Diploma Projects Management App

Sprint Report

<Texnlog>
<Dimosthenis Pantazis 4136>

VERSIONS HISTORY

Date	Version	Description	Author
14/5/2023	1	Final version of texnlog project	Dimosthenis Pantazis

1 Introduction

This document provides information concerning the **<1>** sprint of the project.

1.1 Purpose

1.2 Document Structure

The rest of this document is structured as follows. Section 2 describes out Scrum team and specifies the this Sprint's backlog. Section 3 specifies the main design concepts for this release of the project.

2 Scrum team and Sprint Backlog

2.1 Scrum team

Product Owner	Dimosthenis Pantazis
Scrum Master	Dimosthenis Pantazis
Development Team	Dimosthenis Pantazis

2.2 Sprints

<List below the sprints that you performed and the user stories that have been realized in each Sprint>

Sprint No	Begin Date	End Date	Number of weeks	User stories
1	**20/2/2023**	**1/4/2023**	**6**	**none**
2	**2/4/2023**	**25/4/2023**	**3.5**	General user stories
3	**26/4/2023**	**5/5/2023**	**2**	Student user stories
4	**6/5/2023**	**14/5/2023**	**1**	Professor user stories

3 Use Cases

<Specify the concrete Use Cases that describe the interaction of the user with the applications, as derived from the abstract user stories. Give a **UML Use Case diagram** and the **detailed use case descriptions**.>

3.1 <Use Case 1>

Use case ID	1
Actors	Instructor
Pre conditions	Create an account giving a username and password.
Main flow of events	1. The use case starts when the instructor select's create an account and types in the fields his username and password 2. The instructor then clicks onto the register button.
Alternative flow 1	If the instructor gives an already existing user the register page will crush
Alternative flow 2	.....
Post conditions	The system returns message of success registration

3.2 <Use Case 2>

Use case ID	2
Actors	Instructor
Pre conditions	Already registered user.
Main flow of events	1. The use case starts when the instructor types in the fields the properly username and password.

	2. The instructor then clicks onto the LOGIN button.
Alternative flow 1	
Alternative flow 2	.....
Post conditions	The system returns the main page of the app.

3.3 <Use Case 3>

Use case ID	3
Actors	Instructor
Pre conditions	The instructor must have login successfully.
Main flow of events	1. The use case starts when the instructor clicks the button “Student” 2. The system displays a table with the Students
Alternative flow 1	……..
Alternative flow 2	.....
Post conditions	The system shows a page of the table with the Students

3.4 <Use Case 4>

Use case ID	4
Actors	Instructor
Pre conditions	The instructor must have login successfully.
Main flow of events	1. The use case starts when the instructor clicks the button “Add Student”. 2. The system shows the page to submit a new Students with the informations.

	3. The instructor fills out the form with the properly data about the student. 4. The instructor clicks the button “Submit”.
Alternative flow 1	If the instructor gives an already existing student the page will crush
Alternative flow 2	.....
Post conditions	The system processed the new student into the database and redirects to the students page.

3.5 <Use Case 5>

Use case ID	5
Actors	Instructor
Pre conditions	The instructor must have login successfully and directs at student page.
Main flow of events	1. The use case starts when the instructor clicks the button “available subject”. 2. The system shows the page with available subjects offered by professors.
Alternative flow 1	
Alternative flow 2	.....
Post conditions	The system shows the page with available subjects offered by professors.

3.6 <Use Case 6>

Use case ID	6
Actors	Instructor

Pre conditions	1. A subject must exist on the available subjects table. 2. A Student must exist on the Students table.
Main flow of events	1. The use case starts when instructor selects apply to available subjects 2. The system shows a page which ask you to choose the student and the subject you want to apply.
Alternative flow 1	
Alternative flow 2	.....
Post conditions	The system does the mapping into the database and redirects to available subjects page.

