

# **School Project Application Report**

Στο PDF αυτό, παρουσιάζεται το Project το οποίο κληθήκαμε να υλοποιήσουμε στα πλαίσια του προγράμματος του reGeneration “Τεχνικών Εφαρμογών Πληροφορικής” σε συνεργασία με τον ΣΕΒ. Θα αναλυθούν τα εργαλεία, οι μέθοδοι και ο τρόπος με τον οποίο υλοποιήθηκε το Project.

## **Ανάλυση Προβλήματος**

Θέλουμε να υλοποιήσουμε μια εφαρμογή διαχείρισης καθηγητών, μαθητών, μαθημάτων αλλά και εγγραφών σε μαθήματα ενός σχολείου ή εκπαιδευτικού προγράμματος. Θα παρέχουμε στον χρήστη υπηρεσίες εισαγωγής καθηγητών, μαθητών και μαθημάτων, διαγραφή και ανανέωση αυτών, αλλά και εγγραφής μαθητών στα διάφορα μαθήματα που δημιουργούμε.

Θα υπάρχουν συγκεκριμένοι περιορισμοί, όπως για παράδειγμα κάθε όνομα, επώνυμο ή περιγραφή να ξεπερνά σε μήκος τους 2 χαρακτήρες ή να είναι απαραίτητο να υπάρχει ένας καθηγητής για να δημιουργήσουμε ένα μάθημα με αυτόν.

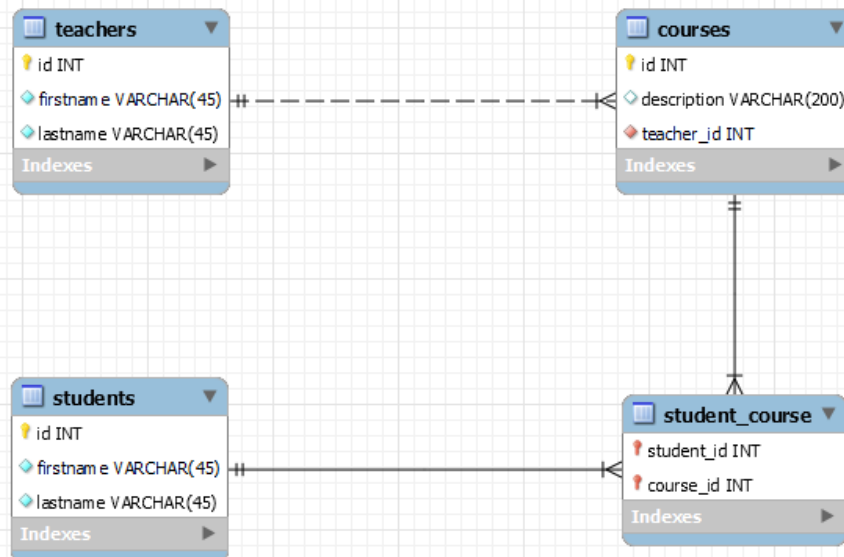
Επίσης, κάθε διαγραφή ή ανανέωση που θα λαμβάνει μέρος σε κάθε οντότητα της εφαρμογής μας, θα πρέπει με τη σειρά της να γίνεται και στους αντίστοιχους πίνακες με τους οποίους συνδέεται η οντότητα αυτή.

## Βάση Δεδομένων


Για την υλοποίηση του Project χρησιμοποίησα βάση δεδομένων mySQL της Oracle και το εργαλείο mySQL WorkBench. Παρακάτω παρουσιάζονται φωτογραφίες από το μοντέλο ER της βάσης μας, το schema και τα πεδία των πινάκων. Οι πίνακες μας είναι οι εξής:




1. TEACHERS(id, firstname, lastname)
2. STUDENTS(id, firstname, lastname)
3. COURSES(id, description, teacher\_id)
4. STUDENT\_COURSE(student\_id, course\_id)

Τυχόν updates και deletes, γίνονται cascade στη βάση λόγω της μικρής πολυπλοκότητας των πινάκων μας.







teachers - Table ×

 Table Name:


Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
 id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
 firstname	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
 lastname	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>




students - Table ×

 Table Name:


Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
 id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
 firstname	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
 lastname	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>



courses - Table ×

 Table Name:



Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
 id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
 description	VARCHAR(200)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
 teacher_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

student\_course - Table ×

 Table Name:


Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
 student_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
 course_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>


courses - Table ×

 Table Name:  Schema: **schoolDB** 



Foreign Key Name	Column	Referenced Column
teacher_fk	<input type="checkbox"/> id	
	<input type="checkbox"/> description	
	<input checked="" type="checkbox"/> teacher_id	id

Foreign Key Options

On Update:  


On Delete:  


student\_course - Table ×

 Table Name:  Schema: **schoolDB** 

Foreign Key Name	Column	Referenced Column
courses_fk		
students_fk		

Foreign Key Options

On Update:  

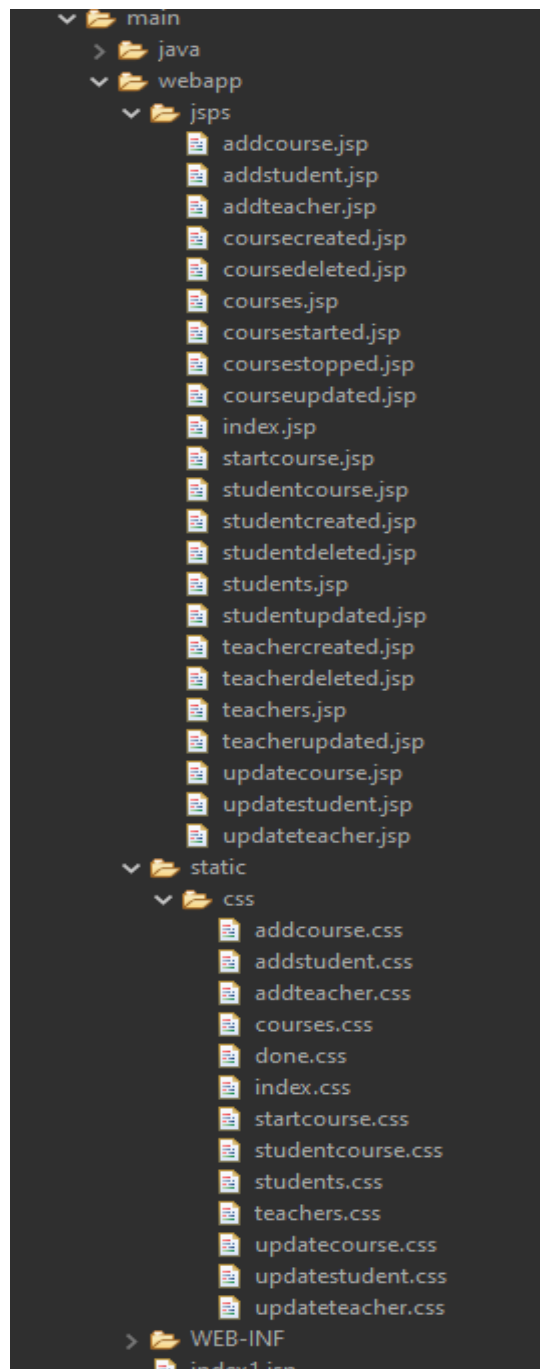
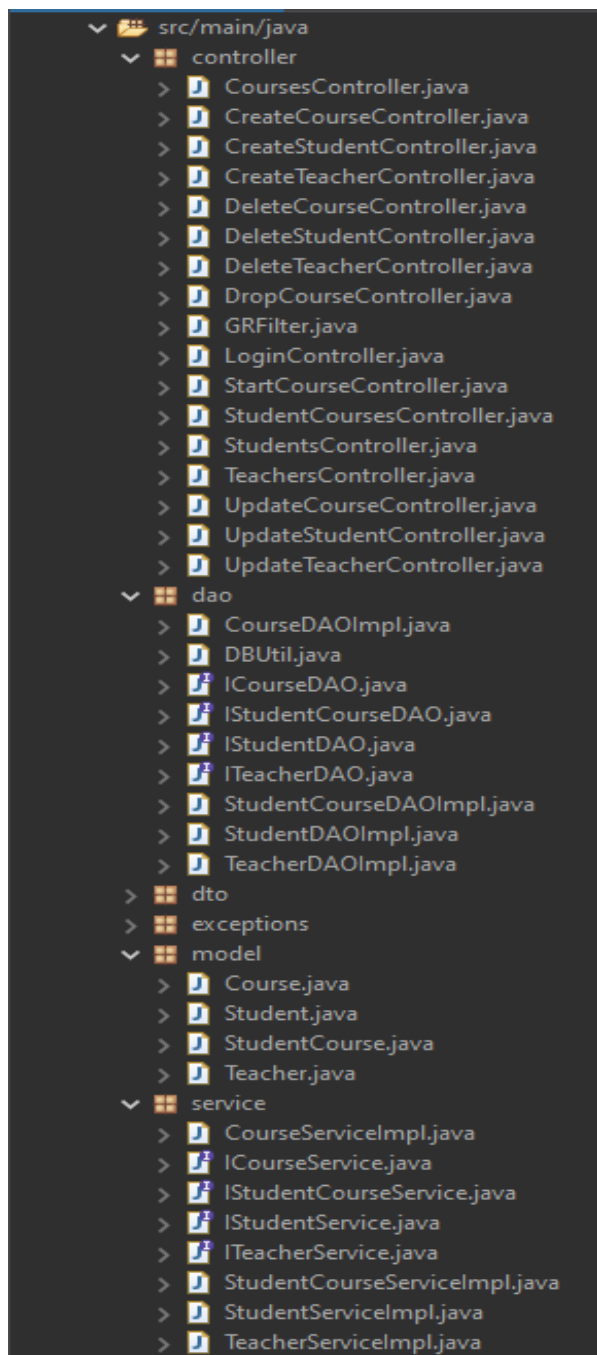
On Delete:  

