



*Bilkent University*

Department of Computer Engineering

## **CS319- OBJECT ORIENTED SOFTWARE ENGINEERING**

### **PROJECT ANALYSIS REPORT**

# **Grader++**

Group Members: Ali Burak Erdoğan, Süleyman Can Özülkü, Ömer Eren,

Yusuf Said Canbaz

---

## **Table of Contents:**

- [1. Introduction](#)
- [2. Requirement Analysis](#)
  - [2.1. Overview](#)
  - [2.2. Functional Requirements](#)
    - [2.2.1 Student](#)
    - [2.2.2 TA \(Teaching Assistant\)](#)
    - [2.2.3 Instructor](#)
  - [2.3. Non-Functional Requirements](#)
  - [2.4. Constraints](#)
  - [2.5. Use Case Models](#)
  - [2.6. User Interface](#)
- [3. Analysis](#)
  - [3.1. Object Model](#)
    - [3.1.1. Class Diagram\(s\)](#)
  - [3.2. Dynamic Models](#)
    - [3.2.1. State Chart](#)
    - [3.2.2. Sequence Diagram](#)
- [4. Conclusion](#)

# 1. Introduction

Grader++ is a code evaluation system we plan to develop in aid to the instructors, teaching assistants, and students. Most of the time, instructors' method of grading students' codes involves steps that can be automated by a software system. For example, an instructor firstly compiles student's code and checks if there is a compilation error, afterwards, he/she checks if the code has memory leaks, and tries a few test cases and compares output to solution. Steps like this can be easily automated by a software and takes a lot of time of the instructors grading code submissions.

There are problems automating these kind of tasks. Students might submit malicious codes which can be caught by instructor by looking at the code, automating this process and handling dangerous codes is difficult. We plan to make grader++ secure by using technologies like containers and make it extendable.

Grader++ has two parts: Core part and UI part. Core part will be an extendible system composed of evaluators, evaluating submissions concurrently. UI part will give ability to instructors to add homeworks to the system and students to submit their code. Core part will be a restful api server, accepting task descriptions and submissions. UI part will interact core part through http requests and will have a web frontend in which instructors and students could interact core part with ease. Separating core part from UI will enable us to easily extend our system and will enable us to use more than one server, so that UI part will be accessible under heavy load.

## **2. Requirement Analysis**

### **2.1. Overview**

As the group #9 we decided Grader++ for our project's name . We aim to develop an automated grading service. This service can be useful for a lot of different purposes, however we decided to develop Grader++ for helping with grading programming homeworks, and for helping students with the development of their programming skills by providing them with analysis tools. We want to make a service which is somewhat similar to hackerrank and codeforces. We decided to use JAVA for implementation. There is a brief information about Grader++ in "Introduction" part of this report. After short description of the game, there is an explanation of why we have chosen to work on Grader++. In this report, requirements of the project will be given under two headings: Functional Requirements and Nonfunctional Requirements. System models of the service will also be described in detail. System models section includes domain analysis, use case analysis, sequence diagrams, state and activity diagrams. After System models section, detailed description of Grader++ features will be provided.

## **2.2. Functional Requirements**

### **2.2.1 Student**

#### **Login**

User name and password required log-in mechanism. Login system is just a standard log-in mechanism. This is common to for all types of users. (TA, instructor, student) Because it is standard among other user types, it is not going to be rewritten.

#### **Home Page**

Home page is going to have 2 lists and 2 couple of buttons. Each of lists will be relevant to a couple of button. Button couples are located at the bottom of the lists. A list will be tasks, the other will be courses. Tasks will be the tasks which are assigned to the courses which the current student subscribed. Task list items will be selectable. To start an action about task, user must be selected a task. User is not allowed to select more than 1 task. If there is no selected task, then 2 buttons will be disabled. One button to show submissions made by the student, another button to generate a new submission for selected task. The other list will be the course list. If the student is subscribed to a course then it denoted with a tick near the course. To start an action about course, user must be selected a course. User is not allowed to select more than 1 course. If there is no selected course, then 2 buttons will be disabled. One button to subscribe to the selected course, another button to to unsubscribe from the selected course.

#### **Submit Code**

Among the user types only "Student" can submit code for a task. Firstly, student should select the task to upload from main page. Then, the student is allowed to upload his or her file for that task. The student is going to upload his or her source code file for that assignment. Other file types are not allowed to upload.

### **View Submissions.**

All user types are allowed to view submissions that are submitted by students for a particular task (except that students cannot view other students submissions).

This is just a function for viewing submissions' status, Users cannot add or change something from this view. Firstly student should select the task to upload from main page. Then, the student is allowed to view submissions to the selected task.

## **2.2.2 TA (Teaching Assistant)**

### **Home Page**

This page will also be similar to the students' home page. It will have a list of tasks and 2 buttons. Tasks will be assigned by instructor. Without selecting a task, TA is not allowed to click a button. Buttons will be disabled when there is no selected task. There will be one button to show submissions which are made by students, another button to edit a selected task.

### **Edit Task**

TA can change some properties of the task. TA can add or delete test cases. TA can change due date of the task.

### **View Submissions.**

This function is very similar among all users. TA is allowed to view all the submissions for the selected task.

### **2.2.3 Instructor**

#### **Home Page**

Instructors' home page and Students' home page is very similar to each other. Home page is going to have 2 lists and 2 couple of buttons. Each of lists will be relevant to a couple of button. Button couples are located at the bottom of the lists. A list will be tasks, the other will be courses. Tasks will be the tasks which are generated by the current Instructor. Task list items will be selectable. 1 Button to delete the selected list items, another button to add a new task. User is allowed to select more than 1 task. If there is no selected task, then delete button will be disabled. The other list will be the course list. Courses that current Instructor generated will be the items of the list. Course list items will be selectable. 1 Button to delete the selected list items, another button to add a new course. User is allowed to select more than 1 course. If there is no selected course, then delete button will be disabled.

#### **View Submissions**

This view is very similar among all users. Instructor is allowed to view all the submissions for the selected task.

#### **Add Task**

Instructors are allowed to add task for their courses. Add task requires to type name of the task, to select course name, to type description for the task, to select due date for the task, to assign a TA for task, select test files etc. This part is not certain and details could change after implementation.

### **Add Course**

Instructors are allowed to add courses. Add course requires to type name of the course, to type description of the course, to select last day of the course, to specify maximum participator count, to select TA.

## **2.3. Non-Functional Requirements**

### **Ability to evaluate codes concurrently**

We plan to make grader++ robust. Most of the courses taught in CS departments have large student base, and it is possible that there will be a significant amount of students with procrastinatory tendencies who will submit their homework at the last minute. Hence, there will be a huge load at times, and grader++ should handle more than one evaluation at the same time.



## **Extendibility**

Since every CS course different requirements in their homeworks, grader++ should be highly extendable. For example, a data structure course might need time and memory constraints, whereas an OS course might have constraints on number of processes etc.

## **User-Friendly User Interface**

Since most of the courses given has lifespan of one semester, our project need to be adopted easily, with a basic learning curve. User-Interface must have help notifications and introduction screens.

## **Performance**

Project needs to evaluate student's code quickly. This should be achieved by using meaningful constraints (max time limit etc) and concurrent grading.