3.7 <Use Case 7>

Use case ID	7
Actors	Instructor
Pre conditions	Already registered user.
Main flow of events	1. The use case starts when instructor selects Professor at home page
Alternative flow 1	
Alternative flow 2	.....
Post conditions	The system returns the Professor page.

3.8 <Use Case 8>

Use case ID	8
Actors	Instructor

Pre conditions	Already registered user.
Main flow of events	1. The use case starts when the instructor clicks the button Add Professor at professors page. 2. The system shows the page to submit a new Professor with its information. 3. The instructor fills out the form with the properly data about the Professor. 4. The instructor clicks the button “Submit”.
Alternative flow 1	
Alternative flow 2	.....
Post conditions	The system processed the new Professor into the database and redirects to the professors page.

3.9 <Use Case 9>

Use case ID	9
Actors	Instructor
Pre conditions	Already registered user.
Main flow of events	1. The use case starts when the instructor clicks the button Add Subject at professors page. 2. The system shows the List Professor page and instructor clicks add Subject again. 3. The system shows the page to submit a new Subject with its information. 4. The instructor fills out the form with the properly data about the Subject. 5. The instructor clicks the button “Submit”.
Alternative flow 1	
Alternative flow 2	.....

Post conditions	The system processed the new Subject into the database and redirects to the professors page.

3.10 <Use Case 10>

Use case ID	10
Actors	Instructor
Pre conditions	Successfully login.
Main flow of events	2. The use case starts when instructor selects logout at home page
Alternative flow 1	
Alternative flow 2	.....
Post conditions	The system redirects at login page and returns message of successfully logged out.

4 Design

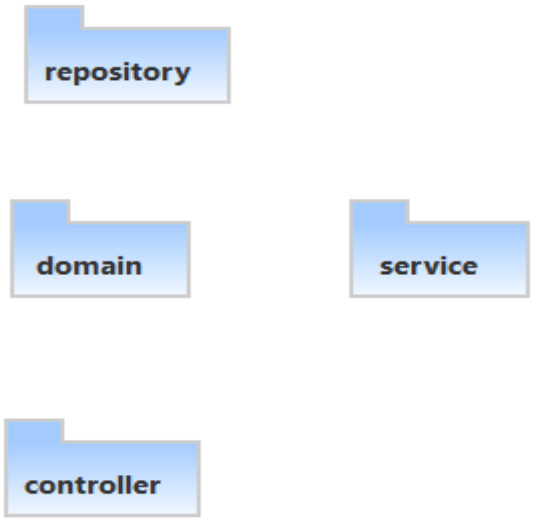
4.1 Architecture

The architecture model used to create the application is

Model-View-Controller (MVC). In this model the application is divided into three

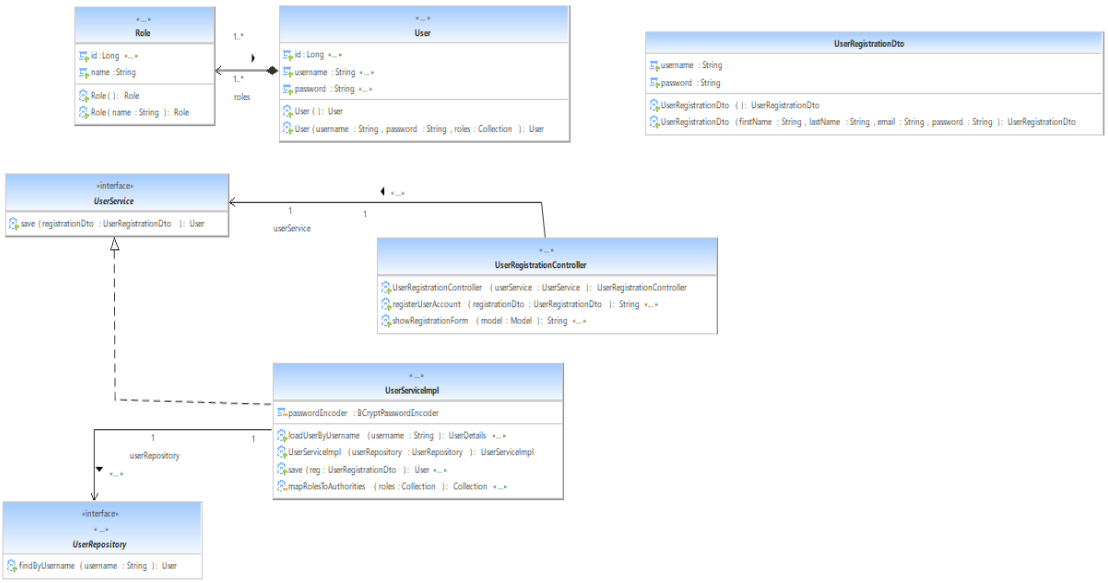
interconnected parts to separate the presentation of information to the user from

