Faculty of Computers, Informatics and Microelectronics

Technical University of Moldova

Object-Oriented Modeling and Analysis

Course Work

# Analysis and modelling of an online courses platform

*Author:*
Cernei Liviu
st. gr. FAF-161

*Supervisor:*
Mihail Gavriliță
university assistant

Chișinau 2018

# Contents

# 1 Intro and Theory

## 1.1 Project Description, User Stories

The new **St. Courses** platform is a great place to learn new skills and boost your knowledge. It provides a web based interaction between students and teachers. Because of it's simplicity and security, it can be used by schools and universities. On St. Courses, the user gets personalized help in order to customize his online classes. The website is user-friendly, engaging and easy to use. It also has a customizable front-end, advanced analytics and much more.

The platform is populated by 3 kinds of actors: administrators, teachers and students.

**Courses:** A course has a title, description and content. It is placed in time by the start date and end date. It is created by a teacher and can have enroled students. The content is structured in chapters and can contain text, images and interactive elements. Every teacher who logged in can create and edit a course. To delete it, he must complete a request to an administrator, containing the cause of removal.

**Tests:** The teacher optionally can create a test to evaluate the quality of the course. For this he must fill in the questions, provide the coorect answers and create an grading system. For the student, a test is obligatory. At the end of the course the students get marks based on their activity.

**Feedback:** Both teachers and students can give feedback which wil be assigned to the course history.

**Enrolment to a course:** The courses are free, meaning that everyone with an account can enrol to any corse. However, there are recomended courses, based on age, fields of interest, difficulty level. A student can lose access to a course if he unenroled himself, or the teacher removed him. In both cases the student loses the progress, tests, marks gained in that course.

**Invitations:** A teacher can send invitations to a course. The invited students will be enroled automatically. With this method the teacher can assemble an real-life grop and assure access to the course for them.

- An administrator (moderator) is responsible for handling reports, managing courser and users..
- A teacher can create a new course, edit an existing one, approve or reject an student to a course, create tests.
- A student can enrol to a course, read the material, unenrol from a course, pass a test, give feedback.

## 1.2 Modeling Languages

A modeling language is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules. The rules are used for interpretation of the meaning of components in the structure.

A modeling language can be graphical or textual.

**Graphical** modeling languages use a diagram technique with named symbols that represent concepts and lines that connect the symbols and represent relationships and various other graphical notation to represent constraints.

**Textual** modeling languages may use standardized keywords accompanied by parameters or natural language terms and phrases to make computer-interpretable expressions.

Not all modeling languages are executable, and for those that are, the use of them doesn't necessarily mean that programmers are no longer required. On the contrary, executable modeling languages are intended to amplify the productivity of skilled programmers, so that they can address more challenging problems, such as parallel computing and distributed systems.

### 1.3 Conceptual OO Analysis / Technical OO Design

**Object-Oriented Analysis**

- Elicit requirements: Define what does the software need to do, and what's the problem the software trying to solve.
- Specify requirements: Describe the requirements, usually, using use cases (and scenarios) or user stories.
- Conceptual model: Identify the important objects, refine them, and define their relationships and behavior and draw them in a simple diagram.

**Conceptual Model**

- Identifying Objects
- Refining Objects
- Drawing Objects
- Identifying Object Relationships
- Identifying Object Behaviors

### 1.4 Abstraction, Encapsulation, Decomposition, Generalization

Most modern programming languages support and encourage object-oriented programming (OOP). It's main principles are:

**Abstraction** is the idea of simplifying a concept to its bare essentials in some context. It allows you to better understand the concept by stripping it down to a simplified version.

**Encapsulation** can be thought of as putting something inside a capsule - you limit its exposure to the outside world. In software, restricting access to inner objects and properties helps with data integrity.

**Decomposition** is the action of splitting an object into multiple separate smaller parts. Said parts are easier to understand, maintain and program.

**Generalization** might be the most important design principle - it is the process of extracting shared characteristics and combining them in one place. All of us know about the concept of functions and class inheritance —both are a kind of generalization.

### 1.5 Coupling and Cohesion, Separation of Concerns, Information Hiding, Conceptual integrity

**Coupling** - An indication of the strength of interconnections between program units. Highly coupled have program units dependent on each other. Loosely coupled are made up of units that are independent or almost independent.

**Cohesion** - Measure of how well module fits together. A component should implement a single logical function or single logical entity. All the parts should contribute to the implementation.

**Separation of Concerns** (SoC) is a design principle for separating a computer program into distinct sections, such that each section addresses a separate concern. A concern is a set of information that affects the code of a computer program. A concern can be as general as the details of the hardware the code is being optimized for, or as specific as the name of a class to instantiate. A program that embodies SoC well is called a modular program. Modularity, and hence separation of concerns, is achieved by encapsulating information inside a section of code that has a well-defined interface. Encapsulation is a means of **information hiding**. Layered designs in information systems are another embodiment of separation of concerns (e.g., presentation layer, business logic layer, data access layer, persistence layer).

**Conceptual integrity** is the principle that anywhere you look in your system, you can tell that the design is part of the same overall design. This includes low-level issues such as formatting and identifier naming, but also issues such as how modules and classes are designed, etc.

### 1.6 Solid / Grasp Principles

**SOLID Principles are principles of class design.**

- SRP: Single Responsibility Principle–An object should have only a single responsibility and all the responsibility should be entirely encapsulated by the class.–There should never be more than one reason for a class to change
- OCP: Open/Closed Principle–Software entities should be open for extension, but closed for modification
- LSP: Liskov Substituion Principle–Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program
- ISP: Interface Segregation Principle–many client specific interfaces are better than one general purpose interface–once an interface has gotten too 'fat' split it into smaller and more specific interfaces so that any clients of the interface will only know about the methods that pertain to them. No client should be forced to depend on methods it does not use

– DIP: Dependency Inversion Principle–Depend upon Abstractions. Do not depend upon concretions.Dependency Injection (DI) is one method of following this principle.

**GRASP - Acronym for General Responsibility Assignment Software Patterns.**

– Assigning responsibilities to classes is a critical aspect of object-oriented design.

– Appropriate assignment of responsibilities to classes is the key to successful design.

– There are fundamental principles in assigning responsibilities that experienced designers apply.

– These principles are summarized in the GRASP patterns.

– Has nine core principles that object-oriented designers apply when assigning responsibilities to classes and designing message interactions.

### 1.7 Software Development Process

In software engineering, a **software development process** is the process of dividing software development work into distinct phases to improve design, product management, and project management. It is also known as a software development life cycle. The methodology may include the pre-definition of specific deliverables and artifacts that are created and completed by a project team to develop or maintain an application.

Most modern development processes can be vaguely described as agile. Other methodologies include waterfall, prototyping, iterative and incremental development, spiral development, rapid application development, and extreme programming.

### 1.8 Top-down and Bottom-up Design

Top-down and bottom-up are both strategies of information processing and knowledge ordering, used in a variety of fields including software, humanistic and scientific theories (see systemics), and management and organization. In practice, they can be seen as a style of thinking, teaching, or leadership.

