Háskólinn í Reykjavík, 29. apríl 2016
Daníel Brandur Sigurgeirsson

Coder

Design report

Group 7
Andri Andrésson
Gunnar Torfi Steinarsson
Orri Arnarsson
Steinar Freyr Kristinsson
Þorgrímur Jónasarason

Table of Contents

Introduction

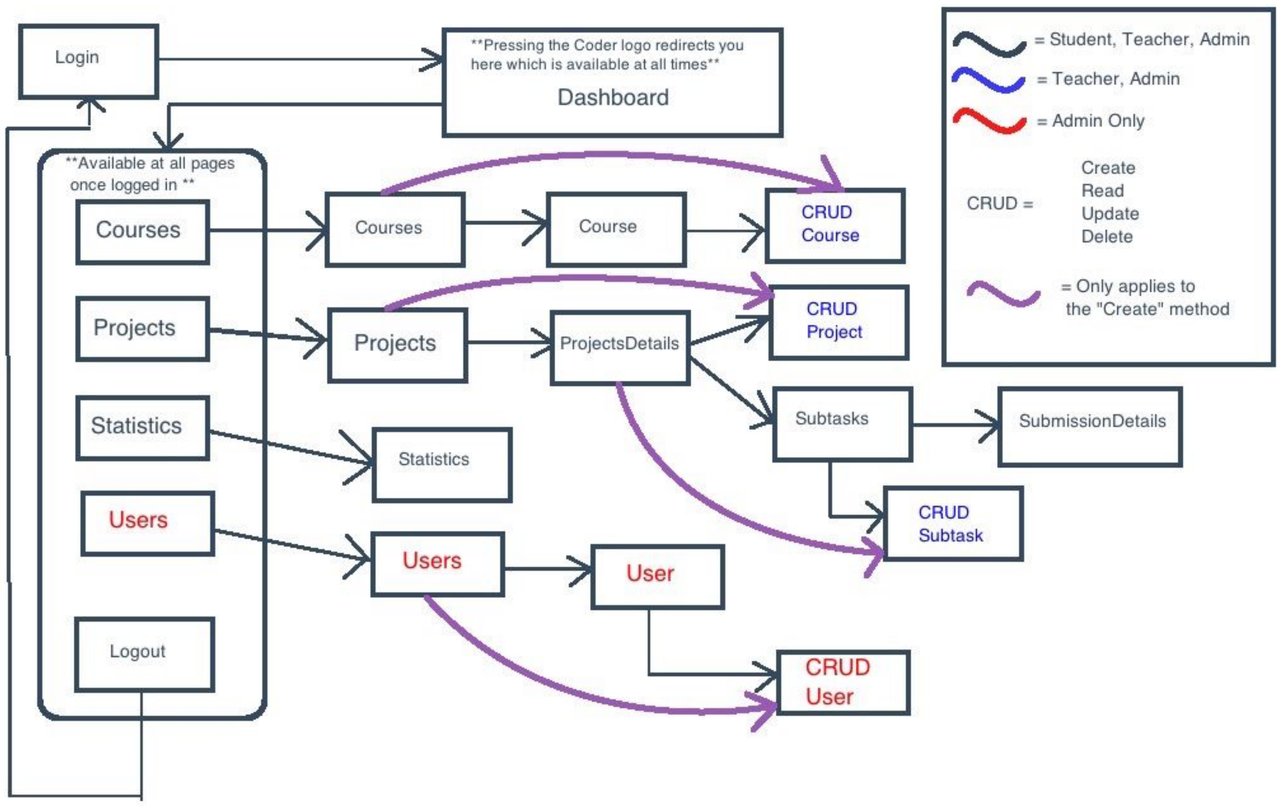
The Coder requirement report looked into “**what”** the system should do, the Coder design report will look into “**how”** it will be done. In this report we will look into navigation diagrams, page prototypes, class diagrams, coding conventions, sequence diagrams and state diagrams in Coder. These diagrams and prototypes are to help programmers with the implementation of the program.

Coder is a new web application tool which is supposed to be a better and more user friendly version of Mooshak, which is the current programming assignment checker. The general idea is to offer students studying programming instant feedback when solving programming problems posed by their teachers. The teacher defines an assignment with a given input and output and the students can submit their solutions and see clearly if their solution is solving the problem correctly.

Coder should be very easy to use and should display user and course statistics clearly. The main difference from the current Mooshak is that Coder will not be as contest minded as its precursor. Mooshak does a lot of things very well, but there is always room for improvement. The goal of Coder is to be the next generation of Mooshak.

Navigation Diagram

The navigation diagrams looks similar for all users (the colors on the diagram indicate the differences between user types). That way navigating the site will be easy, even when you are using the system with different login roles (e.g student and TA). The goal was to keep the navigation as simple as possible to ensure all users could learn to use it without much trouble. The diagram can be seen here below with explanations:


Login
Pressing the Coder logo redirects you here which is available at all times
Dashboard
**Available at all pages once logged in **
Courses
Projects
Statistics
Users
Logout
Courses
Course
CRUD Course
Projects
ProjectsDetails
CRUD Project
Subtasks
SubmissionDetails
CRUD Subtask
Statistics
Users
User
CRUD User
= Student, Teacher, Admin
= Teacher, Admin
= Admin Only
CRUD = Create Read Update Delete
= Only applies to the "Create" method

Page Prototypes

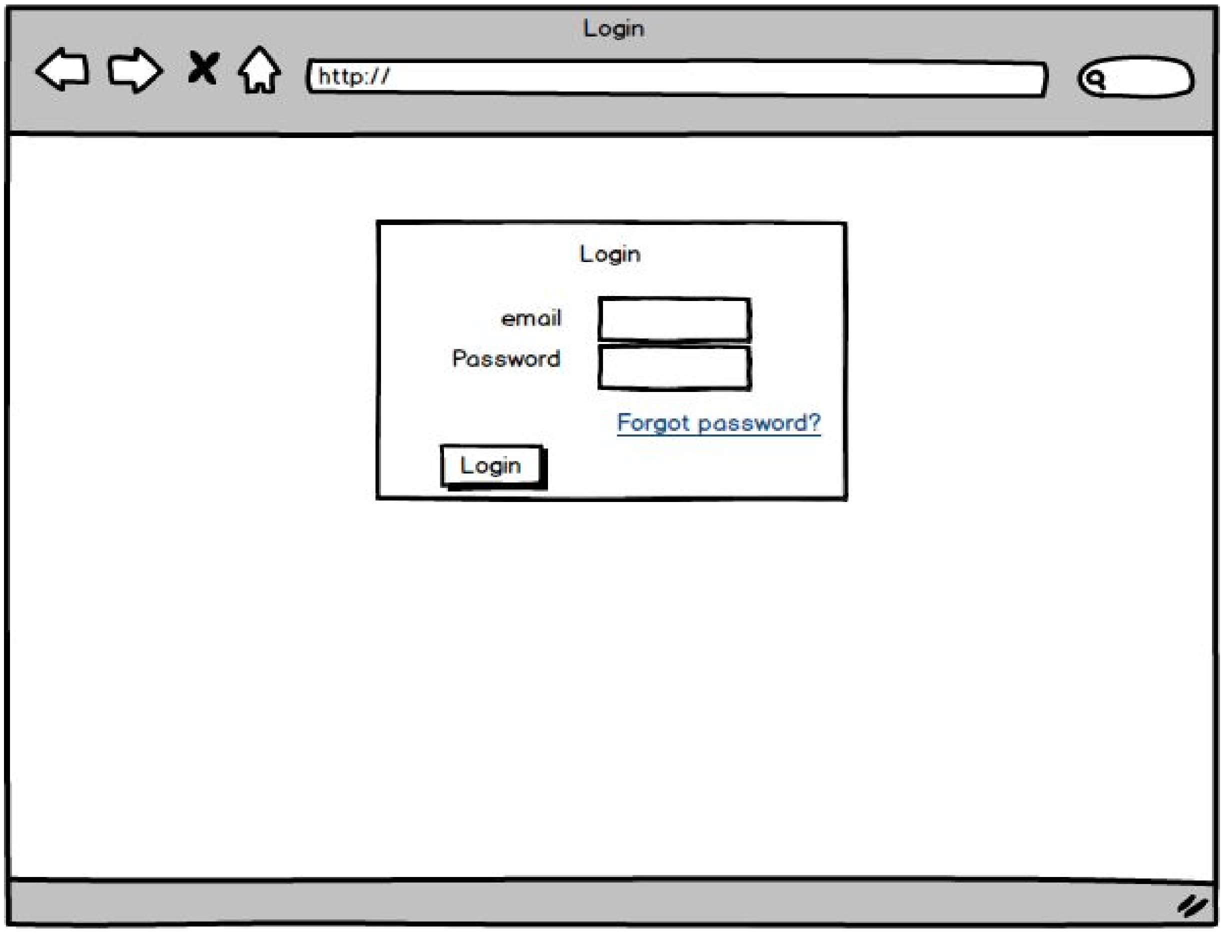
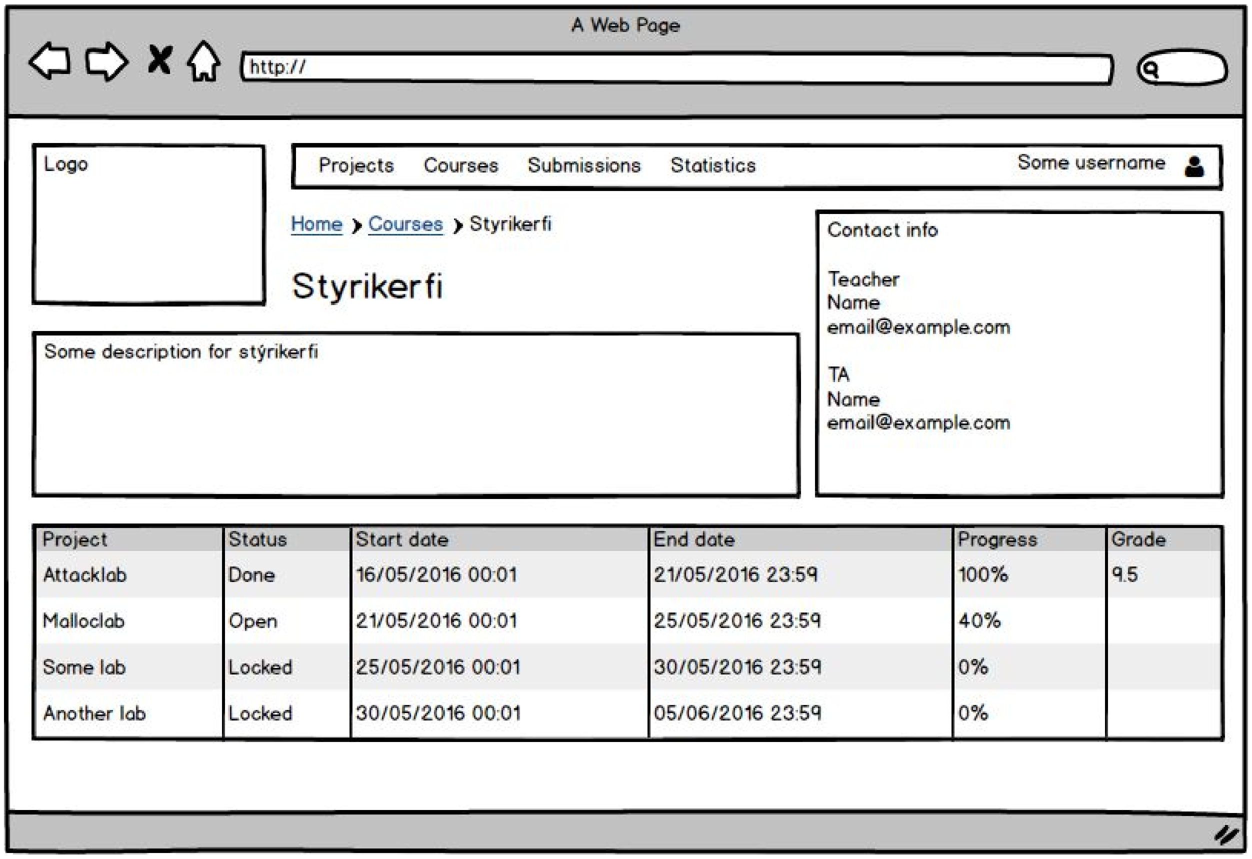