## **Security**

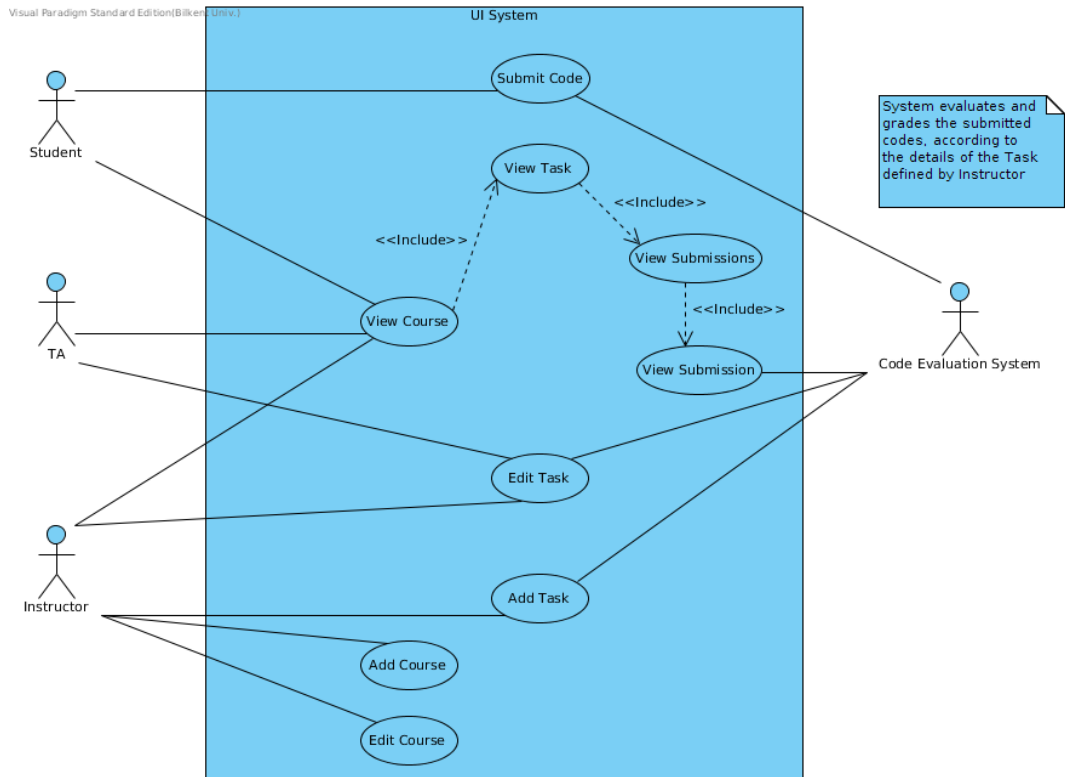
Plagiarism is a big problem faced by many instructors throughout their teaching career. Our project needs to isolate users code from the system, and should disable students access to other students accounts and codes.

## **2.4. Constraints**

- The user interface should be in the web platform. Thus, the user can interact with the system through a web browser.
- The web-based user interface should be built with Java and by following the MVC (Model-View-Controller) principle in order to handle the future modifications quickly.
- The back-end mechanism, the essential part of the code evaluating system should also be implemented in Java, and if needed C++.
- The live server machine which will serve to code compiling, evaluating and grading, should use Ubuntu 12 or above.

## **2.5. Use Case Models**

This sections explains the main use cases of Grader++ in detail.



### 2.5.1 Submit Code

**Use Case Name:** Submit Code

**Primary Actor:** Student

**Stakeholders and Interests:**

- Student aims to make his/her submission to the related Task of his/her Course.
- Submission must be received by System and put into progress of evaluation and grading.

**Pre-condition:** Student must be logged in. Also the Course that Student is taking must have a Task.

**Post-condition:** The submission must be obtained by System and saved. System should evaluate the submission in terms of requirements criteria.

**Entry Condition:** Student presses the “Quick Submission” link from Main Page or “Submit Code” button from a particular Task’s page.

**Exit Condition:** Student presses the “Submit” button of the submission form.

**Event Flow:**

1. Student selects a Task from the list.
2. Student selects the “.zip” file which contains all the required files from “Browse” button.
3. Student presses the “Submit” button.

**Alternative Flows:**

- A. If Student wants to cancel submission:
  - A.1. Presses the “Cancel Submission” button of the form.
  - A.2. He/she returns to the Main Page.

## **2.5.2 View Task**

**Use Case Name:** View Task

**Primary Actor:** Student, TA, Instructor

**Stakeholders and Interests:**

- Student should see the details and evaluation state of the submission he/she made.
- Instructor and TA should see the details and evaluation state of the submissions made by students on the selected Task.
- Student, Instructor, TA should see all the details of a specific Task.

**Pre-condition:** User must be logged in.

**Post-condition:** -

**Entry Condition:** Instructor, TA or Student presses a particular Task's "View Submissions & Details" link which is located on a specific Course's page.

**Exit Condition:** User presses the "Return to Main Page" button.

**Event Flow:**

1. User should see a data table which consists of rows that each represents a submission, and columns such as submitter name, submission date, evaluation grade, the name of the Task that is submitted to etc.

**Alternative Flows:**

- A. If User wants to see the details of the submission:
  - a. Presses the button at the left corner of the row of Submission.
  - b. User should see the details in a new page.
- B. If User wants to reorder the results by a column in the table:

- a. Presses the head of the column.
- b. All the rows should be reordered by the column (ascending or descending)

### **2.5.3 Add Task**

**Use Case Name:** Add Task

**Primary Actor:** Instructor

**Stakeholders and Interests:**

- User should be able to define a new Task and determine its qualifications.

Also, User can assign a TA as a manager.

**Pre-condition:** User must be logged in. User should be authorized to add a Task.

**Post-condition:** The Task should be saved to the System.

**Entry Condition:** User presses the “Add Task” button from a particular Course’s page.

**Exit Condition:** User presses the “Return to Main Page” button.

**Event Flow:**

1. User types the name of Task into the relevant text box.
2. User types the description of Task into the relevant text box.
3. User uploads test files for the new Task using “Add a Test File” button.
4. User specifies the grading value of each test case (out of 100).

5. User specifies the penalties for defined conditions (e.g. Memory Leak penalty, Late Submission penalty)
6. User chooses a due-date through a date-picker widget.
7. User defines the code compiling options via uploading a “makefile” file.
8. User assigns a Teaching Assistant as a manager for this new Task by selecting a TA by its name in the select-box.
9. User presses the “Add Task” button of the form.

#### **Alternative Flows:**

- A. If User wants to remove test-files or “makefile”s uploaded:
  - a. Presses the “Remove” button at the right corner of the file icon which is located at file list panel.

#### **2.5.4 Edit Task**

**Use Case Name:** Edit Task

**Primary Actor:** TA, Instructor

#### **Stakeholders and Interests:**

- User should be able to modify a specific Task’s qualifications such as task description, grading policy, due date, required files, makefiles, test files.

