Software Architecture Specification

Group 4

Group Members:

Nico Taljaard (10153285)
Neels van Rooyen (29052735)
Stephan Viljoen (11008408)
Martin Scoeman (10651994)
Jacques Lewis (28183488)
Uteshlen Nadesan (28163304)
Moeletji Semenya (12349136)

Version 1.0

03/03/2013	Version 1.0	Document Created	Nico Taljaard
12/03/2013	Version 1.0	API UML Diagrams	Neels van Rooyen
12/03/2013	Version 1.0	System Class Diagrams	Nico Taljaard
12/03/2013	Version 1.0	Framework & Technologies	Stephan Viljoen
13/03/2013	Version 1.0	Database ERD & Description	Uteshlen Nadesan
13/03/2013	Version 1.0	Protocals & Libraries	Martin Scoeman
13/03/2013	Version 1.0	Sequence Diagrams	Jacques Lewis
13/03/2013	Version 2.0	Added Diagrams & Edited layout	Nico Taljaard

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to declare and illustrate all lower levels of the system giving a more in-depth view of how the system should be implemented. This is done by designing UMLs, class diagrams and ERDs to depicted how each subsystem should be create as well as how the interconnection between components are to be create and utilized.

1.2 Related Documents

Master Requirement Specification: Provided by the University of Pretoria

2 System Description

These software architecture design decisions are based on the constraints defined by the master requirement specification document:

2.1 Technologies

Programming Languages

- Python
  - A high-level object oriented programming language that can be used for scripting,prototyping, testing and very easy to learn and implement.

- HTML
  - The system must be accessible to users and HTML Markup language will be used to for creating structured displayable content in a web browsers (Mozilla Firefox, Google Chrome, Apple Safari and Microsoft Internet Explorer). Reference 4.1.1.
- CSS
  - Used for formatting of HTML documents creating rich web interfaces.
- JavaScript
  - Dynamic programming language used for client side interaction.
- SQL
  - Used for data managing (insert, delete, update and read) of your relational data using the Object-Relational Mapper.
- Java
  - A Powerful language with vast libraries with which the android interface will be created. Reference 4.1.2

Application Server

- A Django application server will be used to run and deploy the system within the cs.up.ac.za Apache web server. The webserver must be published as SOAP–based.

Database Technology MySQL implement the quality requirements stated by the master specification and it decouples the system from the database.

- Scalability and Flexibility:
  Provides scalability, sporting the capacity to handle deeply embedded applications
  Platform flexibility (Linux, UNIX, Mac and Windows)

- Security
  Offering exceptional features that ensure absolute data protection when looking at database authentication, backup and recovery.

2.2 Frameworks

Object-Relational Mapper

- A technique for converting data between incompatible types in object-oriented languages. With our application data to outlive the applications process. The persistence to the relational database must be done using the Object-Relational Mapper bundled with Django. Django is the chosen Object-Relational mapper used to obtain, retrieve and transfer data between a variety of platforms such as from the MySQL database to the web client and android interface. You can define your data model in Python and access through the database API.

Web Service Frame Work

- The Django framework is based on the model view controller, which is used for creating complex database driven websites with the ability of reusability of components and rapid development. The key quality requirements that Django accommodate are scalability, security, reliability and integration at a lesser extent.

2.3 Protocols

The protocols we will be using are HTTPS for security, LDAP for the computer science web site, XML and JSON.

The two main protocols for data transfer in a web application environment are JSON and XML. We will be using JSON as our main data transfer protocol, but also have support for XML.
The reasons will be explained in accordance with the quality requirements.

Security:

JSON in limited to storing only classical data like numbers and text, but with XML you can store any data type you want. That could be a security risk, because you can store an executable as well.

Auditability:

The simplicity of JSON and the fact that it's only text base, makes it very easy to audit.

Testability:

JSON and XML are easy to test, because both of their implementation is self describing and is cross platform independent.

Usability:

Both XML and JSON is human readable, but JSON is more so. The reason is the JSON files are more restrictive and a lot less data formats that it supports. JSON uses an array to store data, where XML uses a tree. Seeing as arrays are more a structure of object oriented languages than trees are, it will be much easier the parse the data in the array than in the tree.

Scalability:

Using a JSON document, without a schema makes it very easy to scale the JSON document. JSON also has no max size or max length of object. So you can send and receive a document without being depended on size. The limitations of the size will be determined by the server or browser.

Performance:

The performance of JSON is very high, because of the fact it is only text base. So there will be no support for large files like images, videos and so forth.

