



Manage Project Homework: Design Document

Riccardo Ancona - 782025
Alessandro Ditta - 781482

December 22, 2011

Contents

1	Introduction	4
1.1	Purpose	4
1.2	Scope	4
1.3	Definitions, acronyms, and abbreviations	4
1.4	References	5
2	Functional overview of modules and components	5
2.1	Database Management System	6
2.2	Server Module	6
2.3	Client Module	7
2.4	Shared Class Library	7
3	Detailed System Architecture Description	8
3.1	Database Management System	10
3.1.1	Conceptual Entity-Relationship Diagram	11
3.1.2	Logic Entity-Relationship Diagram	11
3.1.2.1	Operations	11
3.1.2.2	Tables	11
3.1.2.3	Foreign Keys	12
3.1.3	Detailed Table Description	12
3.1.3.1	Professor	12
3.1.3.2	Student	13
3.1.3.3	Course	13
3.1.3.4	Project	14
3.1.3.5	Deliverable	14
3.1.3.6	ProjectTeam	15
3.1.3.7	StudentBelongsTeam	15
3.1.3.8	StudentRequestsMembership	15
3.1.3.9	Artifact	16
3.2	Server Module	17
3.2.1	User	19
3.2.2	Student	20
3.2.3	Professor	21
3.2.4	Course	22
3.2.5	Project	23
3.2.6	ProjectId	24
3.2.7	Team	25
3.2.8	TeamId	26
3.2.9	Deliverable	27
3.2.10	DeliverableId	28
3.2.11	Artifact	29
3.2.12	ArtifactId	30
3.2.13	Session Beans	31
3.2.14	Local	33

3.2.15	Remote	35
3.3	Shared Module	37
3.3.1	Data Transfer Objects	38
3.3.2	Data Transfer Objects Identifiers	40
3.3.3	Exceptions	41
3.3.4	Sessions	42
3.3.4.1	Artifact	43
3.3.4.2	Course	43
3.3.4.3	Deliverable	43
3.3.4.4	Project	43
3.3.4.5	Team	43
3.3.4.6	User	43
3.4	Client Module	44
3.4.1	Connection Package	45
3.4.2	Student GUI Package	46
3.4.3	Professor GUI Package	48
3.4.4	RegistrationScreen	50
3.4.5	LoginScreen	51
3.4.6	StudentMainScreen	52
3.4.7	StudentsScreen	53
3.4.8	ProjectSelectionScreen	54
3.4.9	ProjectsInfoScreen	55
3.4.10	ProfessorMainScreen	56
3.4.11	NewProjectScreen	57
3.4.12	DeliveredArtifactsScreen	58
3.4.13	TeamInfoScreen	59
4	Reuse and relationship to other products	59
4.1	Java and Java tools	59
4.1.1	Programming Language	59
4.1.2	External Libraries	59
4.1.2.1	RMIIO	60
4.1.3	IDE	60
4.2	Database and Reporting	60
5	Design Decisions and Tradeoffs	61
5.1	Three Tier Design	61
A	Pseudocode	61
A.1	DBMS Processing	61

1 Introduction

1.1 Purpose

This software design document describes the architecture and system design of Manage Project Homework

It is a living document that is expected to evolve throughout the design process. The focus during conceptual design is of describing enough of the design to allow an examination of the design's suitability in meeting the system requirements contained in the Requirements Analysis and Specification Requirements document.

The intended audience of this document includes the prospective developers of the tool.

1.2 Scope

The software system to be produced is a projects management tool which will be referred to as Manage Project Homework (MPH) throughout this document.

MPH will allow professors to publish a project description and to define the set of the corresponding deliverables. It will allow students to join project teams and submit deliverables by uploading them into the system.

The professor will also be able to evaluate the project deliverables assigning a score to them and leaving to the system the computation of the final score based on the average of the individual scores. MPH will also provide some information sharing functionalities among different teams.

1.3 Definitions, acronyms, and abbreviations

MPH: Manage Project Homework, the software system to be produced.

Team: a set of 1, 2 or 3 students who work together on the same project.

Admin: the system administrator

Deliverable: the model of a tangible object produced as a result of a specific phase of the project.

Artifact: the effectively tangible object associated to a deliverable.

Deadline: the date by which the artifact associated to a deliverable must be delivered.

RASD: Requirements Analysis and Specification Document

DD: Design Document

DBMS: DataBase Management System

GUI: Graphical User Interface

RMI: Remote Method Invocation

JEE: Java Enterprise Edition

UX Diagram: User Experience Diagram

DTO: Data Transfer Object

1.4 References

- Description of the project: <http://corsi.metid.polimi.it>
- MPH RASD
- The Interactive User Experience (UX.html) dynamic web page attached to this document
- IEEE Std. 1016-2009: IEEE Standard for Information Technology—Systems Design— Software Design Descriptions: <http://ieeexplore...number=5167253>
- Carlo Ghezzi, Mehdi Jazayeri, Dino Mandrioli, Ingegneria del Software - Fondamenti e Principi, Pearson Education Italia, 2004

2 Functional overview of modules and components

The MPH system to be developed consists in a *Client-Server* application with *fully remote presentation*. The software is integrated with a database accessible by the Internet. The design process takes great care in designing a framework which can be updated easily based on a multi-tier architecture. There are three basic, logical components of the system: the Database Management System (Data Tier), the Server Module (Business Tier), and the Client Module (Presentation Tier). The system architecture follows the Java Enterprise Edition (JEE) paradigm.

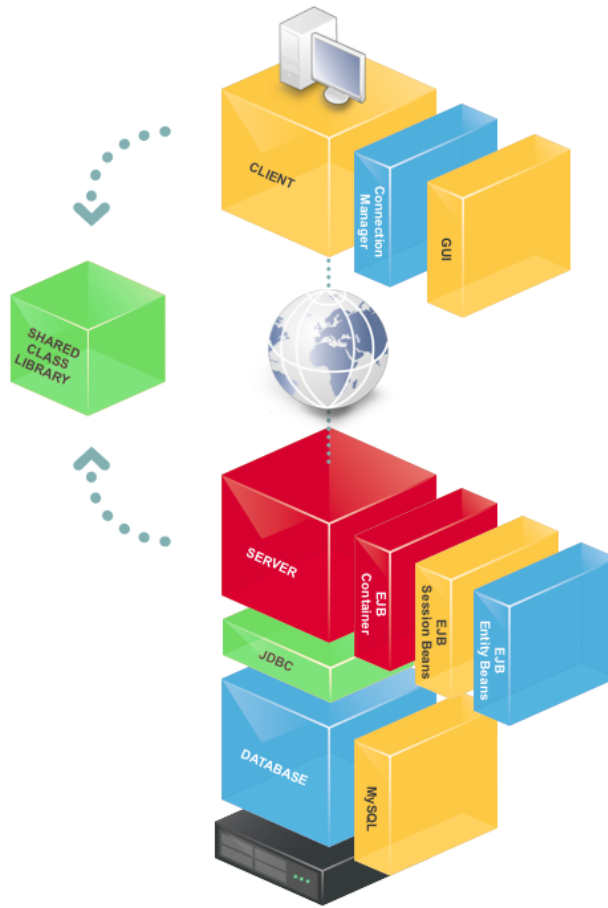


Figure 1: Architecture Logic Tiers

2.1 Database Management System

- MySQL will be used as COTS.
- Hosts the back-end database which is used for central data storage.
- The JEE framework will take care of its maintenance and updates.
- It is “virtually” hidden by the JEE framework, which provides an Database object-oriented management to the programmer.

2.2 Server Module

- Implemented in Java

- Provides methods and procedures that can be invoked remotely by a client application
- It is composed by a EJB Container
- Retrieves data from the Database using EJB Entity Beans
- Executes the business logic using EJB Session Beans
- Central process which can make all decisions that arise due to the distributed nature of this application
- The server can coordinate conflict resolution with the client application (which can ask the user questions, if necessary)

2.3 Client Module

- Implemented in Java
- Contains all presentation logic
- Interacts exclusively with the user
- Uses the class library package defined below
- Communicates with the server application through the server interface contained in the class library

2.4 Shared Class Library

- Java package which holds all Java classes (and "Java Beans") that both the server and client applications use to represent the data in memory
- Provides a uniform representation on both sides of the network connection, providing local methods for invoking remote methods on the server application hiding the details of how communication with the server is achieved
- Simplifies client/server communication for both the ends of communication
- Client applications can use these objects to represent data, separating it completely from the server-side representation
- Takes care of serializing the objects passed to and from the client application so they can be sent over the network
- If a change in communication strategy is desired, this would only require updating this class
- Updates to the package must be backwards compatible

3 Detailed System Architecture Description

The software will be deployed in two executable jars, one for the client, and one for the server. The **JBOSS** application server and the **MySQL DMBS** are needed to run on the server host.

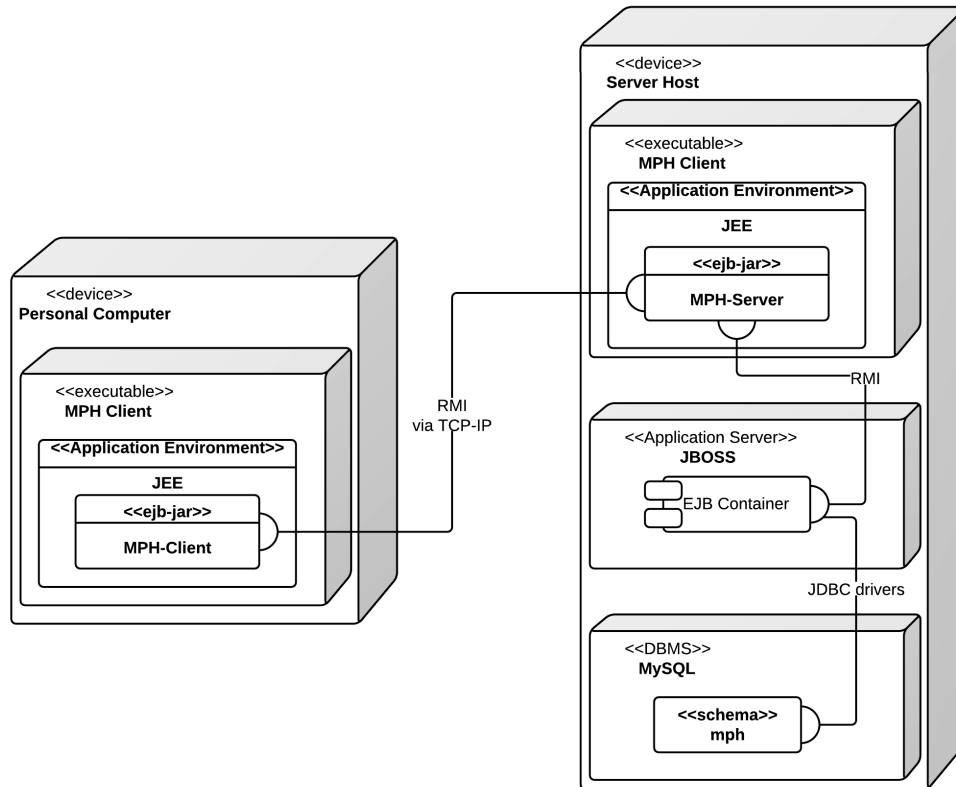


Figure 2: MPH Deployment diagram

The general modules of the software are now presented in a basic overview, followed by a package diagram of our system's major components.

Database Management System It will contain tables to represent the project data. We will be using MySQL as our database software.

Server Module The server software will directly manipulate the contents of the database, based on commands received from the Client.