**Pre-condition:** User must be logged in. User should be authorized to modify the Task.

**Post-condition:** The Task's specifications on the System should be modified accordingly.

**Entry Condition:** User presses the "Edit Task" button which belongs to a specific Task from View Tasks page.

**Exit Condition:** User presses the "Discard Changes" button.

**Event Flow:**

1. User makes required modifications on the selected Task.
2. User presses the "Save Changes" button of the form.

**Alternative Flows:**

- A. If User wants to change the name of the Task:
  - a. Types the new name into the relevant text box.
- B. If User wants to change the description of the Task:
  - a. Types the new description into the relevant text box.
- C. If User wants to change the grading policy:
  - a. Specifies the grading value of each test case (out of 100).
  - b. Specifies the penalties for defined conditions (e.g. Memory Leak penalty, Late Submission penalty)
- D. If User wants to change due date of submission:
  - a. Chooses the date through a date-picker widget.
- E. If User wants to define the code compiling options:
  - a. Uploads a "makefile" file to be run by System.
- F. If User wants to add test files:
  - a. Uploads test file with "Add a Test File" button.
  - b. Enters the grading value of this test file.
- G. If User wants to remove test-files or "makefile"s:
  - a. Presses the "Remove" button at the right corner of the file icon which is located at file list panel.



### **2.5.5 View Submissions**

**Use Case Name:** My Submissions

**Primary Actor:** Student

**Stakeholders and Interests:**

- Student should be able to see all the details of a Task.

**Pre-condition:** Student must be logged in.

**Post-condition:** -

**Entry Condition:** Student presses “My Submissions” link on the Main Page.

**Exit Condition:** User presses the “Return to Main Page” button.

**Event Flow:**

1. Student should see a data-table which consists of all the Submissions made by him/her.

**Alternative Flows:**

- A. If User wants to see detailed information of a specific Submission:
  - a. Presses the “Details” button of this Submission.
  - b. User should be redirected to the Submission page.

## 2.5.6 Add Course

**Use Case Name:** Add Course

**Primary Actor:** Instructor

**Stakeholders and Interests:**

- User should be able to define a new Course and determine its qualifications. Also, User can assign a TA as a manager.

**Pre-condition:** User must be logged in. User should be authorized to add a Task.

**Post-condition:** The Task should be saved to the System.

**Entry Condition:** User presses the “Add Task” button from View Course page.

**Exit Condition:** User presses the “Return to Main Page” button.

**Event Flow:**

1. User types the name of Task into the relevant text box.
2. User types the description of Task into the relevant text box.
3. User uploads test files for the new Task using “Add a Test File” button.
4. User specifies the grading value of each test case (out of 100).
5. User specifies the penalties for defined conditions (e.g. Memory Leak penalty, Late Submission penalty)
6. User chooses a due-date through a date-picker widget.
7. User defines the code compiling options via uploading a “makefile” file.

8. User assigns a Teaching Assistant as a manager for this new Task by selecting a TA by its name in the select-box.
9. User presses the “Add Task” button of the form.

**Alternative Flows:**

- A. If User wants to remove test-files or “makefile”s uploaded:
  - a. Presses the “Remove” button at the right corner of the file icon which is located at file list panel.

### **2.5.7 View Course**

**Use Case Name:** View Course

**Primary Actor:** Student, TA, Instructor

**Stakeholders and Interests:**

- User should be able to see all the details of a Course, including all the Tasks assigned by Instructor to that Course.

**Pre-condition:** User must be logged in.

**Post-condition:** -

**Entry Condition:** User presses the “Courses” button from Main Page.

**Exit Condition:** User presses the “Return to Main Page” button.

**Event Flow:**

1. User selects one of the Courses which he/she is related with, from the select-box by the Course’s name.
2. A part of the page reloads with the details of the selected Course.

**Alternative Flows:**

- A. If User wants to view a Task of that Course:
  - a. Presses one of the Tasks' "Details" button on the panel.
  - b. User should be redirected to that specific Task's page.
- B. If User is an Instructor and wants to add a new Task to the Course:
  - a. Presses the "Add Task" button on the panel.
  - b. User is redirected to the Add Task page.

## 2.6. User Interface

### 2.6.1 Instructor Add Course Screen

The screenshot shows a web browser window titled "Add Course". The address bar contains "http://". The breadcrumb navigation is "Main Page > Courses > Add Course > ...". The user is identified as "Instructor: Hüseyin Özgür" with a user icon. The form is titled "Add Course" and contains the following fields:

- Name:** A text input field.
- Description:** A text input field.
- last day of Course:** A date input field showing "20/ 10 / 2015" with a calendar icon.
- max student count:** A slider control.
- select TAs:** A list of checkboxes with the following names:
  - ☐ Hamzah Agari
  - ☒ Murat Demirbüken
  - ☐ Emre Kurt
  - ☐ Ali Ölmez

## 2.6.2 Instructor Add task

Add Task

←

→


✕

🏠

http://

🔍

[Main Page](#) > [Courses](#) > [Add Task](#) > ...

Instructor: Hüseyin Özgür 

Add Task

Course Code

CS 201


▼

Name

Description

Due date

20/ 10 / 2015



Manager TA

Select one of TAs...

▼

Makefile

Browse

Test files

Browse

+ Add more...

test1.cpp

20

Points

test2.cpp

30

Points

Penalties

Late Submission

▼

+ Add more...

Memory Leak

40

Points

Add Task

Return to Main Page

## 2.6.3 Instructor Home Page

Home Page

http://

[Home Page](#) > ...

Instructor: Özgür Tan

Task	Due Date	Result
cs 201	16.10.2015	56/100
cs 202	16.10.2015	56/100
cs 102	16.10.2015	56/100
cs 101	16.10.2015	56/100
cs 315	16.10.2015	56/100
cs 319	16.10.2015	56/100

add Task

delete Task

Course	Student Count
cs 101	45 / 55
cs 201	49 / 55

add Course

delete Course

## 2.6.4 Login page

The image is a hand-drawn sketch of a web browser window. The window's title bar is labeled "Login". The address bar contains the text "http://". Below the address bar, the page content is displayed. At the top left of the content area, there is a link labeled "Login Page" followed by a right-pointing arrow and an ellipsis. Below this, there are two input fields. The first is labeled "User Name:" with a small person icon to its left. The second is labeled "Password" with a small key icon to its left. At the bottom of the form, there are two buttons: one labeled "login" and one labeled "create account". The entire sketch is rendered in a simple, hand-drawn style with black outlines and no color.

## 2.6.5 Instructor View Submissions Page

Submissions

http://

q

[Main Page](#) > [Tasks](#) > [Submissions](#) > ...

Instructor: Özgür Tan

View Submissions

### Submissions of cs202\_hw1



Submitter	Submission Date	Task	Grade
Giacomo Guilizzoni	20/09/2015	cs202_hw1	80
Marco Botton	20/09/2015	cs202_hw1	Processing...
Tuttofare			
Mariah MacLachlan	19/09/2015	cs202_hw1	90
Better Half			
Valerie Liberty	09/09/2015	cs202_hw1	30
Head Chef			
Guido Jack Guilizzoni	20/09/2015	cs202_hw1	Rejected <a href="#">x</a>

back




## 2.6.6 Student Home Page

Submit Code



[Home Page](#) > ...