format stored on the system.

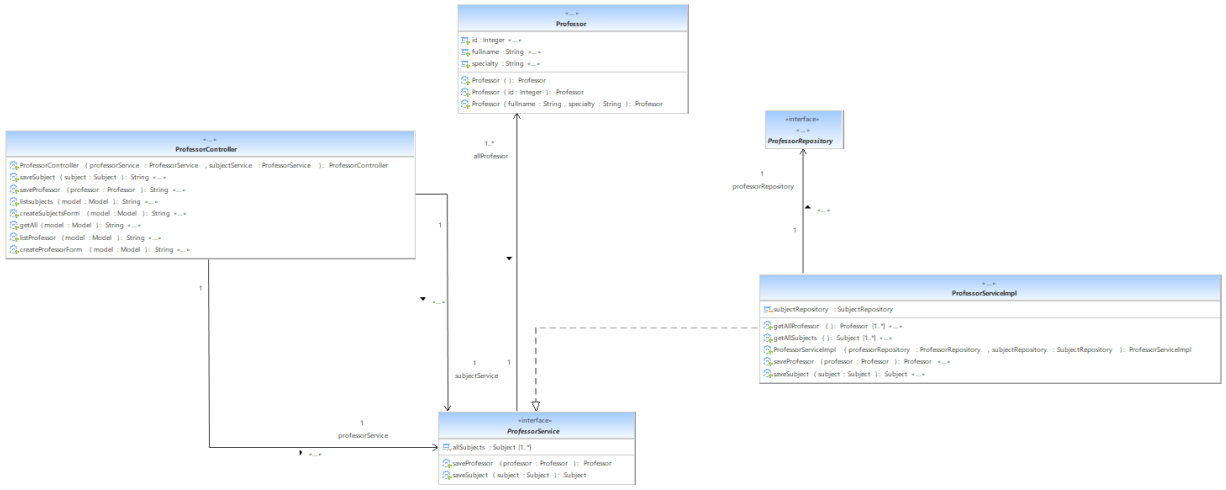


- controller package : Contains the controller classes that are responsible for the communication between backend and frontend.
- domain package : Contains the domain classes of the project, each of one represents a table in the database with the properly data.
- service package : Contains the service classes that are responsible to retrieve data from the database.
- repository package : Contains the repository interfaces that handles the query operations to the database.