Client Module The Client Software will reside on a desktop computer. Its purpose is to present data to the user as requested, and provide an interface

so that the user can easily update project information and other data fetched from the server.

Shared Class Library Module The shared class library will provide some common interfaces for the client and server module, as well as some common useful utility classes.

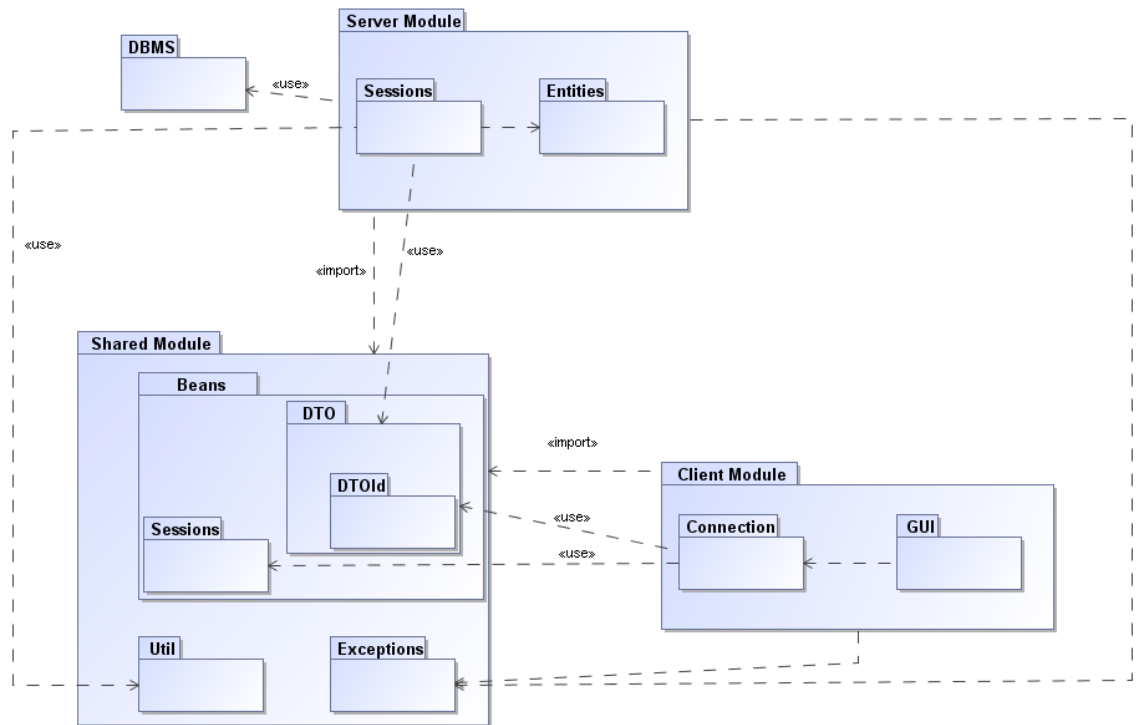


Figure 3: MPH general architecture

The following tables will provide more details on each module and component.

3.1 Database Management System

Identification	Database Management System Software
Type	Module
<i>Purpose</i>	Provides data management and storage for the server module.
<i>Function</i>	Takes EJQL/SQL QUERY and UPDATE commands from the server module, and stores or updates the data according to those commands.
<i>Subordinates</i>	To see the subordinates data structure, check the following E-R diagram.
<i>Dependencies</i>	A system administrator user must perform database setup functions, adding and modifying the structure of tables so that the server module can store the data as appropriate.
<i>Interfaces</i>	Apart from administrative setup, all modification of the database items will be performed via JDBC by the server application, exploiting the mapping between Entity Beans and Database Tables.
<i>Resources</i>	<p>The database that will be used is MySQL 5.0. All tables will be created by the software at run-time.</p> <p>The software module will connect to the database using JDBC connectivity. Once the server gets a request for a certain data type from a client, the server will connect to the database, execute the query and get a java ResultSet back. At this point, it will convert the ResultSet to the Object that was requested by the client and send it back to the client via RMI over the network.</p> <p>Calls to the database should not take longer than (but should be much less than) 30 seconds and transfers of the each Object over the network should not take longer than 1 minute. Conversion of a ResultSet to a Java Object should take less than 1 second. A client may also send an updated object to the server, in this case it will convert to a Set and then do an UPDATE to the database. This is a basic database call so it should not take longer than 1 second from start to finish.</p>
<i>Processing</i>	When the server module execute a request for the client, it will query the database for some information, then will perform some kind of operation and possibly modify the database state. For an example you can go and check a sample pseudocode (see A.1) in the appendix of this document.
<i>Data</i>	The data in the database will be filled by the server module except for the cases stated above (during extraordinary maintenance by the system administrator). We will use valid EJQL or SQL statements called within Java through JDBC to modify and update the data. In every case we will either get a ResultSet (for queries), a working return code or a SQLException if the command does not work for any reason. Exceptions will be handled according to their type.

Table 1: Database Management System detailed info table

3.1.1 Conceptual Entity-Relationship Diagram

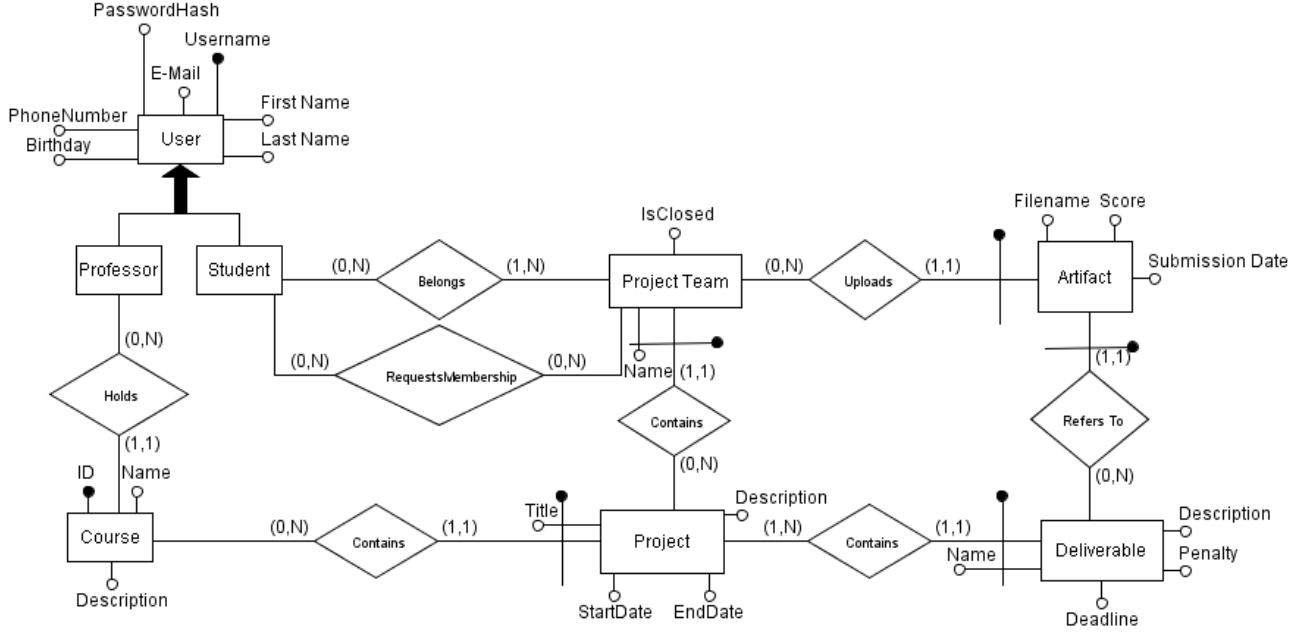


Figure 4: Conceptual Entity-Relationship Diagram

3.1.2 Logic Entity-Relationship Diagram

3.1.2.1 Operations The *User* generalization was removed by sub-classing it into the tables *Professor* and *Student*: although they shared the same data, they were deeply different respect to other relationships throughout the scheme, and to remark this difference, it was decided to keep them separated. Every weak entity, such as *Project Team*, *Project*, *Deliverable* and *Artifact* are modeled as a standalone entity that contains its own primary key and the primary keys of the entities on which they depend. The many-to-many *Belongs* relationship was resolved by creating a bridge table *StudentBelongsTeam* containing the primary keys of both the entities that joined the relationship. Other operations are self-explanatory.

3.1.2.2 Tables

- **Professor**(Username, PasswordHash, FirstName, LastName, Birthday, E-Mail, Phone Number)
- **Student**(Username, PasswordHash, FirstName, LastName, Birthday, E-Mail, Phone Number)

- **Course**(ID, Name, ProfessorUsername, Description)
- **Project**(CourseID, Title, StartDate, EndDate, Description)
- **Deliverable**(CourseID, ProjectTitle, Name, Deadline, Description, Penalty)
- **ProjectTeam**(CourseID, ProjectTitle, Name, IsClosed)
- **StudentBelongsTeam**(StudentUsername, TeamName, ProjectTitle, CourseID)
- **StudentRequestsMembership**(StudentUsername, TeamName, ProjectTitle, CourseID)
- **Artifact**(CourseID, ProjectTitle, ProjectTeamName, DeliverableName, FileName, SubmissionDate, Score)

3.1.2.3 Foreign Keys

- **Course**.*ProfessorUsername* → **Professor**.*Username*
- **Project**.*CourseID* → **Course**.*ID*
- **Deliverable**.(*CourseID*, *ProjectTitle*) → **Project**.(*CourseID*, *Title*)
- **ProjectTeam**.(*CourseID*, *ProjectTitle*) → **Project**.(*CourseID*, *Title*)
- **StudentBelongsTeam**.*StudentUsername* → **Student**.*Username*
- **StudentBelongsTeam**.(*CourseID*, *ProjectTitle*, *TeamName*) → **ProjectTeam**.(*CourseID*, *ProjectTitle*, *Name*)
- **StudentRequestsMembership**.*StudentUsername* → **Student**.*Username*
- **StudentRequestsMembership**.(*CourseID*, *ProjectTitle*, *TeamName*) → **ProjectTeam**.(*CourseID*, *ProjectTitle*, *Name*)
- **Artifact**.(*CourseID*, *ProjectTitle*, *TeamName*) → **ProjectTeam**.(*CourseID*, *ProjectTitle*, *Name*)
- **Artifact**.(*CourseID*, *ProjectTitle*, *DeliverableName*) → **Deliverable**.(*CourseID*, *ProjectTitle*, *Name*)

3.1.3 Detailed Table Description

3.1.3.1 Professor This table contains the *Professors* registered into the MPH system. Every professor is identified by a username. The *PasswordHash* attribute must contain the MD5 hash digest for the real password of the user. The other attributes represent the personal data of the professor.

- **Constraints**

- *NOT NULL Attributes*

- * *Username*
- * *Password*
- * *E-Mail*
- *Birthday* must be a *Date* posterior the 01/01/1900 and anterior the actual Timestamp
- **Deleting Policy**
 - Cascade
- **Updating Policy**
 - Cascade

3.1.3.2 Student This table contains the *Students* registered into the MPH system. Every student is identified by a username. The *PasswordHash* attribute must contain the MD5 hash digest for the real password of the user. The other attributes represent the personal data of the student.

- **Constraints**
 - *NOT NULL Attributes*
 - * *Username*
 - * *Password*
 - * *E-Mail*
 - *Birthday* must be a *Date* posterior the 01/01/1900 and anterior the actual Timestamp
- **Deleting Policy**
 - Cascade
- **Updating Policy**
 - Cascade

3.1.3.3 Course This table contains the *Courses* registered into the MPH system. Every course is identified by a unique ID and contains the username of the professor holding the course. It also has got a name and a description.