Student: Ali Burak Erdoğan 

Task	Due Date	Result
cs 201	16.10.2015	56/100
cs 202	16.10.2015	never submitted
cs 102	16.10.2015	56/100
cs 101	16.10.2015	56/100
cs 315	16.10.2015	56/100
cs 319	16.10.2015	56/100

Show Submissions

Submit

## 2.6.7 Student Submit Code Page

The screenshot shows a web browser window titled "Submit Code". The address bar contains "http://". The breadcrumb trail is "Main Page > Submit Code > ...". The student's name, "Student: Ali Burak Erdoğan", is displayed in the top right corner next to a user icon. The main content area is titled "Submit Code for CS 201 Task HW02" and contains a large text input field. Below the input field, there are four buttons: "Select your zip file..." (with a file icon), "Browse", "Submit", and "Cancel Submission".

Submit Code

http://

Main Page > Submit Code > ...

Student: Ali Burak Erdoğan

Submit Code for CS 201 Task HW02

Select your zip file... Browse

Submit Cancel Submission

## 2.6.8 Student View Submissions Page

Submissions

←

→


×

🏠

http://

🔍

[Main Page](#) > [Tasks](#) > [Submissions](#) > ...

Student: Ali Burak Erdoğan 

View Submissions

Submissions of cs202\_hw1

Submitter	Submission Date	Task	Grade
Giacomo Guilizzoni	20/09/2015	cs202_hw1	80
Marco Botton	20/09/2015	cs202_hw1	Processing...
Tuttofare	19/09/2015	cs202_hw1	90
Mariah MacLachlan	09/09/2015	cs202_hw1	30
Better Half	20/09/2015	cs202_hw1	Rejected <a href="#">x</a>
Valerie Liberty			
Head Chef			
Guido Jack Guilizzoni			

back

## 2.6.9 TA home page

Home Page

http://

Home Page > ...

TA: Murat Demirbügen

Task	Due Date	Result
cs 201	16.10.2015	56/100
cs 202	16.10.2015	56/100
cs 102	16.10.2015	56/100
cs 101	16.10.2015	56/100
cs 315	16.10.2015	56/100
cs 319	16.10.2015	56/100

Show Submissions Edit

## 2.6.10 TA Edit Task

← → × ↩

http://

⌕

[Main Page](#) > [Courses](#) > [Edit Task](#) > ...

TA: Murat Demirbükten

⚙

Edit Task CS 201 HW01

Due date

20/ 10 / 2015

📅

Makefile

Browse

test files

test02.cpp

test03.cpp

test04.cpp

test05.cpp

add

delete

done

## 2.6.11 TA View Submissions

Submissions

←

→


✕

🏠

http://

🔍

[Main Page](#) > [Tasks](#) > [Submissions](#) > ...

TA: Murat Demirbügen 

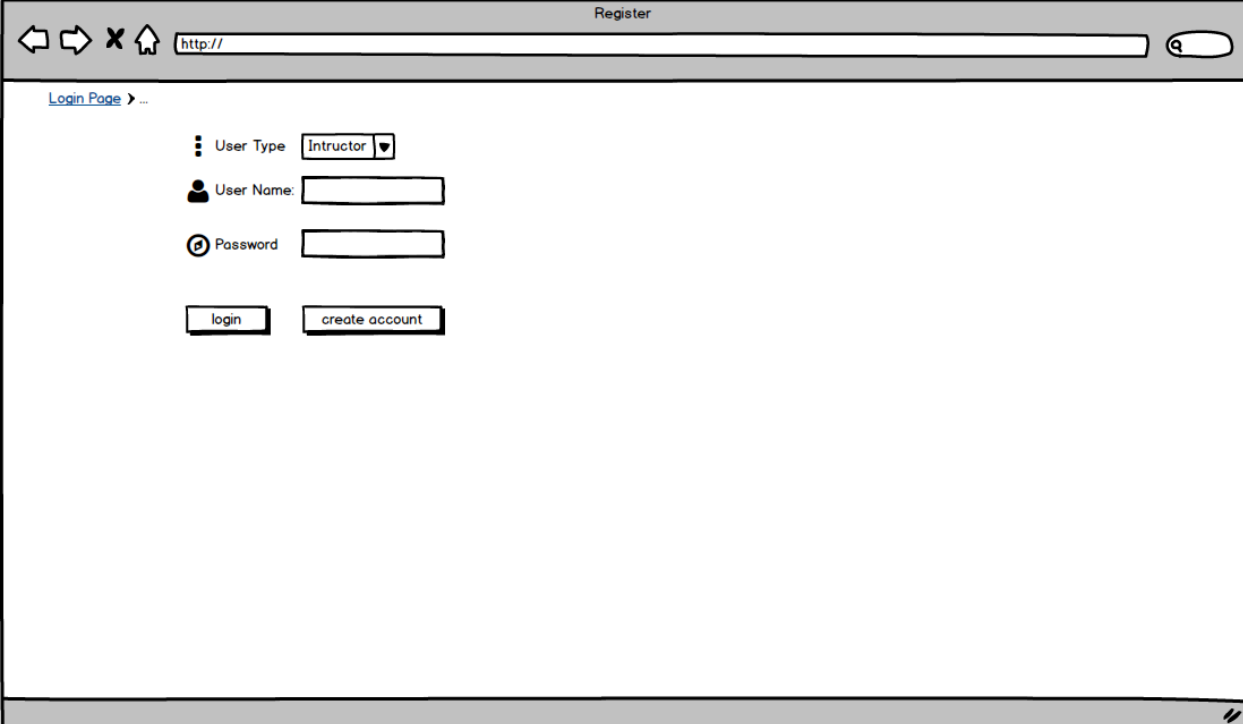
View Submissions

Submissions of cs202\_hw1

Submitter	Submission Date	Task	Grade
Giacomo Guilizzoni	20/09/2015	cs202_hw1	80
Marco Botton Tuttofare	20/09/2015	cs202_hw1	Processing...
Mariah MacLachlan Better Half	19/09/2015	cs202_hw1	90
Valerie Liberty Head Chef	09/09/2015	cs202_hw1	30
Guido Jack Guilizzoni	20/09/2015	cs202_hw1	Rejected <a href="#">x</a>

back

## 2.6.12 Register Page



A screenshot of a web browser window titled "Register". The address bar shows "http://". The page content includes a breadcrumb link "Login Page > ...", a "User Type" dropdown menu set to "Instructor", a "User Name:" label with a text input field, a "Password:" label with a text input field, and two buttons labeled "login" and "create account".