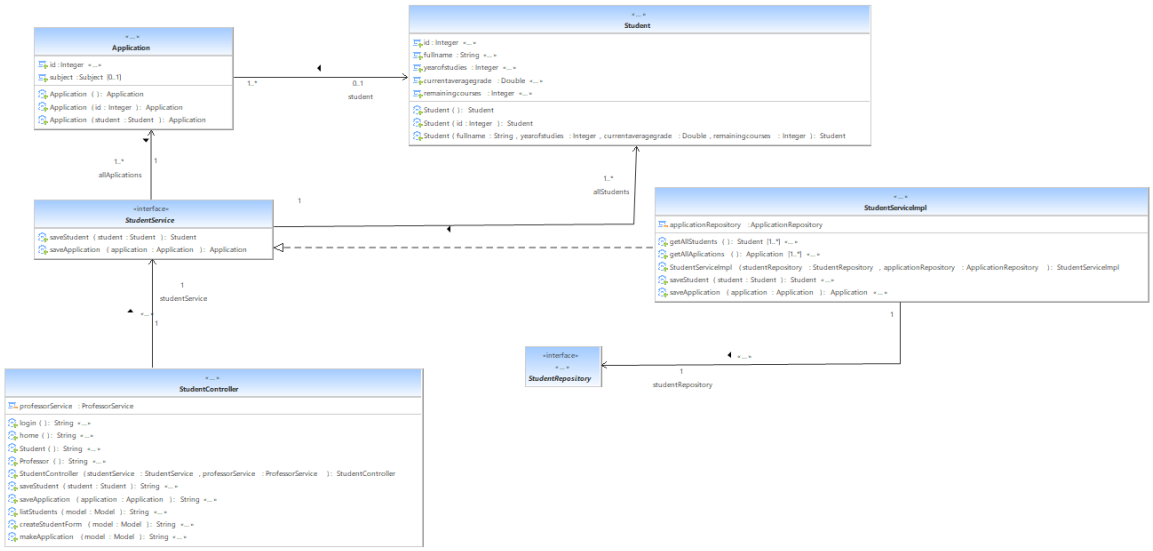
User


Role
id : Long
name : String
Role ( ) : Role
Role (name : String) : Role
User
id : Long
username : String
password : String
User ( ) : User
User (username : String , password : String , roles : Collection) : User
1..*
roles
UserRegistrationDto
username : String
password : String
UserRegistrationDto ( ) : UserRegistrationDto
UserRegistrationDto (firstName : String , lastName : String , email : String , password : String) : UserRegistrationDto
«interface»
UserService
save (registrationDto : UserRegistrationDto) : User
userService
UserRegistrationController
UserRegistrationController (userService : UserService) : UserRegistrationController
registerUserAccount (registrationDto : UserRegistrationDto) : String
showRegistrationForm (model : Model) : String
UserServiceImpl
passwordEncoder : BCryptPasswordEncoder
loadUserByUsername (username : String) : UserDetails
UserServiceImpl (userRepository : UserRepository) : UserServiceImpl
save (reg : UserRegistrationDto) : User
mapRolesToAuthorities (roles : Collection) : Collection
userRepository
«interface»
UserRepository
findByUsername (username : String) : User