Conclusion:

JSON is selected as our main protocol, because it only supports text and no other formats. Seeing how our system will only be transferring text, JSON is the best option. Also JSON is simpler to use, easier to read and many people are changing from XML to JSON, because it is the newer technology.

2.4 Libraries

The libraries we will be using, will mostly if not all consist of open source libraries.

PDF generator:

The system must be able to generate PDFs on two different platforms, android and with a web service. For android we will be using the iText library that supports creating a PDF on android. For our server side, we will be using a PHP library called TCPDF, which will allow us to generate a PDF server side.

JSON marshalling/de-marshalling:

For our JSON marshalling/de-marshalling, we will be using the library EclipseLink MOXy. It works with XML, but also works great with JSON. There is a lot of support with this library, if one needed help.

LDAP integration:

We will be using django-auth-ldap 1.1.4 for the Django authentication backend that will authenticate against an LDAP service. For python we will be using the python-ldap library to allow us to use python to communicate with LDAP.

3 Overall Architecture

3.1 Architectural Strategies

3.1.1 Thread Pooling:

3.1.2 Clustering:

3.1.3 Interception:

3.1.4 Run-Time Lookups:

3.1.5 Queuing:

3.2 Architectural Patterns

3.2.1 Layering:

3.2.2 Model-View-Controller(MVC:

3.2.3 Pipes and Filters:

3.3 Reference Architectures

3.4 Technology and Framework Selection

3.4.1 Database Servers

3.4.2 Web Server

4 Details of System

4.0.3 Class Diagram

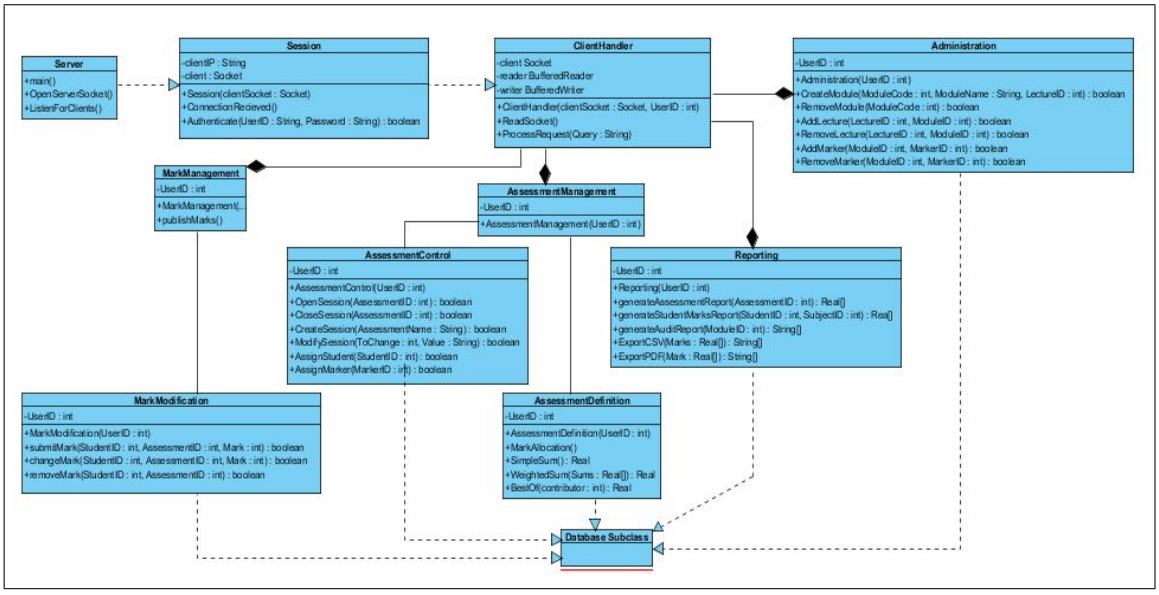

Figure 1: System Class Diagram

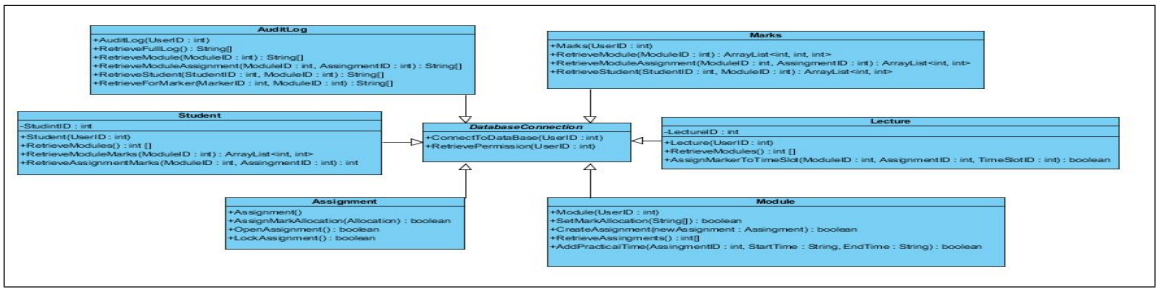

Figure 2: Database Class Diagram

4.1 Database Design

The Database software that is to be used is MySQL. The design above will be implemented in MySQL. The population of the database will be taken from the LDAP database of the Department of Computer Science. These will include, but not limited to, student numbers, names, surnames, courses, lecturers' details, etc.