Register

[Login Page > ...](#)

User Type:

User Name:

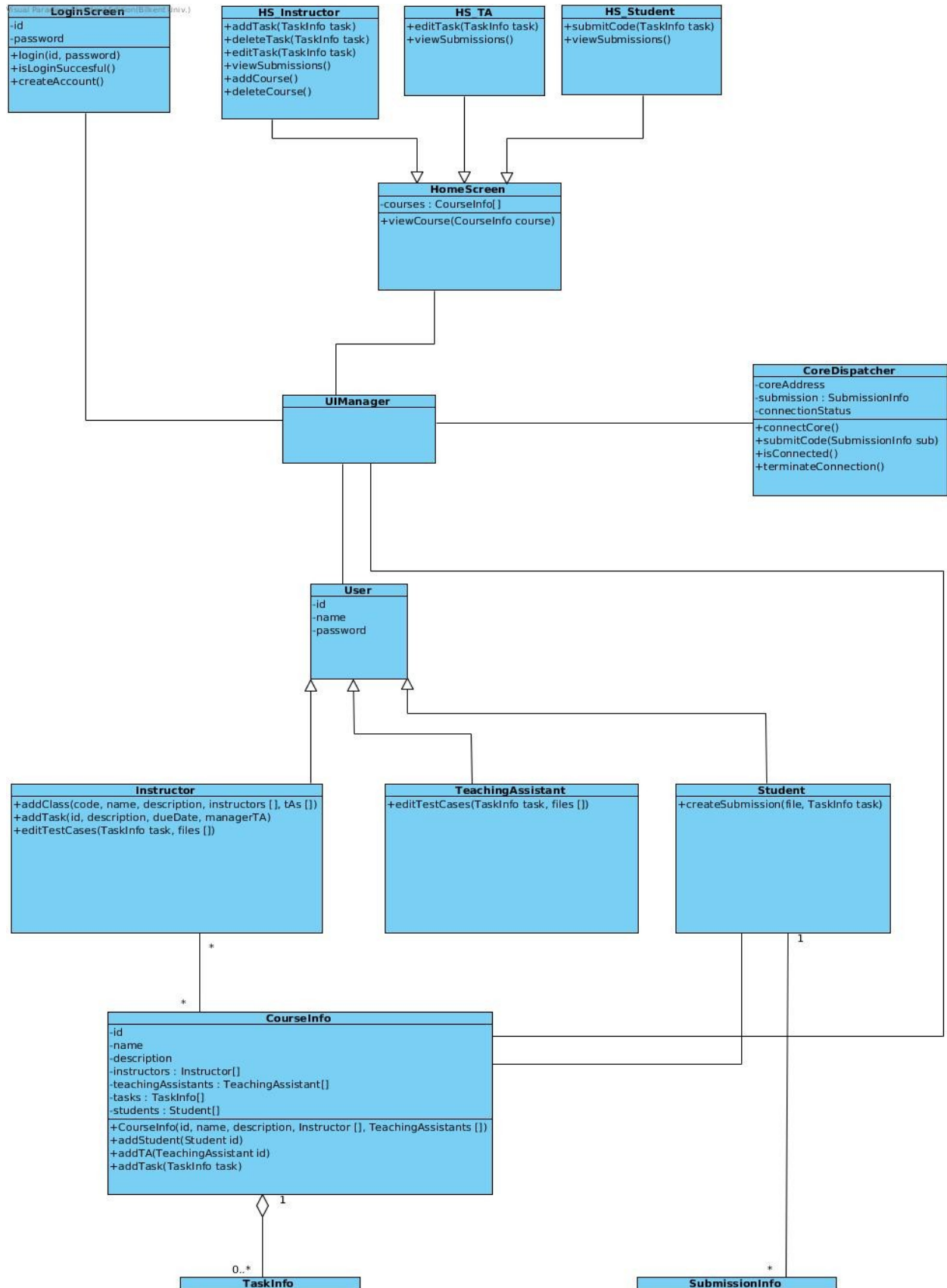
Password:

## **3. Analysis**

### **3.1. Class Diagram(s)**

#### **3.1.2. Client Class Diagram**





We have a number of classes to fulfill the different needs. We have some Controller classes such as LoginScreen, HomeScreen, UIManager, CoreDispatcher as intermediates between Views and Models. The Views will be web pages. The Models are CourseInfo, TaskInfo, SubmissionInfo, User.

- LoginScreen: It is a controller class which will be used to handle the logic of the login page. It holds id, password as members.

createAccount and login are among the methods of this class.

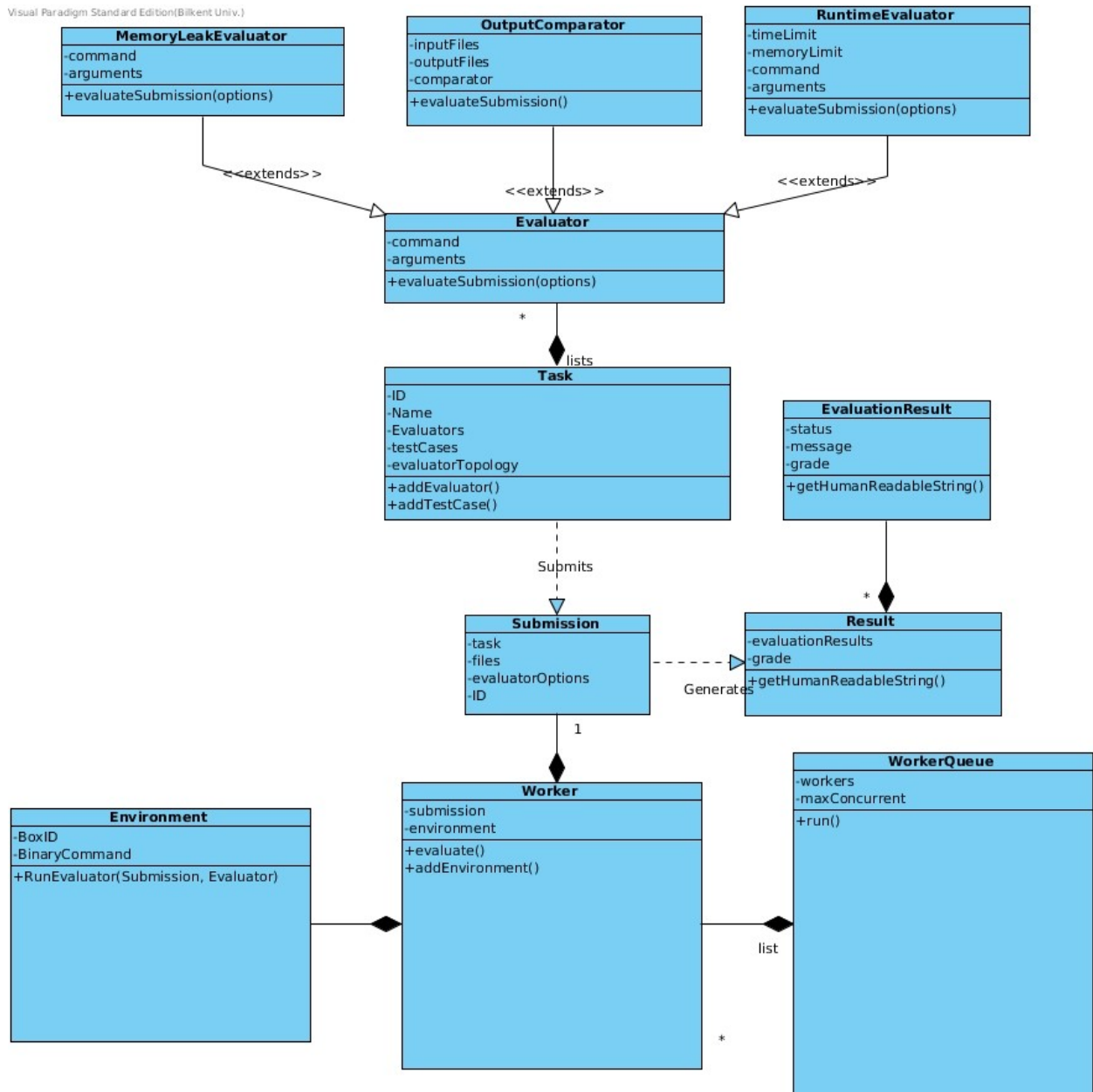
- HomeScreen: This is the controller class of the main page which a User can interact with the core system. This class is inherited by HsInstructor, HsTA, HsStudent classes. Each of the children of this class supplies different functionalities. For instance, when a User logged in to the system as an Instructor, a HsInstructor class is instantiated. Therefore, functions such as addTask, deleteTask can be called.