# Υλοποίηση Project

Για την υλοποίηση του Project χρησιμοποίησα JavaEE με jsp και servlet στο Eclipse IDE.

Η προσέγγισή μου είναι μία προσέγγισή αρχιτεκτονικής SOA μαζί με το MVC (Model, View, Controller) design pattern, καθώς είναι αρκετά διαδεδομένη για προβλήματα ίδιου ή παρόμοιου τύπου με το δικό μας.

Αυτό φαίνεται στη δομή που έχουν τα αρχεία του Project στο Eclipse παρακάτω.



## Οντότητα Καθηγητή

Ας αναλύσουμε το κομμάτι των υπηρεσιών που θα παρέχει στον χρήστη το κομμάτι της υλοποίησης του καθηγητή.

Μας αφορά ο χρήστης να μπορεί να Εισάγει, Διαγράψει, Ανανεώσει αλλά και να δει, κάθε εγγραφή καθηγητή που υπάρχει στη βάση.

Για να αποθηκεύουμε αλλά και να χρησιμοποιούμε τα δεδομένα που εξάγουμε από τη βάση χρησιμοποιούμε την κλάση Teacher που βρίσκεται στον φάκελο models.

```
public class Teacher
{
    private int id;
    private String firstname;
    private String lastname;

    public Teacher() {}

    public Teacher(int id, String firstname, String lastname) {
        super();
        this.id = id;
        this.firstname = firstname;
        this.lastname = lastname;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getFirstname() {
        return firstname;
    }

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    public String getLastname() {
        return lastname;
    }

    public void setLastname(String lastname) {
        this.lastname = lastname;
    }
}
```

Πριν δημιουργήσουμε το DAO layer της εφαρμογής για την πραγματοποίηση CRUD πράξεων στη βάση δεδομένων, φτιάχνουμε μία κλάση DBUtil με την οποία επιτυγχάνεται η σύνδεση μας με τη βάση μας.

```
public class DBUtil
{
    private static Connection conn;

    /**
     * No instances of this class should be available.
     */
    private DBUtil() {}

    public static Connection getConnection() throws SQLException
    {
        try
        {
            Class.forName("com.mysql.cj.jdbc.Driver");

            String url = "jdbc:mysql://localhost:3306/schoolDB?serverTimezone=UTC";

            String username = "jim";

            String password = "123456";

            conn = DriverManager.getConnection(url, username, password);

            return conn;
        }
        catch(ClassNotFoundException exception)
        {
            exception.printStackTrace();
            return null;
        }
        catch(Exception exception)
        {
            exception.printStackTrace();
            return null;
        }
    }

    public static void closeConnection() throws SQLException
    {
        conn.close();
    }
}
```

Χρησιμοποιώντας πλέον τη βοηθητική κλάση DBUtil προχωράμε στη δημιουργία του DAO με υπηρεσίες που χρειαζόμαστε για τον πίνακα των καθηγητών.

Interface με τις υπηρεσίες μας:

```
package dao;

import java.sql.SQLException;

public interface ITeacherDAO
{
    boolean insert(Teacher teacher) throws SQLException;
    boolean delete(int id) throws SQLException;
    boolean update(int id, String firstname, String lastname) throws SQLException;
    Teacher getTeacherById(int id) throws SQLException;
    List<Teacher> getAll() throws SQLException;
}
```

Κλάση με την υλοποίηση των υπηρεσιών:

```
package dao;

import java.sql.Connection;

public class TeacherDAOImpl implements ITeacherDAO
{
    @Override
    public boolean insert(Teacher teacher) throws SQLException
    {
        PreparedStatement statement = null;
        Connection connection = getConnection();

        try
        {
            statement = connection.prepareStatement("INSERT INTO TEACHERS (FIRSTNAME, LASTNAME) VALUES (?, ?)");
            statement.setString(1, teacher.getFirstname());
            statement.setString(2, teacher.getLastname());

            statement.executeUpdate();
            return true;
        }
        catch(SQLException exception)
        {
            exception.printStackTrace();
            throw exception;
        }
        finally
        {
            if (statement != null) statement.close();
            if (connection != null) closeConnection();
        }
    }
}
```

```

@Override
public boolean delete(int id) throws SQLException
{
    PreparedStatement statement = null;
    Connection connection = getConnection();

    try
    {
        statement = connection.prepareStatement("DELETE FROM TEACHERS WHERE ID = ?");
        statement.setInt(1, id);
        statement.executeUpdate();

        return true;
    }
    catch(SQLException exception)
    {
        exception.printStackTrace();
        throw exception;
    }
    finally
    {
        if(statement != null) statement.close();
        if(connection != null) closeConnection();
    }
}

```

```

@Override
public boolean update(int id, String firstname, String lastname) throws SQLException
{
    PreparedStatement statement = null;
    Connection connection = getConnection();

    try
    {
        statement = connection.prepareStatement("UPDATE TEACHERS SET FIRSTNAME = ?, LASTNAME = ? WHERE ID = ?");
        statement.setString(1, firstname);
        statement.setString(2, lastname);
        statement.setInt(3, id);

        statement.executeUpdate();

        return true;
    }
    catch(SQLException exception)
    {
        exception.printStackTrace();
        throw exception;
    }
    finally
    {
        if(statement != null) statement.close();
        if(connection != null) closeConnection();
    }
}

```



```

@Override
public Teacher getTeacherById(int id) throws SQLException
{
    PreparedStatement statement = null;
    Connection connection = getConnection();
    ResultSet results = null;
    Teacher teacher = null;

    try
    {
        statement = connection.prepareStatement("SELECT * FROM TEACHERS WHERE ID LIKE ?");
        statement.setInt(1, id);

        results = statement.executeQuery();

        if (results.next())
        {
            teacher = new Teacher();
            teacher.setId(results.getInt("ID"));
            teacher.setFirstname(results.getString("FIRSTNAME"));
            teacher.setLastname(results.getString("LASTNAME"));
        }

        return teacher;
    }
    catch(SQLException exception)
    {
        exception.printStackTrace();
        throw exception;
    }
    finally
    {
        if (statement != null) statement.close();
        if (connection != null) closeConnection();
    }
}

```

```

public List<Teacher> getAll() throws SQLException
{
    PreparedStatement statement = null;
    Connection connection = getConnection();
    List<Teacher> teachers = new ArrayList<>();
    ResultSet results = null;

    try
    {
        statement = connection.prepareStatement("SELECT * FROM TEACHERS");

        results = statement.executeQuery();

        while (results.next())
        {
            Teacher teacher = new Teacher();
            teacher.setId(results.getInt("ID"));
            teacher.setFirstname(results.getString("FIRSTNAME"));
            teacher.setLastname(results.getString("LASTNAME"));

            teachers.add(teacher);
        }

        return teachers;
    }
    catch(SQLException exception)
    {
        exception.printStackTrace();
        throw exception;
    }
    finally
    {
        if (statement != null) statement.close();
        if (connection != null) closeConnection();
    }
}

```

Με χρήση κατάλληλων SQL ερωτημάτων, εκτελούμε στη βάση μας CRUD πράξεις, και αποθηκεύουμε τα αποτελέσματα σε αντικείμενα ή λίστες αντικειμένων των κλάσεων του model μας. Στη συγκεκριμένη περίπτωση του Teacher.

Για την κλήση των DAO υπηρεσιών είναι υπεύθυνο το Service Layer.