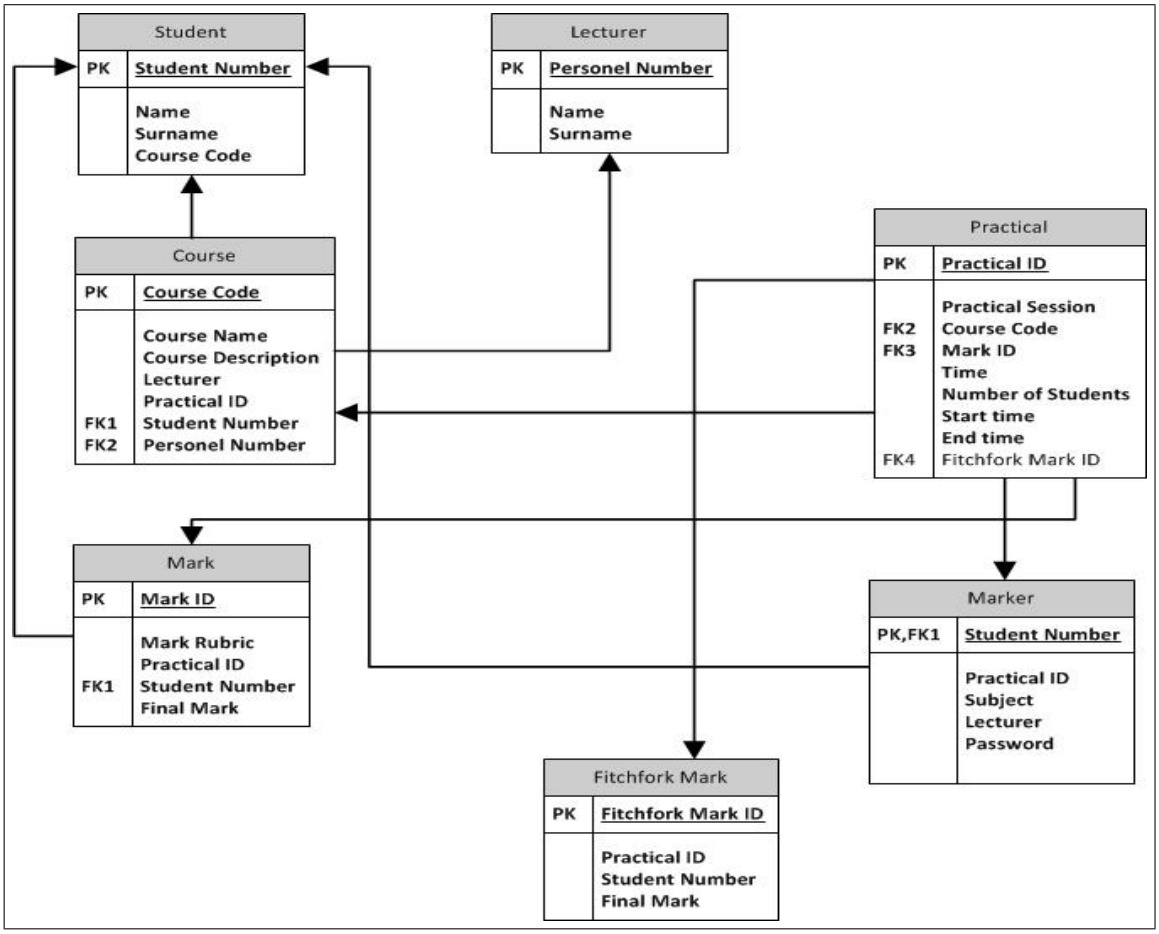

Figure 3: Database ERD

- Student
  - All student details will be imported and added to complete the student table of the database. All students have a unique student number that identifies them, names, surnames and courses can then be added. Adding these details will enable access to all additional tables.
- Practical
  - The Practical table has the session for that specific practical the number of students assigned to that practical, the Fitchfork marks that can be added although it is not mandatory. The start and end times are also added as they are essential for the application to know when to open and close editing of the database. The practicals are indirectly linked to the lecturers who will have access to their subject's practicals.
- Mark
  - A table of marks will capture the marks for each student along with the final mark each student will receive for the practical. Marks are linked to the student and the practical itself via the student number and practical identification number. This will simplify access, not only for the students, but also for the markers and lecturers combined.
- Marker
  - Markers are in a separate database and include a password specific to that marker for accessing the android application and include the overseeing lecturer and subject and practical the marker is assigned to.
- Fitchfork
  - Marks that need to be imported from Fitchfork will be done separately and then added to the above database tables to complete the database entries for the practical.