Image 1. Login
On this image we can see the login view. Here all the registered users (by admin) are able to login and/or navigate to a forgot password view.



Styrikerfi

Some description for stýrikerfi

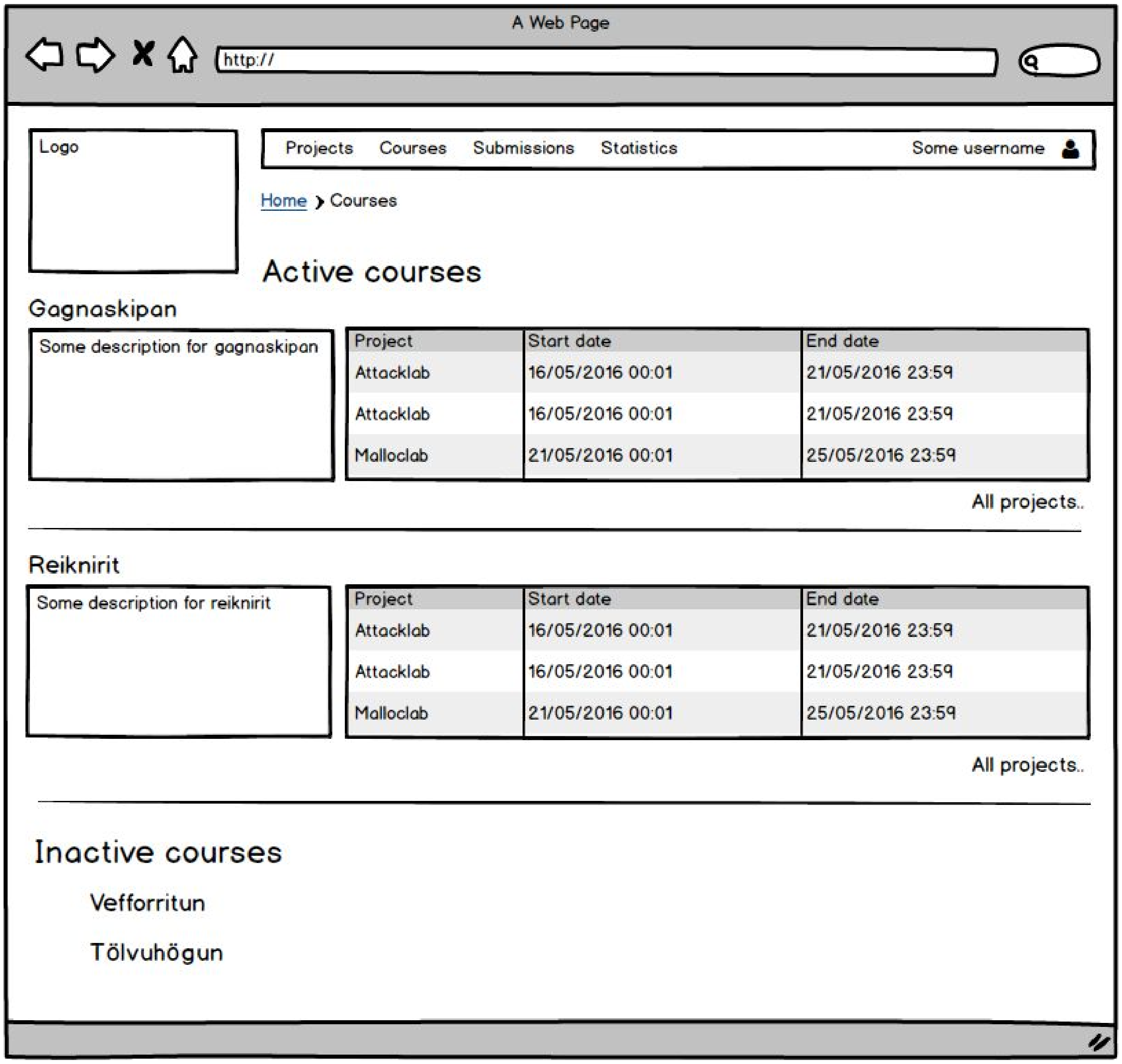
Contact info

Teacher
Name
email@example.com

TA
Name
email@example.com

Project	Status	Start date	End date	Progress	Grade
Attacklab	Done	16/05/2016 00:01	21/05/2016 23:59	100%	9.5
Malloclab	Open	21/05/2016 00:01	25/05/2016 23:59	40%	
Some lab	Locked	25/05/2016 00:01	30/05/2016 23:59	0%	
Another lab	Locked	30/05/2016 00:01	05/06/2016 23:59	0%	

Image 2. Course details
This view displays details about a chosen course. Users (who are registered in the course) can also see all projects linked to this course and navigate to any given project.



A Web Page

http://

Logo

Projects Courses Submissions Statistics Some username

Home > Courses

Active courses

Gagnaskipan

Some description for gagnaskipan

Project	Start date	End date
Attacklab	16/05/2016 00:01	21/05/2016 23:59
Attacklab	16/05/2016 00:01	21/05/2016 23:59
Malloclab	21/05/2016 00:01	25/05/2016 23:59

All projects..

Reiknirit

Some description for reiknirit

Project	Start date	End date
Attacklab	16/05/2016 00:01	21/05/2016 23:59
Attacklab	16/05/2016 00:01	21/05/2016 23:59
Malloclab	21/05/2016 00:01	25/05/2016 23:59

All projects..

Inactive courses

Vefforritun

Tölvuhögun

Image 3. Active courses

This view displays all courses where the user is currently enrolled in. The user is also able to see and navigate to any given project displayed under given course.