Service Layer Interface:

```
public interface ITeacherService
{
    /**
     * Adds a teacher in the teacher list.
     * @param teacherDTO Added teacher information.
     * @throws SQLException
     */
    boolean createTeacher(TeacherDTO dto) throws SQLException, EmptyFieldException, FirstLastNameLengthException;

    /**
     * Deletes a teacher from the list.
     * @param id Deleted teacher ID.
     * @throws SQLException
     */
    boolean deleteTeacher(int id) throws SQLException;

    /**
     * Updates a teacher from the list.
     * @param id Teacher to be updated ID.
     * @param firstname Teacher to be updated first name.
     * @param lastname Teacher to be updated last name.
     * @throws SQLException
     */
    boolean updateTeacher(int id, String firstname, String lastname) throws SQLException, EmptyFieldException, FirstLastNameLengthException;

    /**
     * Searches a teacher from the list.
     * @param id ID of teacher to search.
     * @return Information of teacher found or null if not found.
     * @throws SQLException
     */
    Teacher searchTeacherById(int id) throws SQLException;

    /**
     * Gets every teacher from the list.
     * @return A list of every teacher on the list or a null list if list is empty.
     * @throws SQLException
     */
    List<Teacher> getAllTeachers() throws SQLException;
}
```

Παρόμοιες μέθοδοι με αυτές του DAO, με τη διαφορά ότι τα ονόματα τους είναι πιο φιλικά προς τον χρήστη και περιέχουν και documentation για την ευκολία του χρήστη.

## Service Layer Implementation:

```
public class TeacherServiceImpl implements ITeacherService
{
    private final ITeacherDAO dao;

    public TeacherServiceImpl(ITeacherDAO dao)
    {
        this.dao = dao;
    }

    @Override
    public boolean createTeacher(TeacherDTO dto) throws SQLException, EmptyFieldException, FirstLastNameLengthException
    {
        if(dto.getFirstname().equals("") || dto.getLastname().equals("")) throw new EmptyFieldException();

        if(dto.getFirstname().length() <= 2 || dto.getLastname().length() <= 2) throw new FirstLastNameLengthException();

        Teacher teacher = extract(dto);
        try
        {
            dao.insert(teacher);
            return true;
        }
        catch(SQLException exception)
        {
            throw exception;
        }
    }
}
```

Η επικοινωνία DAO και service επιτυγχάνεται μέσω Dependency Injection του DAO Interface στον Constructor του Service.

```
@Override
public boolean deleteTeacher(int id) throws SQLException
{
    try
    {
        dao.delete(id);
        return true;
    }
    catch(SQLException exception)
    {
        throw exception;
    }
}
```

```
@Override
public boolean updateTeacher(int id, String firstname, String lastname) throws SQLException, EmptyFieldException, FirstLastNameLengthException
{
    if(firstname.equals("") || lastname.equals("")) throw new EmptyFieldException();

    if(firstname.length() <= 2 || lastname.length() <= 2) throw new FirstLastNameLengthException();

    try
    {
        dao.update(id, firstname, lastname);
        return true;
    }
    catch(SQLException exception)
    {
        exception.printStackTrace();
        throw exception;
    }
}
```

```

@Override
public Teacher searchTeacherById(int id) throws SQLException
{
    try
    {
        return dao.getTeacherById(id);
    }
    catch(SQLException exception)
    {
        exception.printStackTrace();
        throw exception;
    }
}

```

```

public List<Teacher> getAllTeachers() throws SQLException
{
    try
    {
        return dao.getAll();
    }
    catch(SQLException exception)
    {
        exception.printStackTrace();
        throw exception;
    }
}

private Teacher extract(TeacherDTO dto)
{
    Teacher teacher = new Teacher();
    teacher.setId(dto.getId());
    teacher.setFirstname(dto.getFirstname());
    teacher.setLastname(dto.getLastname());

    return teacher;
}

```

η μέθοδος extract αναλαμβάνει τη μετατροπή αντικειμένων των κλάσεων DTO σε model. Οι κλάσεις DTO είναι ακριβώς ίδιες με αυτές του model και τις χρησιμοποιούμε για να μεταφέρουμε δεδομένα από το DAO στο View(χρήστη) και αντίστροφα βάση του MVC Design Pattern. Για την οντότητα του μαθητή η διαδικασία είναι ίδια, συνεπώς δε θα την αναλύσουμε.

## Οντότητα Μάθημα

Όπως με τον καθηγητή και το μαθητή, για τις υπηρεσίες του μαθήματος ακολουθούμε την ίδια μεθοδολογία.

Model:

```
package model;

public class Course
{
    private int id;
    private String description;
    private Integer teacherId;

    public Course () {}

    public Course(int id, String description) {
        super();
        this.id = id;
        this.description = description;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Integer getTeacherId() {
        return teacherId;
    }

    public void setTeacherId(Integer teacherId) {
        this.teacherId = teacherId;
    }
}
```

## DAO Layer:

```
1 package dao;
2
3 import java.sql.SQLException;
4
5 public interface ICourseDAO
6 {
7     boolean insert(String description, int teacherId) throws SQLException;
8     boolean delete(int id) throws SQLException;
9     boolean update(int id, String description, int teacherId) throws SQLException;
10    List<Course> getAll() throws SQLException;
11 }
12
```

```
public class CourseDAOImpl implements ICourseDAO
{
    public boolean insert(String description, int teacherId) throws SQLException
    {
        PreparedStatement statement = null;
        Connection connection = getConnection();

        try
        {
            statement = connection.prepareStatement("INSERT INTO COURSES (DESCRIPTION, TEACHER_ID) VALUES (?, ?)");
            statement.setString(1, description);
            statement.setInt(2, teacherId);

            statement.executeUpdate();

            return true;
        }
        catch(SQLException exception)
        {
            exception.printStackTrace();
            throw exception;
        }
        finally
        {
            if (statement != null) statement.close();
            if (connection != null) closeConnection();
        }
    }
}
```

```

@Override
public boolean delete(int id) throws SQLException
{
    PreparedStatement statement = null;
    Connection connection = getConnection();

    try
    {
        statement = connection.prepareStatement("DELETE FROM COURSES WHERE ID = ?");
        statement.setInt(1, id);
        statement.executeUpdate();

        return true;
    }
    catch(SQLException exception)
    {
        exception.printStackTrace();
        throw exception;
    }
    finally
    {
        if (statement != null) statement.close();
        if (connection != null) closeConnection();
    }
}

```

```

@Override
public boolean update(int id, String description, int teacherId) throws SQLException
{
    PreparedStatement statement = null;
    Connection connection = getConnection();

    try
    {
        statement = connection.prepareStatement("UPDATE COURSES SET DESCRIPTION = ?, TEACHER_ID = ? WHERE ID = ?");
        statement.setString(1, description);
        statement.setInt(2, teacherId);
        statement.setInt(3, id);

        statement.executeUpdate();

        return true;
    }
    catch(SQLException exception)
    {
        exception.printStackTrace();
        throw exception;
    }
    finally
    {
        if (statement != null) statement.close();
        if (statement != null) closeConnection();
    }
}

```

```

public List<Course> getAll() throws SQLException
{
    PreparedStatement statement = null;
    Connection connection = getConnection();
    ResultSet results = null;
    List<Course> courses = new ArrayList<>();

    try
    {
        statement = connection.prepareStatement("SELECT * FROM COURSES");

        results = statement.executeQuery();

        while (results.next())
        {
            Course course = new Course();
            course.setId(results.getInt("ID"));
            course.setDescription(results.getString("DESCRIPTION"));
            course.setTeacherId(results.getInt("TEACHER_ID"));

            courses.add(course);
        }

        return courses;
    }
    catch(SQLException exception)
    {
        exception.printStackTrace();
        throw exception;
    }
    finally
    {
        if (statement != null) statement.close();
        if (connection != null) closeConnection();
    }
}

```

Οι διαφορές με τον καθηγητή και τον μαθητή, είναι ότι το μάθημα έχει διαφορετικά πεδία, τα οποία είναι ο κωδικός του μαθήματος, η περιγραφή του και ο κωδικός του καθηγητή ο οποίος είναι ξένο κλειδί στον πίνακα των μαθημάτων από τον πίνακα των καθηγητών. Γιαυτό και υπάρχει διαφοροποίηση στα SQL ερωτήματα.