- Lecturer
  - All courses that require marks to be entered into this database will have a lecturer assigned to the course and hence, the practical.

4.2 Logic View

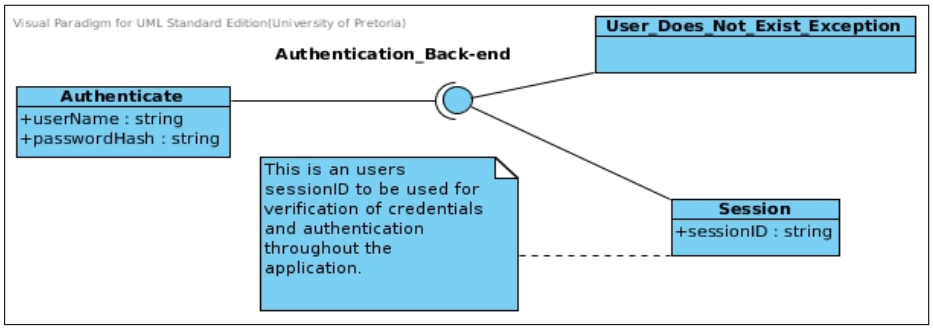

Figure 4: Authentication API

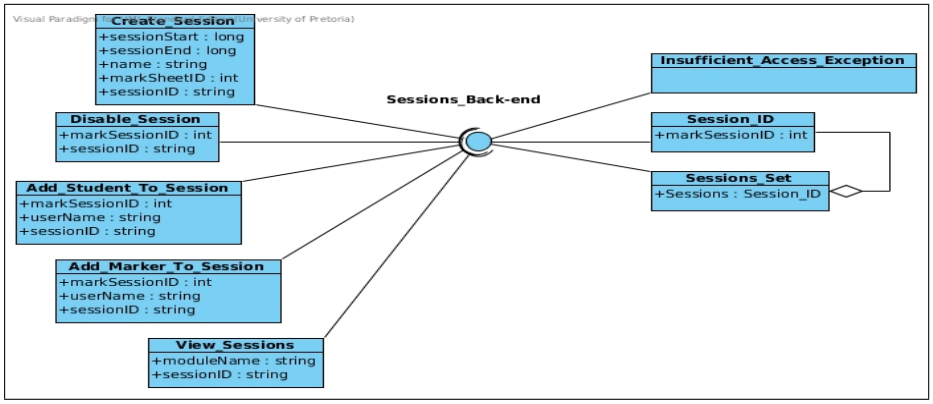