A **top-down** approach (also known as stepwise design and in some cases used as a synonym of decomposition) is essentially the breaking down of a system to gain insight into its compositional sub-systems in a reverse engineering fashion. In a top-down approach an overview of the system is formulated, specifying, but not detailing, any first-level subsystems. Each subsystem is then refined in yet greater detail, sometimes in many additional subsystem levels, until the entire specification is reduced to base elements. A top-down model is often specified with the assistance of "black boxes", which makes it easier to manipulate. However, black boxes may fail to clarify elementary mechanisms or be detailed enough to realistically validate the model. Top down approach starts with the big picture. It breaks down from there into smaller segments.

A **bottom-up** approach is the piecing together of systems to give rise to more complex systems, thus making the original systems sub-systems of the emergent system. Bottom-up processing is a type of information processing based on incoming data from the environment to form a perception. From

a cognitive psychology perspective, information enters the eyes in one direction (sensory input, or the "bottom"), and is then turned into an image by the brain that can be interpreted and recognized as a perception (output that is "built up" from processing to final cognition). In a bottom-up approach the individual base elements of the system are first specified in great detail. These elements are then linked together to form larger subsystems, which then in turn are linked, sometimes in many levels, until a complete top-level system is formed. This strategy often resembles a "seed" model, by which the beginnings are small but eventually grow in complexity and completeness. However, "organic strategies" may result in a tangle of elements and subsystems, developed in isolation and subject to local optimization as opposed to meeting a global purpose.

### 1.9 OntoUML

**OntoUML** is a pattern-based and ontologically well-founded version of the Unified Modeling Language (UML). Its meta-model has been designed in compliance with the ontological distinctions of a well-grounded theory, named the Unified Foundational Ontology (UFO). OntoUML includes a system of interrelated axiomatic theories, providing modeling foundations for all Conceptual Modeling major concepts, including theories for: types and taxonomic structures (including roles), part-whole relations, events, formal and material relations, dependent (weak) entities, attributes and attribute value and measurement spaces (roughly datatypes).

## 2 Analysis and Modeling

### 2.1 Main and Alternative Flows

- **Student Use Case diagram - Enrol in course**

  **Success scenario:**

  - 1. The student navigates to "Courses" page
  - 2. The student selects from the list a course
  - 3. The student clicks the [Enrol] button.
  - 4. The system outputs success message.

  **Alternate flow:**

  - 1.a The student is enroled to the course by a teacher, other steps are not performed.
  - 2.a The student inputs name of course in the search box

- **Teacher Use Case diagram - Create course**

  **Success scenario:**

  - 1. The teacher navigates to "Courses" page

- 2. The teacher clicks the [New] button.
- 3. The teacher writes the title, date, content.
- 4. The teacher clicks the [Save] button.
- 5. The system outputs success message.

**Alternate flow:**

- 3.a The teacher writes the title, date and imports the pdf file with the content.

- **Admin Use Case diagram - Login**

  **Success scenario:**

  - 1. The admin navigates to "Login" page
  - 2. The admin inputs correct username and correct password.
  - 3. The admin clicks the [Login] button.
  - 4. The system outputs success message.

  **Alternate flow:**

  - 2.a The admin inputs wrong username or wrong password, fails to login but attempts one more time with correct username and correct password.
  - 3.a The admin presses the [Enter] key on the keyboard.

### 2.2 Requirements

**Functional**

- Provide access to intersting courses
- Teachers can create new courses
- Students can take tests
- Students can give feedback
- Teachers can invite and enroll students

**Non-Functional**

- The site supports multiple users at the same time
- The information is stored and encripted safely
- The site provides very fast access to data
- The site can provides access in multiple countries.
- The information is processed real-time.

### 2.3 Technologies

**Model–view–controller** is commonly used for developing software that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development.

Traditionally used for desktop graphical user interfaces (GUIs), this architecture has become popular for designing web applications and even mobile, desktop and other clients. Popular programming languages like Java, C#, Ruby, PHP and others have popular MVC frameworks that are currently being used in web application development straight out of the box.

### 2.4 SWOT Analysis

**Strengths**

- Customer-centric design and messaging
- Useful and relevant content
- Intuitive navigation and search
- Quick and easy enrollment process
- Responsive design with full mobile support

**Weaknesses**

- The lack of a content specialist within the web team.
- Minor updates and not thinking strategically.
- Limitations over the type of content published

**Opportunities**

- New technologies to improve user experience
- Emerging new and untapped markets
- New niches and market segments
- New design trends to better convey messages
- More effective marketing tactics
- Positive changes in social factors

**Threats**

- Competitors copying features or ideas

– Emergence of new competitors

– Changing customer needs

– New laws or regulations

– SPAM & unsolicited advertising

– Upgraded browser software

### 2.5 Domain Analysis

**Online courses** are aimed at unlimited participation and open access via the web. In addition to traditional course materials such as filmed lectures, readings, and problem sets, many online courses provide interactive user forums to support community interactions among students, professors, and teaching assistants (TAs).

Online courses are regarded by many as an important tool to widen access to Higher Education (HE) for millions of people, including those in the developing world, and ultimately enhance their quality of life. Online courses may be regarded as contributing to the democratisation of HE, not only locally or regionally but globally as well. Online courses can help democratise content and make knowledge reachable for everyone. Students are able to access complete courses offered by universities all over the world, something previously unattainable. With the availability of affordable technologies, online courses increase access to an extraordinary number of courses offered by world-renowned institutions and teachers.

There are already similar platforms. **"Coursera"** emerged as the top ranked online courses platform and the best overall choice due to its impressive selection of learning pathways and course features. **"edX"** and **"Udacity"** were also found to be strong contenders.

### 2.6 Project Setup. Delivery / Installation Documentation

The last step of website development is **deployment**. The process of"getting your website on the web" consists of a number of steps:

– Finding a domain name

– Finding a hosting service

– Uploading files with SFTP

– Deploying server-side applications

But the client is not required to download or install anything. He simply navigates to the webpage. The only step he should take is to create an account.

# 3 Analysis and Modeling in UML

## 3.1 Use Case Diagram

In Figure 3.1 is represented the Use Case diagram for the actor "student".

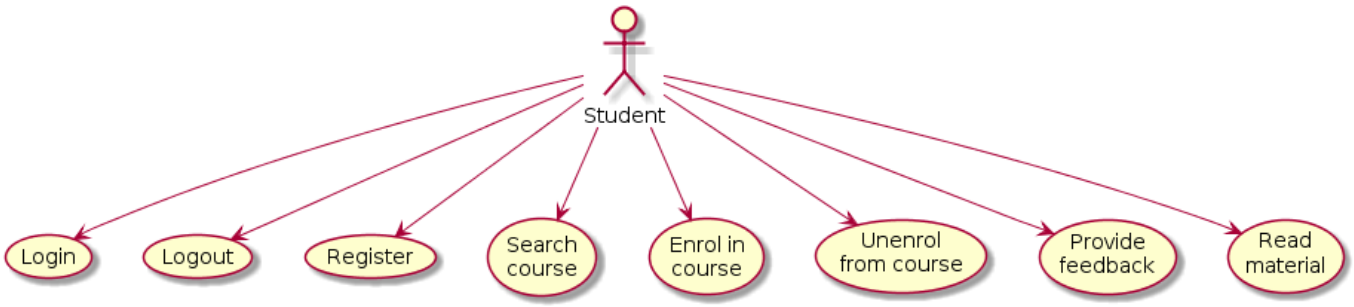


Figure 3.1 – Student Use Case diagram

In order to be able to login the student should have an account. Also, to logout the student should firstly login. Another case is the registration - the student must provide personal information (first / last name, year of study, university, faculty. group etc.). After registration, the user searches a course, the student can input the name of teacher or the title of the lecture.