## Οντότητα Μαθητή Μάθημα

Στον συγκεκριμένο πίνακα, τα SQL ερωτήματα αλλά και οι υπηρεσίες τις οποίες θέλουμε να μας παρέχει, είναι λίγο πιο περίπλοκες διότι έχουμε να κάνουμε με έναν πίνακα συσχέτισης μεταξύ μαθητή και μάθημα. Είναι ο πίνακας αυτός ο οποίος θέλουμε να κρατάει κωδικούς μαθημάτων αλλά και μαθητών, έτσι ώστε να συμβολίζονται οι εγγραφές των μαθητών στα μαθήματα. Τα πεδία του είναι ξένα κλειδιά άλλων πινάκων, συνεπώς κάποιες από τις υπηρεσίες του περιλαμβάνουν SQL JOIN ερωτήματα.

Model:

```
package model;

public class StudentCourse
{
    private int courseId;
    private int studentId;

    public StudentCourse() {}

    public StudentCourse(int courseId, int studentId) {
        super();
        this.courseId = courseId;
        this.studentId = studentId;
    }

    public int getCourseId() {
        return courseId;
    }

    public void setCourseId(int courseId) {
        this.courseId = courseId;
    }

    public int getStudentId() {
        return studentId;
    }

    public void setStudentId(int studentId) {
        this.studentId = studentId;
    }
}
```

## DAO Layer:

```
package dao;

import java.sql.SQLException;

public interface IStudentCourseDAO
{
    boolean insert(int studentId, int courseId) throws SQLException;
    boolean delete(int courseId) throws SQLException;
    List<Course> getAll(int studentId) throws SQLException;
    List<Course> getNotStarted(int studentId) throws SQLException;
}
```

```
import static dao.DBUtil.closeConnection;

public class StudentCourseDAOImpl implements IStudentCourseDAO
{
    @Override
    public boolean insert(int studentId, int courseId) throws SQLException
    {
        PreparedStatement statement = null;
        Connection connection = getConnection();

        try
        {
            statement = connection.prepareStatement("INSERT INTO STUDENT_COURSE (STUDENT_ID, COURSE_ID) VALUES (?, ?)");
            statement.setInt(1, studentId);
            statement.setInt(2, courseId);

            statement.executeUpdate();

            return true;
        }
        catch(SQLException exception)
        {
            exception.printStackTrace();
            throw exception;
        }
        finally
        {
            if (statement != null) statement.close();
            if (connection != null) closeConnection();
        }
    }
}
```

```
@Override
public boolean delete(int courseId) throws SQLException
{
    PreparedStatement statement = null;
    Connection connection = getConnection();

    try
    {
        statement = connection.prepareStatement("DELETE FROM STUDENT_COURSE WHERE COURSE_ID = ?");
        statement.setInt(1, courseId);
        statement.executeUpdate();

        return true;
    }
    catch(SQLException exception)
    {
        exception.printStackTrace();
        throw exception;
    }
    finally
    {
        if (statement != null) statement.close();
        if (connection != null) closeConnection();
    }
}
```

```

public List<Course> getAll(int studentId) throws SQLException
{
    PreparedStatement statement = null;
    Connection connection = getConnection();
    ResultSet results = null;
    List<Course> courses = new ArrayList<>();

    try
    {
        statement = connection.prepareStatement("SELECT COURSES.ID, COURSES.DESCRPTION "
            + "FROM COURSES, STUDENTS, STUDENT_COURSE "
            + "WHERE STUDENTS.ID = ? AND COURSES.ID = STUDENT_COURSE.COURSE_ID AND STUDENTS.ID = STUDENT_COURSE.STUDENT_ID");
        statement.setInt(1, studentId);
        results = statement.executeQuery();

        while (results.next())
        {
            Course course = new Course();
            course.setId(results.getInt("ID"));
            course.setDescription(results.getString("DESCRIPTION"));

            courses.add(course);
        }

        return courses;
    }
    catch(SQLException exception)
    {
        exception.printStackTrace();
        throw exception;
    }
    finally
    {
        if (statement != null) statement.close();
        if (connection != null) closeConnection();
    }
}

```

```

public List<Course> getNotStarted(int studentId) throws SQLException
{
    PreparedStatement statement = null;
    Connection connection = getConnection();
    ResultSet results = null;
    List<Course> courses = new ArrayList<>();

    try
    {
        statement = connection.prepareStatement("SELECT COURSES.ID, COURSES.DESCRPTION FROM COURSES WHERE COURSES.ID NOT IN (SELECT COURSE_ID FROM STUDENT_COURSE WHERE STUDENT_ID = ?)");
        statement.setInt(1, studentId);
        results = statement.executeQuery();

        while (results.next())
        {
            Course course = new Course();
            course.setId(results.getInt("ID"));
            course.setDescription(results.getString("DESCRIPTION"));

            courses.add(course);
        }

        return courses;
    }
    catch(SQLException exception)
    {
        exception.printStackTrace();
        throw exception;
    }
    finally
    {
        if (statement != null) statement.close();
        if (connection != null) closeConnection();
    }
}

```

Πέραν από τις CRUDE insert, update υπηρεσίες, μας παρέχει επίσης την υπηρεσία getAll, η οποία μας επιστρέφει τους κωδικούς και τις περιγραφές όλων των μαθημάτων που είναι εγγεγραμμένος ένας μαθητής.

Ενώ και την υπηρεσία getNotStarted, η οποία μας επιστρέφει μας επιστρέφει τα μαθήματα στα οποία δεν είναι εγγεγραμμένος ένας συγκεκριμένος μαθητής.

## Service Layer:

```
public class StudentCourseServiceImpl implements IStudentCourseService
{
    private final IStudentCourseDAO dao;

    public StudentCourseServiceImpl(IStudentCourseDAO dao)
    {
        this.dao = dao;
    }

    @Override
    public boolean startCourse(int studentId, int courseId) throws SQLException
    {
        try
        {
            dao.insert(studentId, courseId);
            return true;
        }
        catch(SQLException exception)
        {
            throw exception;
        }
    }

    @Override
    public boolean stopCourse(int courseId) throws SQLException
    {
        try
        {
            dao.delete(courseId);
            return true;
        }
        catch(SQLException exception)
        {
            throw exception;
        }
    }
}
```

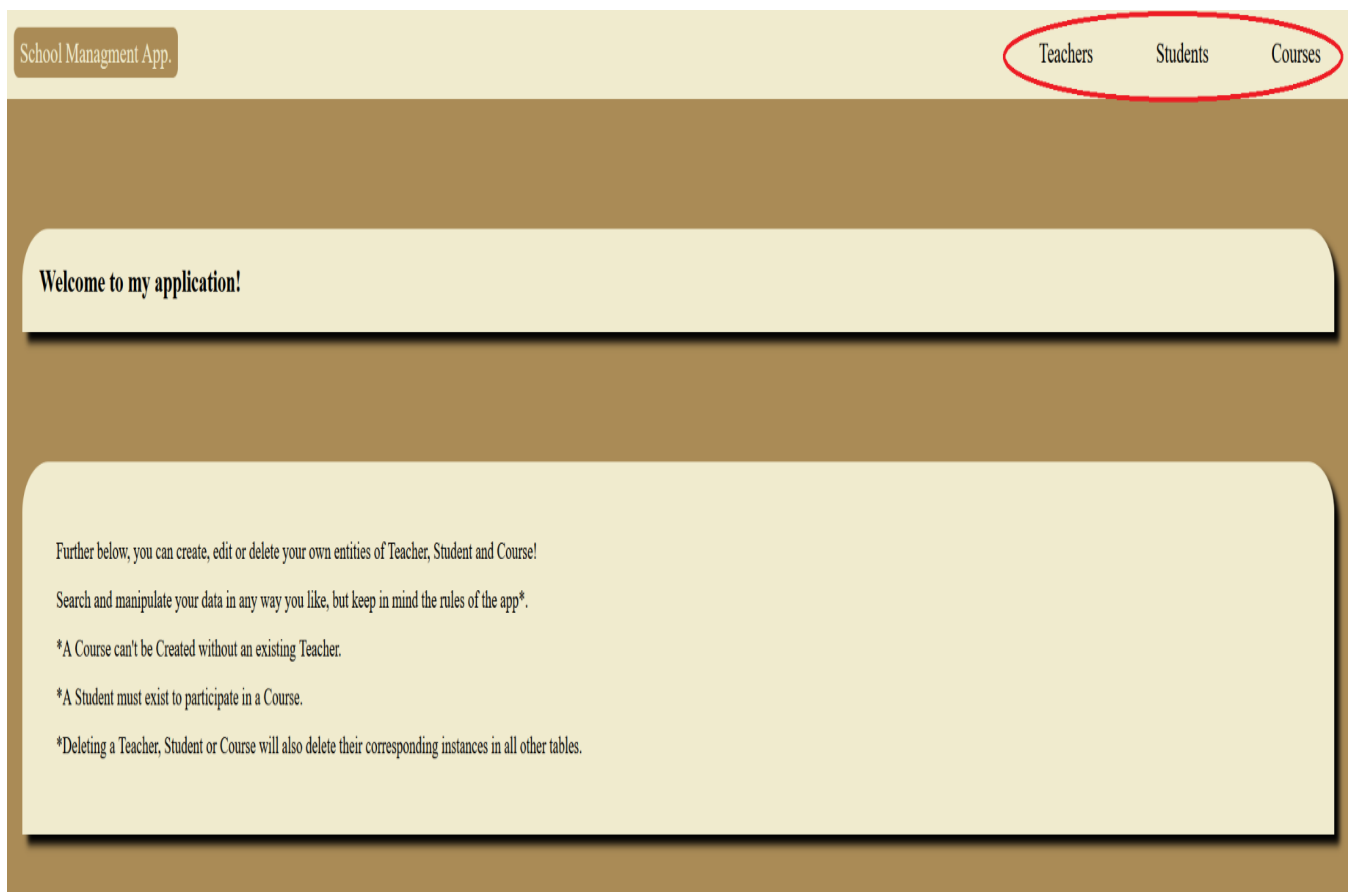
```
@Override
public List<Course> getCourses(int studentId) throws SQLException
{
    try
    {
        return dao.getAll(studentId);
    }
    catch(SQLException exception)
    {
        throw exception;
    }
}

@Override
public List<Course> getNotStartedCourses(int studentId) throws SQLException
{
    try
    {
        return dao.getNotStarted(studentId);
    }
    catch(SQLException exception)
    {
        throw exception;
    }
}
```