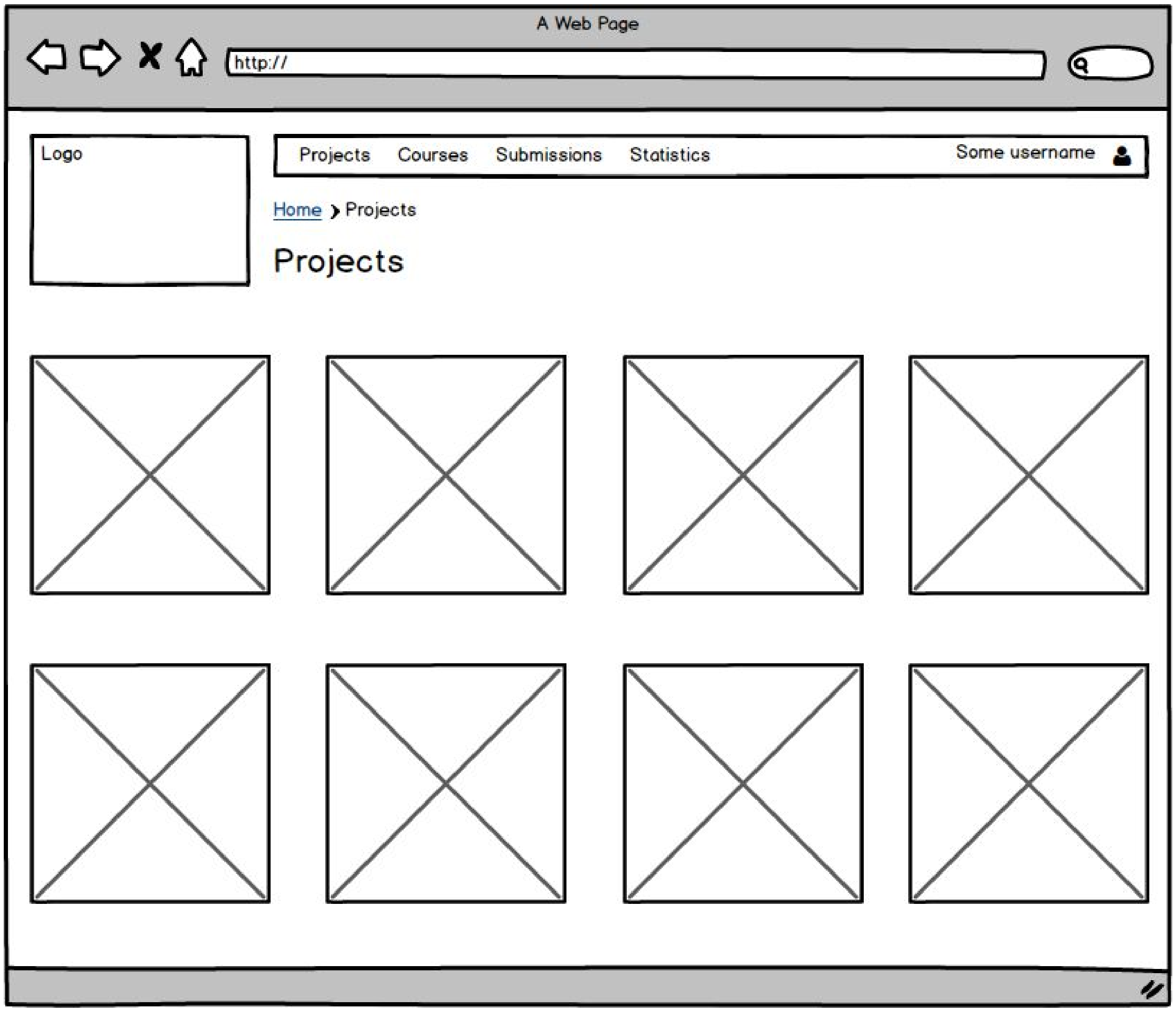
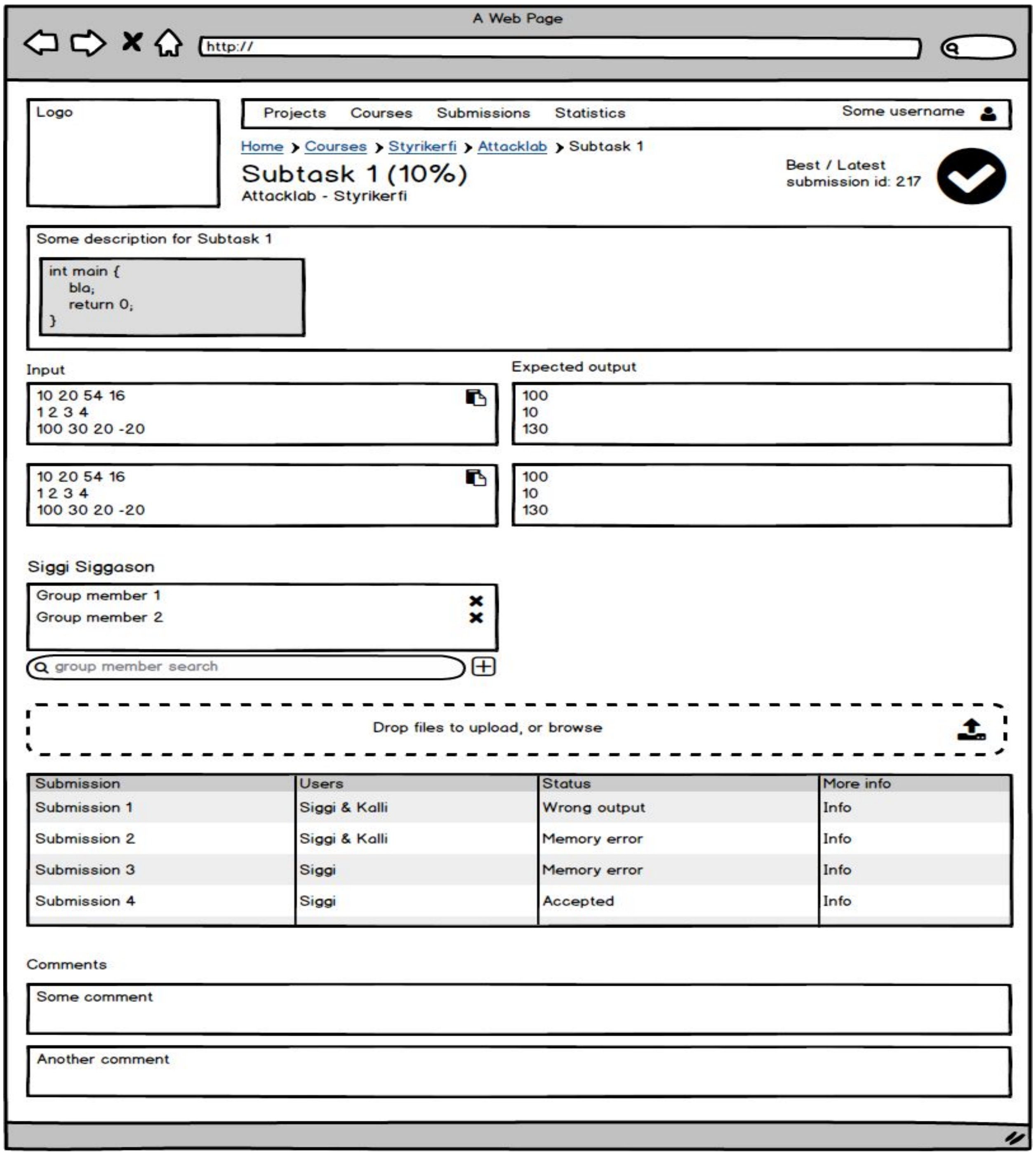


Image 4. Student dashboard
This view displays student's upcoming and currently active projects.



Subtask 1 (10%)

Attacklab - Styrikerfi

Some description for Subtask 1

```
int main {
    bla;
    return 0;
}
```

Input:

```
10 20 54 16
1 2 3 4
100 30 20 -20
```

Expected output:

```
100
10
130
```

Input:

```
10 20 54 16
1 2 3 4
100 30 20 -20
```

Expected output:

```
100
10
130
```

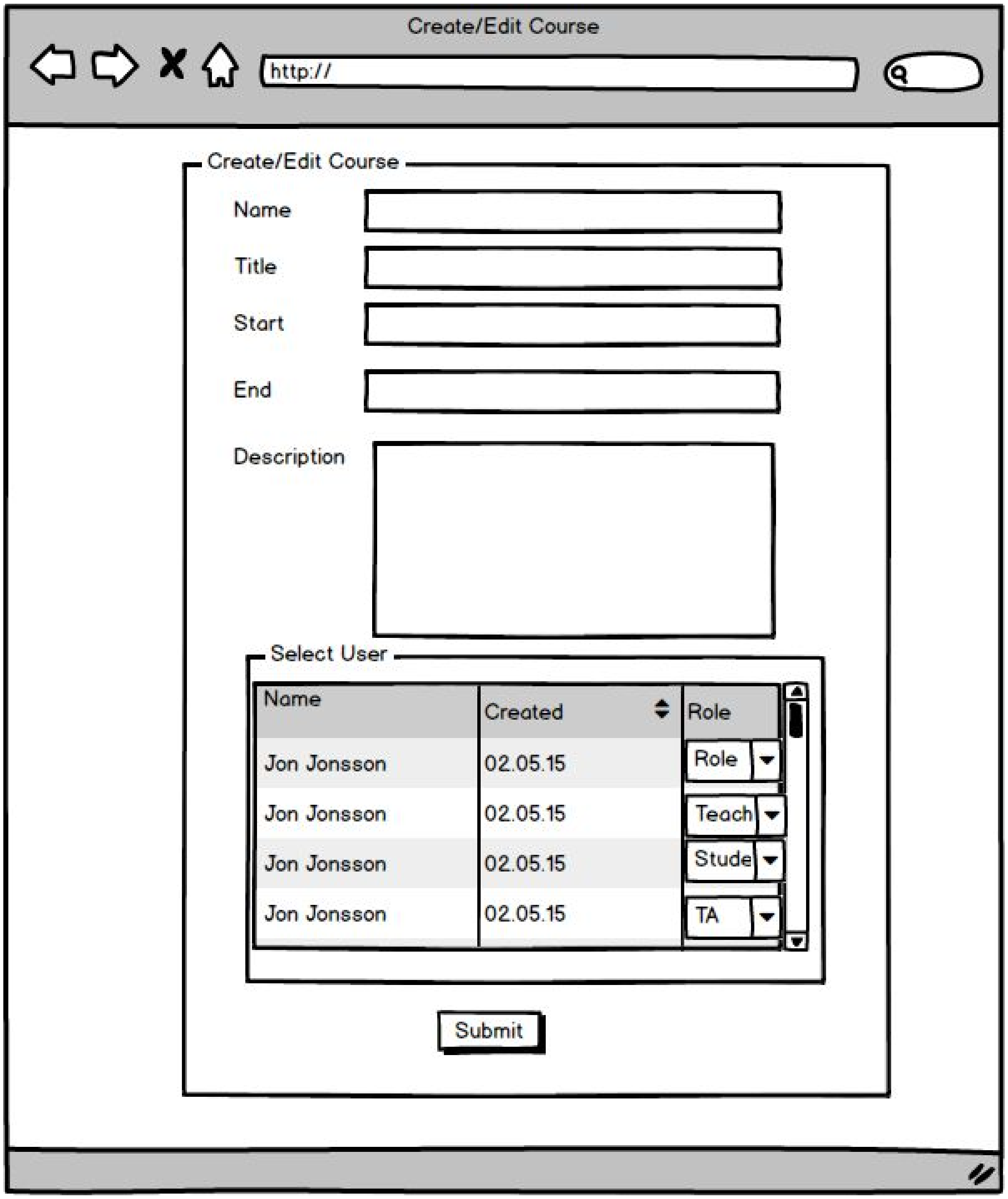
Submission	Users	Status	More info
Submission 1	Siggi & Kalli	Wrong output	Info
Submission 2	Siggi & Kalli	Memory error	Info
Submission 3	Siggi	Memory error	Info
Submission 4	Siggi	Accepted	Info

