @author asım zorlu

\*/

public class StudentRepositoryImpl implements StudentRepositoryCustom {

@Autowired

private MongoTemplate mongoTemplate;

@Override

public void addStudents(final List<Student> students) {

// System.out.println("In database.");

this.mongoTemplate.insert(students, DbUtil.PROJ\_COLL);

}

@Override

public Student getStudentByNameSurname(final String name, final String surname) {

final Query query = new Query();

final Criteria criteria = Criteria.where("name").is(name).and("surname").is(surname);

query.addCriteria(criteria);

return this.mongoTemplate.findOne(query, Student.class, DbUtil.PROJ\_COLL);

}

@Override

public List<Student> getStudentsByGroupNo(final int groupNo) {

final Query query = new Query();

final Criteria criteria = Criteria.where("groupNo").is(groupNo);

query.addCriteria(criteria);

return this.mongoTemplate.find(query, Student.class, DbUtil.PROJ\_COLL);

}

@Override

public void updateStudent(final Student updated) {

final Query query = new Query();

final Criteria criteria = Criteria.where("studentNumber").is(updated.getStudentNumber());

query.addCriteria(criteria);

final Update update = new Update();

update.set("name", updated.getName());

update.set("surname", updated.getSurname());

update.set("grade", updated.getGrade());

update.set("groupNo", updated.getGroupNo());

this.mongoTemplate.updateFirst(query, update, DbUtil.PROJ\_COLL);

}

}

package base.database;

import org.springframework.data.repository.CrudRepository;

import base.data.User;

/\*\*

\* Combines the CRUD and custom functionality and makes it available to clients.

\*

\* @author anıl öztürk

\* @author ahmet gül

\* @author asım zorlu

\*/

public interface UserRepository extends CrudRepository<User, String>, UserRepositoryCustom {

}

package base.database;

import base.data.User;

/\*\*

\* To enrich a repository with custom functionality.

\*

\* @author anıl öztürk

\* @author ahmet gül

\* @author asım zorlu

\*/

public interface UserRepositoryCustom {

public User getUserByEmailAndPass(String email, String pass);

}

package base.database;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.mongodb.core.MongoTemplate;

import org.springframework.data.mongodb.core.query.Criteria;

import org.springframework.data.mongodb.core.query.Query;

import base.data.User;

import base.util.DbUtil;

/\*\*

\* This implementation uses MongoTemplate to achieve the database operations.

\*

\* @author anıl öztürk

\* @author ahmet gül

\* @author asım zorlu

\*/

public class UserRepositoryImpl implements UserRepositoryCustom {

@Autowired

private MongoTemplate mongoTemplate;

@Override

public User getUserByEmailAndPass(final String email, final String pass) {

final Query query = new Query();

final Criteria criteria = Criteria.where("email").is(email).and("password").is(pass);

query.addCriteria(criteria);

return this.mongoTemplate.findOne(query, User.class, DbUtil.USER\_COLL);

}

}

package base.mail;

import java.io.IOException;

import java.util.List;

import java.util.Properties;

import javax.mail.Flags;

import javax.mail.Folder;

import javax.mail.Message;

import javax.mail.MessagingException;

import javax.mail.Multipart;

import javax.mail.Session;

import javax.mail.Store;

import javax.mail.internet.MimeBodyPart;

import javax.mail.search.FlagTerm;

import base.beans.StudentBean;

import base.data.Student;

import base.util.DbUtil;

/\*\*

\* It takes studentBean for add newly parsed students to database, uses 'imaps' protocol.

\*

\* @author anıl öztürk

\* @author ahmet gül

\* @author asım zorlu

\*/

public class MailListener implements Runnable {

private static final String host = "imap.gmail.com";

private final StudentBean studentBean;

public MailListener(final StudentBean studentBean) {

this.studentBean = studentBean;

}

@Override

public void run() {

try {

// create properties field

final Properties properties = new Properties();

properties.put("mail.store.protocol", "imaps");

final Session emailSession = Session.getDefaultInstance(properties);

// create the POP3 store object and connect with the pop server

final Store store = emailSession.getStore("imaps");

store.connect(host, DbUtil.USER, DbUtil.PASS);

// create the folder object and open it

final Folder emailFolder = store.getFolder("INBOX");

emailFolder.open(Folder.READ\_WRITE);

final Flags seen = new Flags(Flags.Flag.SEEN);

final FlagTerm unseenFlagTerm = new FlagTerm(seen, false);

final Message[] messages = emailFolder.search(unseenFlagTerm);