Enrolment in a course is performed by selecting a course, clicking the [Enrol] button and accepting the confirmation message. The same with the unenrollment procedure. Finally, the student selects a course in which he is already enroled and can start reading the content. At the bottom of the page the student also can find the "Feedback" form.

In Figure 3.2 is represented the Use Case diagram for the actor "teacher".

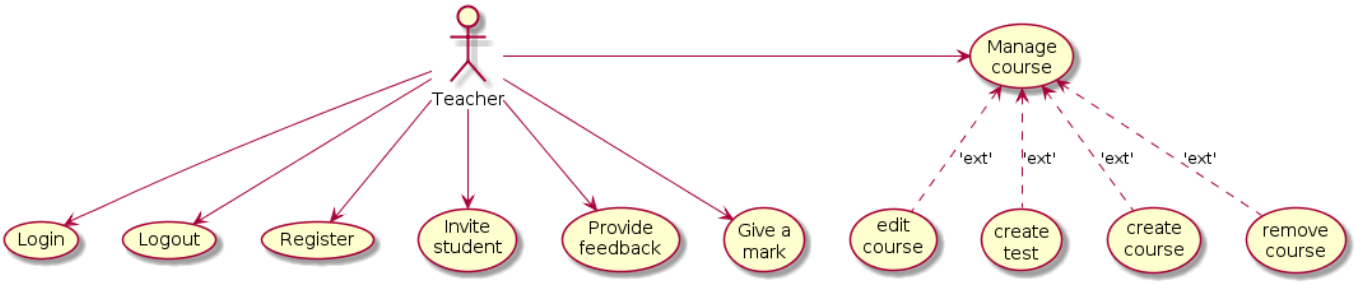


Figure 3.2– Teacher Use Case diagram

The teacher has access like the student user, but with additional privileges. He invite a student to one of his own courses. The teacher is responsible for giving marks or evaluating the student at the end of course.

Also, the techer is in control of the content of the course - we can say that he can perform "CRUD" operations on his own courses.

In Figure 3.3 is represented the Use Case diagram for the actor "administrator".

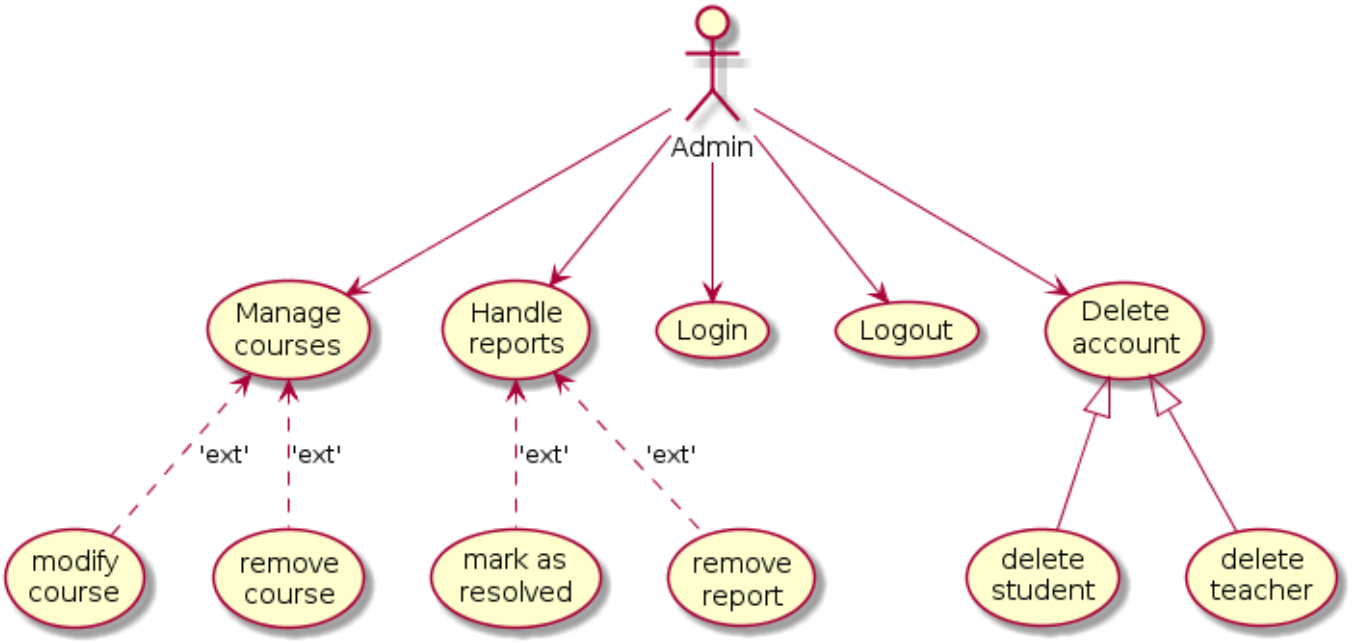


Figure 3.3– Admin Use Case diagram

The administrator can do almost everything he wants. He can manage courses from any teacher, can remove any accounts. Also he is responsible for handling reports.

### 3.2 Sequence Diagram

In Figure 3.4 is represented the sequence diagram for the login operation.