- UIManager: This class is responsible for handling all the interaction between User and the interface. It holds references to controller classes of all the web pages, model classes to deal with the models such as CourseInfo, SubmissionInfo and so on. It is also linked with the CoreDispatcher object to make requests to the server.

- CoreDispatcher: This class is used for establishing the connection between User Interface and the Core System. Core system is a live server which can receive submitted codes and compile it, then evaluate it according to the preferences. This CoreDispatcher class can make all the required network requests such as connectCore, submitCode etc.

- User: This is a model class which represents a single user of UI. When a person is signed in to the system, according to the role of the actor, an instance of a child class of User is instantiated. (e.g. if TA is logged in, TeachingAssistant object is instantiated)

### 3.1.2 Core Class Diagram



Our core package, which will be standalone web server, accepting https requests of task submissions and task definitions will be separated from our UI part. Core server will not know about users, and it will be stateless (It will not hold the submission results, it will just return it and forget it). However it will task definitions in a database. There several classes which will fulfill the core's task description. Evaluators will be runned on the submission according to given topology in task description, their log will be returned in submission object, which will be consist of evaluatorResults. Here is the description of core classes:

- **MemoryLeakEvaluator:** MemoryLeakEvaluator will run the submission against test case and checks if there is a memory leak and it will generate an evaluatorResult with a binary grade (1 means there isn't a memory leak, 0 means there is a memory leak).
- **OutputComparator:** OutputComparator will compare each resulting output of testcases against official outputs. Simplest version of OutputComparator will check if output generated by submission is same as the official output provided by instructor. For some cases, it might not be enough and because of that OutputComparator have a field called comparator which can be specified by instructor to do more complex comparison tasks.
- **RuntimeEvaluator:** Runtime evaluator will check constraints on submissions, specified by task description. These constraints will be memory and time constraints. For example: A task might need for every submission to be run at most one second for each test case and for every run it will only use 256 mb of data segment and no heap segment.
- **Evaluator:** MemoryLeakEvaluator, OutputComparator, RuntimeEvaluator's base class is the Evaluator class. Evaluator class will be an abstract class, which will ensure all evaluators have the evaluateSubmission function and it will have some general private helper function to aid child classes perform their evaluations.

- Task: Task will hold the description of a task specified by instructor. Each task will have test cases, an ID and a Name(which will be used when communicating with client). Also each task will have a evaluator topology, which will define relationship between evaluators, so that no more job than necessary will be done (For example, if there is a relationship between MemoryLeakEvaluator and RuntimeEvaluator, RuntimeEvaluator will not be run on submission until MemoryLeakEvaluator has a positive grade).

- Submission: Submission Class will hold all the information about a user's submission, and this information will be used by workers to evaluate the submission.

- WorkerQueue: WorkerQueue will be a queue of workers. Since, core system will run evaluations concurrently which is specified by non functional requirements, WorkerQueue will orchestrate when a worker should run on which evaluator for which submission.

- Worker: Worker class will run a submission for one evaluator. Main job of worker queue is to have a environment for evaluation and handling low level operations(file operations, setting up a docker environment etc.). By using workers and worker module, handling concurrency will be much easier.

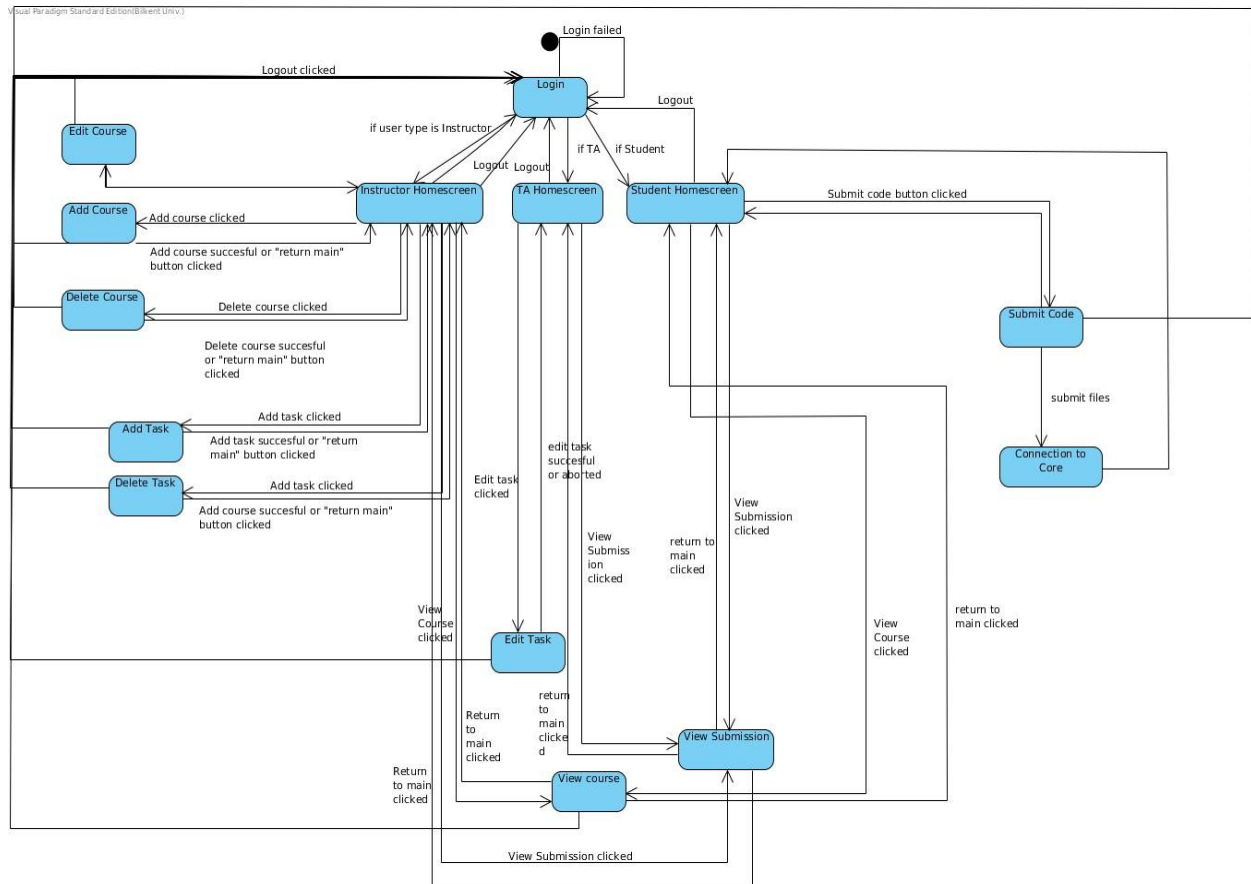
- Result: Result will be a list of evaluationResults and a aggregated grade. It will generated when all the workers are done on a submission and will be returned back to the client.