System.out.println("messages.length---" + messages.length);

for (int i = 0, n = messages.length; i < n; i++) {

final Message message = messages[i];

System.out.println("---------------------------------");

System.out.println("Email Number " + (i + 1));

System.out.println("Subject: " + message.getSubject());

System.out.println("From: " + message.getFrom()[0]);

final StringBuilder builder = new StringBuilder();

if (message.getContentType().contains("TEXT/PLAIN")) {

final Object content = message.getContent();

if (content != null) {

builder.append(content);

}

} else if (message.getContentType().contains("multipart")) {

final Multipart mp = (Multipart) message.getContent();

final int numParts = mp.getCount();

for (int count = 0; count < numParts; count++) {

final MimeBodyPart part = (MimeBodyPart) mp.getBodyPart(count);

final String content = part.getContent().toString();

if (part.getContentType().contains("TEXT/PLAIN")) {

builder.append(content);

}

}

}

MessageManager.addMessage(builder.toString());

// send confirmation mail.

MailSender.send(message.getFrom());

}

// close the store and folder objects

emailFolder.close(false);

store.close();

} catch (final MessagingException | IOException e) {

e.printStackTrace();

}

final List<Student> newStudents = MessageParser.parse();

// System.out.println("Parsed.");

this.studentBean.addStudents(newStudents);

// System.out.println("Going out.");

}

}

package base.mail;

import java.util.Properties;

import javax.mail.Address;

import javax.mail.Message;

import javax.mail.MessagingException;

import javax.mail.Session;

import javax.mail.Transport;

import javax.mail.internet.InternetAddress;

import javax.mail.internet.MimeMessage;

import base.util.DbUtil;

/\*\*

\* This class sends a confirmation mail to students which have been sent to group information. Uses

\* 'smtp' protocol.

\*

\* @author anıl öztürk

\* @author ahmet gül

\* @author asım zorlu

\*/

public class MailSender {

public static void send(final Address[] to) {

// Sender's email ID needs to be mentioned

final String from = DbUtil.USER;

final String password = DbUtil.PASS;// change accordingly

// Assuming you are sending email through smtp.gmail.com

final String host = "smtp.gmail.com";

final String port = "465";

final Properties props = System.getProperties();

props.put("mail.smtp.user", from);

props.put("mail.smtp.host", host);

props.put("mail.smtp.port", port);

props.put("mail.smtp.starttls.enable", "true");

props.put("mail.smtp.debug", "true");

props.put("mail.smtp.auth", "true");

props.put("mail.smtp.socketFactory.port", port);

props.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");

props.put("mail.smtp.socketFactory.fallback", "false");

// Get the Session object.

final Session session = Session.getDefaultInstance(props);

session.setDebug(true);

try {

// Create a default MimeMessage object.

final Message message = new MimeMessage(session);

// Set From: header field of the header.

message.setFrom(new InternetAddress(from));

// Set To: header field of the header.

message.setRecipients(Message.RecipientType.TO, to);

// Set Subject: header field

message.setSubject("Sunucu Yazılım Dersi Projesi Hk.");

// Now set the actual message

message.setText("Yolladığınız grup bilgileri alınmıştır.\nBu bir otomatik maildir.");

// Send message

final Transport transport = session.getTransport("smtps");

transport.connect(host, 465, from, password);

transport.sendMessage(message, message.getAllRecipients());

transport.close();

System.out.println("Sent message successfully....");

} catch (final MessagingException e) {

throw new RuntimeException(e);

}

}

}

package base.mail;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* This class like a container for messages. It holds the messages and give them for use.

\*

\* @author anıl öztürk

\* @author ahmet gül

\* @author asım zorlu

\*/

public class MessageManager {

private static List<String> messages = new ArrayList<String>();

public static void addMessage(final String message) {

messages.add(message);

}

public static void clear() {

messages = new ArrayList<String>();

}

public static List<String> getMessages() {

return messages;

}

public static int numberOfMessages() {

return messages.size();

}

}

package base.mail;

import java.util.ArrayList;

import java.util.List;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import base.data.Student;

/\*\*

\* This class parses mails and construct Student data structure from them.