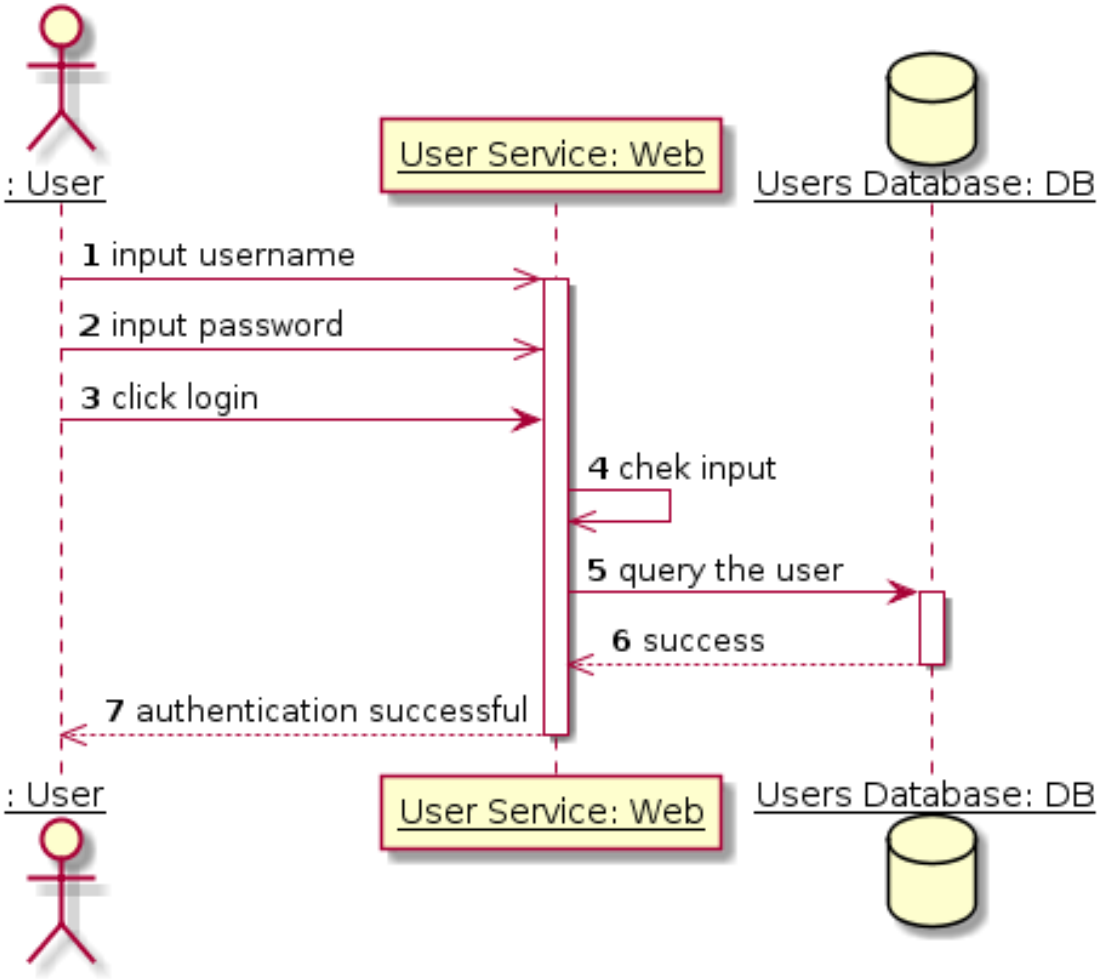


Figure 3.4– Login sequence diagram

We observe the sequence of operations needed to login any user: student, teacher, or even admin. It is a generalized operation, thats why I used an anonymous calss. The user inputs username and password, the user service check it and sends the information further to the database. The database returns an ”success” message, which is propagated to the user by the service. This is the case when all information was correct, and results with the user authenticated.

In Figure 3.5 is represented the sequence diagram for creation of a new course.

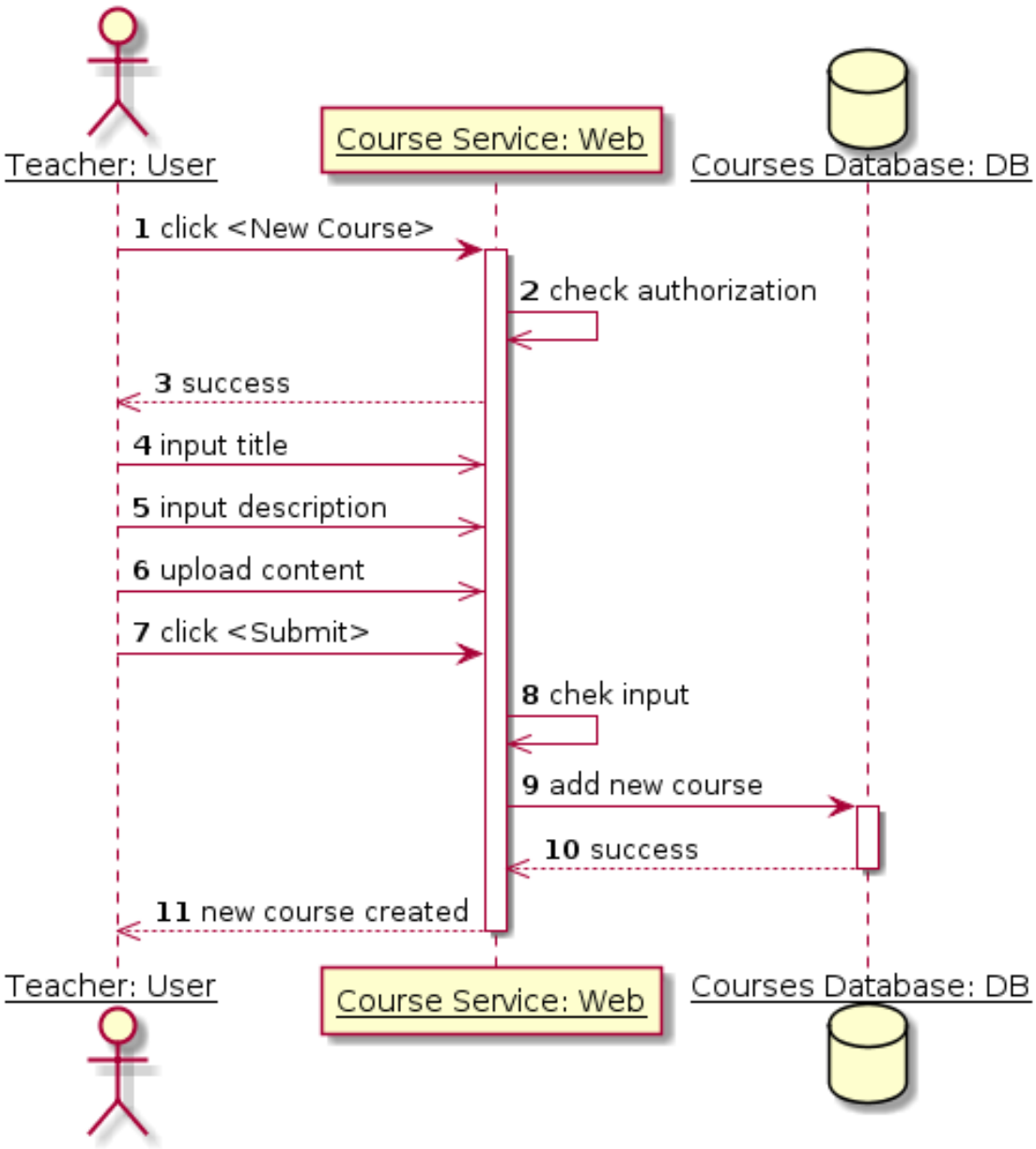


Figure 3.5– New course sequence diagram

This diagram is specifically for the teacher. In order to create a course, the teacher should have assignet a special authorization key. If it is found, he can begin to input the course information. When submitted, it is checked for invalid fields, then it is sent to the course database. In case of success the teacher is announced through the course service.

In Figure 3.6 is represented the sequence diagram for giving feedback for a course.