- **Constraints**
 - *NOT NULL Attributes*
 - * *ID*
 - * *ProfessorUsername*

- **Deleting Policy**

- Cascade

- **Updating Policy**

- Cascade

3.1.3.4 Project This table contains the *Projects* registered into the MPH system. Every course is identified by the corresponding course ID and a title. It contains a description as well as a start and an end date and flag which, if set to true, means the project is closed.

- **Constraints**

- *NOT NULL Attributes*

- *CourseID*

- *Title*

- *StartDate*

- *EndDate*

- **Deleting Policy**

- Cascade

- **Updating Policy**

- Cascade

3.1.3.5 Deliverable This table contains the *Deliverables* associated with a project. Every deliverable is identified by the title of the associated project, the course ID and the name. Every deliverable has also got a deadline, a description and a penalty.

- **Constraints**

- *NOT NULL Attributes*

- * *CourseID*

- * *ProjectTitle*

- * *Name*

- * *Deadline*

- * *Penalty*

- **Deleting Policy**

- Cascade

- **Updating Policy**

- Cascade

3.1.3.6 ProjectTeam This table contains the *Project Teams* enrolled in a project. Every team is identified by the title of the enrolled project, the course ID and the team name.

- **Constraints**

- *NOT NULL Attributes*
 - * *CourseID*
 - * *ProjectTitle*
 - * *Name*

- **Deleting Policy**

- Cascade

- **Updating Policy**

- Cascade

3.1.3.7 StudentBelongsTeam This table is a bridge between the Student table and the ProjectTeam table, it associates a team with every student belonging to it.

- **Constraints**

- *NOT NULL Attributes*
 - * *StudentUsername*
 - * *TeamName*
 - * *CourseID*
 - * *ProjectTitle*

- **Deleting Policy**

- Cascade

- **Updating Policy**

- Cascade

3.1.3.8 StudentRequestsMembership This table is a bridge between the Student table and the ProjectTeam table, it associates a team with every student requesting to join it.

- **Constraints**

- *NOT NULL Attributes*
 - * *StudentUsername*

- * *TeamName*
- * *CourseID*
- * *ProjectTitle*

- **Deleting Policy**

- Cascade

- **Updating Policy**

- Cascade

3.1.3.9 Artifact This table contains the *Artifacts* delivered by the teams for a project deliverable. Every artifact is identified by the project team key it and the project deliverable key. It also contains the name of the uploaded file, the submission date and the score given by the professor.

- **Constraints**

- *NOT NULL Attributes*

- * *CourseID*
- * *ProjectTitle*
- * *ProjectTeamName*
- * *DeliverableName*
- * *FileName*
- * *SubmissionDate*

- *Score* is an integer between 1 and 10

- **Deleting Policy**

- Cascade

- **Updating Policy**

- Cascade

3.2 Server Module

Identification	Entities
Type	Package
<i>Purpose</i>	Provides the classes that will be mapped into tuples in the corresponding tables of the database.
<i>Function</i>	This is simply a package of classes. Each class is a <i>JEE Entity</i> . An entity is a lightweight persistence domain object. Typically an entity represents a table in a relational database, and each entity instance corresponds to a row in that table. The persistent state of an entity is represented either through persistent fields or persistent properties. These fields or properties use object/relational mapping annotations to map the entities and entity relationships to the relational data in the underlying data store.
<i>Subordinates</i>	<p>Inside this package there are the following classes:</p> <ul style="list-style-type: none"> • Artifact • ArtifactId • Course • Deliverable • DeliverableId • Professor • Project • ProjectId • Team • TeamId • Student • User
<i>Dependencies</i>	The Server software use the class package to know the representation of every single table in the database, letting the programmer to manipulate objects instead of tables to get and modify data stored in the database. In particular, there is a huge amount of data requested and modified by the session beans class in the package <i>Sessions</i> .
<i>Interfaces</i>	All classes in the class library will implement the Serializable interface, so that they can easily be marshalled over the network by the server as well as saved to disk on the client. Each Entity will have its set of accessor and mutator methods.
<i>Resources</i>	The specific resources required for the class library are JDBC and Java's serialization mechanism.
<i>Processing</i>	None.
<i>Data</i>	None.

Table 2: Entities Package

3.2.1 User

Identification	User
Type	Class
<i>Purpose</i>	Represents an User entity and encapsulates all necessary data for storing an user into the database.
<i>Function</i>	<p>A user has the following data items. These are the field names as defined in the object class, but the database schema may have different column names. Each field will be accessed and mutated via getter and and setter methods.</p> <ul style="list-style-type: none"> • String username; • String password; • String firstName; • String lastName; • String email; • Date birthday; • String telephoneNumber; <p>All the attributes map the corresponding column in the database. For more information about their meaning, see the the Detailed Table Description on page 12.</p>
<i>Subordinates</i>	Defined in the <i>Function</i> sub-section.
<i>Dependencies</i>	For more information, see the Dependencies row in the Server Module package table on page 17.
<i>Interfaces</i>	Defined in the <i>Function</i> sub-section.
<i>Resources</i>	None.
<i>Processing</i>	None.
<i>Data</i>	Defined in the <i>Function</i> sub-section.

Table 3: User Class

3.2.2 Student

Identification	Student
Type	Class
<i>Purpose</i>	Represents a Student entity and encapsulates all necessary data for storing an user into the database.
<i>Function</i>	<p>The class <i>Student</i> is a subclass of the <i>User</i> class, thus it has the <i>User</i> data items plus the following data items. These are the field names as defined in the object class, but the database schema may have different column names.</p> <p>Each field will be accessed and mutated via getter and setter methods.</p> <ul style="list-style-type: none"> • Set<Team> studentTeams; • Set<Team> studentTeamEnrollingRequests; <p>This two fields represent the <i>StudentBelongsTeam</i> and <i>StudentRequestsMembership</i> tuple: they are in fact a set of values for each instance of student (whose instance represent a single tuple in the student table), thus realizing a many to many relationship. For more information about this two relationships, see the Detailed Table Description on page 12.</p>
<i>Subordinates</i>	Defined in the <i>Function</i> sub-section.
<i>Dependencies</i>	The Student class has an association with the Team class, because every student can belong to a particular team modeled by the <i>Team</i> class as shown in the Server Module package class diagram on page 17. For more information, see the Dependencies row in the Server Module package table on page 17.
<i>Interfaces</i>	Defined in the <i>Function</i> sub-section.
<i>Resources</i>	None.
<i>Processing</i>	None.
<i>Data</i>	Defined in the <i>Function</i> sub-section.

Table 4: Student Class

3.2.3 Professor

Identification	Student
Type	Class
<i>Purpose</i>	Represents a Professor entity and encapsulates all necessary data for storing an user into the database.
<i>Function</i>	<p>The class <i>Professor</i> is a subclass of the <i>User</i> class, thus it has the <i>User</i> data items plus the following data items. These are the field names as defined in the object class, but the database schema may have different column names.</p> <p>Each field will be accessed and mutated via getter and and setter methods.</p> <ul style="list-style-type: none"> • Set<Course> coursesHeld; <p>This field represent the one to many relationship between the Course entity and the Professor entity. For more information about this two relationships, see the the Detailed Table Description on page 12.</p>
<i>Subordinates</i>	Defined in the <i>Function</i> sub-section.
<i>Dependencies</i>	The Professor class has an association with the Course class, because every course is held by one and only one professor, as shown in the Server Module package class diagram on page 17. For more information, see the Dependencies row in the Server Module package table on page 17.
<i>Interfaces</i>	Defined in the <i>Function</i> sub-section.
<i>Resources</i>	None.
<i>Processing</i>	None.
<i>Data</i>	Defined in the <i>Function</i> sub-section.

Table 5: Professor Class

3.2.4 Course

Identification	Course
Type	Class
<i>Purpose</i>	Represents a Course entity and encapsulates all necessary data for storing an user into the database.
<i>Function</i>	<p>The class <i>Course</i> contains the following data items. These are the field names as defined in the object class, but the database schema may have different column names.</p> <p>Each field will be accessed and mutated via getter and and setter methods.</p> <ul style="list-style-type: none"> • protected Long uid; • protected String name; • protected String description; • protected Professor professor; <p>All the fields map the corresponding column in the database. In particular, the professor field maps the one to many association with the Professor entity. For more information about their meaning, see the the Detailed Table Description on page 12.</p>
<i>Subordinates</i>	Defined in the <i>Function</i> sub-section.
<i>Dependencies</i>	The Course class has an association with the Professor class, because every course is held by one and only one professor, as shown in the Server Module package class diagram on page 17. For more information, see the Dependencies row in the Server Module package table on page 17.
<i>Interfaces</i>	Defined in the <i>Function</i> sub-section.
<i>Resources</i>	None.
<i>Processing</i>	None.
<i>Data</i>	Defined in the <i>Function</i> sub-section.

Table 6: Course Class

3.2.5 Project

Identification	Project
Type	Class
<i>Purpose</i>	Represents a Project entity and encapsulates all necessary data for storing an user into the database.
<i>Function</i>	<p>The class <i>Project</i> contains the following data items. These are the field names as defined in the object class, but the database schema may have different column names.</p> <p>Each field will be accessed and mutated via getter and and setter methods.</p> <ul style="list-style-type: none"> • private ProjectId id; • private Date startDate; • private Date endDate; • private String description; <p>All the fields map the corresponding column in the database. For more information about their meaning, see the the Detailed Table Description on page 12. In particular, the id field maps the primary key of the Project table. For more information about the key, check the ProjectId on the following page</p>
<i>Subordinates</i>	Defined in the <i>Function</i> sub-section.
<i>Dependencies</i>	The Project class is related to the Team, Course, Deliverable classes, because every class listed before is linked to a particular project, as shown in the Server Module package class diagram on page 17. For more information, see the Dependencies row in the Server Module package table on page 17.
<i>Interfaces</i>	Defined in the <i>Function</i> sub-section.
<i>Resources</i>	None.
<i>Processing</i>	None.
<i>Data</i>	Defined in the <i>Function</i> sub-section.

Table 7: Project Class

3.2.6 ProjectId

Identification	ProjectId
Type	Class
<i>Purpose</i>	Represents a Project Identifier, an embeddable object representing the primary key of the Project entity.
<i>Function</i>	<p>The class <i>ProjectId</i> contains the following data items. These are the field names as defined in the object class, but the database schema may have different column names.</p> <p>Each field will be accessed and mutated via getter and setter methods.</p> <ul style="list-style-type: none"> • private Course course; • private String title; <p>The course field map the corresponding <i>entity</i> in the database, while the field title represent the name of the Project. For more information about their meaning, see the Entity Mapping in Hibernate and JBoss.</p>
<i>Subordinates</i>	Defined in the <i>Function</i> sub-section.
<i>Dependencies</i>	The ProjectId class is strictly related to the Project class, because it represent its identifier.
<i>Interfaces</i>	Defined in the <i>Function</i> sub-section.
<i>Resources</i>	None.
<i>Processing</i>	None.
<i>Data</i>	Defined in the <i>Function</i> sub-section.