Το View Layer με το οποίο αλληλεπιδρά ο χρήστης επικοινωνεί με το service μέσω των controllers, ειδικά servlets τα οποία διαχειρίζονται δεδομένα αλλά και λάθη μεταξύ του Frontend και του Backend μιας εφαρμογής. Οι controllers στέλνουν και δέχονται δεδομένα από και προς τις jsp σελίδες του View μέσω ειδικών μεθόδων GET και POST. Στην συνέχεια θα δούμε μερικά παραδείγματα από controllers, jsps αλλά και το πως λειτουργούν στην εφαρμογή μας.

## Controller, View & Jsps

Στην αρχική σελίδα της εφαρμογής, index.jsp μπορούμε να δούμε πάνω δεξιά το μενού μας με τις επιλογές Teachers, Students, Courses.



Επιλέγοντας για παράδειγμα την επιλογή Courses, καλείται το servlet CoursesController.

```

1 package controller;
2
3 import java.io.IOException;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19 @WebServlet("/courses")
20 public class CoursesController extends HttpServlet
21 {
22     private static final long serialVersionUID = 1L;
23
24     ICourseDAO dao = new CourseDAOImpl();
25     ICourseService service = new CourseServiceImpl(dao);
26
27
28     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
29     {
30         response.setContentType("text/html; charset=UTF-8");
31
32         try
33         {
34             List<Course> courses = service.getCourses();
35             if(courses.size() == 0)
36             {
37                 request.getRequestDispatcher("/jsps/courses.jsp").forward(request, response);
38             }
39             else
40             {
41                 request.setAttribute("courses", courses);
42                 request.getRequestDispatcher("/jsps/courses.jsp").forward(request, response);
43             }
44         }
45         catch(SQLException exception)
46         {
47             request.setAttribute("SQLException", true);
48             request.getRequestDispatcher("/jsps/index").forward(request, response);
49         }
50     }
51 }

```

Ο controller αυτός με τη σειρά του, καλεί το service των μαθημάτων και μέσω του DAO επιστρέφει μία λίστα με τα μαθήματα τα οποία είναι διαθέσιμα στο μαθητές της εφαρμογής, αν υπάρχουν ή όχι.

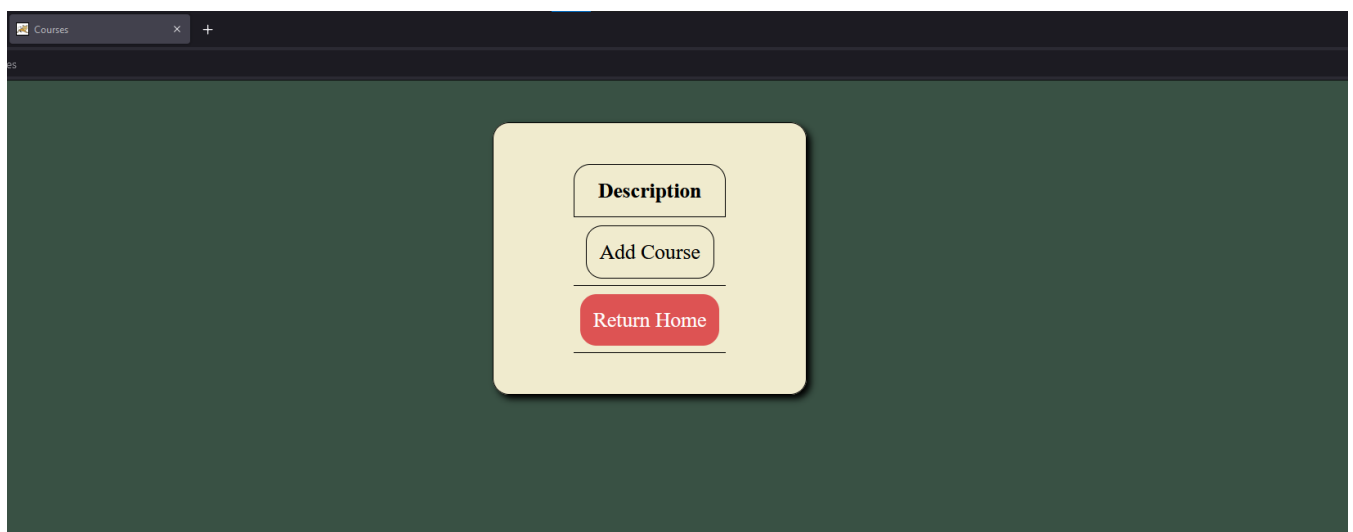
Στη συνέχεια αυτά τα δεδομένα στέλνονται σε σελίδες jsp (courses.jsp) οι οποίες παρουσιάζουν τα δεδομένα μας στον χρήστη.

Στο συγκεκριμένο παράδειγμα θα δούμε ότι δεν υπάρχει κανένα μάθημα, και πως μπορεί ο χρήστης να δημιουργήσει μαθήματα, επιλέγοντας την περιγραφή και τον καθηγητή που θέλει ο ίδιος.

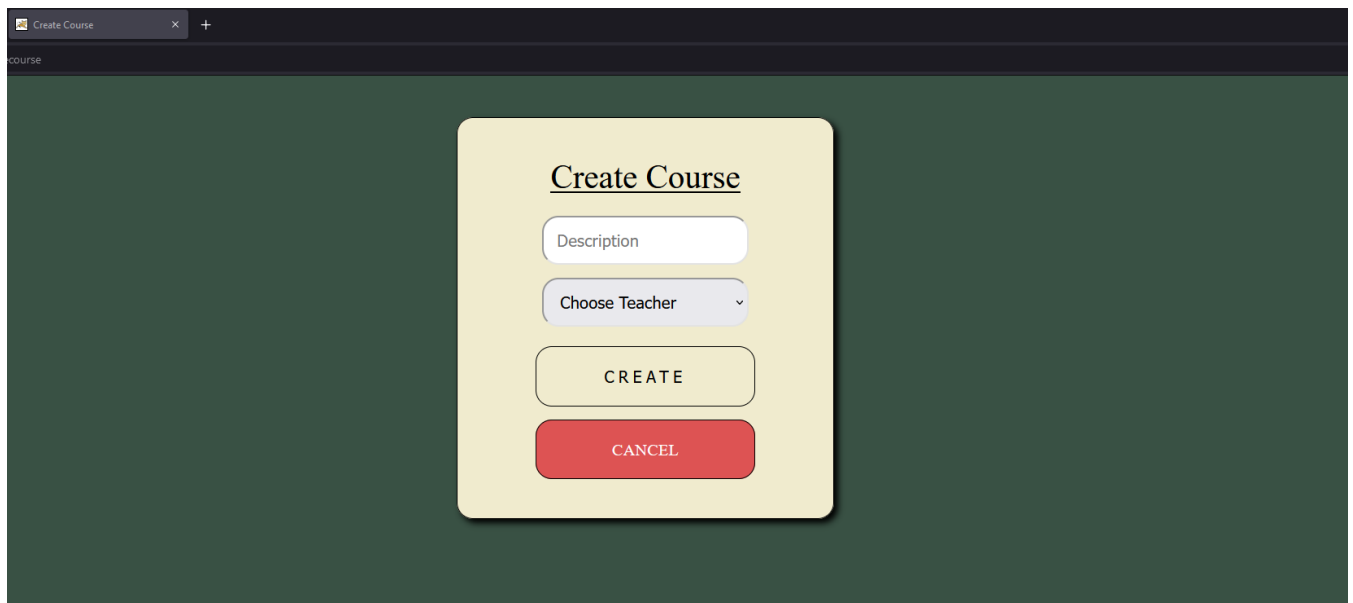
## courses.jsp:

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="${pageContext.request.contextPath}/static/css/courses.css">
  <title>Courses</title>
</head>
<body>
  <section id="courses-table">
    <table>
      <thead>
        <tr>
          <th hidden="hidden">ID</th>
          <th>Description</th>
        </tr>
      </thead>
      <tbody>
        <c:forEach var="course" items="${courses}">
          <tr>
            <td hidden="hidden">${course.id}</td>
            <td>${course.description}</td>
            <td hidden="hidden">${course.teacherId}</td>
            <td>
              <a href="${pageContext.request.contextPath}/updatecourse?id=${course.id}&description=${course.description}&teacherid=${course.teacherId}">Update</a>
            </td>
            <td>
              <a href="${pageContext.request.contextPath}/deletecourse?id=${course.id}">Delete</a>
            </td>
          </tr>
        </c:forEach>
        <tr>
          <td id="addbutton"><a href="${pageContext.request.contextPath}/createcourse">Add Course</a></td>
        </tr>
        <tr>
          <td id="returnbutton"><a href="/jsps/index.jsp">Return Home</a></td>
        </tr>
      </tbody>
    </table>
  </section>
</body>
</html>
```

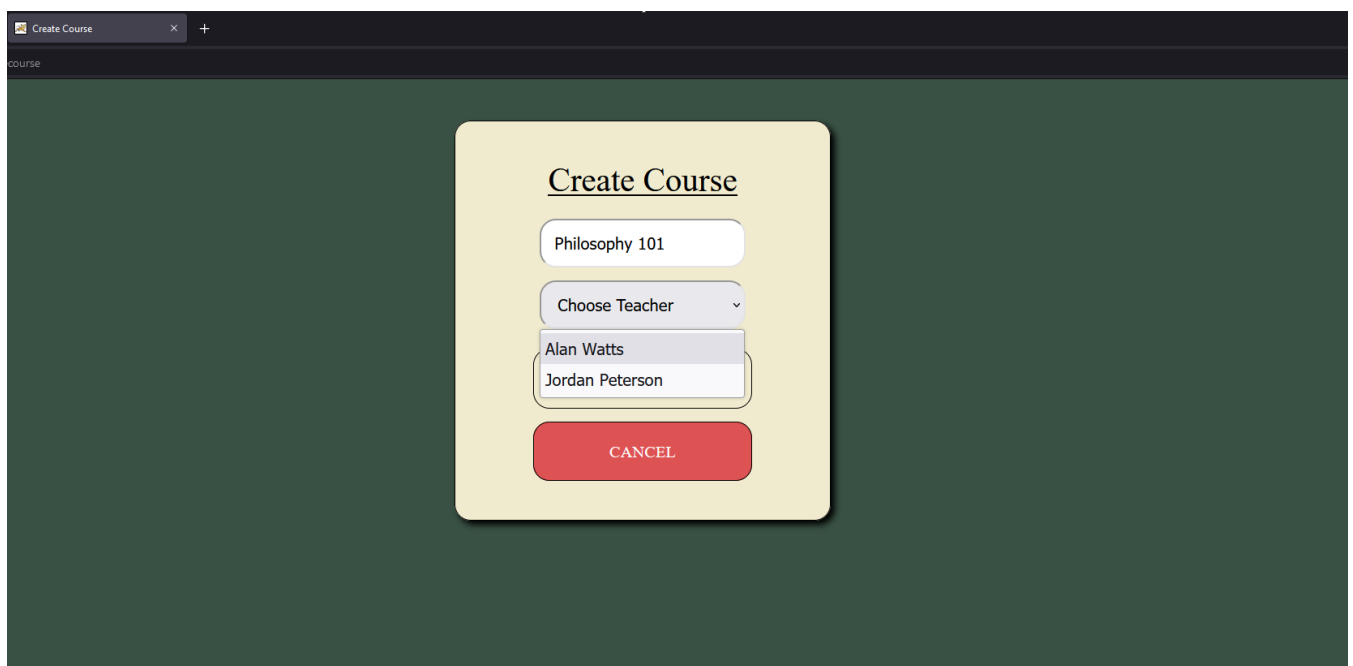
## courses.jsp View:



Πατώντας το κουμπί Add Course, στο επόμενο view παρουσιάζεται μία φόρμα δημιουργίας μαθήματος με περιγραφή αλλά και ένα Dropdown μενού με όλους τους διαθέσιμους καθηγητές.



A screenshot of a web browser window with a dark header bar containing a tab labeled 'Create Course' and a plus sign. The main content area has a dark green background. In the center is a light yellow rounded rectangle containing the title 'Create Course' in a serif font. Below the title are two input fields: the first is labeled 'Description' and is empty; the second is a dropdown menu labeled 'Choose Teacher' with a downward arrow. At the bottom of the form are two buttons: a yellow 'CREATE' button and a red 'CANCEL' button.



A screenshot of the same 'Create Course' form. The 'Description' field now contains the text 'Philosophy 101'. The 'Choose Teacher' dropdown menu is open, showing a list of two names: 'Alan Watts' and 'Jordan Peterson'. The 'CREATE' button is still present but not visible in this frame, while the 'CANCEL' button remains at the bottom.

Αφού επιλέξουμε περιγραφή και καθηγητή της επιλογής μας, πατώντας CREATE, μεταβαίνουμε σε μία νέα σελίδα στην οποία επιβεβαιώνεται η εισαγωγή ή μας εμφανίζει μηνύματα λάθους αν προκύψει κάποιο θέμα.



A screenshot of a confirmation message box. It features a light yellow background with a dark brown border. On the left, the text 'Course created successfully' is displayed in a green serif font. To the right of this text are two buttons: a yellow 'Return' button and a red 'Return Home' button.



την μετάβαση αυτή διαχειρίζεται ο CreateCourseController με κλήσεις GET για να δημιουργηθεί το Dropdown των καθηγητών, και POST για να γίνει η εισαγωγή μαθήματος.

GET:

```
ICourseDAO courseDao = new CourseDAOImpl();
ICourseService courseService = new CourseServiceImpl(courseDao);

ITeacherDAO teacherDao = new TeacherDAOImpl();
ITeacherService teacherService = new TeacherServiceImpl(teacherDao);