Figure 5: Sessions API

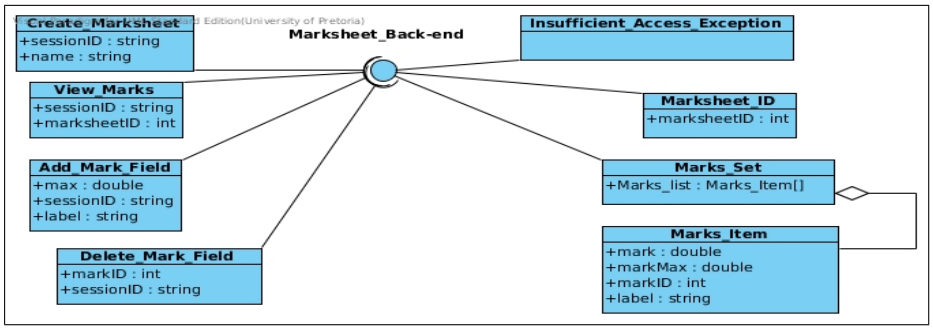

Figure 6: Marksheet API

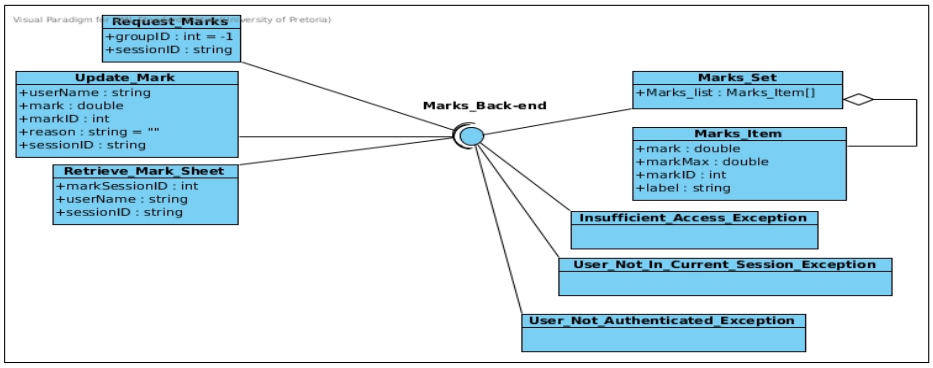

Figure 7: Marks API

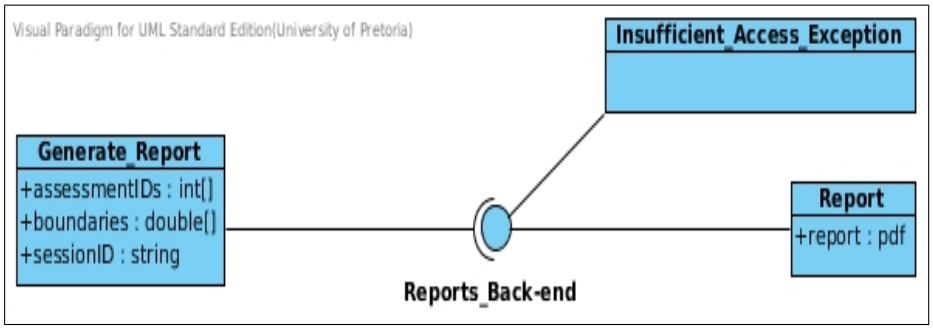

Figure 8: Reports API

4.2.1 Communication Diagram

4.2.2 Sequence Diagram