Table 8: ProjectId Class

3.2.7 Team

Identification	Team
Type	Class
<i>Purpose</i>	Represents a Team entity and encapsulates all necessary data for storing an user into the database.
<i>Function</i>	<p>The class <i>Team</i> contains the following data items. These are the field names as defined in the object class, but the database schema may have different column names.</p> <p>Each field will be accessed and mutated via getter and setter methods.</p> <ul style="list-style-type: none"> • private TeamId id; • private boolean closed; • Set<Student> studentList; • Set<Student> studentEnrollingRequests; <p>All the fields map the corresponding column in the database. For more information about their meaning, see the Detailed Table Description on page 12. In particular, the id field maps the primary key of the Team table. For more information about the key, check the ProjectId on the preceding page</p>
<i>Subordinates</i>	Defined in the <i>Function</i> sub-section.
<i>Dependencies</i>	The Team class is related to the Project and Artifact classes, because every class is linked to a particular team, as shown in the Server Module package class diagram on page 17. For more information, see the Dependencies row in the Server Module package table on page 17.
<i>Interfaces</i>	Defined in the <i>Function</i> sub-section.
<i>Resources</i>	None.
<i>Processing</i>	None.
<i>Data</i>	Defined in the <i>Function</i> sub-section.

Table 9: Team Class

3.2.8 TeamId

Identification	TeamId
Type	Class
<i>Purpose</i>	Represents a Team Identifier, an embeddable object representing the primary key of the Team entity.
<i>Function</i>	<p>The class <i>TeamId</i> contains the following data items. These are the field names as defined in the object class, but the database schema may have different column names.</p> <p>Each field will be accessed and mutated via getter and setter methods.</p> <ul style="list-style-type: none"> • private Project project; • private String name; <p>The project field map the corresponding <i>entity</i> in the database, while the field name represent the name of the Team. For more information about their meaning, see the Entity Mapping in Hibernate and JBoss.</p>
<i>Subordinates</i>	Defined in the <i>Function</i> sub-section.
<i>Dependencies</i>	The TeamId class is strictly related to the Team class, because it represent its identifier.
<i>Interfaces</i>	Defined in the <i>Function</i> sub-section.
<i>Resources</i>	None.
<i>Processing</i>	None.
<i>Data</i>	Defined in the <i>Function</i> sub-section.

Table 10: TeamId Class

3.2.9 Deliverable

Identification	Deliverable
Type	Class
<i>Purpose</i>	Represents a Deliverable entity and encapsulates all necessary data for storing an user into the database.
<i>Function</i>	<p>The class <i>Deliverable</i> contains the following data items. These are the field names as defined in the object class, but the database schema may have different column names.</p> <p>Each field will be accessed and mutated via getter and and setter methods.</p> <ul style="list-style-type: none"> • private DeliverableId id; • private Date deadline; • private int penalty; • private String description; <p>All the fields map the corresponding column in the database. For more information about their meaning, see the the Detailed Table Description on page 12. In particular, the id field maps the primary key of the Deliverable table. For more information about the key, check the ProjectId on page 24</p>
<i>Subordinates</i>	Defined in the <i>Function</i> sub-section.
<i>Dependencies</i>	The Deliverable class is related to the Project and Artifact classes, because every deliverable is linked to a particular project and every artifact is linked to a particular deliverable, as shown in the Server Module package class diagram on page 17. For more information, see the Dependencies row in the Server Module package table on page 17.
<i>Interfaces</i>	Defined in the <i>Function</i> sub-section.
<i>Resources</i>	None.
<i>Processing</i>	None.
<i>Data</i>	Defined in the <i>Function</i> sub-section.

Table 11: Deliverable Class

3.2.10 DeliverableId

Identification	DeliverableId
Type	Class
<i>Purpose</i>	Represents a Deliverable Identifier, an embeddable object representing the primary key of the Deliverable entity.
<i>Function</i>	<p>The class <i>DeliverableId</i> contains the following data items.</p> <p>These are the field names as defined in the object class, but the database schema may have different column names.</p> <p>Each field will be accessed and mutated via getter and setter methods.</p> <ul style="list-style-type: none"> • private Project project; • private String name; <p>The project field map the corresponding <i>entity</i> in the database, while the field name represent the name of the Deliverable. For more information about their meaning, see the Entity Mapping in Hibernate and JBoss.</p>
<i>Subordinates</i>	Defined in the <i>Function</i> sub-section.
<i>Dependencies</i>	The DeliverableId class is strictly related to the Deliverable class, because it represent its identifier.
<i>Interfaces</i>	Defined in the <i>Function</i> sub-section.
<i>Resources</i>	None.
<i>Processing</i>	None.
<i>Data</i>	Defined in the <i>Function</i> sub-section.

Table 12: DeliverableId Class

3.2.11 Artifact

Identification	Artifact
Type	Class
<i>Purpose</i>	Represents an Artifact entity and encapsulates all necessary data for storing an user into the database.
<i>Function</i>	<p>The class <i>Artifact</i> contains the following data items. These are the field names as defined in the object class, but the database schema may have different column names.</p> <p>Each field will be accessed and mutated via getter and and setter methods.</p> <ul style="list-style-type: none"> • private ArtifactId id; • private Date submissionDate; • private int score; • private String fileName; <p>All the fields map the corresponding column in the database. For more information about their meaning, see the the Detailed Table Description on page 12. In particular, the id field maps the primary key of the Artifact table. For more information about the key, check the ProjectId on page 24</p>
<i>Subordinates</i>	Defined in the <i>Function</i> sub-section.
<i>Dependencies</i>	The Deliverable class is related to the Deliverable and Team classes, because every artifact is linked to a particular deliverable and every artifact is linked to a particular team who created it, as shown in the Server Module package class diagram on page 17. For more information, see the Dependencies row in the Server Module package table on page 17.
<i>Interfaces</i>	Defined in the <i>Function</i> sub-section.
<i>Resources</i>	None.
<i>Processing</i>	None.
<i>Data</i>	Defined in the <i>Function</i> sub-section.

Table 13: Artifact Class

3.2.12 ArtifactId

Identification	ArtifactId
Type	Class
<i>Purpose</i>	Represents an Artifact Identifier, an embeddable object representing the primary key of the Artifact entity.
<i>Function</i>	<p>The class <i>DeliverableId</i> contains the following data items.</p> <p>These are the field names as defined in the object class, but the database schema may have different column names.</p> <p>Each field will be accessed and mutated via getter and setter methods.</p> <ul style="list-style-type: none"> • private Deliverable deliverable; • private Team team; <p>The deliverable and the team fields map the corresponding <i>entity</i> in the database. For more information about their meaning, see the Entity Mapping in Hibernate and JBoss.</p>
<i>Subordinates</i>	Defined in the <i>Function</i> sub-section.
<i>Dependencies</i>	The ArtifactId class is strictly related to the ArtifactId class, because it represent its identifier.
<i>Interfaces</i>	Defined in the <i>Function</i> sub-section.
<i>Resources</i>	None.
<i>Processing</i>	None.
<i>Data</i>	Defined in the <i>Function</i> sub-section.

Table 14: ArtifactId Class

3.2.13 Session Beans

Identification	Sessions
Type	Package
<i>Purpose</i>	Provides the classes that will control and modify the entity beans in the package Server Module on page 17.
<i>Function</i>	This is simply a package of packages. Each package contains a particular type of <i>JEE Session bean</i> . A session bean performs operations, such as calculations or database access, for the client, querying for entities and modifying them. The session contained here can query and modify data related to a professor's tasks or student's tasks.
<i>Subordinates</i>	<p>Inside this package there are the following packages:</p> <ul style="list-style-type: none"> • Local • Remote
<i>Dependencies</i>	The Server software use the class package to manipulate every single table in the database, letting the programmer to manipulate objects instead of tables to get and modify data stored in the database.
<i>Interfaces</i>	All classes in the class library will implement the Serializable interface, so that they can easily be marshalled over the network by the server as well as saved to disk on the client. Each Entity will have its set of accessor and mutator methods.
<i>Resources</i>	The specific resources required for the class library are JDBC, Java's serialization mechanism, and the RMIIO library to transfer data transfer objects (DTO's).
<i>Processing</i>	None.
<i>Data</i>	None.

Table 15: Entities Package

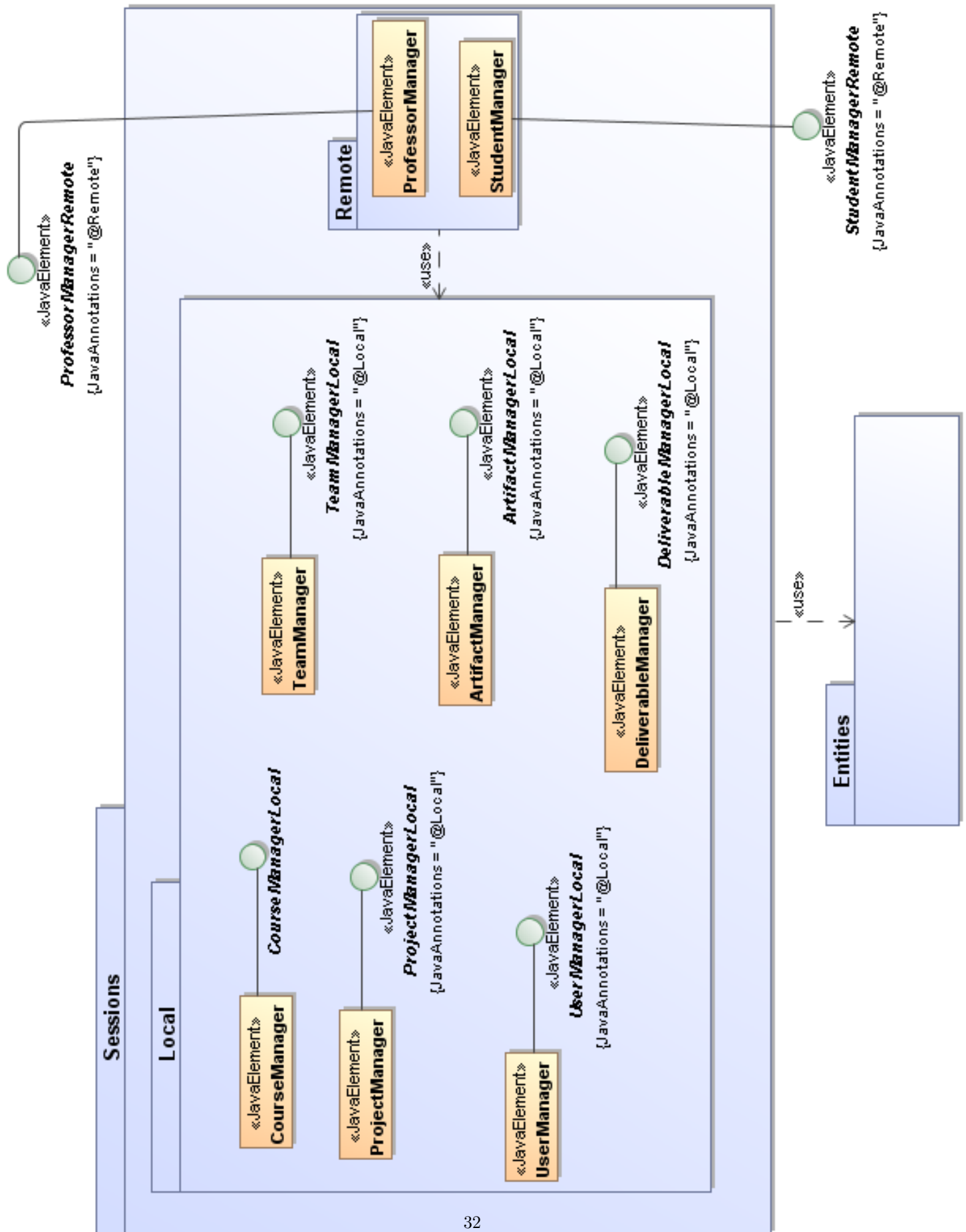


Figure 6: Sessions Class Diagram

3.2.14 Local

Identification	Sessions.Local
Type	Package
<i>Purpose</i>	Provides the classes that will control and modify the entity beans in the package Server Module on page 17.
<i>Function</i>	This is simply a package of classes. Each class is a <i>local JEE Session bean</i> . A local session bean performs operations, such as calculations or database access, and can be accessed only by the server .
<i>Subordinates</i>	<p>Inside this package there are the following classes:</p> <ul style="list-style-type: none"> • UserManager • CourseManager • ProjectManager • DeliverableManager • TeamManager • ArtifactManager
<i>Dependencies</i>	The Server software use the class package to manipulate every single table in the database, letting the programmer to manipulate objects instead of tables to get and modify data stored in the database. These classes will be used by other stateful beans in other to reuse the code between different stateful sessions.
<i>Interfaces</i>	All classes implements their local interfaces, provided in the same package, ad is identified by the name of the class with the suffix "Local". For example, the interface of UserManager is located in the interface UserManagerLocal.
<i>Resources</i>	The specific resources required for the class library are JDBC, Java's serialization mechanism, and the RMIO library to transfer data transfer objects (DTO's).
<i>Processing</i>	None.
<i>Data</i>	None.