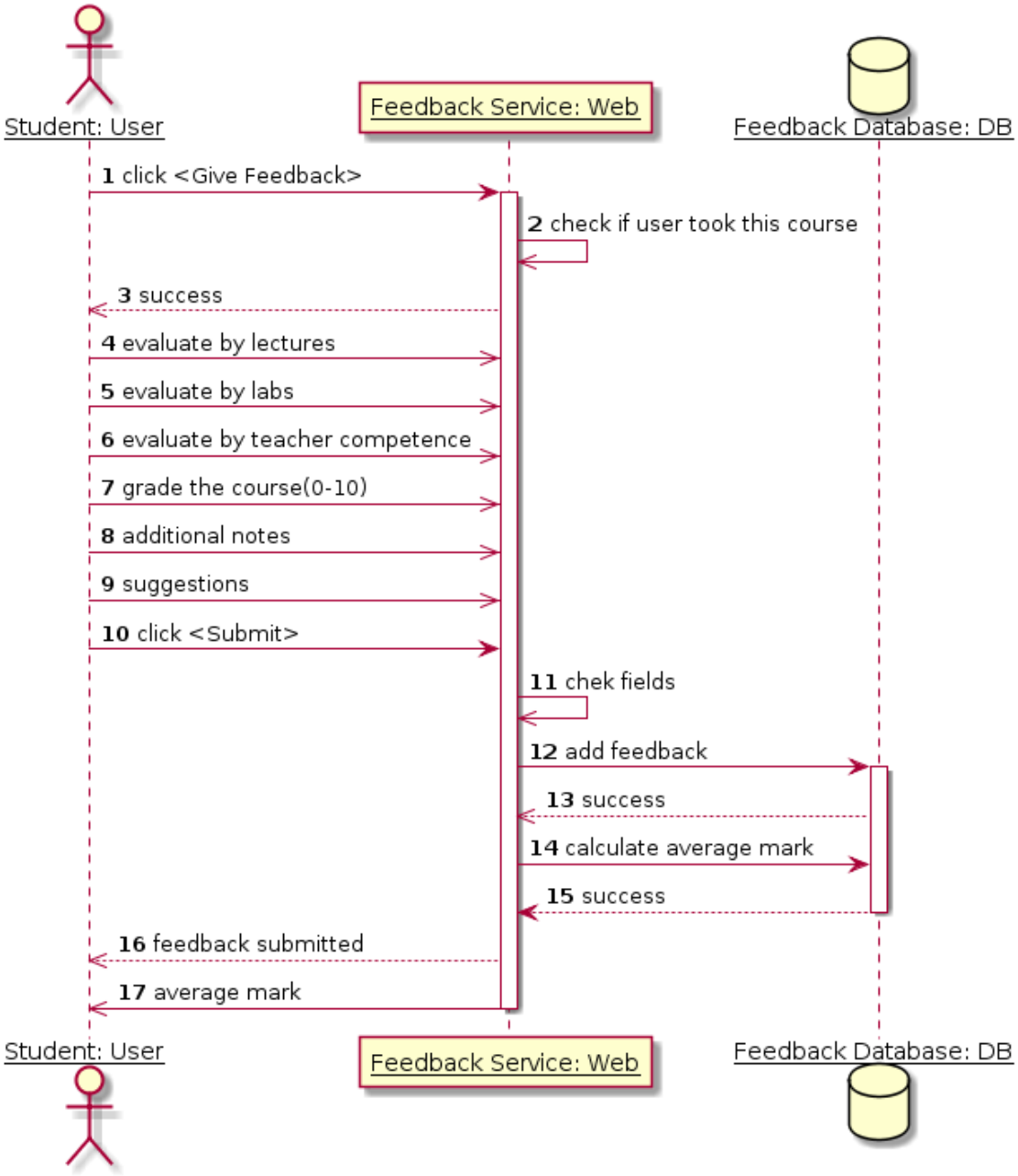


Figure 3.6– Feedback sequence diagram

To be eligible to give feedback, the studnt should have already completed the course. If the condition is satisfied, he can fill the form. After submission, all fields are checked and the information is sent to the database. After a "success" message, the feedback service requests the average mark for that course. Then the student receives an acceptance message and can see the overall quality of the course.

### 3.3 Collaboration Diagram

In Figure 3.7 is represented the collaboration diagram for the login operation.

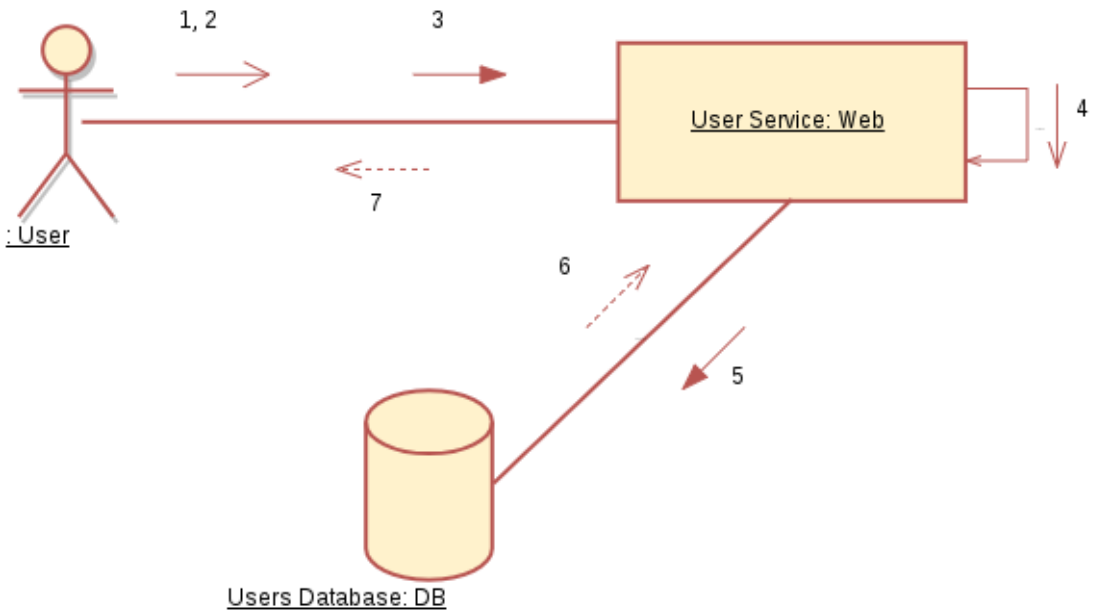


Figure 3.7– Login collaboration diagram

Here we can see the interaction beween objects in our project. These diagrams are the equivalent of the previous sequence diagrams.

1 - input username; 2 - input password; 3 - click login; 4 - chek input; 5 - query the user; 6 - success; 7 - authentication successful;

In Figure 3.8 is represented the collaboration diagram for creation of a new course.