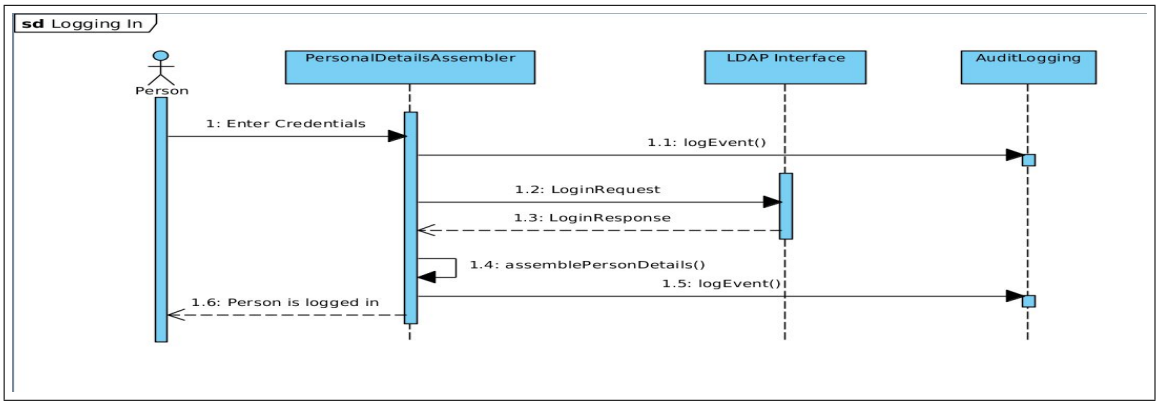

Figure 9: Login

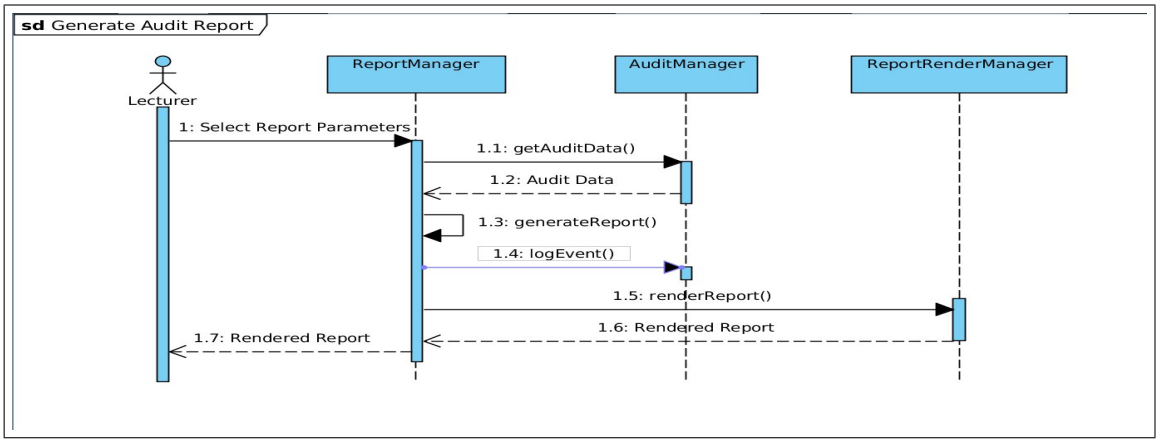

Figure 10: Generate Audit Report

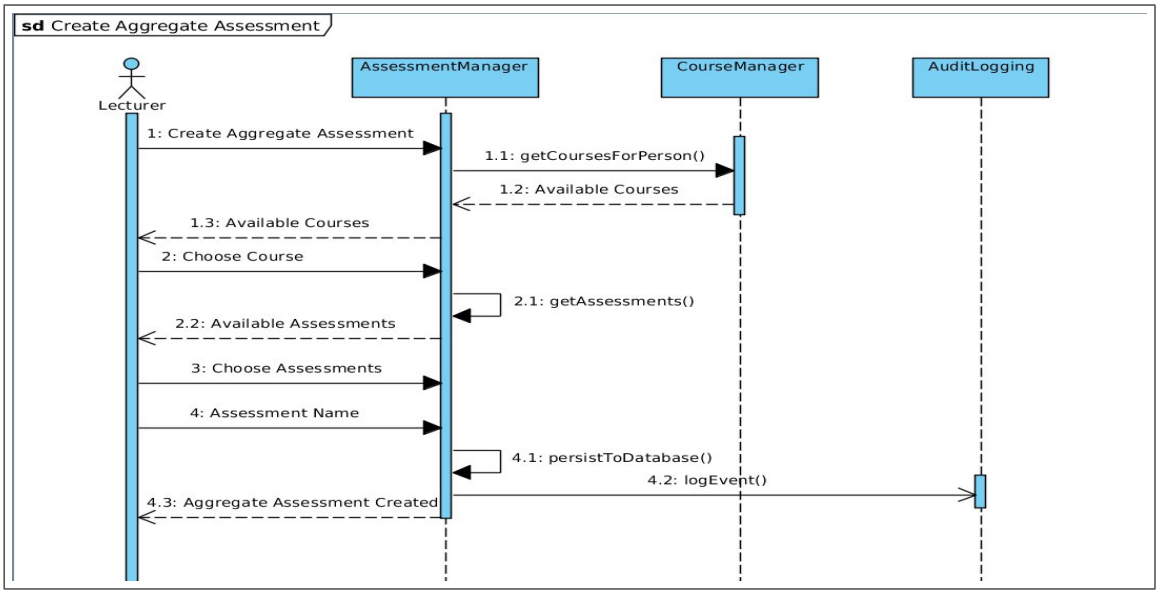

Figure 11: Create Aggregate Assessment

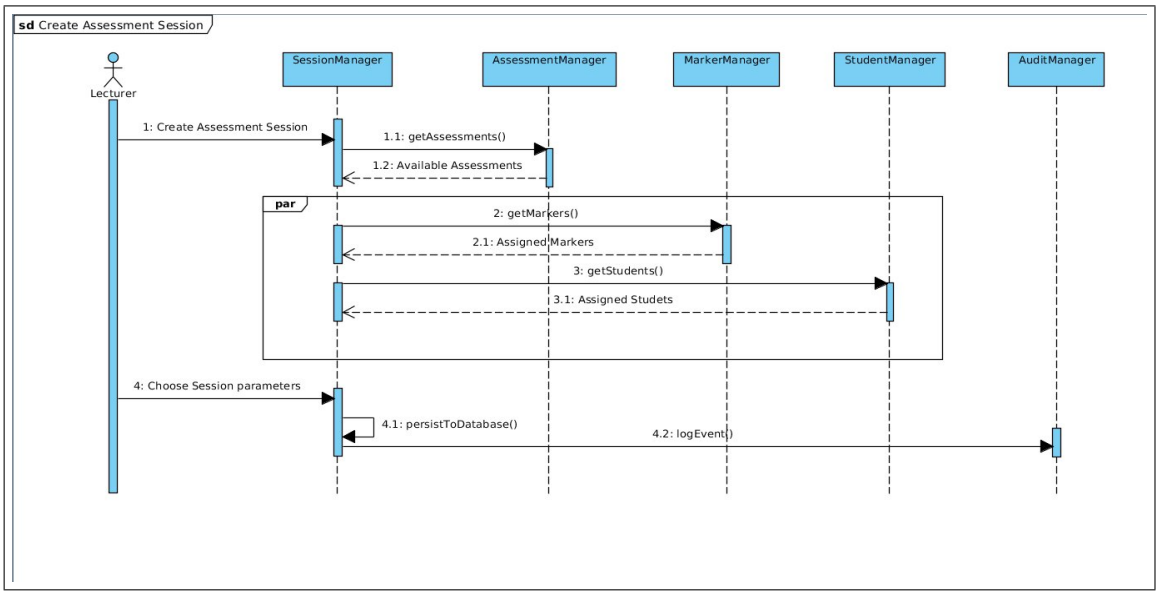