Table 16: Sessions.Local Package

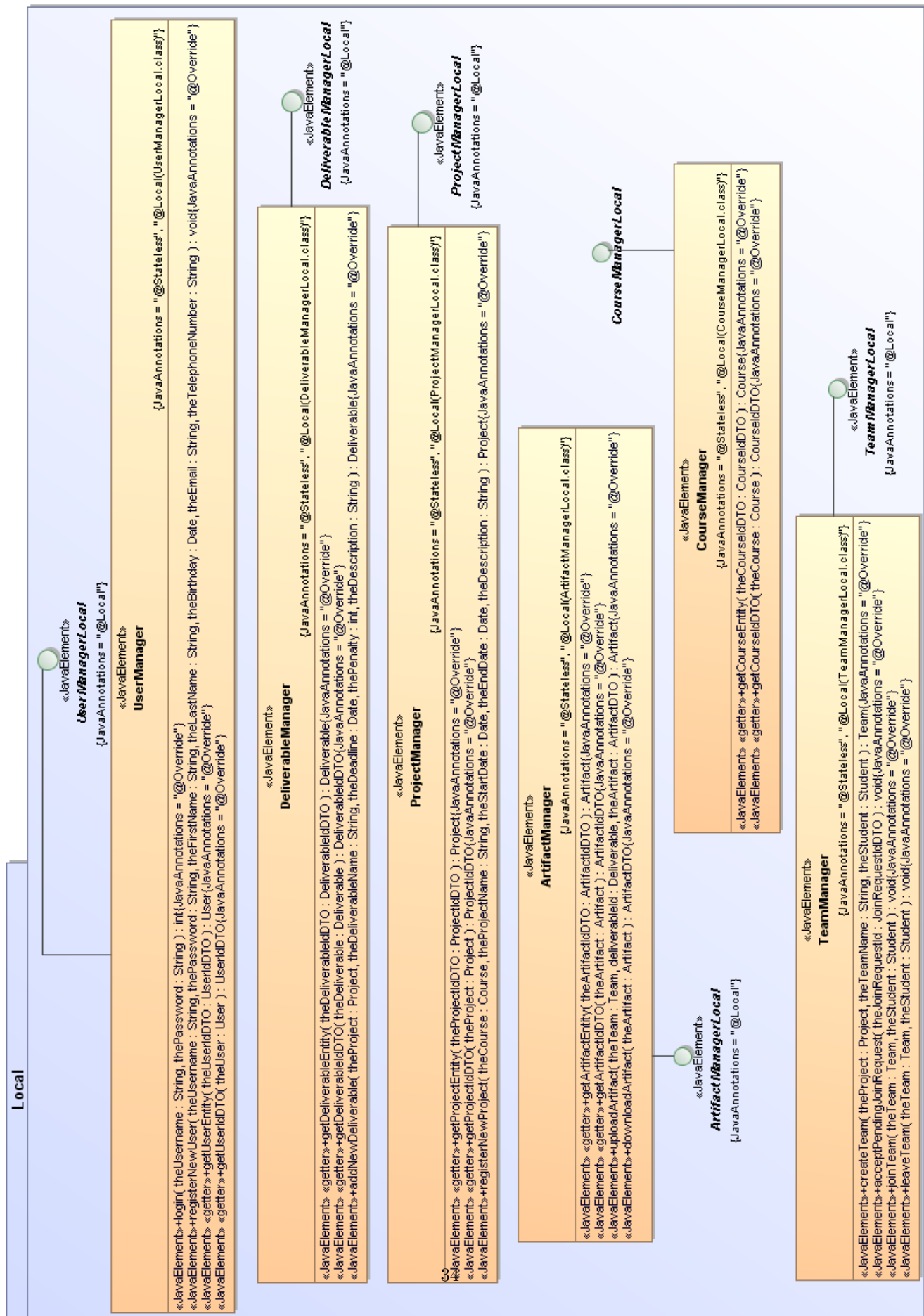
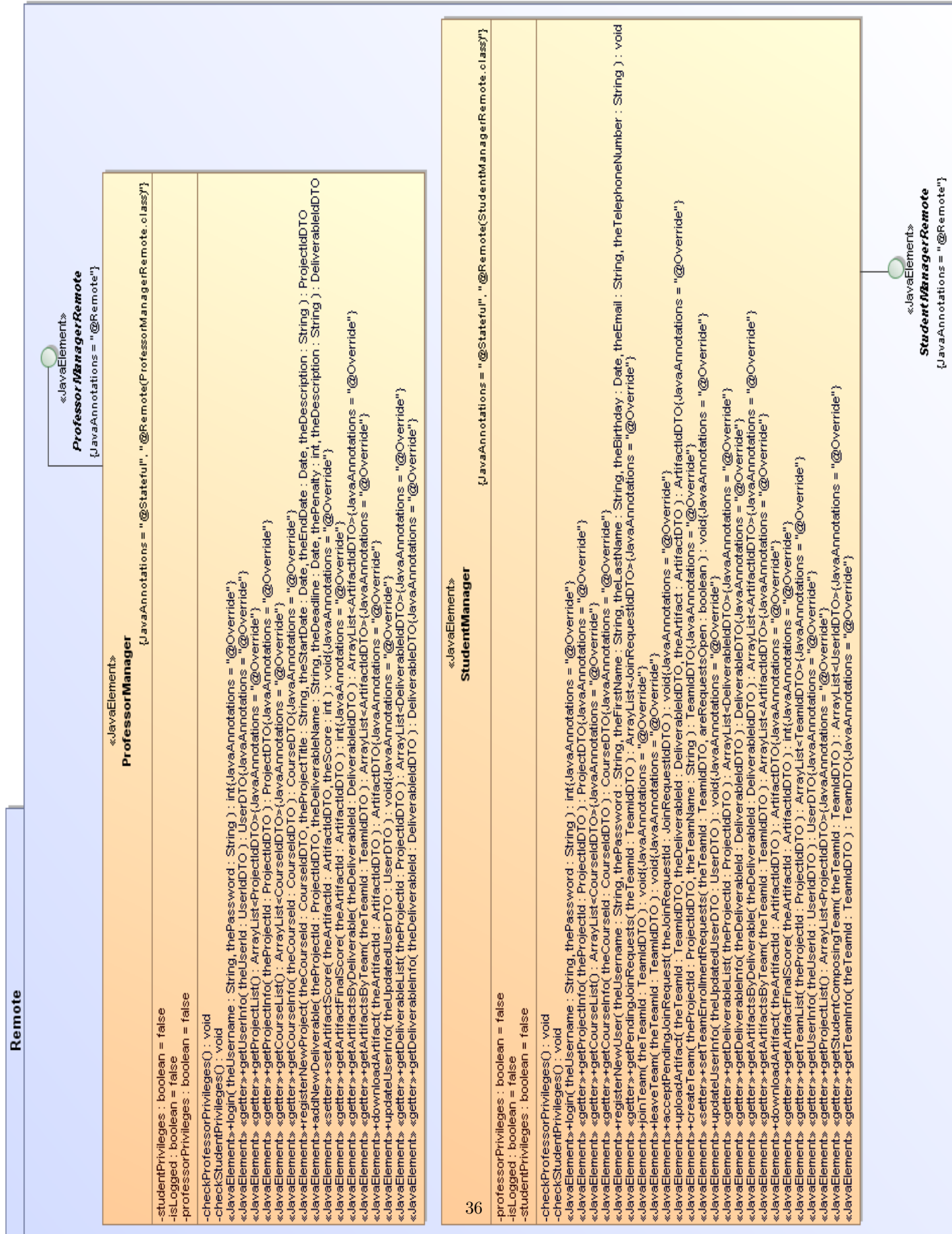


Figure 7: Sessions.Local Class Diagram

3.2.15 Remote

Identification	Sessions.Remote
Type	Package
<i>Purpose</i>	Provides the classes that will execute the client required actions.
<i>Function</i>	This is simply a package of classes. Each class is a <i>remote JEE Session bean</i> . A remote session bean performs operations, such as calculations or database access, and can be accessed by the client.
<i>Subordinates</i>	Inside this package there are the following packages: <ul style="list-style-type: none"> • StudentManager • ProfessorManager
<i>Dependencies</i>	The Server software use the class package to manipulate every single table in the database, letting the programmer to manipulate objects instead of tables to get and modify data stored in the database. The classes will provide an implementation of the logic business of the software.
<i>Interfaces</i>	All classes in the package implement a shared interface with the client, contained in the shared class library.
<i>Resources</i>	The specific resources required for the class library are JDBC, Java's serialization mechanism, and the RMIO library to transfer data transfer objects (DTO's).
<i>Processing</i>	None.
<i>Data</i>	None.

Table 17: Sessions.Remote Package



3.3 Shared Module

Identification	Shared
Type	Module
<i>Purpose</i>	This module contains classes for the interaction between the Server and Client modules. It provides Data Transfer Objects and a Server Interface for the client, which will use to get receive data from the Server.
<i>Function</i>	<ul style="list-style-type: none"> • Provide DTO which contains the data transferred during the communication between the Server and Client modules • Contain the exceptions classes used by the system modules • Provide a Server interface for the Client
<i>Subordinates</i>	<p>It contains the following packages:</p> <ul style="list-style-type: none"> • DTO • Exceptions • Sessions • Util
<i>Dependencies</i>	The Shared Module does not depend on any other system modules, instead the other two modules depend on this one.
<i>Interfaces</i>	<ul style="list-style-type: none"> • ProfessorManagerRemote.java • StudentManagerRemote.java • UserManagerRemote.java
<i>Resources</i>	The RMIIIO library used for file transfer.
<i>Processing</i>	None.
<i>Data</i>	Defined in the <i>Function</i> sub-section.