Some comment

Another comment

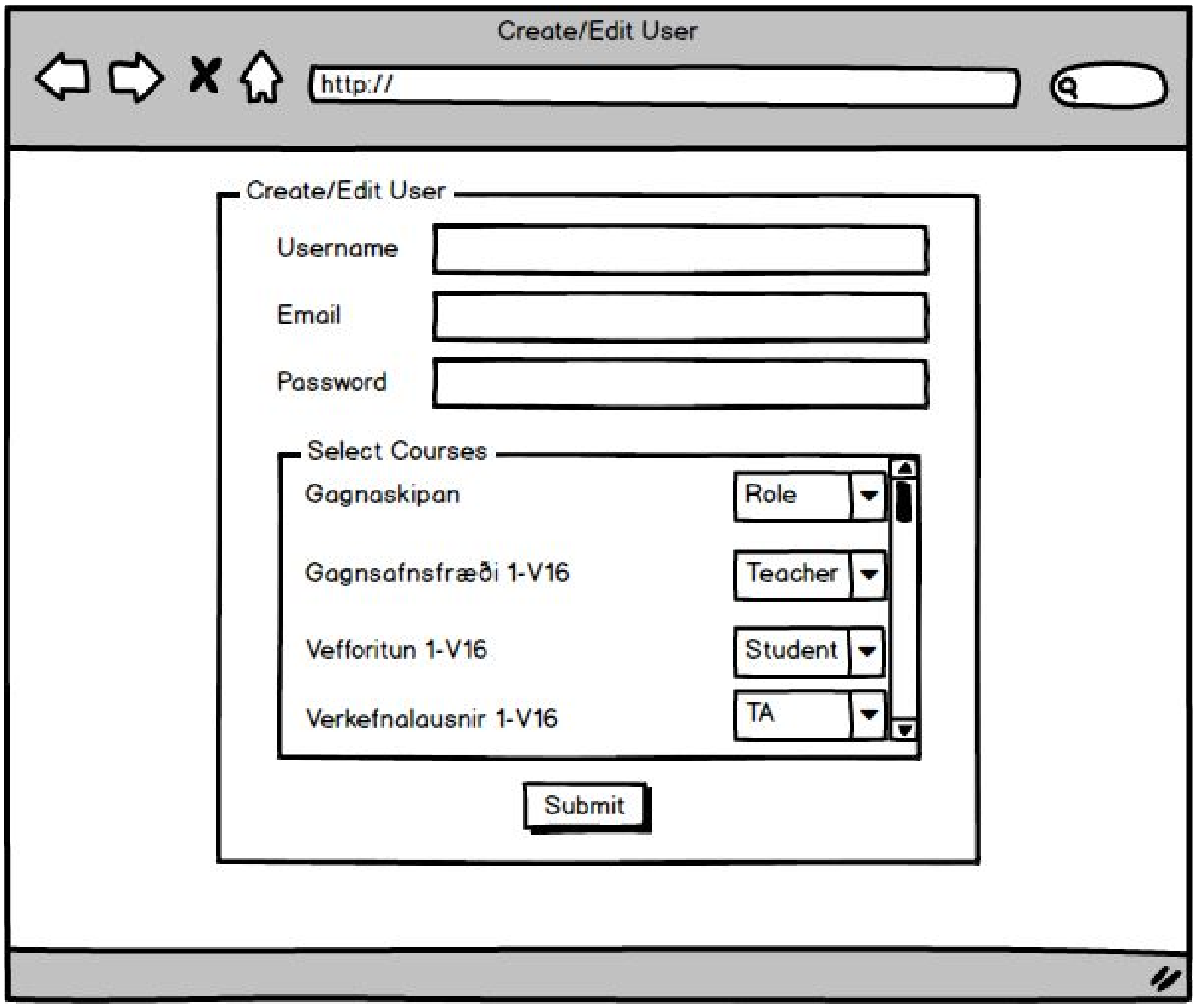
Image 5. Subtasks

This view displays detailed subtask. Students are able to submit a solution for the subtask and view previous submissions. Student can submit alone or as a group. Any user linked to the subtask is able to leave a comment.



Name	Created	Role
Jon Jonsson	02.05.15	Role
Jon Jonsson	02.05.15	Teach
Jon Jonsson	02.05.15	Stude
Jon Jonsson	02.05.15	TA

Image 6. Create/edit course.
This view displays how a course is created and/or edited. This view belongs to the teacher and also admin. Above name, dates and description can be edited and below students/teachers can be linked to the course.



Gagnaskipan	Role
Gagnsafnsfræði 1-V16	Teacher
Vefforitun 1-V16	Student
Verkefnalausnir 1-V16	TA

Image 7. Create/edit user
This view displays how a user is created and edited. This view only belongs to the admin. Above name, email and password can be edited and below the user gets his role, which can differ between courses.

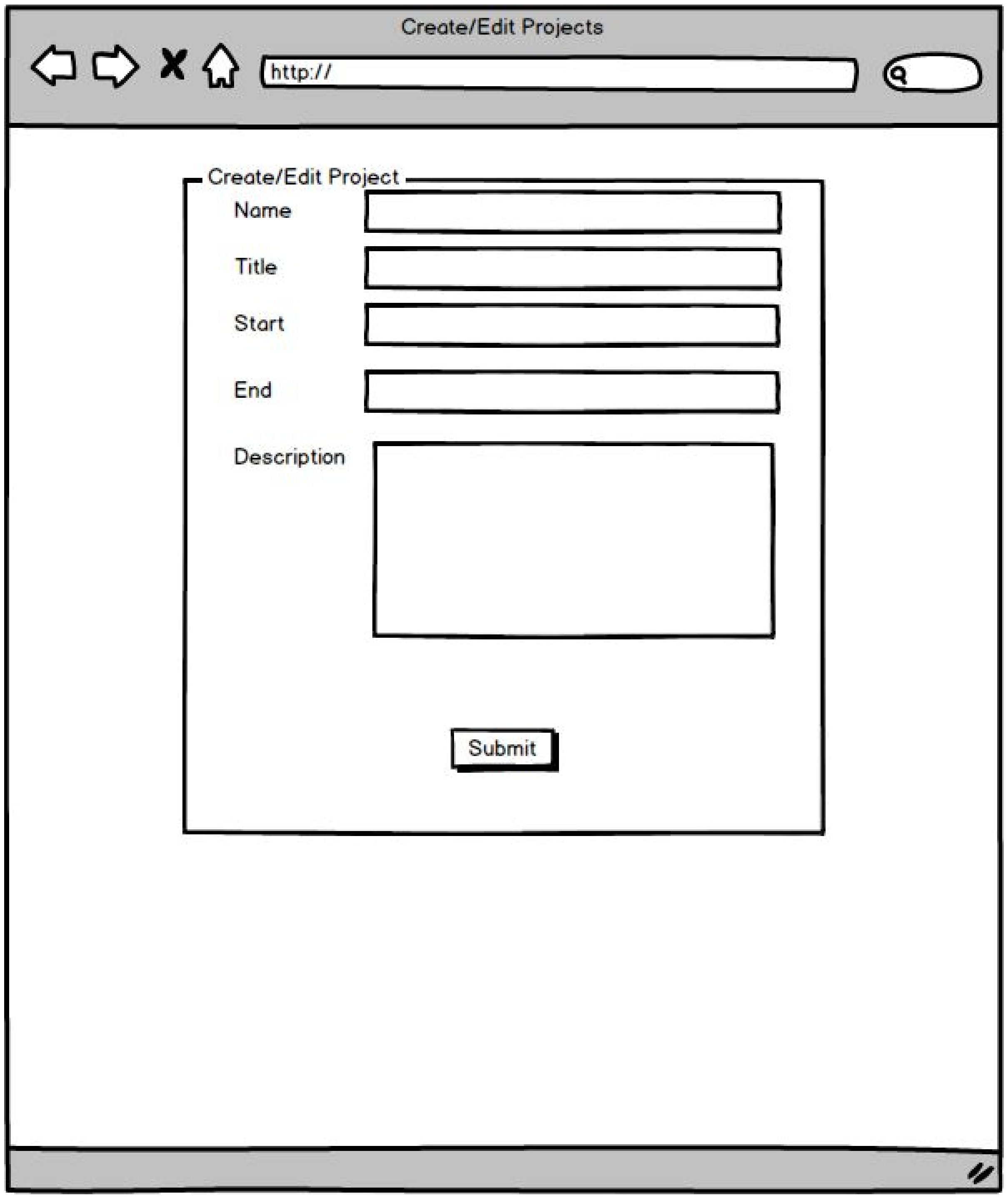


Image 8.Create/edit projects.
This view displays how projects are created and edited within a course. This view belongs to the teachers and admins. This is only the project itself, image 9 shows how subtasks are created.