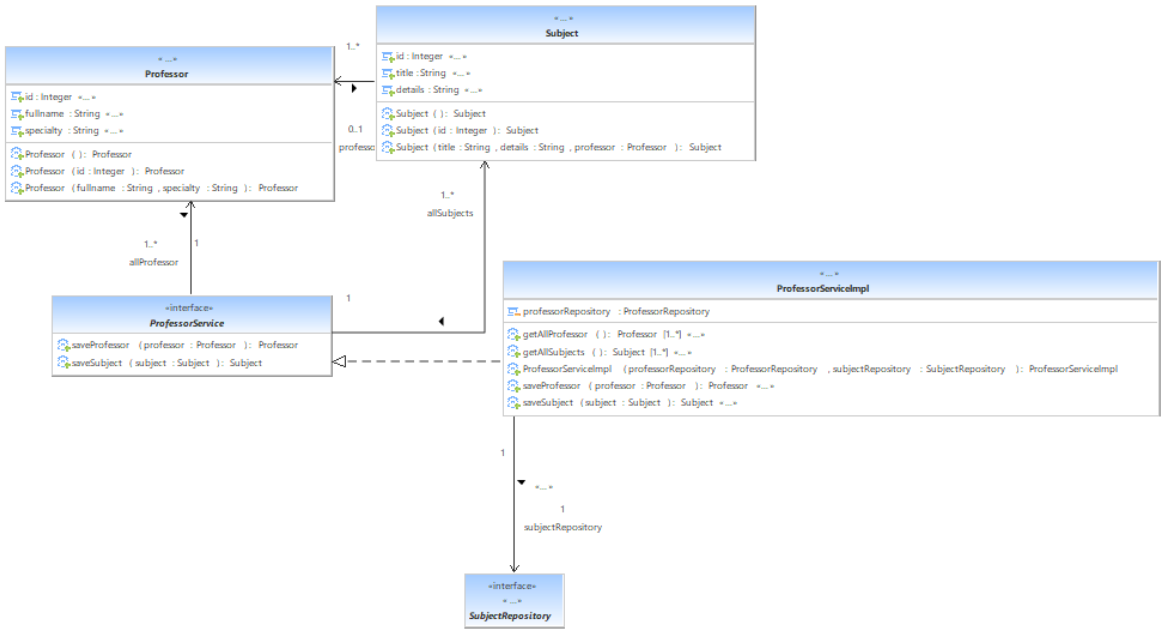
Professor


Professor
id : Integer
fullname : String
specialty : String
Professor ( ) : Professor
Professor (id : Integer) : Professor
Professor (fullname : String , specialty : String) : Professor
ProfessorController
ProfessorController (professorService : ProfessorService , subjectService : ProfessorService) : ProfessorController
saveSubject (subject : Subject) : String
saveProfessor (professor : Professor) : String
listaSubjects (model : Model) : String
createSubjectsForm (model : Model) : String
getAll (model : Model) : String
listProfessor (model : Model) : String
createProfessorForm (model : Model) : String
allProfessor
«interface»
ProfessorRepository
professorRepository
ProfessorServiceImpl
subjectRepository : SubjectRepository
getAllProfessor ( ) : Professor [1..*]
getAllSubjects ( ) : Subject [1..*]
ProfessorServiceImpl (professorRepository : ProfessorRepository , subjectRepository : SubjectRepository) : ProfessorServiceImpl
saveProfessor (professor : Professor) : Professor
saveSubject (subject : Subject) : Subject
subjectService
professorService
«interface»
ProfessorService
allSubjects : Subject [1..*]
saveProfessor (professor : Professor) : Professor
saveSubject (subject : Subject) : Subject