Figure 12: Create Assessment Session

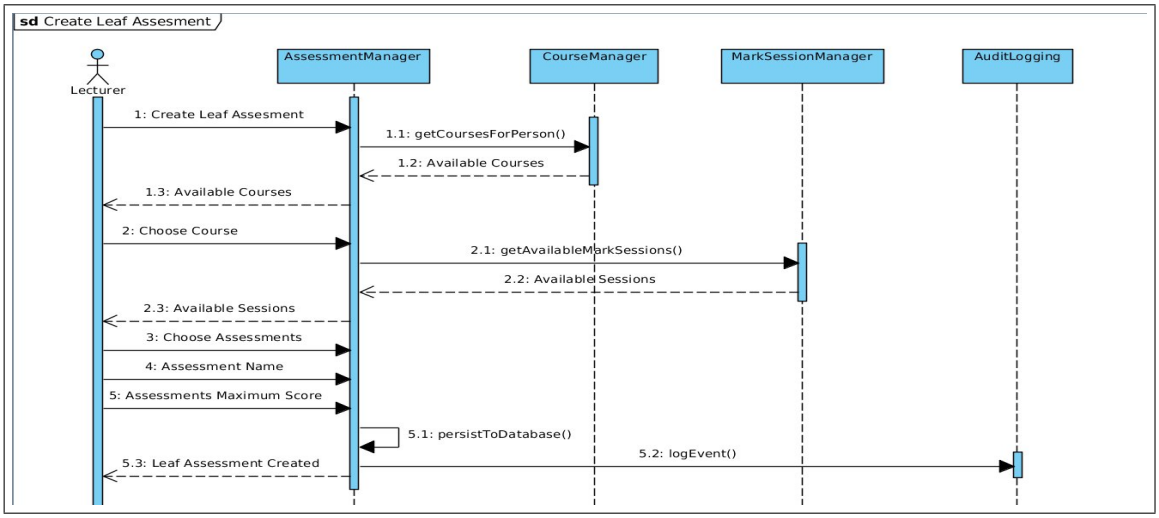

Figure 13: Create Leaf Assessment

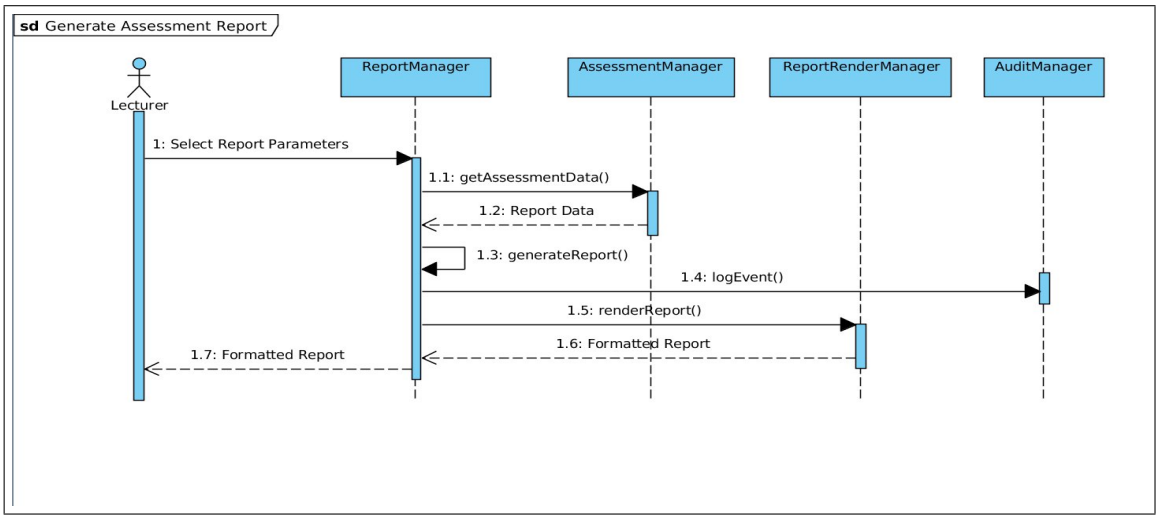

Figure 14: Generate Assessment Report

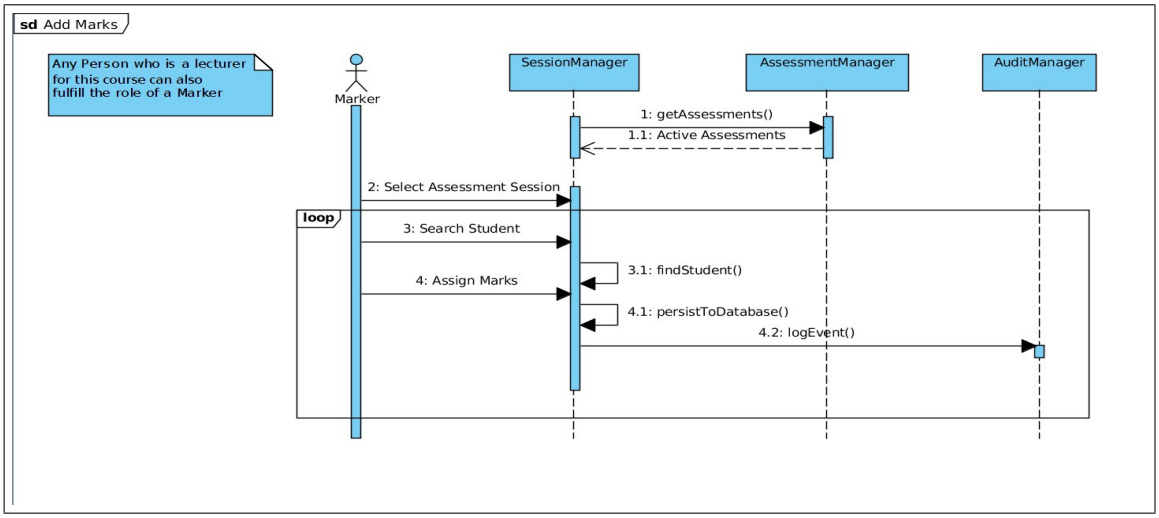

Figure 15: Add Marks