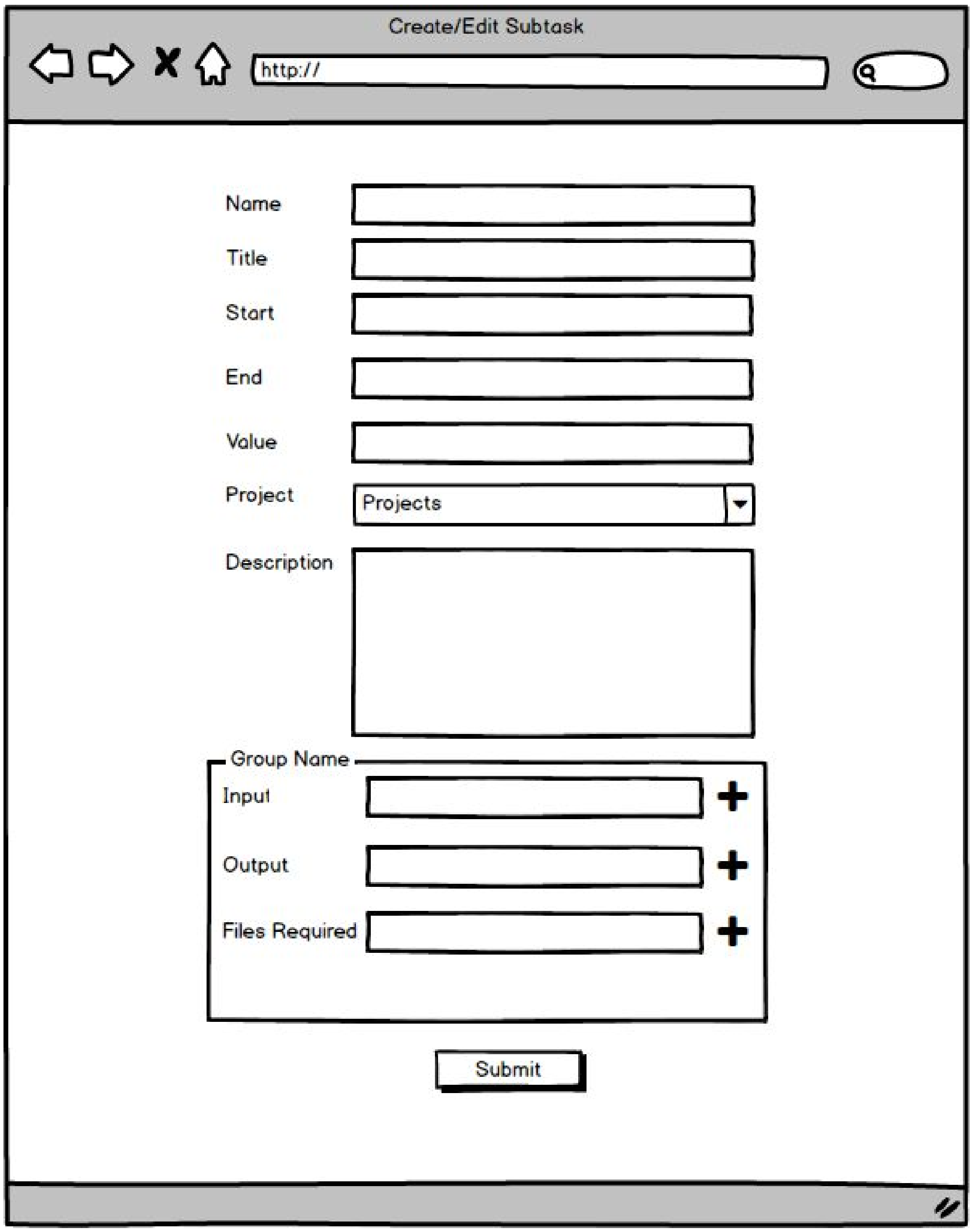


Image 9. Create/edit subtask
This view displays how subtasks are created and edited within a course. This view belongs to the teachers and admins. Here the teacher gives the subtask a name, title, date, description, value and relates it to a project. At the bottom the teacher also creates the desired input, output and how many files are required (including the filename/s).

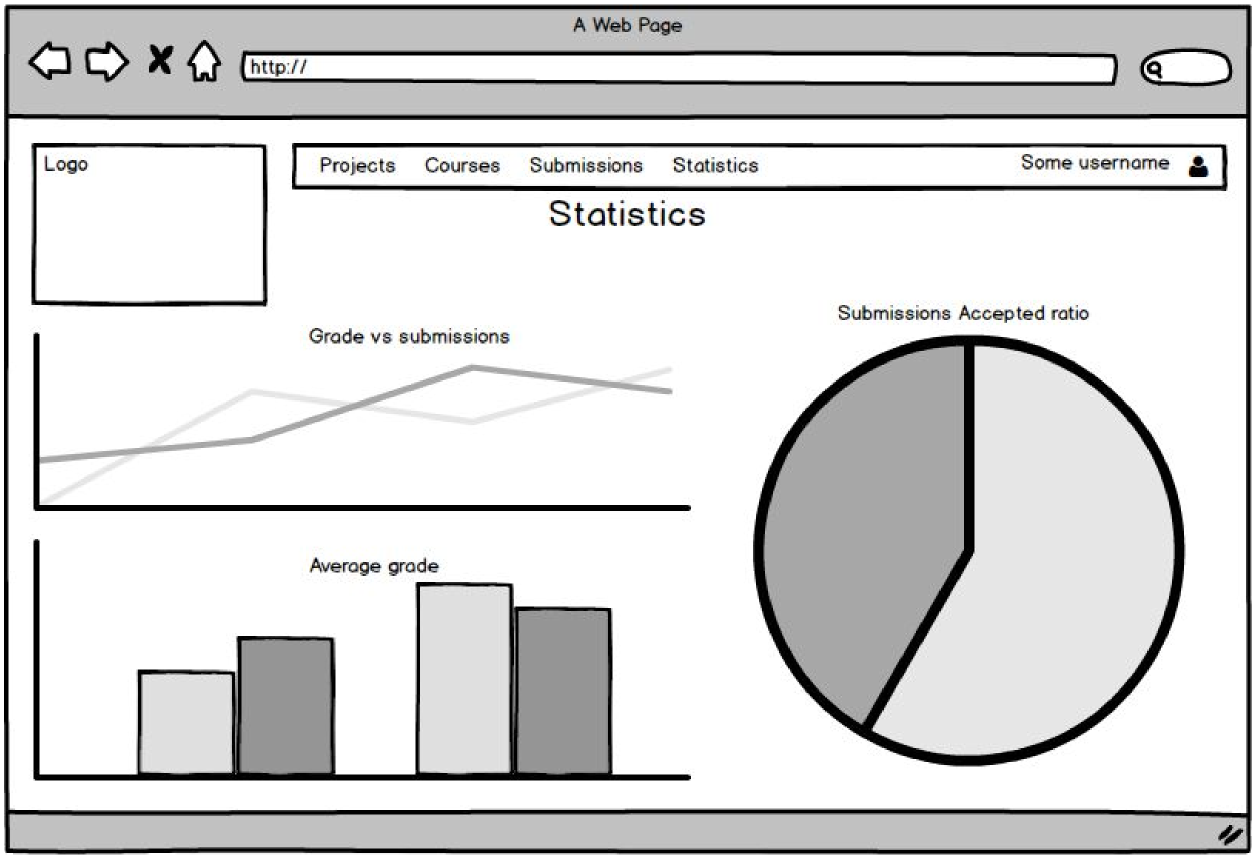
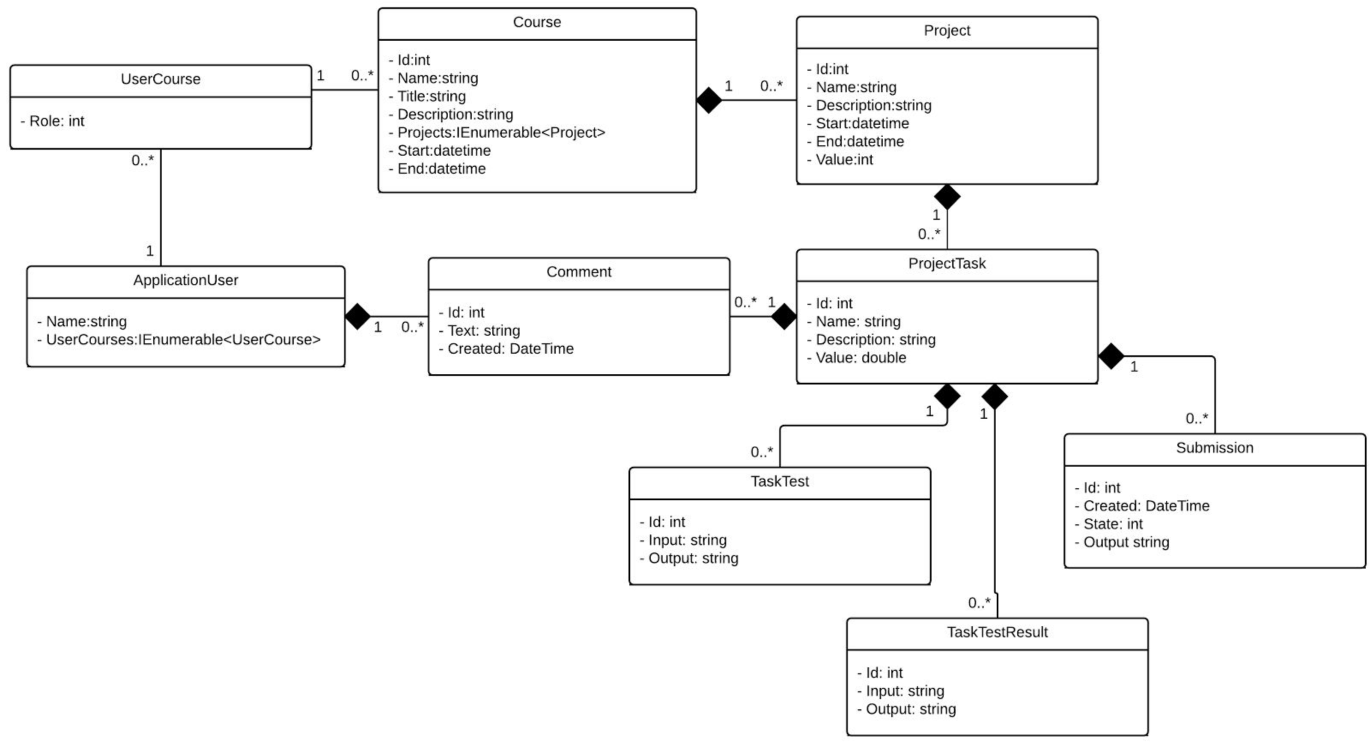