- EvaluatorResult: EvaluatorResult will hold a grade for a evaluator's result on a submission. It will be both used by result object and topology specified in task description.

- Environment: Environment will ensure security functionality specified by non functional requirements by handling the docker environment. Docker is a linux container technology that creates an isolated environment.

## 3.2. Dynamic Models

### 3.2.1. State Chart



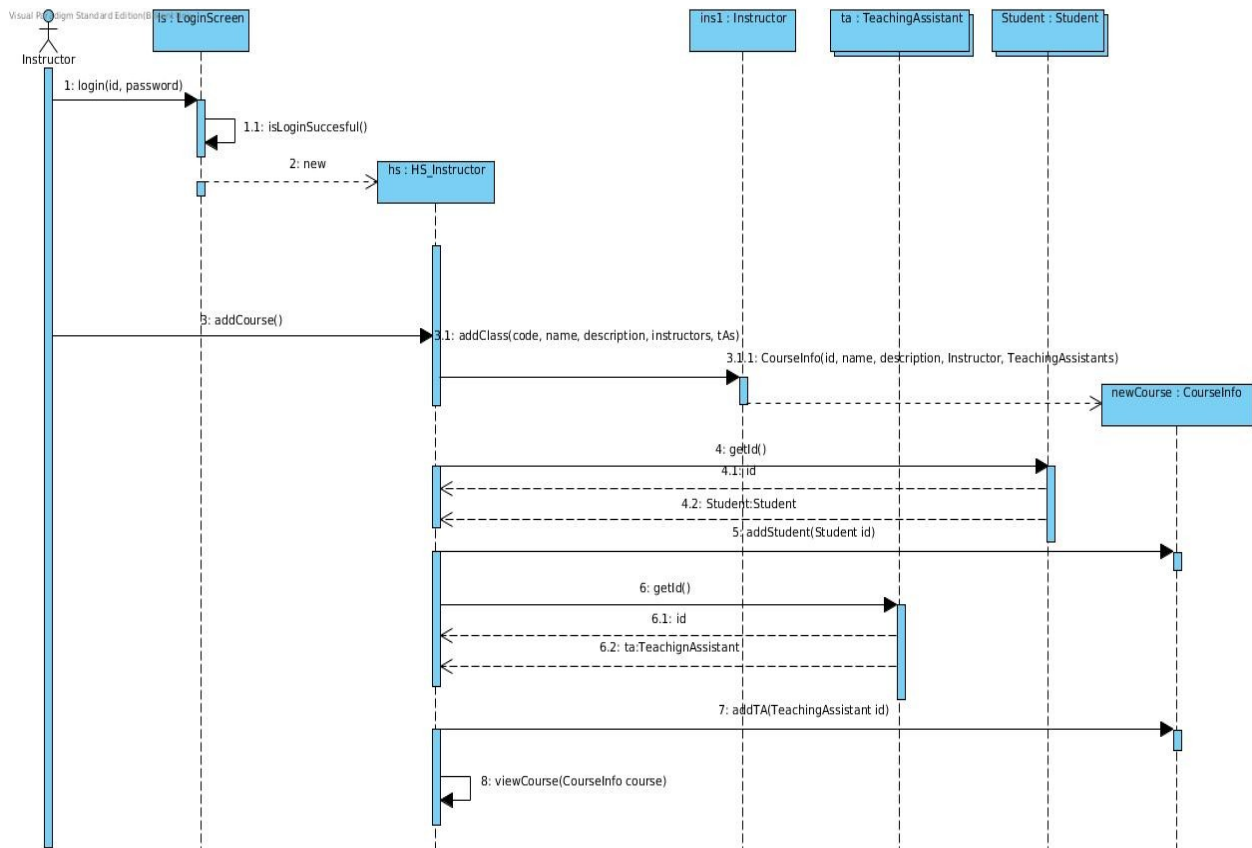
### 3.2.2. Sequence Diagrams

After user accesses the login screen and successfully enters his/hers id and password the system will provide the appropriate homepage depending on the type of user (instructor, teachign assistant or student). Each user is allowed to do spesific operations on client, for example instructors can add courses to system, add tasks(homeworks) for a course, and students can submit their homeworks through client. Among these operations, significant ones' sequence diagrams are provided below.

### **3.2.2.a) Add Course**

Instructors are the only ones allowed to add courses to system. So, the user should login to the system, and after it is confirmed that user is of type "Instructor" client will provide user with homescreen for instructors. User is provided with add course button in the homescreen, and when he/she clicks on it Instructor class will construct a new CourseInfo object that with the properties indicated by user. After this process user will be able to get the information of students and teaching assistants so that they can be subscribed to the course.

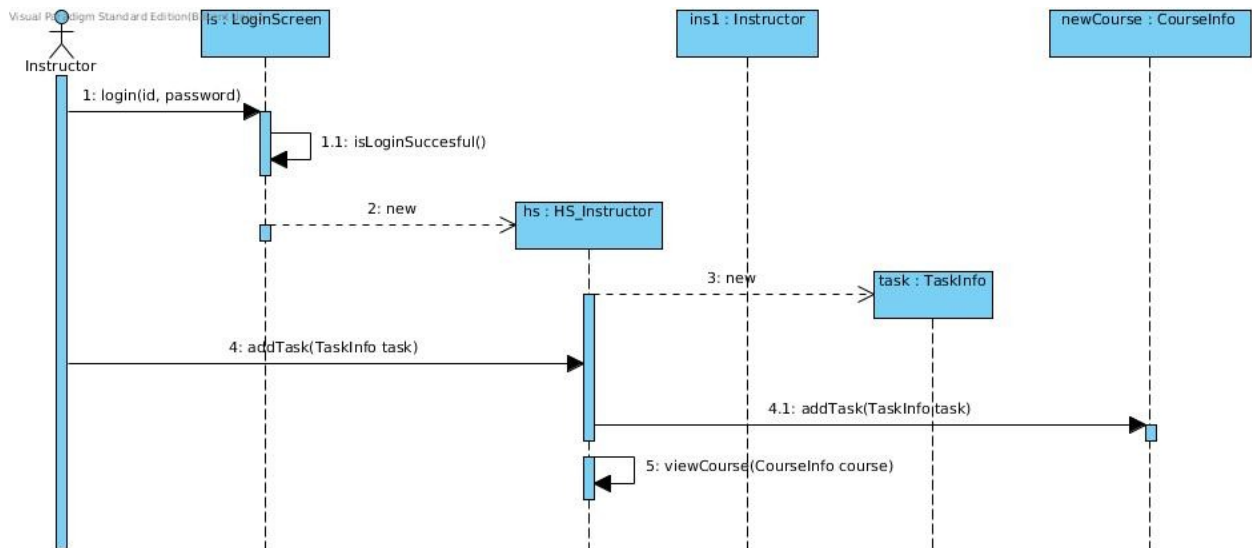
Once these processes are over client will provide user with the viewCourse screen as default.