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{
    try
    {
        List<Teacher> teachers = teacherService.getAllTeachers();
        if(teachers.size() == 0)
        {
            request.setAttribute("EmptyList", true);
            request.getRequestDispatcher("/jsps/courses.jsp").forward(request, response);
        }
        else
        {
            request.setAttribute("teachers", teachers);
            request.getRequestDispatcher("/jsps/addcourse.jsp").forward(request, response);
        }
    }
    catch(SQLException exception)
    {
        request.setAttribute("SQLException", true);
        request.getRequestDispatcher("/jsps/courses.jsp").forward(request, response);
    }
}
```

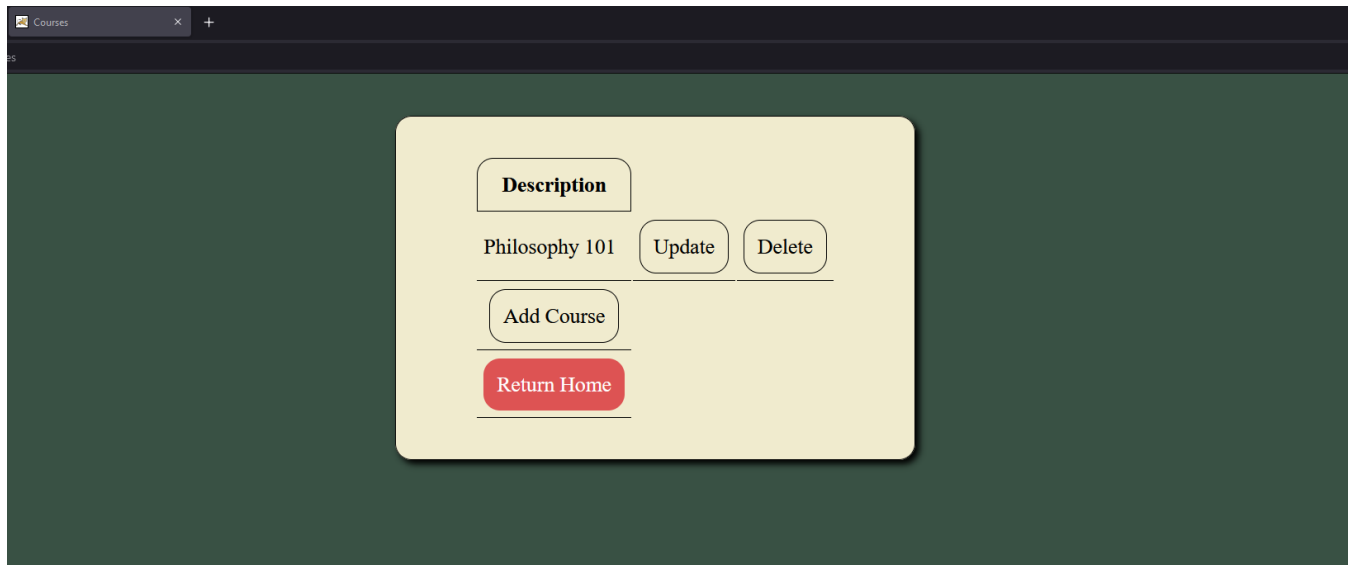
POST:

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{
    response.setContentType("text/html; charset=UTF-8");

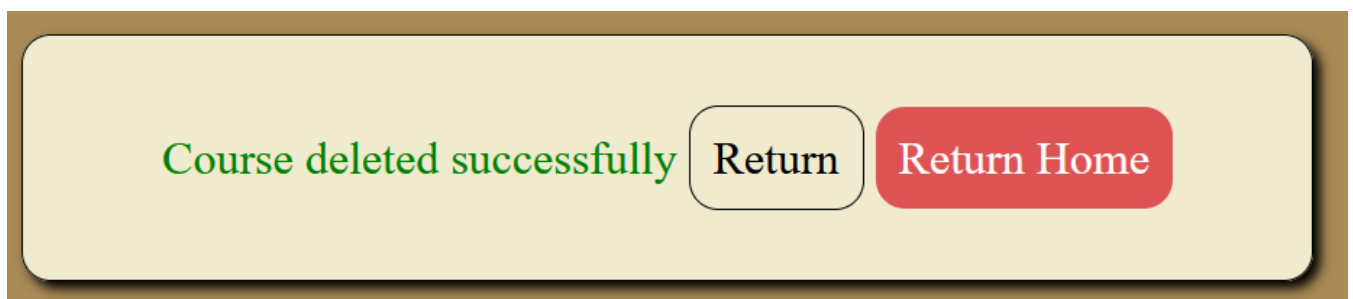
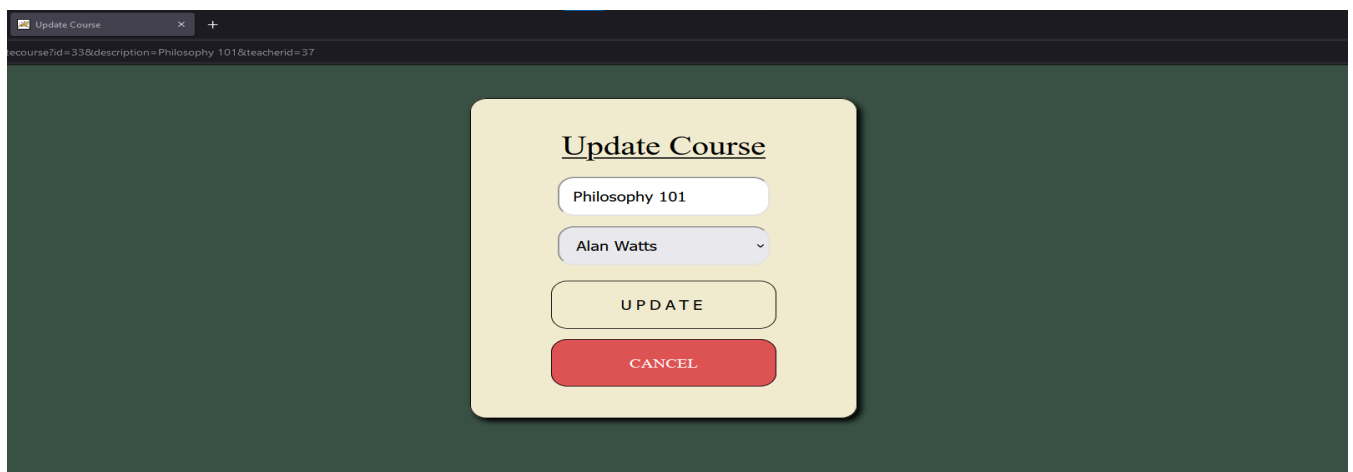
    String description = request.getParameter("description");

    try
    {
        int teacherId = Integer.parseInt(request.getParameter("teacherid"));
        courseService.createCourse(description, teacherId);
        request.setAttribute("description", description);
        request.setAttribute("success", true);
        request.getRequestDispatcher("/jsps/coursecreated.jsp").forward(request, response);
    }
    catch(EmptyFieldException exception)
    {
        request.setAttribute("EmptyField", true);
        request.getRequestDispatcher("/jsps/coursecreated.jsp").forward(request, response);
    }
    catch(DescriptionLengthException exception)
    {
        request.setAttribute("FieldLength", true);
        request.getRequestDispatcher("/jsps/coursecreated.jsp").forward(request, response);
    }
    catch(NumberFormatException exception)
    {
        request.setAttribute("nullFormat", true);
        request.getRequestDispatcher("/jsps/coursecreated.jsp").forward(request, response);
    }
    catch(SQLException exception)
    {
        request.setAttribute("SQLException", true);
        request.getRequestDispatcher("/jsps/coursecreated.jsp").forward(request, response);
    }
}
```

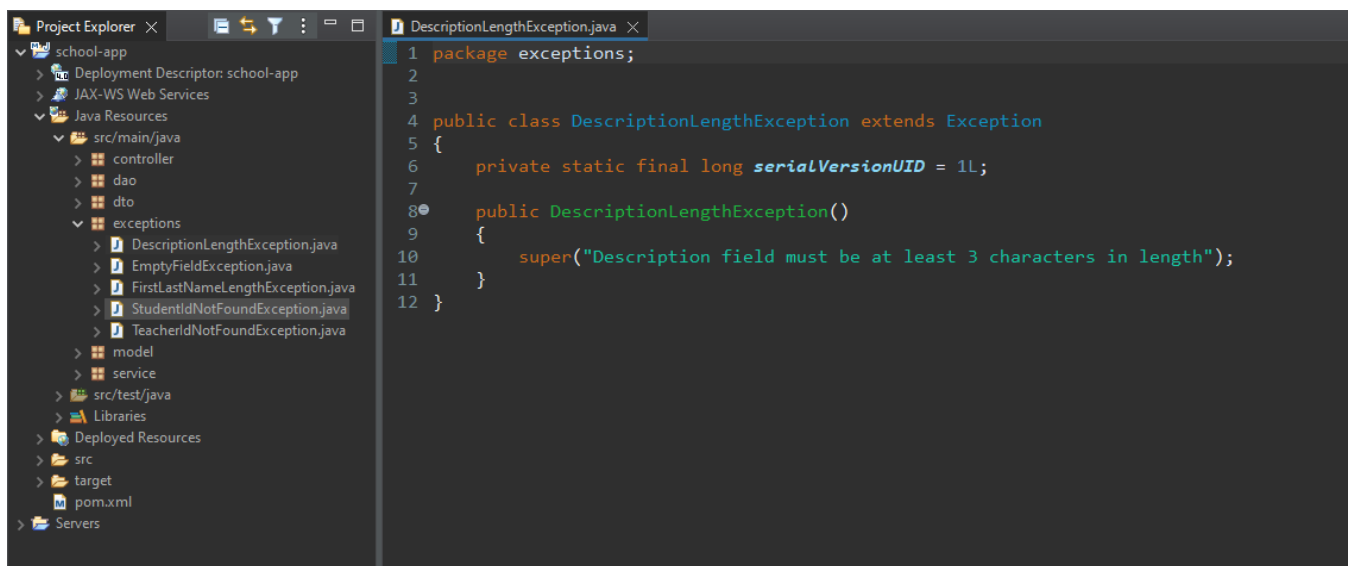
Αφού πάρουμε το μήνυμα επιτυχούς εγγραφής, πατώντας return, γυρνάμε πίσω στη σελίδα των μαθημάτων και βλέπουμε αυτή τη φορά το μάθημα το οποίο δημιουργήσαμε.