Image 10. Statistics.
This view displays somewhat useful statistics, teacher can see overall statistics about students in his course, student can see statistics about his projects and admin can see statistics about, active students/teachers/TA's.

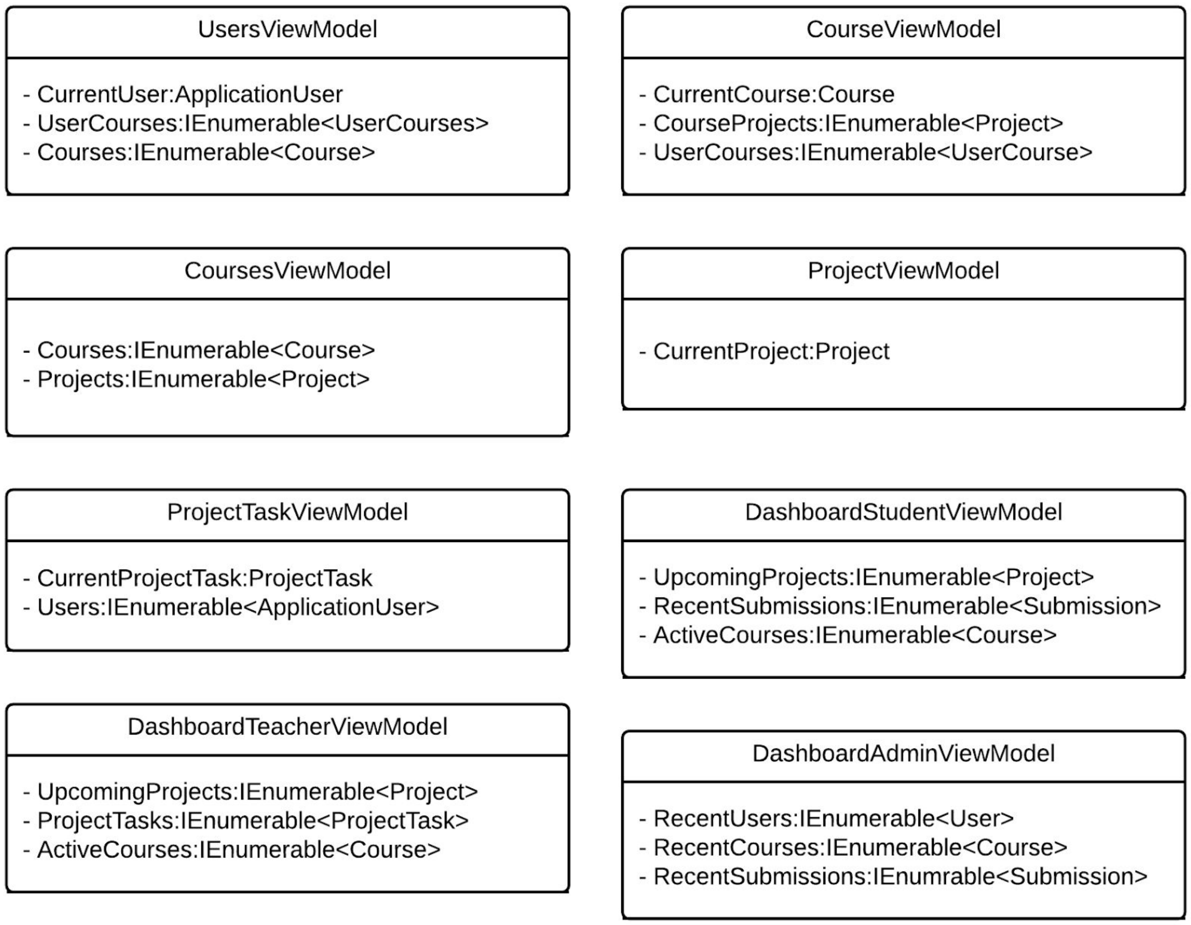
Class Diagrams

In this report, there is a diagram for entity models that shows how each entity class is related to other classes. There is also a diagram for view models, controllers and repositories. Table schema is not in this report, since it would show the same information that the entity model diagram shows.

Entity models



View models



UsersViewModel

- CurrentUser:ApplicationUser
- UserCourses:IEnumerable<UserCourses>
- Courses:IEnumerable<Course>

CourseViewModel

- CurrentCourse:Course
- CourseProjects:IEnumerable<Project>
- UserCourses:IEnumerable<UserCourse>

CoursesViewModel

- Courses:IEnumerable<Course>
- Projects:IEnumerable<Project>

ProjectViewModel

- CurrentProject:Project

ProjectTaskViewModel

- CurrentProjectTask:ProjectTask
- Users:IEnumerable<ApplicationUser>

DashboardStudentViewModel

- UpcomingProjects:IEnumerable<Project>
- RecentSubmissions:IEnumerable<Submission>
- ActiveCourses:IEnumerable<Course>

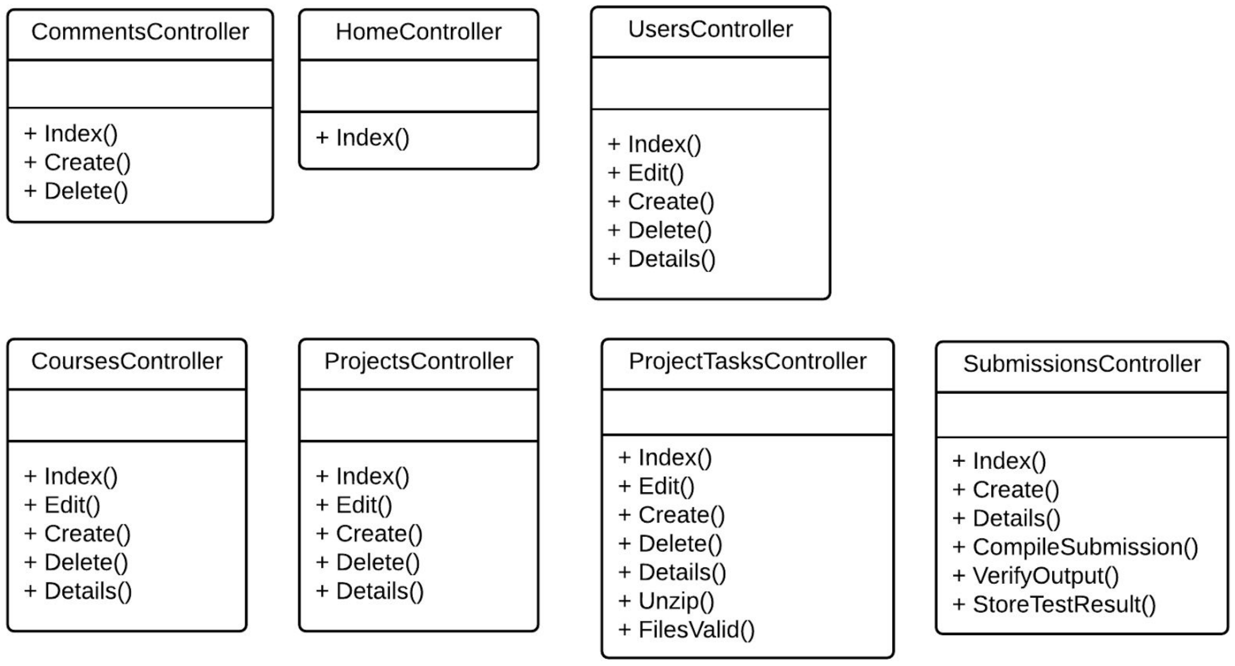
DashboardTeacherViewModel

- UpcomingProjects:IEnumerable<Project>
- ProjectTasks:IEnumerable<ProjectTask>
- ActiveCourses:IEnumerable<Course>