### 3.2.2.b) Add Task

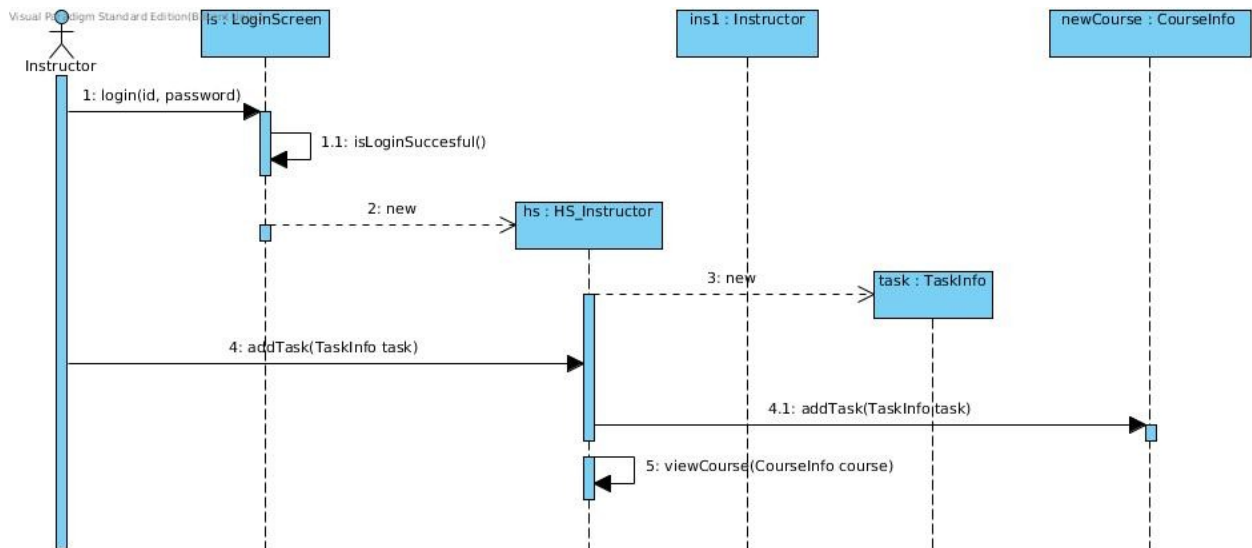
Instructors can add tasks (or homeworks) as part of a course. After the login process is completed and the appropriate homescreen is provided to instructor, the button for adding task will be visible to user. Following the click on the button there will be a new TaskInfo object constructed according to the specifications indicated by instructor. Then, this newly created task will be added to the appropriate CourseInfo instance.





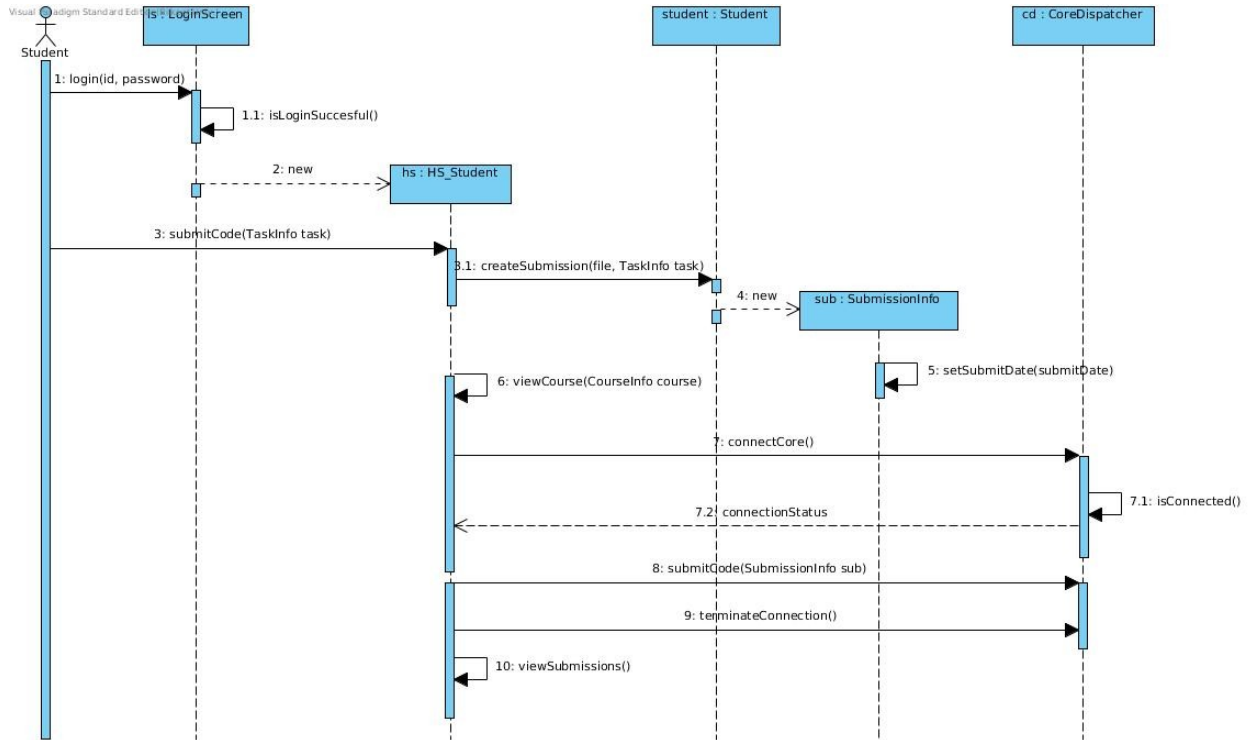
### 3.2.2.c) Edit Task

In some cases instructor may need to edit task info, for example the test inputs may had been initially wrong and instructor would want to change the wrong inputs with the correct ones, or the responsible TA might want to try his own test inputs. In order to do this user should login, and once the homescreen is ready he/she should select a particular task and click on Edit Task button. After the button is clicked there will be a new TaskInfo object constructed with respect to the attributes indicated by user, and then the copy constructor of the original TaskInfo object will take the new TaskInfo as it's parameter. Hence, the task will be edited.



### 3.2.2.c) Submit Code

In order to submit code to the system students should first login, and then click the submit code button. This will trigger the `createSubmission()` method of `Student` object and it will eventually create a `SubmissionInfo` object and deliver it to the core (evaluator) for analysis and grading.



## 4. Conclusion

In the analysis report, we have focused on what the system should do, what functions it should include, etc. However we have not designed a detailed system enough for implementation. Expected features of the system and given requirements were our main helper to form use case models and their scenarios.

Writing the analysis report was a useful process because it helped us to think about the design process thoroughly even before we started the implementation phase. In this report, our main target was to make people understand the analysis of our system and explain our user-friendly system for students and instructors. To achieve this goal, we project design tools such as Visual Paradigm, and used Unified Modeling Language (UML). Since UML is a unique programming language, it really helped us to communicate with other people easily. We used Visual Paradigm to create our UML diagrams such as Use Case diagrams, Class diagrams, Sequence diagrams, State and Activity Diagrams etc.

Finally, in the Analysis Report, we also discussed and eventually decided how user interface should be formed. We decided how system should respond to the user actions. We tried to provide understandable and informative user interface. We want our users to reach their goal with just a few steps. Consequently, we detailed our core and UI to implement them in object-oriented fashion.