Table 18: Shared Module

3.3.1 Data Transfer Objects

Identification	dto
Type	Package
<i>Purpose</i>	A Transfer Object is used to encapsulate the business data. A single method call is used to send and retrieve the Transfer Object. When the client requests the enterprise bean for the business data, the enterprise bean can construct the Transfer Object, populate it with its attribute values, and pass it by value to the client.
<i>Function</i>	<ul style="list-style-type: none"> • Provide DTO which contains the data transferred during the communication between the Server and Client modules • Contain the exceptions classes used by the system modules • Provide a Server interface for the Client
<i>Subordinates</i>	<p>It contains the dto.ids Package (3.3.2) and the following classes:</p> <ul style="list-style-type: none"> • ArtifactDTO.java • CourseDTO.java • DeliverableDTO.java • ProjectDTO.java • TeamDTO.java • UserDTO.java
<i>Dependencies</i>	The Client will use DTO ID's to identify a DTO and retrieve information from it, so the class that will implement the ProfessorManagerRemote and StudentManagerRemote interfaces will use these DTO to send and receive data.
<i>Interfaces</i>	They all implement the Serializable interface.
<i>Resources</i>	None.
<i>Processing</i>	None.
<i>Data</i>	DTO are initialized by their constructors, the data are given as parameters.

Table 19: DTO Package

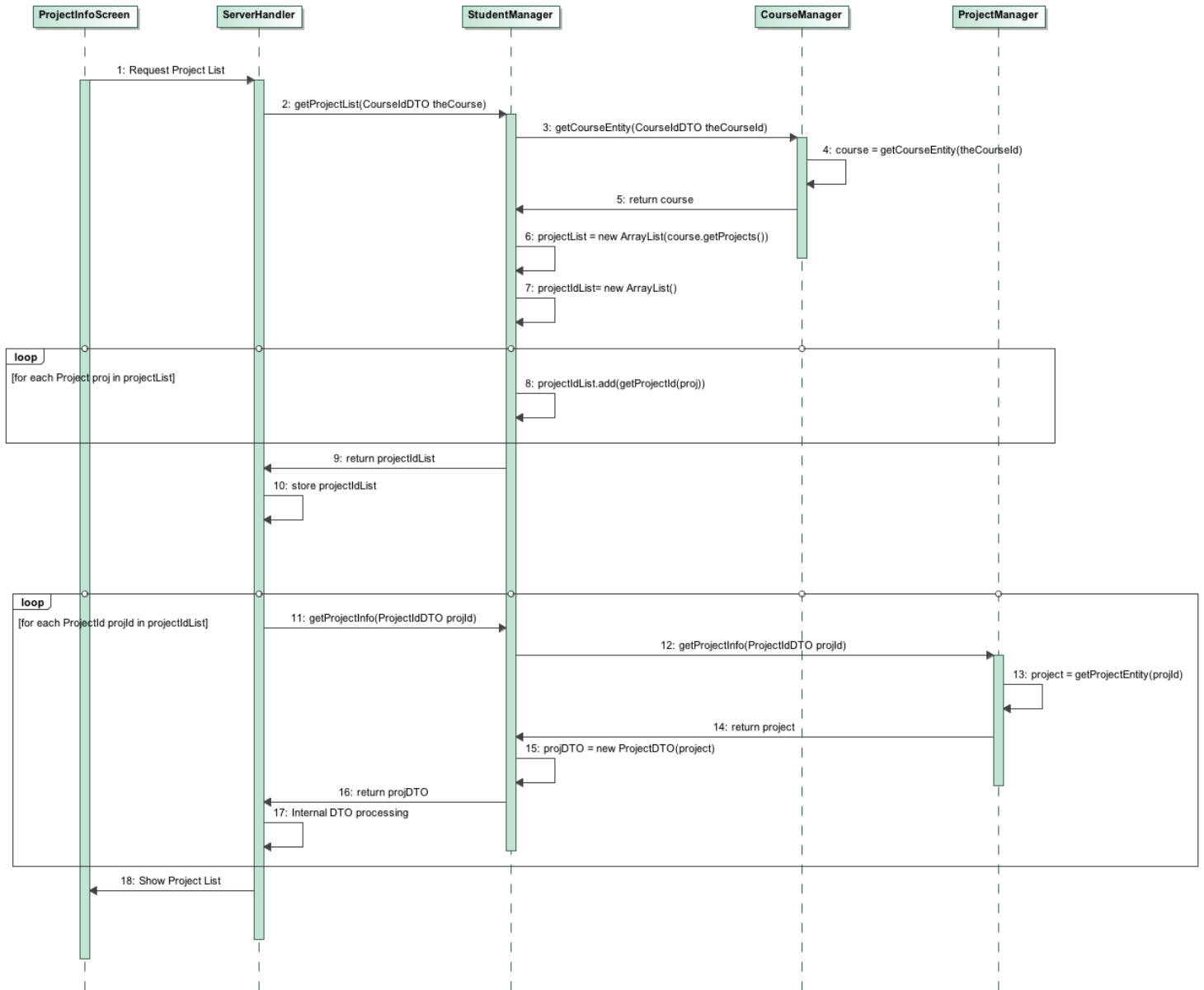


Figure 9: Example of DTO passing between Client and Server

3.3.2 Data Transfer Objects Identifiers

Identification	dto.ids
Type	Package
<i>Purpose</i>	When an enterprise bean uses a Transfer Object, the client makes a single remote method invocation to the enterprise bean to request the Transfer Object instead of numerous remote method calls to get individual attribute values. The enterprise bean then constructs a new Transfer Object instance, copies values into the object and returns it to the client. The client receives the Transfer Object through the DTO ID and can then invoke getter methods on the Transfer Object to get the individual attribute values from the Transfer Object. Because the Transfer Object is passed by value to the client, all calls to the Transfer Object instance are local calls instead of remote method invocations.
<i>Function</i>	This is simply a package of classes. Each class is a <i>Data Transfer Object ID</i> .
<i>Subordinates</i>	<p>Inside this package there are the following classes:</p> <ul style="list-style-type: none"> • ArtifactIdDTO • CourseIdDTO • DeliverableIdDTO • JoinRequestIdDTO • ProfessorIdDTO • ProjectIdDTO • StudentIdDTO • TeamIdDTO • UserIdDTO
<i>Dependencies</i>	The Client will use DTO ID's to identify a DTO and retrieve information from it.
<i>Interfaces</i>	They all implements the <i>Serializable</i> interface.
<i>Resources</i>	None.
<i>Processing</i>	None.
<i>Data</i>	DTO are initialized by their constructors, the data are given as parameters.

Table 20: DTO Package

3.3.3 Exceptions

AlreadyLoggedInException occurs when there is an attempt to login using the credentials of a user already logged in

CourseNotHeldException occurs when a professor tries to execute operations for a course which it is not held by him/her

DownloadErrorException occurs when there is an error during the download of an artifact by a Client

InvalidEmailException occurs when the inserted email address does not respect the syntax local-part@domain

EntityNotFoundException occurs when the searched entity is not found in the Database

FailedRegistrationException occurs when the registration procedure fails

FilePathNotFoundException occurs when the file path of an artifact is incorrect

ForbiddenOperationException occurs when the entity invoking a method does not have the rights to perform that operation

InconsistentDateException occurs when there is an insertion of date inconsistent with the current date or the ones stored in the Database

NegativeValueException occurs when a negative value is inserted for data which are positive by definition

PasswordMismatchException occurs when the password used to log in does not match with the one stored in the Database

ProjectNotOwnedException occurs when a professor tries to execute operations for a project whose he/she is not the owner

StudentNotEnrolledException occurs when a student tries to execute operations in a project team that he/she does not belong to

TeamClosedException occurs when there is an attempt to join a team which does not accept any new membership requests

UploadErrorException occurs when there is an error during the upload of an artifact to the Server

3.3.4 Sessions

Identification	sessions
Type	Package
<i>Purpose</i>	This package contains the interfaces implemented by the Server session beans. They are grouped by functionality, this ensures the modularization of the system, making easy for a developer to add functionalities by simply adding new interfaces. This decomposition makes sure that the Client does not know the real methods invoked but only the list of functionalities it can access.
<i>Function</i>	<ul style="list-style-type: none"> • Provide protected remote interface extended by the Professor and Student remote interfaces
<i>Subordinates</i>	<p>It contains the following sets of interfaces:</p> <ul style="list-style-type: none"> • Artifact 3.3.4.1 • Course 3.3.4.2 • Deliverable 3.3.4.3 • Project 3.3.4.4 • Team 3.3.4.5 • User 3.3.4.6 <p>and the remote interfaces which extend the needed interfaces:</p> <ul style="list-style-type: none"> • ProfessorManagerRemote.java • StudentManagerRemote.java
<i>Dependencies</i>	The Server session beans implement all the methods declared in the functionalities interfaces.
<i>Interfaces</i>	ProfessorManagerRemote and StudentManagerRemote implement different functionalities interfaces according to the specifications.
<i>Resources</i>	None.
<i>Processing</i>	None.
<i>Data</i>	None.

Table 21: Sessions Package

3.3.4.1 Artifact

- ArtifactEditingFunctionalities.java
- ArtifactEvaluatorFunctionalities.java
- ArtifactViewerFunctionalities.java

3.3.4.2 Course

- CourseViewerFunctionalities.java

3.3.4.3 Deliverable

- DeliverableEditingFunctionalities.java
- DeliverableViewerFunctionalities.java

3.3.4.4 Project

- ProjectEditingFunctionalities.java
- ProjectViewerFunctionalities.java

3.3.4.5 Team

- TeamBasicFunctionalities.java
- TeamCreationFunctionalities.java
- TeamManagementFunctionalities.java
- TeamViewerFunctionalities.java

3.3.4.6 User

- UserEditingFunctionalities.java
- UserLoginFunctionalities.java
- UserRegisteringFunctionalities.java
- UserViewerFunctionalities.java

3.4 Client Module

Identification	Client
Type	Module
<i>Purpose</i>	The Client is the application used by the users (students and professor) to connect to the MPH system and manage their project stuff. It provides a GUI and a RMI connection manager to send commands to the Server.
<i>Function</i>	<ul style="list-style-type: none"> • Let a professor publish a new project and evaluate the artifacts delivered by the corresponding project teams • Let a student register and log into the MPH system • Let a student manage his/her belonging project team and upload artifacts corresponding to the project deliverables • Send commands to the Server to get and save the required data
<i>Subordinates</i>	Connection Package 3.4.1 Student GUI Package 3.4.2 Professor GUI Package 3.4.3
<i>Dependencies</i>	This component will interact closely with the Server Module for the retrieval of data. Server calls will be synchronous, so there will be no timing issues for the Client to consider. The Client will use various data classes and interfaces found in the Shared Class Library. Data from the Server will be packaged into one of these classes, and the Client can then access that data through the public methods of an appropriate class. See 3.3.1.
<i>Interfaces</i>	The Client uses Swing GUI classes for interaction with the user. The screen formats for these windows are shown in the User Experience Report. It uses the ProfessorManagerRemote and StudentManagerRemote interfaces for interaction with the Server. See 3.3.4.
<i>Resources</i>	It will contain the Shared Module in the build path as it will invoke some of the methods contained in the ProfessorManagerRemote and StudentManagerRemote. See 3.3.4.
<i>Processing</i>	None.
<i>Data</i>	Data received from the user are stored locally then sent via RMI to the Server, which will save them permanently in the Database. Viceversa data requested by the user are sent via RMI by the Server Module.