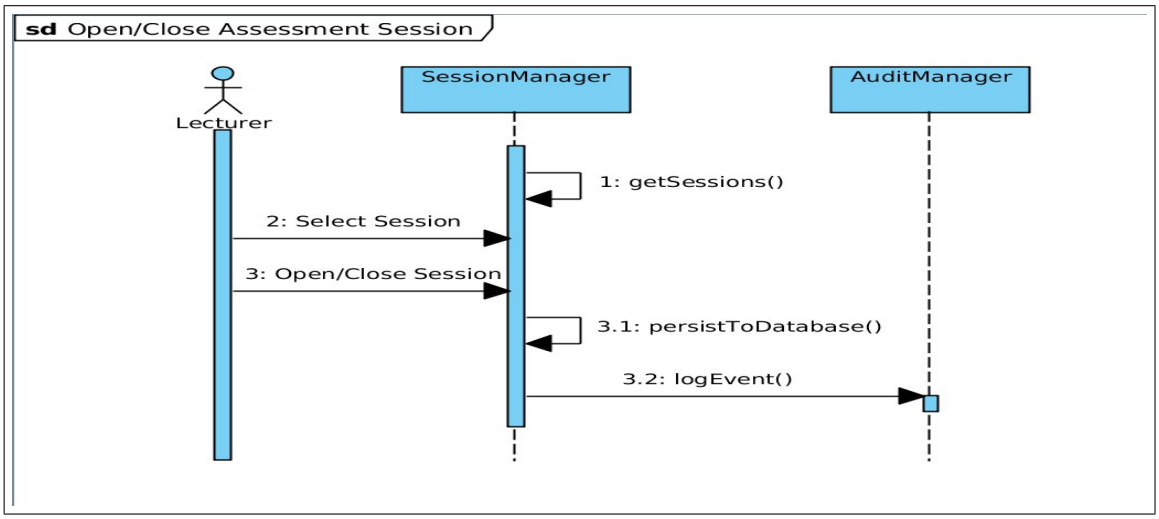

Figure 16: Open Close Session

4.3 Development View

4.3.1 Package Diagram

4.3.2 Component Diagram

4.4 Process View

4.5 Physical View

4.5.1 Deployment Diagram

4.6 Scenarios

4.6.1 Use Cases Diagram

4.7 Component Diagram

5 Traceability Matrix