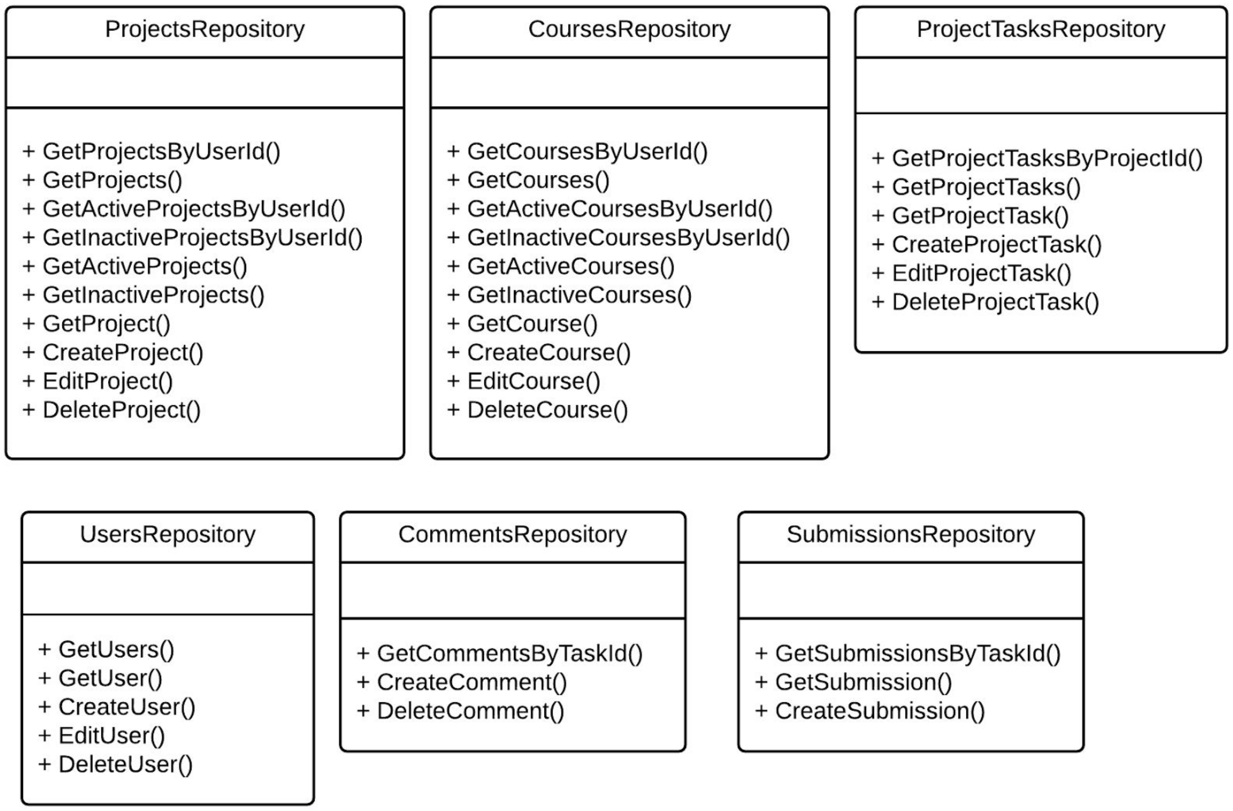
DashboardAdminViewModel

- RecentUsers:IEnumerable<User>
- RecentCourses:IEnumerable<Course>
- RecentSubmissions:IEnumrable<Submission>

Controllers



Repositories



Coding Conventions

Here are the programming rules and guidelines for the system. Having these rules will improve the readability of the program and make maintenance easier.

We will be using the coding rules C# implies, i.e. PascalCasing for properties, functions, structs, classes and namespaces. Variable names will use camelCasing.

We will be using the Stylecop extension to ensure we'll be following these rules.

Curly braces

We will be using the *Allman* standard for our curly braces which looks like this.

```
while (x == y)
{
    something();
    somethingelse();
}
```

Indentions

We'll be using tabs that consists of four spaces for all indentations.

Comments

Code comments will placed on a separate line, not at the end of a line of code.
Comments will be placed below documentation comments, if applicable.

Long lines of code

Very long lines of code shall be broken up into small, readable blocks of text. Let's take a look at a complex query and how it can be made more readable with few new lines.

Example:

```
var projects = (from p in db.Projects join c in db.Courses on p.CourseId
equals c.Id where c.UserCourses.Any(u => u.UserId == userId) select p);
```

Should look like:

```
var projects = (from p in db.Projects
                join c in db.Courses on p.CourseId equals c.Id
                where c.UserCourses.Any(u => u.UserId == userId)
                select p);
```

Initializing variables
Try to initialize local variables as soon as they are declared.

Inter-term spacing
There should be a single space after a comma or a semicolon.
Example:

```
myFunction(a, b, c);
```

Not:

```
myFunction(a,b,c) or myFunction( a, b, c );
```

White Space
Blank lines improve readability. They set off blocks of code which are in themselves logically related. One blank line should always be used between methods.

Number of Declarations per Line
Do not put more than one variable declaration on the same line. Always declare variables in a separate line

Example:

```
int level;
int size;
```

Instead of:

```
int level, size;
```

CSS/SCSS Guidelines

These CSS guidelines are taken from CodeGuide.co written by Mark Otto. We will use them as a guide when writing the CSS styles for the program. We believe they will help make the styles more maintainable and easier to read, but we will not enforce them.

Syntax

- Indent with soft tabs with 4 spaces.
- When grouping selectors, keep individual selectors to a single line.
- Include one space before the opening brace of declaration blocks for legibility.
- Place closing braces of declaration blocks on a new line.

- Include one space after : for each declaration.
- End all declarations with a semicolon.
- Comma-separated property values should include a space after each comma (e.g., box-shadow).
- Don't prefix property values or color parameters with a leading zero (e.g., .5 instead of 0.5 and -.5px instead of -0.5px).
- Lowercase all hex values, e.g., #fff.
- Use shorthand hex values where available, e.g., #fff instead of #ffffff.
- Avoid specifying units for zero values, e.g., margin: 0; instead of margin: 0px;.

Declaration order

Declaration order

Related property declarations should be grouped together following the order:

1. Positioning

```
position: absolute;
top: 0;
right: 0;
bottom: 0;
left: 0;
z-index: 10;
```

2. Box model

```
display: block;
float: right;
width: 100px;
height: 100px;
padding: 10px;
margin: 10px;
```

3. Typographic

```
font: normal 13px "Helvetica Neue", sans-serif;
line-height: 1.5;
color: #333;
text-align: center;
```

4. Visual

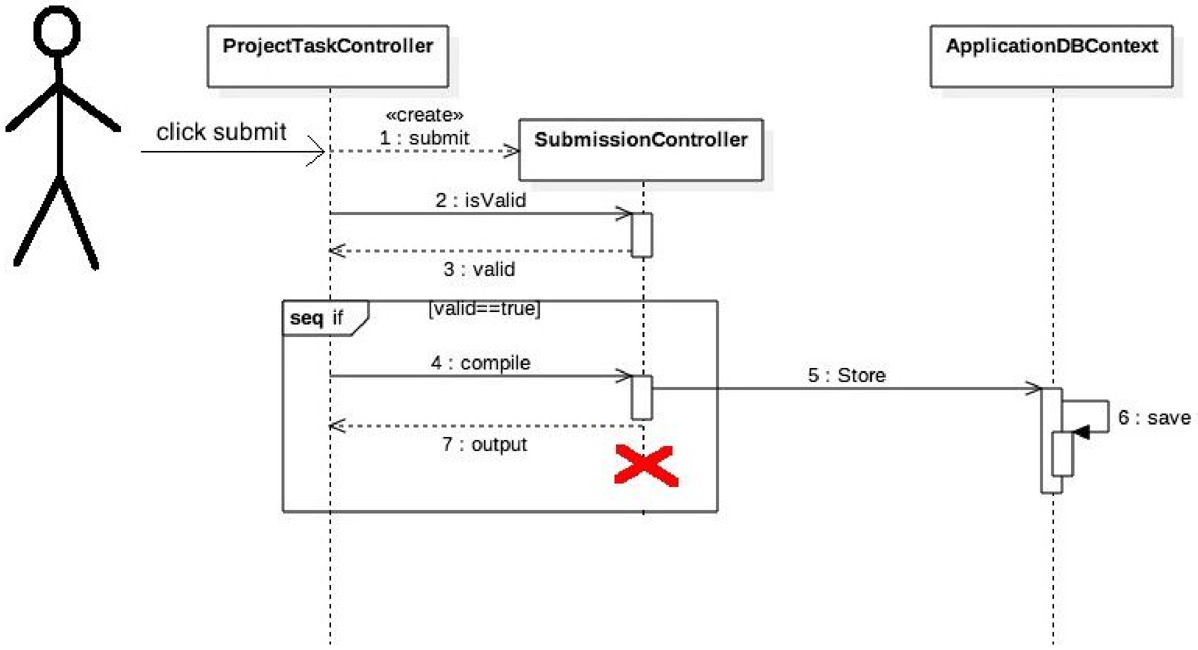
```
background-color: #f5f5f5;
border: 1px solid #e5e5e5;
border-radius: 3px;
```

5. Other
Positioning comes first because it can remove an element from the normal flow of the document and override box model related styles. The box model comes next as it dictates a component's dimensions and placement.