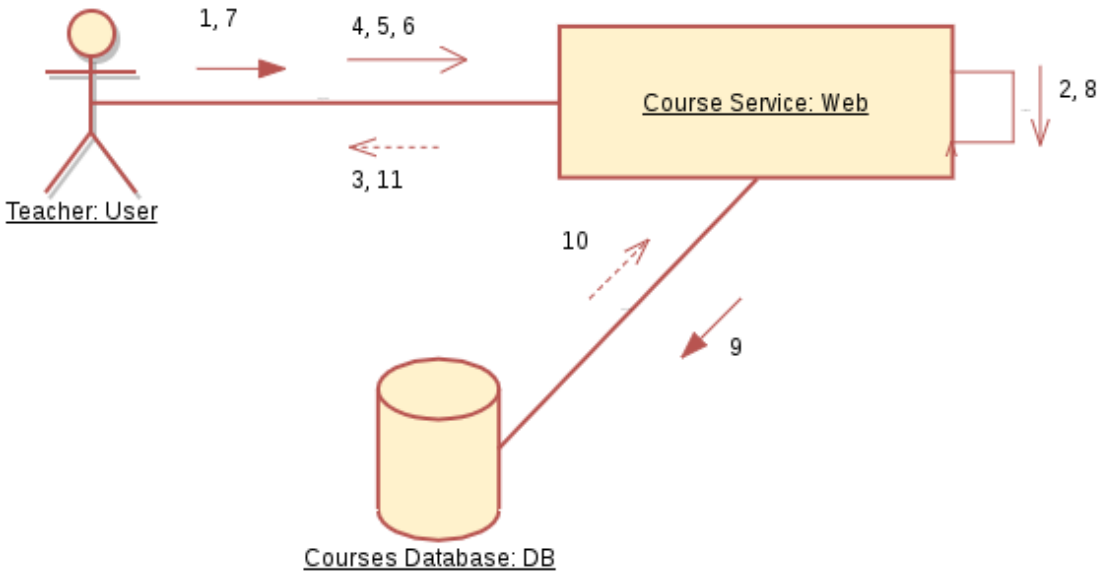


Figure 3.8– New course collaboration diagram

1 - click [New Course]; 2 - check authorization; 3 - success; 4 - input title; 5 - input description; 6 - upload content; 7 - click [Submit]; 8 - chek input; 9 - add new course; 10 - success; 11 - new course created;

In Figure 3.9 is represented the collaboration diagram for giving feedback for a course.

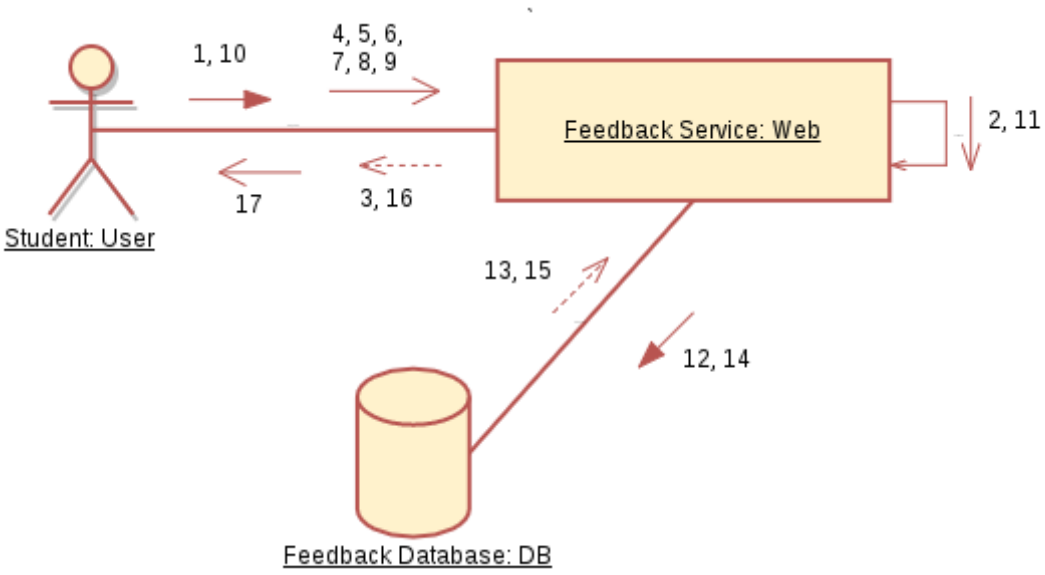


Figure 3.9– Feedback collaboration diagram

1 - click [Give Feedback]; 2 - check if user took this course; 3 - success; 4 - evaluate by lectures; 5 - evaluate by labs; 6 - evaluate by teacher competence; 7 - grade the course(0-10); 8 - additional notes; 9 - suggestions; 10 - click [Submit]; 11 - chek fields; 12 - add feedback; 13 - success; 14 - calculate average mark; 15 - success; 16 - feedback submitted; 17 - average mark;

### 3.4 Class Diagram

In Figure 3.10 is represented the class diagram for the website.

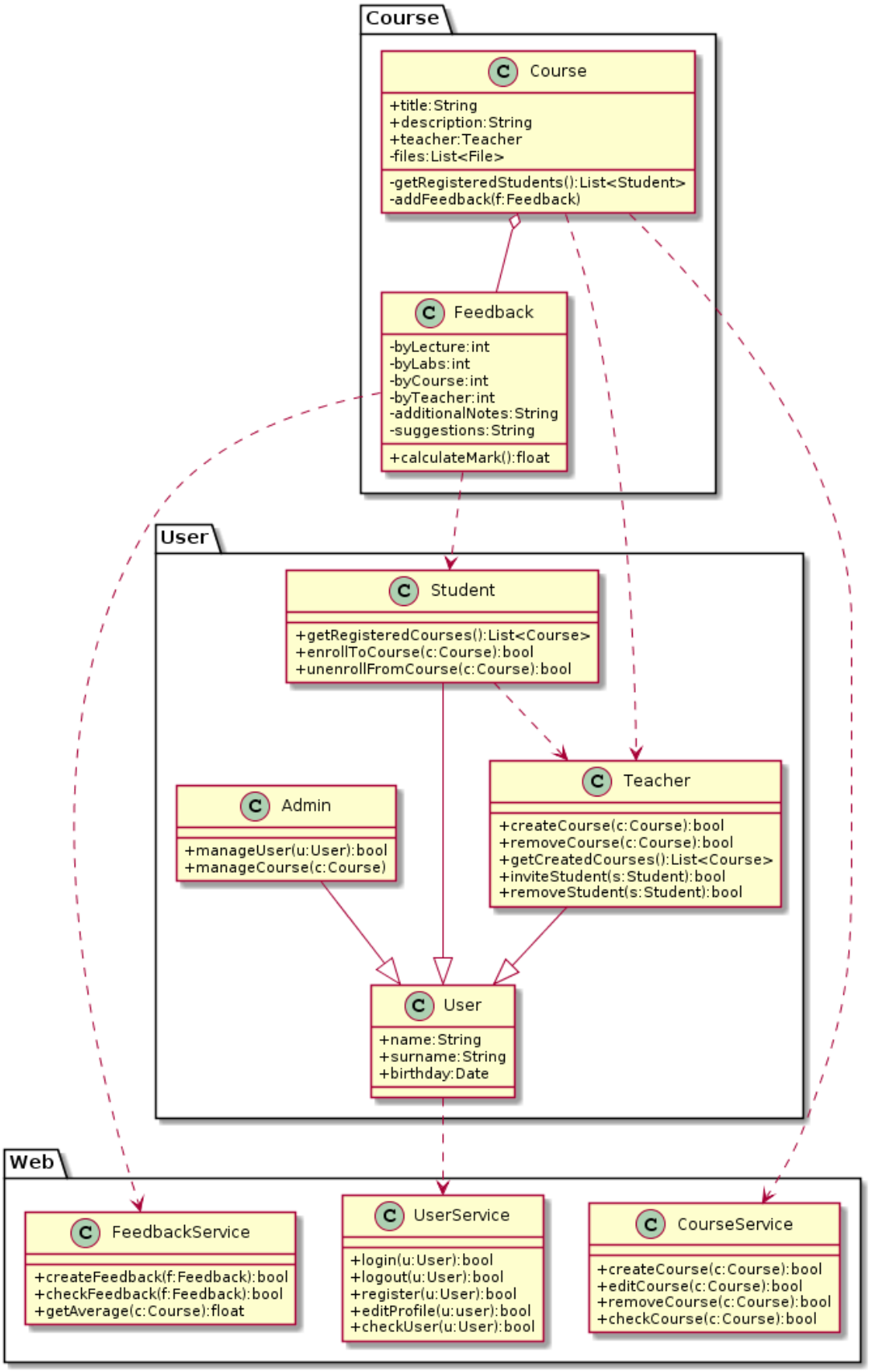


Figure 3.10– Class diagram

The diagram is divided in 3 packages: User, Web, Course. The "Course package" contains the classes Feedback and Course. Feedback is part of Course, that's why they are in agregation relation-

ship. The "User" package contains 3 classses: Admin, Teacher and Student which inherit from the User class. The Feedback depends of Student. The Stdent depends of the Teacher (enroll/unenroll). Also the Course depends of the Teacher.