\*

\* @author anıl öztürk

\* @author ahmet gül

\* @author asım zorlu

\*/

public class MessageParser {

private static int counter = 0; // it will be used as group no

private static Student createStudent(final String[] info) {

final Student student = new Student();

student.setStudentNumber(Long.valueOf(info[0]));

final String[] names = info[1].split(" ");

student.setName(names[0].trim());

student.setSurname(names[1].trim());

student.setGroupNo(counter);

return student;

}

public static List<Student> parse() {

final List<Student> students = new ArrayList<Student>();

final List<String> messages = MessageManager.getMessages();

for (final String message : messages) {

counter++;

final Pattern pattern = Pattern.compile("[0-9]+-[\\p{L}]+\\s+[\\p{L}]+");

final Matcher matcher = pattern.matcher(message);

final ArrayList<String> matches = new ArrayList<String>();

while (matcher.find()) {

matches.add(matcher.group());

}

for (final String match : matches) {

final String[] info = match.split("-");

students.add(createStudent(info));

}

}

MessageManager.clear();

return students;

}

}

package base.util;

public class DbUtil {

public static final String USER\_COLL = "user";

public static final String PROJ\_COLL = "project";

public static final String USER = "sunucuyazilim12@gmail.com";

public static final String PASS = "123456A!";

}

package base.util;

public class PageUtil {

public static final String PROJECT\_PAGE = "project.xhtml";

public static final String LOGIN\_ERROR =

"Your email and/or password is incorrect. Please try again";

}

<?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:h="http://xmlns.jcp.org/jsf/html"

xmlns:f="http://xmlns.jcp.org/jsf/core"

xmlns:ui="http://xmlns.jcp.org/jsf/facelets"

xmlns:p="http://primefaces.org/ui">

<h:head>

<title>Login</title>

<!-- Latest compiled and minified CSS -->

<link rel="stylesheet"

href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"></link>

<link href="login.css" rel="stylesheet" />

<!-- Optional theme -->

<link rel="stylesheet"

href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap-theme.min.css"></link>

<!-- Latest compiled and minified JavaScript -->

<script

src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></script>

</h:head>

<h:body>

<div class="container">

<h:form id="loginForm">

<f:passThroughAttribute name="class" value="form-login"></f:passThroughAttribute>

<h2 class="form-login-heading">Please login</h2>

<h:outputLabel for="inputEmail">Email address

<f:passThroughAttribute name="class" value="sr-only"></f:passThroughAttribute>

</h:outputLabel>

<p:inputText id="inputEmail" value="#{userBean.user.email}" type="email"

placeholder="Email address">

<f:passThroughAttribute name="class" value="form-control"></f:passThroughAttribute>

<f:passThroughAttribute name="required" value="true"></f:passThroughAttribute>

<f:passThroughAttribute name="autofocus" value="true"></f:passThroughAttribute>

</p:inputText>

<h:outputLabel for="inputPassword">Password

<f:passThroughAttribute name="class" value="sr-only"></f:passThroughAttribute>

</h:outputLabel>

<p:password id="inputPassword" value="#{userBean.user.password}"

placeholder="Password">

<f:passThroughAttribute name="class" value="form-control"></f:passThroughAttribute>

<f:passThroughAttribute name="type" value="password"></f:passThroughAttribute>

<f:passThroughAttribute name="required" value="true"></f:passThroughAttribute>

</p:password>

<div class="checkbox">

<label> <h:selectBooleanCheckbox value="#{userBean.user.rememberMe}"></h:selectBooleanCheckbox>

Remember me

</label>

</div>

<p:commandButton id="loginButton" type="submit" value="Login"

ajax="false" action="#{userBean.login}">

<f:passThroughAttribute name="class"

value="btn btn-lg btn-primary btn-block"></f:passThroughAttribute>

</p:commandButton>

<h:message for="loginButton" style="color:red"></h:message>

</h:form>

</div>

<!-- /container -->

</h:body>

</html>

body {

padding-top: 40px;

padding-bottom: 40px;

background-color: #eee;

}

.form-login {

max-width: 330px;

padding: 15px;

margin: 0 auto;

}

.form-login .form-login-heading, .form-login .checkbox {

margin-bottom: 10px;

}

.form-login .checkbox {

font-weight: normal;

}

.form-login .form-control {

position: relative;

height: auto;

-webkit-box-sizing: border-box;

-moz-box-sizing: border-box;

box-sizing: border-box;

padding: 10px;

font-size: 16px;

}

.form-login .form-control:focus {

z-index: 2;

}

.form-login input[type="email"] {

margin-bottom: -1px;

border-bottom-right-radius: 0;

border-bottom-left-radius: 0;

}

.form-login input[type="password"] {

margin-bottom: 10px;

border-top-left-radius: 0;

border-top-right-radius: 0;