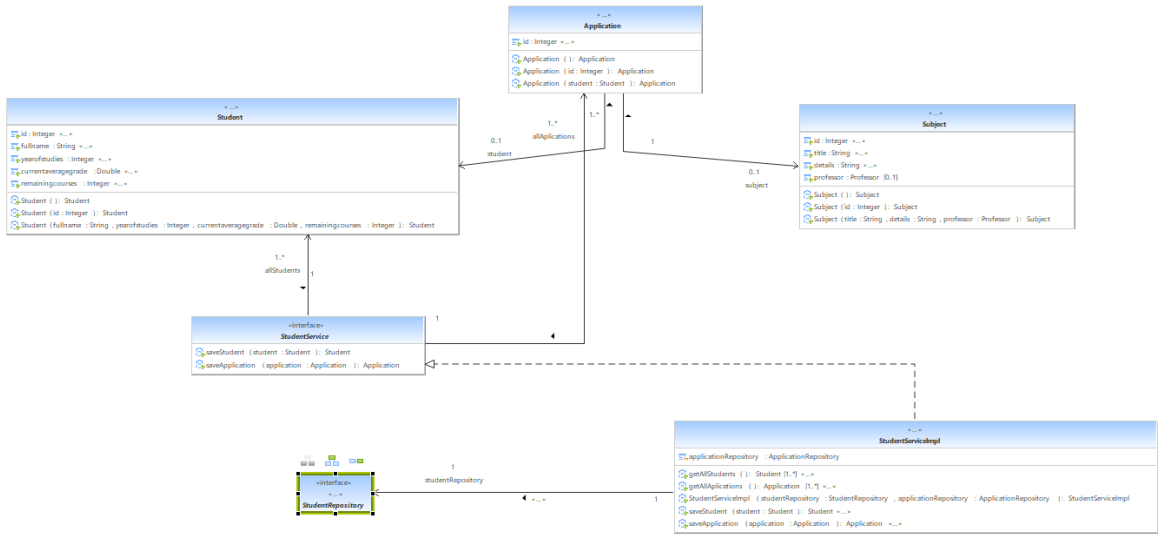
Student



Subject



Application



4.2 Design

<Specify the detailed design for this release in terms of **UML class diagrams**.>

<Document the classes that are included in this release in terms of CRC cards according to the template that is given below.>

Class Name: User	
Responsibilities: ▪ Represents a User of the system ▪ Handles data about User that are matching the data that are stored in the database	**Collaborations:**

Class Name: Professor	
Responsibilities: ▪ Represents a Professor of the system ▪ Handles data about User that are matching the data that are stored in the database	**Collaborations:**

Class Name: Student	
Responsibilities: ▪ Represents a Student of the system ▪ Handles data about User that are matching the data that are stored in the database	**Collaborations:**

Class Name: Subject	
Responsibilities: ▪ Represents a Subject of the system ▪ Handles data about User that are matching the data that are stored in the database	**Collaborations:**

Class Name: Application	
Responsibilities: ▪ Represents a Application of the system ▪ Handles data about User that are matching the data that are stored in the database	**Collaborations:**

Class Name: ProfessorController	
Responsibilities: ▪ Handles the events that occurred in the frontend ▪ Transfer data from the frontend to the backend ▪ Responsible for what user sees on the related pages	**Collaborations:** ▪ Professor ▪ Subjects ▪ ProfessorService

Class Name: StudentController	
Responsibilities: ▪ Handles the events that occurred in the frontend ▪ Transfer data from the frontend to the backend ▪ Responsible for what user sees on the related pages	**Collaborations:** ▪ student ▪ Application ▪ StudentService

Class Name: UserRegistrationController	
Responsibilities: ▪ Handles the events that occurred in the frontend ▪ Transfer data from the frontend to the backend ▪ Responsible for what user sees on the related pages	**Collaborations:** ▪ User ▪ UserService

Class Name: UserServiceImpl	
Responsibilities: ▪ Communicates with the database ▪ Add entries into the table User of the database ▪ Retrieves data of Users from the database	**Collaborations:** ▪ User ▪ Role ▪ UserRepository

Class Name: ProfessorServiceImpl	
Responsibilities: ▪ Communicates with the database ▪ Add entries into the table Professor and Subjects of the database ▪ Retrieves data of Professor and Subject from the database	**Collaborations:** ▪ Professor ▪ Subject ▪ ProfessorRepository ▪ SubjectRepository

Class Name: StudentServiceImpl	
Responsibilities: ▪ Communicates with the database ▪ Add entries into the table Student and Application of the database ▪ Retrieves data of Student and Application from the database	**Collaborations:** ▪ Student ▪ Application ▪ StudentRepository ▪ ApplicationRepository

Class Name: UserRepository	
Responsibilities: ▪ Communicates with the database ▪ Executes crud operation to the database	**Collaborations:**

Class Name: ProfessorRepository	
Responsibilities: ▪ Communicates with the database ▪ Executes crud operation to the database	**Collaborations:**

Class Name: StudentRepository	
Responsibilities: ▪ Communicates with the database ▪ Executes crud operation to the database	**Collaborations:**

Class Name: SubjectRepository	
Responsibilities: ▪ Communicates with the database ▪ Executes crud operation to the database	**Collaborations:**

Class Name: ApplicationRepository	
Responsibilities: ▪ Communicates with the database	**Collaborations:**

▪ Executes crud operation to the database	