There is the third package, "Web". It includes FeedbackService, UserService and CourseService, each being a "dependee" for Feeback, User and Course classes respectively.

### 3.5 Statechart Diagram

In Figure 3.11 is represented the statechart diagram for the user account.

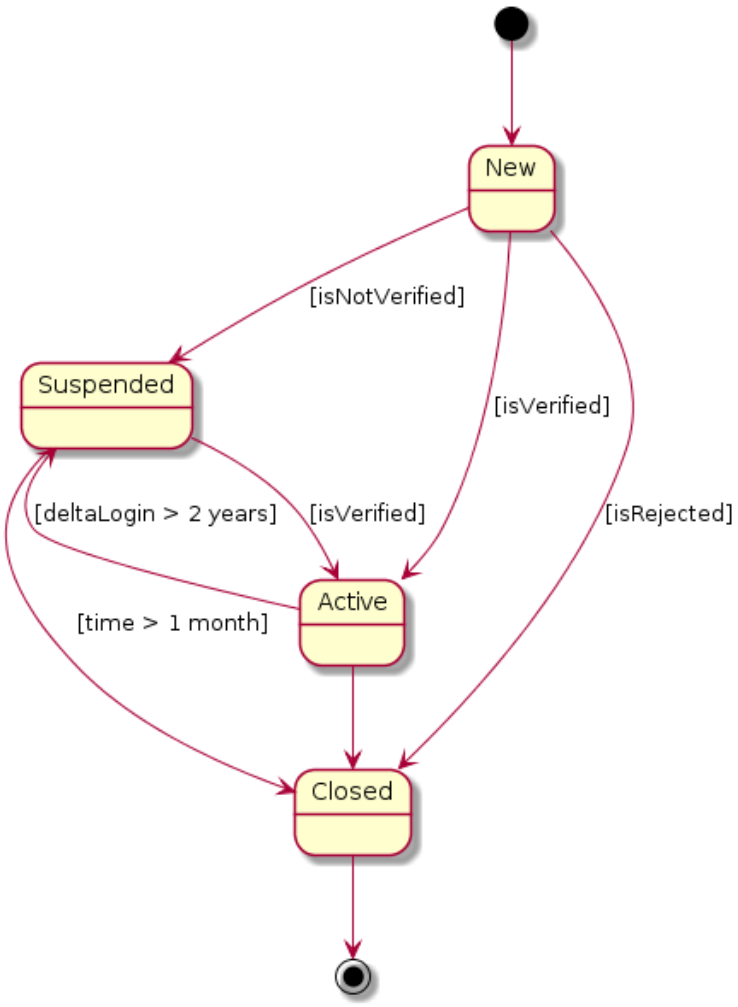


Figure 3.11– User account statechart diagram

Initially all accounts are new (initialization). Those who had valid usernames and passwords, receive a notification by mail. Other ones are rejected, then closed. If the user does not verify the account, it becomes suspended. Suspenden accounts can become active if are verified in less than a month, else they are also closed. An active account with an "Offline" status more than 2 years, becomes supended and receives a new mail notification. Closed accounts have the final state.

In Figure 3.12 is represented the statechart diagram for courses.

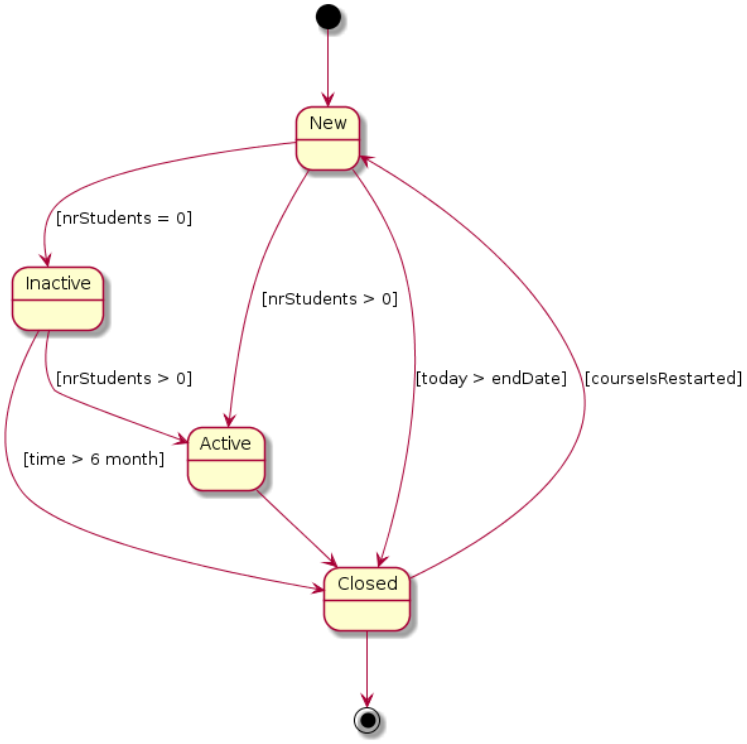


Figure 3.12– Courses statechart diagram

The teacher can create new courses. If he registers some users, the course becomes active. If there are no students at the beginning, the course is inactive. If he can't attract any students in half a year, the course is closed. If the starting date was incorect, the course is also closed. But any course can be restarted, if it was marked as closed.

In Figure 3.13 is represented the statechart diagram for searching a course.