Everything else takes place inside the component or without impacting the previous two sections, and thus they come last.

Sequence Diagrams

This sequence diagram shows the interaction between the classes when submitting a project. The Student has uploaded a file and clicks submit from the **ProjectTaskController**. An instance of **SubmissionController** is created. The submission is validated, if it’s valid the submission is compiled, stored in **ApplicatinoDBContext** (which stores it in the database) and an output is returned.

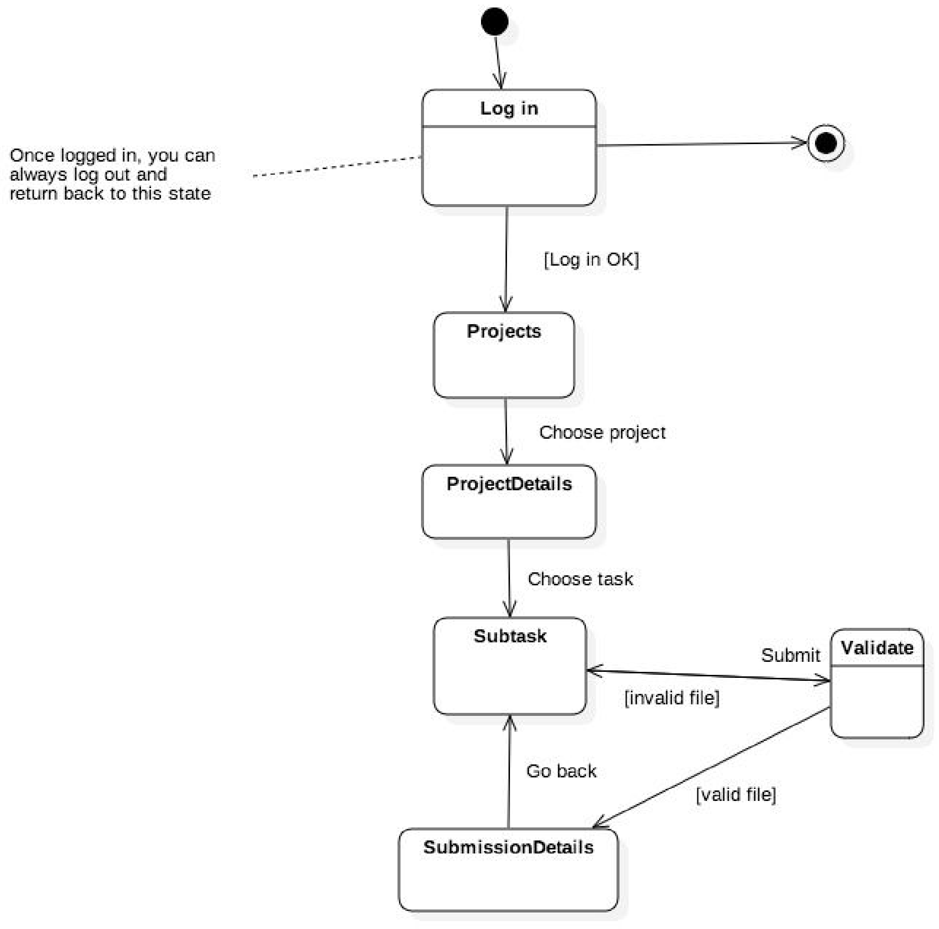


State Diagrams

The following two state diagrams show the behaviour of the system when responding to two common actions. The diagrams show the basic flow of these two actions. Logout is possible at all times once logged in.

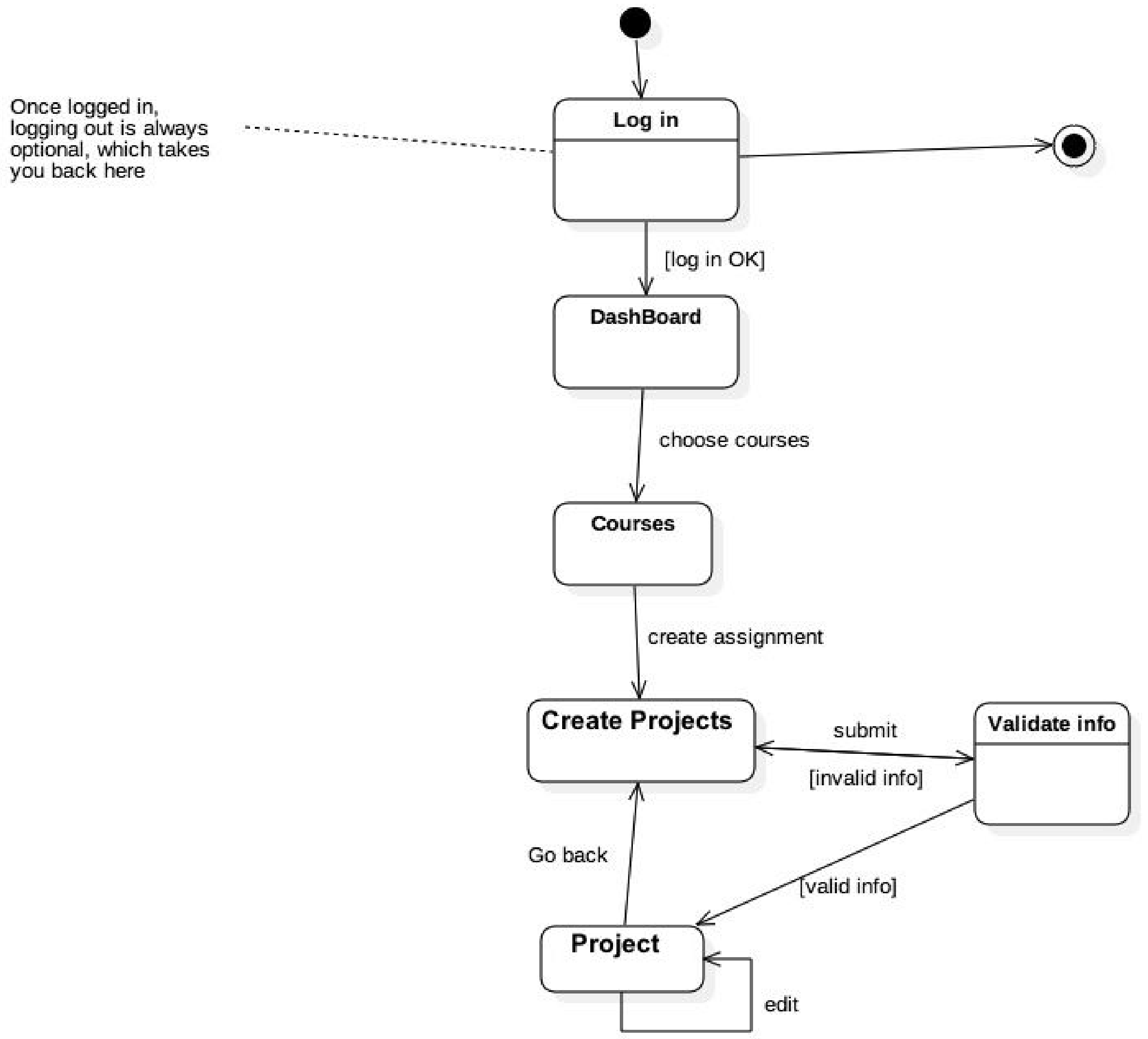
Submission

The former diagram shows the process of submitting a file as a student. The student needs to log in at first. Once logged in he must choose the correct project and then choose the correct subtask. From there the student must upload his file. If the file is valid the student is directed to his submission details where he gets his feedback, otherwise he will get a message ("Your file is not valid"). The process after the student clicks submit can be seen in the sequence diagram above.



Creation of an Assignment

The second diagram shows the process of creating a project as a teacher. The teacher needs to login first. Once logged in the teacher needs to choose a course. From there he can create a project. If the information is valid a new project will be created, otherwise he will get a message ("Information is not valid"). The assignment the teacher has created can now be edited, deleted and viewed with more details.



Final words and conclusion

The purpose of this report is to visualize the design of the Coder system or in other words discuss “**how”** the system part is implemented. It was done by making diagrams, prototypes and a guidelines for the code format used. After experimenting with many variations of navigation, look and function we have come up with a simple, clean, user friendly solution.

The prototypes will be used as a reference while developing the system and views. As are the objects in the diagrams. The next step will be the the biggest one yet, the implementation of the system. This report will be used as a guide throughout the whole development process, as well as the requirement analysis report. After writing this report, the team feels that they are more in control of the tasks at hand.

Appendix

We used the following resources in the making of this report, in the Coding Conventions section:

- https://stylecop.codeplex.com/
- https://en.wikipedia.org/wiki/Indent_style
- http://codeguide.co/

Abbreviations

TA = Teacher assistant
CRUD = Create Read Update Delete