Table 22: Client Module

3.4.1 Connection Package

Identification	Connection
Type	Package
<i>Purpose</i>	This package manages the communication between Client and Server applications This package is responsible to establish a RMI connection with the MPH Server.
<i>Function</i>	<ul style="list-style-type: none"> • Provide methods to establish a connection between a Client and the MPH Server
<i>Subordinates</i>	<ul style="list-style-type: none"> • Connection Manager
<i>Dependencies</i>	Client applications will use this package to retrieve and synchronize data with the server. RMI will be used for the communication thus this package has to invoke methods present in the Shared Class Library.
<i>Interfaces</i>	It calls methods present in the remote interfaces of the Shared module. See 3.3.4
<i>Resources</i>	The communication will use the TCP/IP network protocol, relying on Java's RMI. It also relies on Java's ObjectOutputStream class and Java's serialization mechanism.
<i>Processing</i>	None.
<i>Data</i>	The remote objects containing the data to send or to receive will be serialized

Table 23: Connection Package

3.4.2 Student GUI Package

Identification	GUI.Student
Type	Package
<i>Purpose</i>	This package contains all the classes used for the GUI
<i>Function</i>	<p>It provides a graphical interface to:</p> <ul style="list-style-type: none"> • let the professor publish a new project • let the professor evaluate the artifacts delivered by the project teams
<i>Subordinates</i>	RegistrationScreen 3.4.4 LoginScreen 3.4.5 StudentMainScreen 3.4.6 StudentsScreen 3.4.7 ProjectSelectionScreen 3.4.8 ProjectInfoScreen 3.4.9
<i>Dependencies</i>	The GUI will use the Client Module to get and put data on the server.
<i>Interfaces</i>	The GUI uses Swing classes for interaction with the user. The screen formats for these windows are shown in the User Experience Report.
<i>Resources</i>	<p>The GUI will not have any significantly CPU or memory-intensive tasks however, normal overhead associated with Swing components is to be expected in both cases.</p> <p>Much of the project data to be queried will be accessed on a remote database.</p>
<i>Processing</i>	See the following UX diagram.
<i>Data</i>	The data inserted by the users are sent to the Server via RMI which will store them permanently in the Database.

Table 24: GUI.Student Package

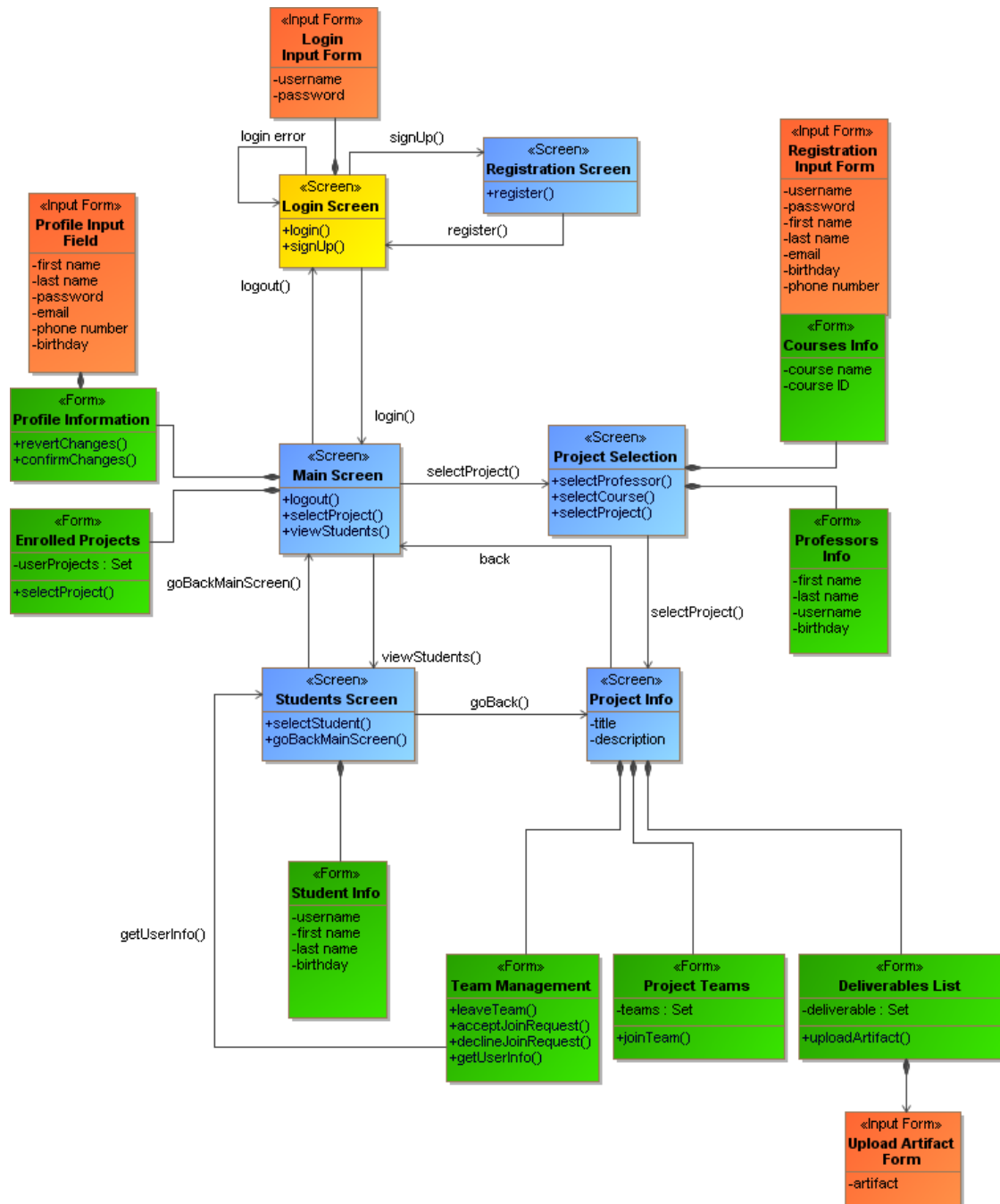


Figure 10: Student UX Diagram

3.4.3 Professor GUI Package

Identification	GUI.Professor
Type	Package
<i>Purpose</i>	This package contains all the classes used for the GUI
<i>Function</i>	<p>It provides a graphical interface to:</p> <ul style="list-style-type: none"> • let the student join project teams and manage them • let the student upload a new artifact
<i>Subordinates</i>	LoginScreen 3.4.5 ProfessorMainScreen 3.4.10 NewProjectScreen 3.4.11 TeamInformationScreen 3.4.13 Delivered ArtifactsScreen 3.4.12 ProjectInfoScreen 3.4.9
<i>Dependencies</i>	The GUI will use the Client Module to get and put data on the server.
<i>Interfaces</i>	The GUI uses Swing classes for interaction with the user. The screen formats for these windows are shown in the User Experience Report.
<i>Resources</i>	<p>The GUI will not have any significantly CPU or memory-intensive tasks however, normal overhead associated with Swing components is to be expected in both cases.</p> <p>Much of the project data to be queried will be accessed on a remote database.</p>
<i>Processing</i>	See the following UX diagram.
<i>Data</i>	The data inserted by the users are sent to the Server via RMI which will store them permanently in the Database.

Table 25: GUI.Professor Package

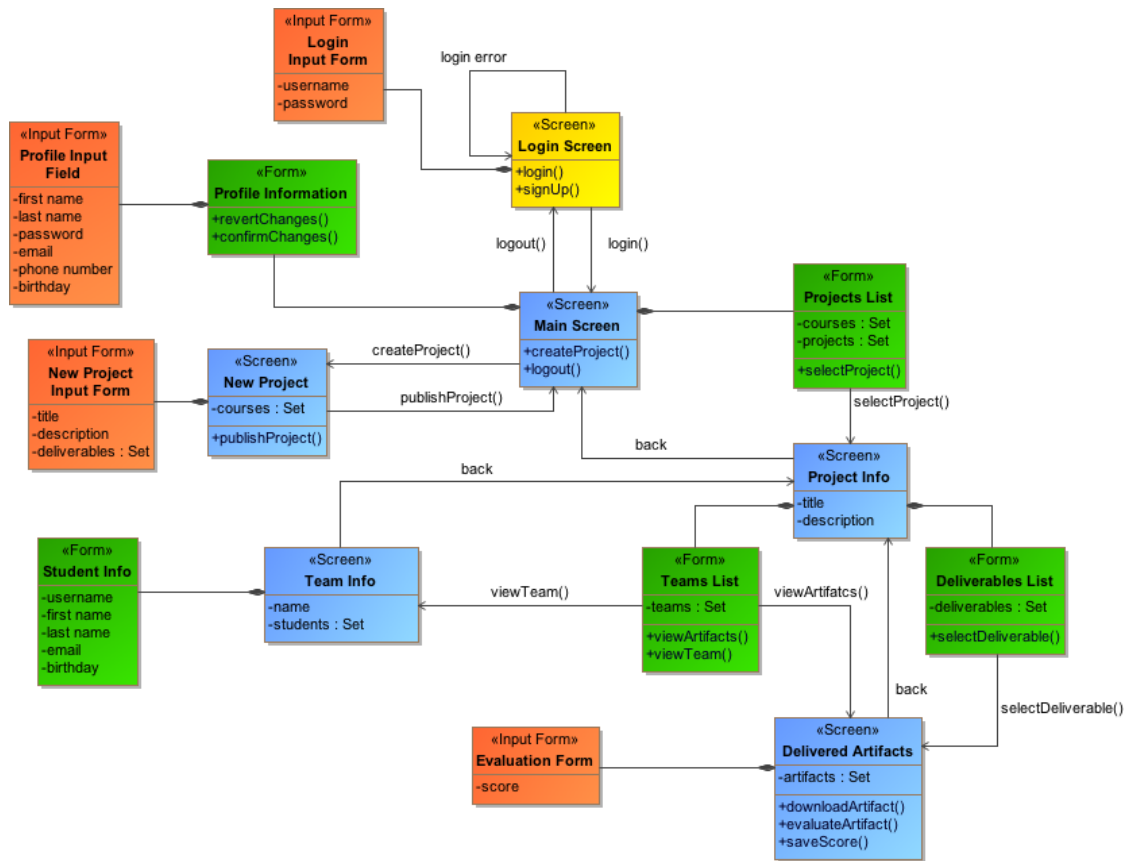


Figure 11: Professor UX Diagram

3.4.4 RegistrationScreen

Identification	Registration Screen
Type	Class
<i>Purpose</i>	This lets the Student register to the MPH system
<i>Function</i>	<ul style="list-style-type: none">• Register a new student into the MPH system• Send to the server application the student registration data via RMI
<i>Subordinates</i>	It contains <ul style="list-style-type: none">• a username text field• two password fields for security reasons• optional text fields for personal information• a “Register” button which takes the student to the Login Screen
<i>Dependencies</i>	Same as other GUI interfaces.
<i>Interfaces</i>	Same as other GUI interfaces.
<i>Resources</i>	Same as other GUI interfaces.
<i>Processing</i>	See User Experience Report.
<i>Data</i>	The student username, password and personal data are serialized and sent to the server which will store them in the Database.

Table 26: Registration Screen

3.4.5 LoginScreen

Identification	LoginScreen
Type	Class
<i>Purpose</i>	This lets a registered Student log into the MPH system
<i>Function</i>	<ul style="list-style-type: none">• Log in a student into the MPH system• Send to the server application the student login credentials via RMI
<i>Subordinates</i>	It contains <ul style="list-style-type: none">• username and password text fields editable by the user• a “Log In” button which, if authentication succeeds, takes the student to the Main Screen
<i>Dependencies</i>	Same as other GUI interfaces.
<i>Interfaces</i>	Same as other GUI interfaces.
<i>Resources</i>	Same as other GUI interfaces.
<i>Processing</i>	See UX.html.
<i>Data</i>	The student username and password are serialized and sent to the server which will check them against the credentials saved in the Database.