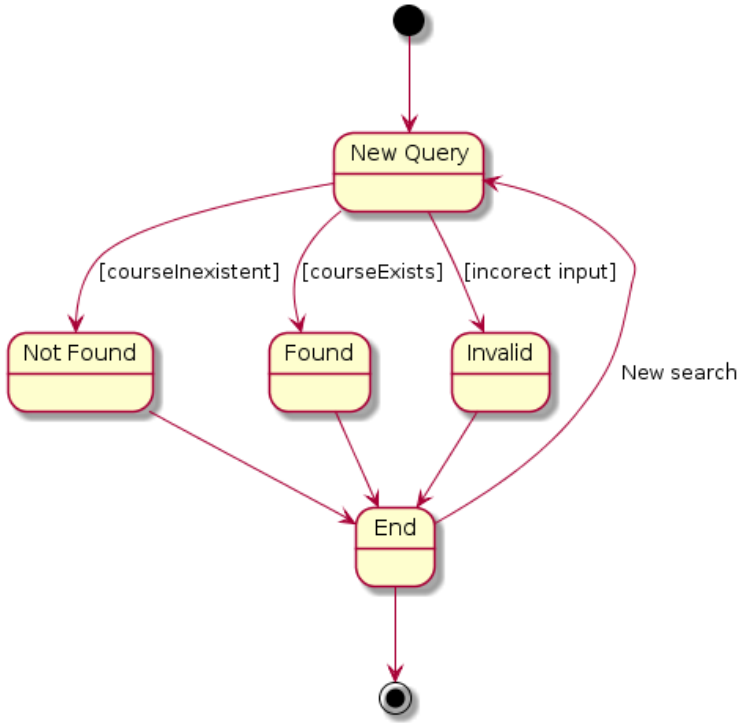


Figure 3.13– Search operation statechart diagram

An user who wants to search a course, creates a new query. If he inputs non-alfanumeric symbols, the result is invalidated. If the course is not found, the result is negative and a propper message appears. Otherwise, the course is found and a positive result with a success message is sent. All states then become finished.

### 3.6 Activity Diagram

In Figure 3.14 is represented the activity diagram for user login.

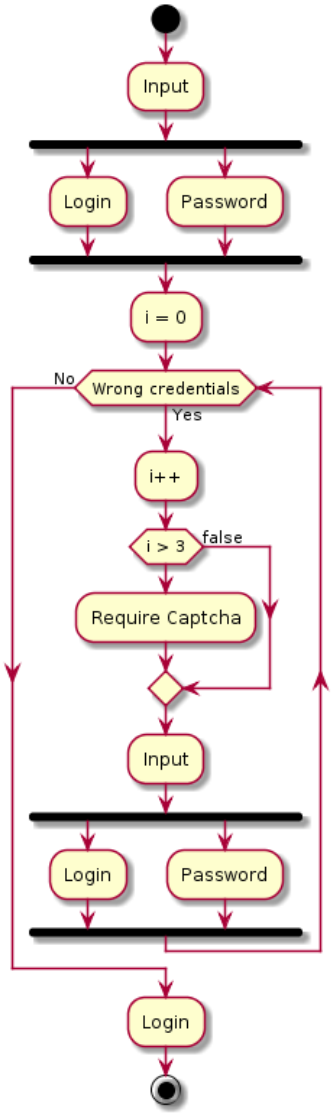


Figure 3.14 – Login activity diagram

To login in an account, the user inputs login and password in any order. Then the input is submitted. If the credentials are right, the user is authenticated successfully. Otherwise, it should try again. If the third attempt is unsuccessful, he is requested to pass the captcha test. Any attempt from that moment will requre the additional test, until he inputs correct data.

### 3.7 Component Diagram

n Figure 3.15 is represented the component diagram for app, using the MVC pattern.

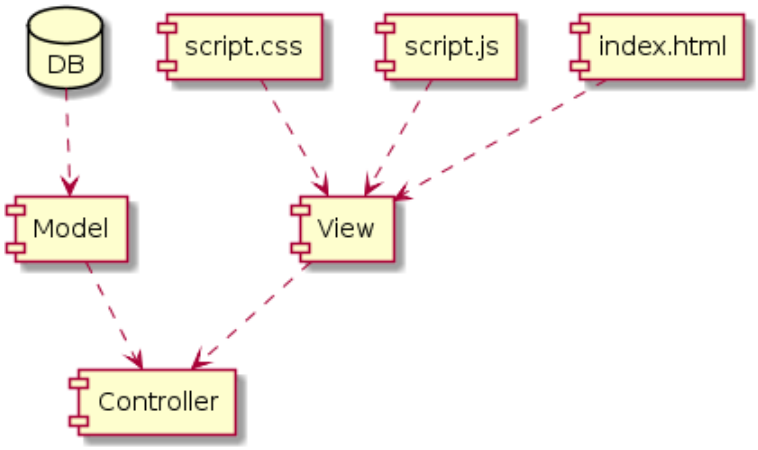


Figure 3.15– MVC component diagram

This is how the architecture of a MVC project looks like. The Model interacts with the database, but depends of the Controller. The View uses the controller to create the final products: .js and .css files which merge into the index.html.

In Figure 3.16 is represented the component diagram for the app, after compiling, minifiing scripts.

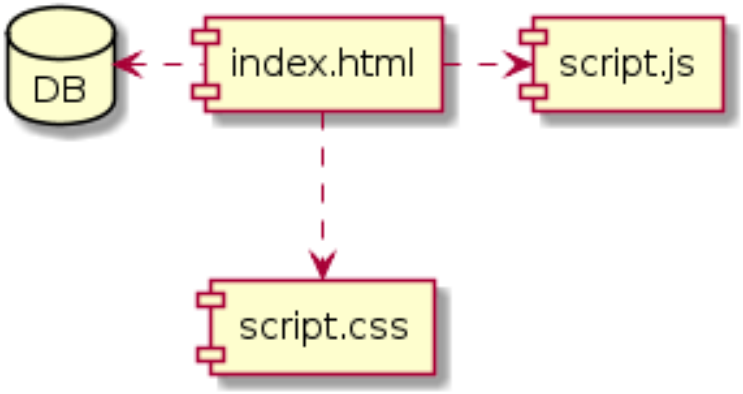


Figure 3.16– Compiled files component diagram

There is represented another step. After building the project, the final product is coposed by index.html file, a style, a scrip file and a database.

### 3.8 Deployment Diagram

In Figure 3.17 is represented the deployment diagram for the app.

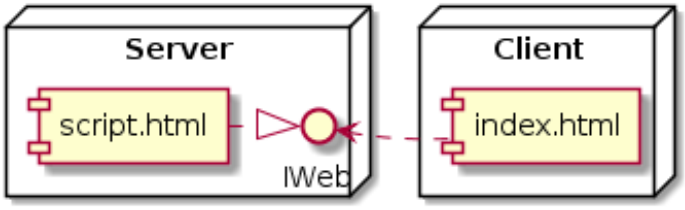


Figure 3.17– Website deployment diagram

The client node has an copy of the final product, provided by the internet. It depends of the interface realised by the server node application.

## 4 Conclusion

UML and Object-Oriented Modeling in general, plays a big role in software development. It forces you to think more abstract and approach OOP from an analytical point of view. Visual modeling creates an actual image of the future application. It produces abstract and accessible descriptions of both system requirements and designs. Also it decreases the "semantic gap" between the system and the real world. Then the system can be constructed using terminology that is almost the same as the non-technical personnel use in everyday business.

At first I had a problem with immagining an entire system. But when I started to build, diagram by diagram, studying every aspect of the project, the picture became clearer and clearer. Also this work made us approach programming products from different sides: SWOT Analysis, Domain Analysis, Top-down and Bottom-up Design, Flow of Events etc. The course achieved it's goal: to build a new set of skills which contribute to the evolution of a good software developer.

## References


1 Xiao He. *A metamodel for the notation of graphical modeling languages*, 2007.

2 Omar El Gabry. *Object-Oriented Analysis And Design - Conceptual Model (Part 2)*, 2017.

3 Stanislav Kozlovski. *A Short Overview of Object Oriented Software Design*, 2018.