}

<?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:h="http://xmlns.jcp.org/jsf/html"

xmlns:f="http://xmlns.jcp.org/jsf/core"

xmlns:ui="http://xmlns.jcp.org/jsf/facelets"

xmlns:p="http://primefaces.org/ui">

<h:head>

<title>Project Page</title>

</h:head>

<h:form id="project">

<p:contextMenu for="table">

<p:menuitem value="Edit" update="studentInfo"

oncomplete="PF('editDialog').show()"></p:menuitem>

<p:menuitem value="Delete" update="table"

actionListener="#{studentBean.doDelete}"></p:menuitem>

</p:contextMenu>

<p:dataTable id="table" value="#{studentBean.students}" var="student"

rowKey="#{student.id}" selectionMode="single"

selection="#{studentBean.selectedStudent}">

<f:facet name="header">

List of Students

</f:facet>

<p:column headerText="Group No">

<p:cellEditor>

<f:facet name="output">

<h:outputText value="#{student.groupNo}" />

</f:facet>

<f:facet name="input">

<p:inputText value="#{student.groupNo}" style="width:100%" />

</f:facet>

</p:cellEditor>

</p:column>

<p:column headerText="Student Number">

<p:cellEditor>

<f:facet name="output">

<h:outputText value="#{student.studentNumber}" />

</f:facet>

<f:facet name="input">

<p:inputText value="#{student.studentNumber}" style="width:100%" />

</f:facet>

</p:cellEditor>

</p:column>

<p:column headerText="Student Name">

<p:cellEditor>

<f:facet name="output">

<h:outputText value="#{student.name}" />

</f:facet>

<f:facet name="input">

<p:inputText value="#{student.name}" style="width:100%" />

</f:facet>

</p:cellEditor>

</p:column>

<p:column headerText="Student Surname">

<p:cellEditor>

<f:facet name="output">

<h:outputText value="#{student.surname}" />

</f:facet>

<f:facet name="input">

<p:inputText value="#{student.surname}" style="width:100%" />

</f:facet>

</p:cellEditor>

</p:column>

<p:column headerText="Grade">

<p:cellEditor>

<f:facet name="output">

<h:outputText value="#{student.grade}" />

</f:facet>

<f:facet name="input">

<p:inputText value="#{student.grade}" style="width:100%" />

</f:facet>

</p:cellEditor>

</p:column>

<f:facet name="footer">

<p:commandButton id="addBttn" value="Add" update=":project:addInfo"

oncomplete="PF('addDialog').show()" />

</f:facet>

</p:dataTable>

<p:dialog header="Edit Student" widgetVar="editDialog" modal="true"

resizable="false">

<p:outputPanel id="studentInfo" style="text-align:center;">

<p:panelGrid columns="2"

rendered="#{not empty studentBean.selectedStudent}"

columnClasses="label,value">

<h:outputText value="Group No:" />

<h:inputText value="#{studentBean.selectedStudent.groupNo}" />

<h:outputText value="Student Number:" />

<h:outputText value="#{studentBean.selectedStudent.studentNumber}" />

<h:outputText value="Student Name:" />

<h:outputText value="#{studentBean.selectedStudent.name}" />

<h:outputText value="Student Surname:" />

<h:outputText value="#{studentBean.selectedStudent.surname}" />

<h:outputText value="Grade:" />

<h:inputText value="#{studentBean.selectedStudent.grade}" />

<p:commandButton id="saveBttn" value="Save" ajax="false"

action="#{studentBean.doSave}" update="project:table"></p:commandButton>

</p:panelGrid>

</p:outputPanel>

</p:dialog>

<p:dialog header="Add Student" widgetVar="addDialog" modal="true"

resizable="false">

<p:outputPanel id="addInfo" style="text-align:center;">

<p:panelGrid columns="2" columnClasses="label,value">

<h:outputText value="Group No:" />

<h:inputText value="#{studentBean.newStudent.groupNo}" />

<h:outputText value="Student Number:" />

<h:inputText value="#{studentBean.newStudent.studentNumber}" />

<h:outputText value="Student Name:" />

<h:inputText value="#{studentBean.newStudent.name}" />

<h:outputText value="Student Surname:" />

<h:inputText value="#{studentBean.newStudent.surname}" />

<h:outputText value="Grade:" />

<h:inputText value="#{studentBean.newStudent.grade}" />

<p:commandButton id="saveAddBttn" value="Save" ajax="false"

action="#{studentBean.doAdd}" update="project:table"></p:commandButton>

</p:panelGrid>

</p:outputPanel>

</p:dialog>

</h:form>

</html>