Table 27: Login Screen

3.4.6 StudentMainScreen

Identification	StudentMainScreen
Type	Class
<i>Purpose</i>	This shows the student the MPH resources he/she can access
<i>Function</i>	<ul style="list-style-type: none"> • Let a student edit his/her personal data • Let a student manage the project teams which he/she belongs to • Let the student search for information about other students, courses and professors
<i>Subordinates</i>	<p>It contains</p> <ul style="list-style-type: none"> • a “Profile Information” form with personal information and buttons to edit them • a “Select Project” button which takes the student to the Project Selection Screen • a “Enrolled Projects” form containing a list of clickable projects which take the student to the Project Info Screen • a “Search for Students” button which takes the student to the Students Screen • a “Log Out” button which takes the student to the Login Screen
<i>Dependencies</i>	Same as other GUI interfaces.
<i>Interfaces</i>	Same as other GUI interfaces.
<i>Resources</i>	Same as other GUI interfaces.
<i>Processing</i>	See User Experience Report.
<i>Data</i>	No data are exchanged in this class.

Table 28: Student Main Screen

3.4.7 StudentsScreen

Identification	Students Screen
Type	Class
<i>Purpose</i>	This shows the student personal information about other students registered in the MPH system
<i>Function</i>	<ul style="list-style-type: none">• Let a student view other student's personal data
<i>Subordinates</i>	<p>It contains</p> <ul style="list-style-type: none">• a search form field• a "Search button"• a list of students matching the search criteria (default: list all students)• the student names are clickable and take the student to the Student Info Screen• a "Back" button which takes the student to the Main Screen
<i>Dependencies</i>	Same as other GUI interfaces.
<i>Interfaces</i>	Same as other GUI interfaces.
<i>Resources</i>	Same as other GUI interfaces.
<i>Processing</i>	See User Experience Report.
<i>Data</i>	The search criteria are sent to the Server which will send the corresponding results taken from the Database

Table 29: Students Screen

3.4.8 ProjectSelectionScreen

Identification	ProjectSelectionScreen
Type	Class
<i>Purpose</i>	This lets the student view a list of projects referring to a selected course held by a professor
<i>Function</i>	<ul style="list-style-type: none">• Let a student view a list of projects
<i>Subordinates</i>	<p>It contains</p> <ul style="list-style-type: none">• a list of selectable professors; clicking on one of them opens a form containing the professor information• a list of selectable courses held by the selected professor; clicking on one of them opens a form containing the professor information• a list of clickable projects referring to the selected projects which take the student to the Project Info Screen• a “Back” button which takes the student to the Main Screen
<i>Dependencies</i>	Same as other GUI interfaces.
<i>Interfaces</i>	Same as other GUI interfaces.
<i>Resources</i>	Same as other GUI interfaces.
<i>Processing</i>	See User Experience Report.
<i>Data</i>	The search criteria are sent to the Server which will send the corresponding results taken from the Database.

Table 30: Project Selection Screen

3.4.9 ProjectsInfoScreen

Identification	ProjectsInfoScreen
Type	Class
<i>Purpose</i>	This shows the student the project information, allowing him/her to view the project deliverables, manage the project team he/she belongs to or join a new project team.
<i>Function</i>	<ul style="list-style-type: none"> • Let a student view the project information and deliverables • Let a student see the list of teams referring to a project and join one of them • Let a student manage his/her project team
<i>Subordinates</i>	<p>This contains</p> <ul style="list-style-type: none"> • the description of the previously selected project • a “Deliverables List” form containing <ul style="list-style-type: none"> – a list of deliverables – a “Upload Artifact” button next to each deliverable which lets the student upload a file from his/her computer • a “Team Management” form containing <ul style="list-style-type: none"> – a list of incoming membership requests by other students – a “Accept Join Request” next to an incoming membership request – a “Decline Join Request” next to an incoming membership request – a “Leave Team” button which lets the student leave the project team • a “Project Teams” form containing <ul style="list-style-type: none"> – a list of teams enrolled in the previously selected project – a “Join Team” button which sends a new Membership Request to the team members • a “Back” button which takes the student to the Main Screen
<i>Dependencies</i>	Same as other GUI interfaces.
<i>Interfaces</i>	Same as other GUI interfaces.
<i>Resources</i>	Same as other GUI interfaces.
<i>Processing</i>	See User Experience Report.
<i>Data</i>	The project data are sent by the server which took them from the Database.

Table 31: Projects Info Screen

3.4.10 ProfessorMainScreen

Identification	ProfessorMainScreen
Type	Class
<i>Purpose</i>	This shows the professor the MPH resources he/she can access
<i>Function</i>	<ul style="list-style-type: none"> • Let a professor see the list of courses held by him/her • Let the professor publish a new project
<i>Subordinates</i>	<p>It contains</p> <ul style="list-style-type: none"> • a frame containing the professor personal information which he/she can edit • a “Publish New Project” button which takes the professor to the New Project Screen • a list of courses and the corresponding projects published by him/her. Selecting a project takes the professor to the Project Info Screen • a “Log Out” button which takes the professor to the Login Screen
<i>Dependencies</i>	Same as other GUI interfaces.
<i>Interfaces</i>	Same as other GUI interfaces.
<i>Resources</i>	Same as other GUI interfaces.
<i>Processing</i>	See User Experience Report.
<i>Data</i>	No data are exchanged in this class.

Table 32: Professor Main Screen

3.4.11 NewProjectScreen

Identification	NewProjectScreen
Type	Class
<i>Purpose</i>	This allows the professor to publish a new project for the desired course so the students can create teams to develop the project.
<i>Function</i>	<ul style="list-style-type: none"> • Let a professor publish a new project for a course • Let the professor define the title and deliverables for the new project
<i>Subordinates</i>	<p>It contains</p> <ul style="list-style-type: none"> • a list of selectable courses held by the professor • text fields for the project title, deliverables' deadlines' dates and description • a "Publish" button which takes the professor to the Main Screen • a "Back" button which takes the professor to the Main Screen
<i>Dependencies</i>	Same as other GUI interfaces.
<i>Interfaces</i>	Same as other GUI interfaces.
<i>Resources</i>	Same as other GUI interfaces.
<i>Processing</i>	See UX.html.
<i>Data</i>	The new project details are sent to the Server via RMI, which will store them permanently in the Database.

Table 33: New Project Screen

3.4.12 DeliveredArtifactsScreen

Identification	DeliveredArtifactsScreen
Type	Class
<i>Purpose</i>	This allows the professor to view the delivered files per deliverable type or per teams and to evaluate them.
<i>Function</i>	<ul style="list-style-type: none"> • Let the professor download the artifacts delivered by the project teams • Let the professor evaluate the delivered artifacts
<i>Subordinates</i>	<p>It contains</p> <ul style="list-style-type: none"> • a list of delivered artifacts corresponding to the selected deliverable or artifacts delivered by the selected team • a “Evaluate” button next to each delivered artifact which opens a small text field where the professor can evaluate the artifact with a score from 1 to 10 • a “Save” button appear after the “Evaluate” button is pressed allowing the professor to save the assigned score • a “Download” button next to each delivered artifact which lets the professor download the artifact • a “Back” button which takes the professor to the Main Screen
<i>Dependencies</i>	Same as other GUI interfaces.
<i>Interfaces</i>	Same as other GUI interfaces.
<i>Resources</i>	Same as other GUI interfaces.
<i>Processing</i>	See User Experience Report.
<i>Data</i>	The delivered artifacts can be downloaded via RMI. The scores assigned by the professor are sent to the server which will store them permanently.

Table 34: Delivered Artifacts Screen

3.4.13 TeamInfoScreen

Identification	TeamInfoScreen
Type	Class
<i>Purpose</i>	This shows the professor information about the students belonging to the teams enrolled in a project.
<i>Function</i>	<ul style="list-style-type: none">• Let a professor view all project teams and students who are part of them
<i>Subordinates</i>	It contains <ul style="list-style-type: none">• a list of selectable projects of the courses held by the professor• a list of teams and students enrolled in the selected project• a “Back” button which takes the professor to the Main Screen
<i>Dependencies</i>	Same as other GUI interfaces.
<i>Interfaces</i>	Same as other GUI interfaces.
<i>Resources</i>	Same as other GUI interfaces.
<i>Processing</i>	See User Experience Report.
<i>Data</i>	The information about the project teams and students are sent by the Server which has taken the data from the Database.

Table 35: Team Information Screen

4 Reuse and relationship to other products

4.1 Java and Java tools

4.1.1 Programming Language

The customer forced the use of the Java programming language and specifically the JEE architecture.

4.1.2 External Libraries

The developer team set the goal of making use of any existing code or COTS to avoid wasting time reinventing the wheel. The team decided to use open source or freeware solutions wherever possible. Because the team is creating software for a defined application, it is possible to use open source technologies in each area.

4.1.2.1 RMIIO The developer team decided to to make use of the **RMIIO** library in order to exchange files between the Server and Client modules.

RMIIO is an open source library which makes it as simple as possible to stream large amounts of data using the RMI framework. The RMI framework makes it very easy to implement remote communication between Java programs but it lacks an immediate solution for the exchange of files over the network.

What is really needed to do is stream data from the Client to the Server, using a framework which does not really expose a streaming model. The RMIIO library was written to fill in that missing gap in the RMI framework. It provides some very powerful classes which enable a Client to stream data to the Server using only a few extra lines of code.

4.1.3 IDE

The developer team chose a development environment that would allow to edit the Java code. **Eclipse** was chosen because it is free and open source and has many plug-ins which allow the use of already existing applications.

The team chose the free Eclipse plug-in **WindowBuilderPro** to create the GUI because it allows to drag and drop frames and edit the generated code all within the same environment.

4.2 Database and Reporting

The project required a Database to store the records. The developer team chose MySQL, because it is open source, has many management tools, and includes **MySQL Workbench**, a DBMS management tool that provides a visual console to easily administer MySQL environments and gain better visibility into databases in a easy but still powerful way.

5 Design Decisions and Tradeoffs

5.1 Three Tier Design

Deciding how to judiciously divide the project between all team members was another design issue. We finally decided on a 3 tier design, which is an application program organized into three major parts: the Database Management System (Data Tier), the Server Module (Business Tier), and the Client Module (Presentation Tier).

A 3-tier application uses the client/server computing model. Since the Database is “virtually” included in the Server module the Client and Server can be developed concurrently by a different programmer. Because the programming for a tier can be changed or relocated without affecting the other tiers, the 3-tier model makes it easier for an enterprise or software packager to continually evolve an application as new needs and opportunities arise. Existing applications or critical parts can be permanently or temporarily retained and encapsulated within the new tier of which it becomes a component. This design idea was very appealing to our team, especially for portability purposes.

A Pseudocode

A.1 DBMS Processing

```
1  public TeamIdDTO createTeam(ProjectIdDTO theProjectId, String theTeamName)
2      throws IllegalArgumentException,
           ForbiddenOperationException {
3
4      //Preliminary operations...
5
6      //Query the database
7      Query q = manager.createQuery("FROM Project p WHERE p.id.course.uid =
           :n AND p.id.title = :m");
8      //Set query parameters
9      q.setParameter("n", aCourseId);
10     q.setParameter("m", aProjectTitle);
11
12     try {
13         //Execute query and check the results
14         if (q.getResultList().isEmpty() == false) {
15
16             Project aProject = (Project) q.
17                 getSingleResult();
18             Team newTeam = new Team(aProject, theTeamName
19                 );
20
21             //Modify the database state
22             manager.persist(newTeam);
23             manager.flush();
24         }
25     } catch (EntityNotFoundException exc) { //Some operations... }
26     catch (NonUniqueResultException exc) { //Some operations... }
27
28     //Aftermath operations...
29
30 }
```