Πατώντας Delete ή Update, μεταβιβάζομαστε στις αντίστοιχες σελίδες με κατάλληλα μηνύματα ή φόρμες για την εκάστοτε υπηρεσία.



Τα τυχόν λάθη ή περιορισμούς τα διαχειρίζονται ειδικά Exception classes που έχουμε δημιουργήσει στον φάκελο exceptions.



```
1 package exceptions;
2
3
4 public class DescriptionLengthException extends Exception
5 {
6     private static final long serialVersionUID = 1L;
7
8     public DescriptionLengthException()
9     {
10         super("Description field must be at least 3 characters in length");
11     }
12 }
```

Ενώ κάποιους άλλους περιορισμούς του διαχειριζόμαστε με το να μη δίνουμε στον χρήστη την επιλογή να κάνει λάθος όπως στο παρακάτω παράδειγμα.



Στη σελίδα των μαθητών, βλέπουμε μία λίστα με τους μαθητές που υπάρχουν εγγεγραμμένοι στη βάση μας. Πατώντας το κουμπί Courses, μπορούμε να δούμε ποια μαθήματα παρακολουθεί ο συγκεκριμένος μαθητής.

Στο παράδειγμά αυτό, δεν υπάρχουν μαθήματα τα οποία παρακολουθεί ο μαθητής, συνεπώς θα προσπαθήσουμε να βάλουμε κάποιο.

Student Courses

×

+

ntcourses?tid=23&firstname=James&lastname=Doe

James Doe

Courses

Add Course

RETURN

Start Course

×

+

ursetid=23&firstname=James&lastname=Doe

Start Course

James

Doe

Philosophy 101

START COURSE

CANCEL

Student Courses

×

+

ntcourses?tid=23&firstname=James&lastname=Doe

James Doe

Courses

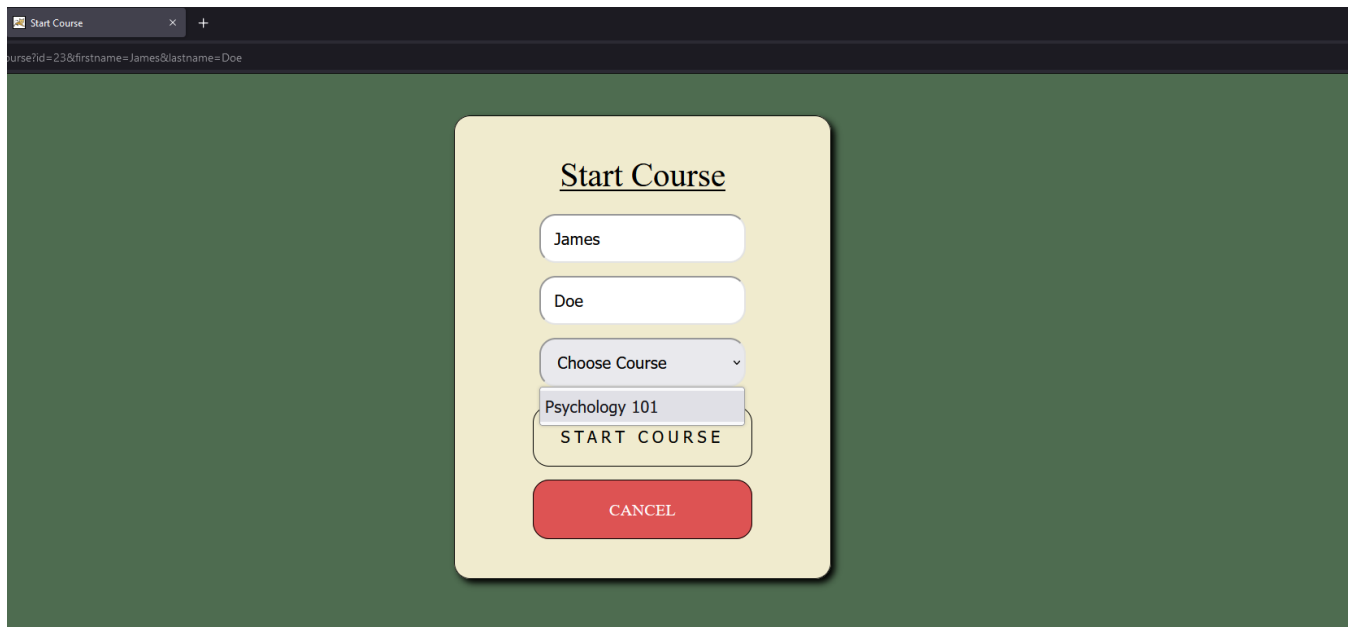
Philosophy 101

Drop Course

Add Course

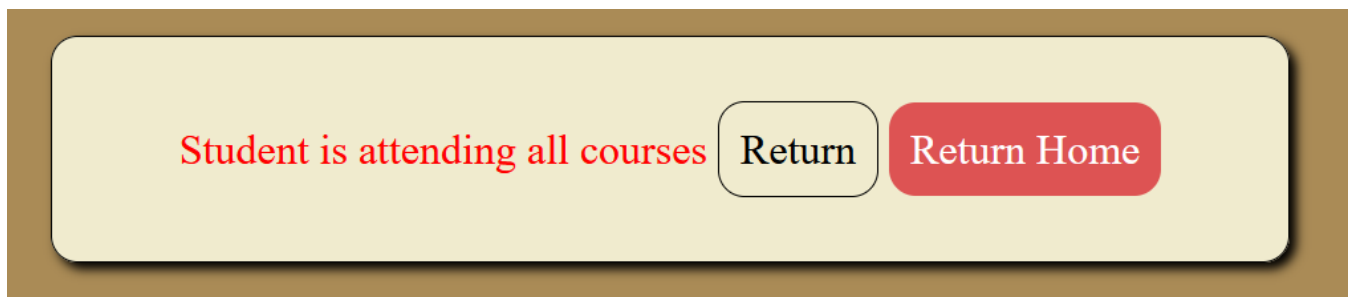
RETURN

Προσπαθώντας στη συνέχεια να εισάγουμε ξανά το ίδιο μάθημα στον μαθητή θα μας ήταν αδύνατο, καθώς τα μαθήματα που ήδη παρακολουθεί ο μαθητής δεν εμφανίζονται πλέον στο Dropdown menu των επιλογών.



The screenshot shows a web browser window with a tab titled 'Start Course'. The address bar contains the URL 'course?id=23&firstname=James&lastname=Doe'. The main content area has a dark green background. In the center is a light yellow rounded rectangle containing the following elements: the title 'Start Course' in black text, a text input field with 'James', another text input field with 'Doe', a dropdown menu labeled 'Choose Course' with a downward arrow, a selected option 'Psychology 101', a yellow button with the text 'START COURSE', and a red button with the text 'CANCEL'.

Ενώ εάν ο μαθητής παρακολουθεί όλα τα υπάρχοντα μαθήματα, εμφανίζεται κατάλληλο μήνυμα.



The screenshot shows a message box with a brown background. Inside is a light yellow rounded rectangle containing the text 'Student is attending all courses' in red. To the right of this text are two buttons: a yellow button with the text 'Return' and a red button with the text 'Return Home'.

## Επίλογος - Βιβλιογραφία

Θα ήθελα να ευχαριστήσω θερμά τον ΣΕΒ αλλά και το Οικονομικό Πανεπιστήμιο Αθηνών για τους πόρους που επένδυσαν στους συμμετέχοντες του προγράμματος.

Επίσης, ένα μεγάλο ευχαριστώ στους καθηγητές Αθανάσιο Ανδρούτσο, Χρυσόστομο Καπέτη, Μάρκο Καραμπάτση, Γιώργο Λεκάκο και Κυριάκο Διακονικολάου για τον χρόνο και την υπομονή τους, και ιδιαίτερα τον κ.Αθανάσιο Ανδρούτσο για την αφοσίωση, το πάθος και την ενέργεια που μας μετέδωσε, και για τις λεπτομερείς σημειώσεις του οι οποίες βοήθησαν αρκετά στην πορεία του προγράμματος.

**Ονοματεπώνυμο:** Δημήτρης Ζώκας

**E-mail:** [jimzokas@outlook.com](mailto:jimzokas@outlook.com)

**GitHub:** [GitHub/SchoolApp](https://github.com/SchoolApp)

**Linkedin:** <https://www.linkedin.com/in/dimitris-zokas-173669243/>

## **Βιβλιογραφία**

- Slides των καθηγητών: [edudz.elearning](https://edudz.elearning)
- [Stack Overflow - Where Developers Learn, Share & Build Careers](https://stackoverflow.com/)
- [W3Schools - Learn to Code](https://www.w3schools.com/)