
**NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS**

**SCHOOL OF SCIENCE**
**DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS**


**BSC THESIS**

# Developing an Electronic Classroom Platform with Ruby on Rails


**Ioannis A. Efthymiou**

**Supervisors:** **Alex Delis,** Professor NKUA
**Panagiotis Liakos,** PhD Candidate



**ATHENS**

**MARCH 2016**

**ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ**

**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ**
**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

# Αναπτύσσοντας μια Πλατφόρμα Ηλεκτρονικής Τάξης με Ruby on Rails

**Ιωάννης Α. Ευθυμίου**

**Επιβλέποντες:** **Αλέξης Δελής,** Καθηγητής ΕΚΠΑ
**Παναγιώτης Λιακός,** Υποψήφιος Διδάκτορας

**ΑΘΗΝΑ**

**ΜΑΡΤΙΟΣ 2016**

**BSC THESIS**

# Developing an Electronic Classroom Platform with Ruby on Rails

**Ioannis A. Efthymiou**
**A.M.:** 1115200600296

**SUPERVISORS:** **Alex Delis,** Professor NKUA
**Panagiotis Liakos,** PhD Candidate

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

# Αναπτύσσοντας μια Πλατφόρμα Ηλεκτρονικής Τάξης με Ruby on Rails

**Ιωάννης Α. Ευθυμίου**
**Α.Μ.:** 1115200600296

**ΕΠΙΒΛΕΠΟΝΤΕΣ:** **Αλέξης Δελής,** Καθηγητής ΕΚΠΑ
**Παναγιώτης Λιακός,** Υποψήφιος Διδάκτορας

# ABSTRACT

Objective of my thesis is to develop an e-class platform, which is a web application aiming to provide assistance to both faculty and students in managing the courses throughout each academic year. Maximizing functionality and user experience were high-valued design goals. As a result, the application feels simple, yet elegant to the end user.

**SUBJECT AREA:** Web Development

**KEYWORDS:** ruby on rails, web application, e-class, ajax, javascript

# ΠΕΡΙΛΗΨΗ

Σκοπός της πτυχιακής μου εργασίας είναι η ανάπτυξη μιας πλατφόρμας ηλεκτρονικής τάξης (e-class), το οποίο είναι μια εφαρμογή διαδικτύου. Ο στόχος της εφαρμογής είναι βοηθήσει τους φοιτητές και το διδακτικό προσωπικό κατά την διάρκεια της ακαδημαϊκής χρονιάς. Η καλύτερη δυνατή λειτουργικότητα και η εμπειρία χρήσης ήταν από τους σημαντικότερους στόχους κατά την σχεδίαση. Ως αποτέλεσμα, η εφαρμογή είναι απλή, αλλά εξαιρετικά λειτουργική για τον χρήστη.

**ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ:** Ανάπτυξη Εφαρμογής Διαδικτύου

**ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ:** ruby on rails, εφαρμόγή διαδικτύου, ηλεκρονική τάξη, ajax, javascript

*Dedicated to my Lily, my friends and family.*

# ACKNOWLEDGEMENTS

I would like to thank my Lily, for everything.

My brother Makis, for always being next to me.

My mother Lalila, for raising me to be the man I am.

My father Antonis, for inspiring me to do what I love.

My brother Theseas, for reminding me what it is to be young and set high goals.

The rest of my family, for sharing my life and providing a loving family.

All of my friends, each of them unique and special in my life.

All the teachers and professors, from kindergarden through university, for their role in shaping me to what I am.

Last but not least PhD candidate Panagiotis Liakos, for his amazing assistance in this thesis, and Professor Alexis Delis for inspiring me to follow this path.

# TABLE OF CONTENTS

# LIST OF FIGURES

# PROLOGUE

The following was written in Athens, Greece, in March 2016. It documents the development of a web application using RoR. It is distributed in hope of assisting others interested in getting started with developing web applications using RoR. However, there are *no guarantees* that the following will always be applicable, due to the everchanging nature of web technologies. Consulting with the official documentation is always recommended.

# 1. INTRODUCTION

Ever since the World Wide Web was made commercially available in the 1990s, it has experienced an unprecedented growth in popularity, as well as itself as an industry. Today, the majority of information we receive is online. Whether it is the news, communicating with other people, or just using the social media, we spend a lot of time online.

It is a logical next step, to take advantage of this power and use it to further assist our education. An electronic class platform (e-class) is a useful tool that makes keeping track of the courses you attend easier.

As an administrator, you can have an overview of the platform, create new courses and assign faculty members to be in charge of them.

Professors have a place where they can inform students about each of their courses, provide updates for those who are attending, give out assignments and receive homework by the students.

Students have the opportunity to find out about all the courses available, keep track of courses they attend and effortlessly enroll and withdraw from them.

## 1.1 Challenges in Design

The design and development of such a platform offers some very interesting challenges. Design challenges include, but are not limited to, the following.

**Consistency:** Each page has to be personalised according to the user viewing it, while maintaining a consistent layout.

**Intelligibility:** A website should be intuitive to use. The users, regardless of their experience or roles in the platform, should be able to easily navigate through the site and accomplish their goals.

**Usefulness:** The application has to provide attractive alternatives to the status quo of course management. For example, posting an assignment online is more efficient than printing it out and handing it to students.

## 1.2 Challenges in Development

Once the hurdles of designing the application are overcome, the problems of developing it arise. Again, here is an example of such challenges, but not a full list of them.

**Proper Modeling:** Creating a proper modelization for the design, that conforms with the Model-View-Controller(MVC) pattern is vital for web applications, and ensures proper modularity.

**Coding Efficiency:** A web application like an e-class offers web pages with similar, but different functionality. Depending on the user role, a page may offer different options to the user. For example, a professor should have different options when viewing a course he is in charge of, than when he is viewing one where he is not. This needs to be achieved with the minumum amount of reduntant code, since duplicate code often is a source of errors and also adds an unnecessary level of complexity to maintaining and debugging the application.

**Security:** Security is always a concern for online applications. Whether the case is a malicious user, or just a simple hiccup on the connection, both incoming and outgoing data needs to be validated. Furthermore, users must be limited to what their role gives them access to.

### 1.3 Tackling the Challenges

RoR is a framework built in Ruby for developing web appications, is a perfect candidate for the task at hand. RoR is based on two pillar stones, as stated on the official RoR website.[1]

- Don't Repeat Yourself: DRY is a principle of software development which states that "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.". By not writing the same information over and over again, the code is more maintainable, more extensible, and less buggy.
- Convention Over Configuration(CoC): Rails has opinions about the best way to do many things in a web application, and defaults to this set of conventions, rather than require that you specify every minutiae through endless configuration files.

RoR is built based on addressing the coding efficiency problem discussed earlier. Additionally, RoR is by default a MVC framework, which makes proper modeling easier. Moreover, it is a widely used, open source software, with a large community. This means that there is already a ton of information out there, and someone, sometime, has already asked the same question. In case that is not true, the rails community will provide a solution soon enough. Which brings us to the next point. Like all programming languages, Ruby — and consequently RoR — has its own libraries, called "Gems".

Gems are extensively used, and quite often hold the solutions to problems that occur. In case a particular feature is requested, it is advisable to first check whether a gem that addresses it already exists, before attempting to implement a solution onelself. It is very likely that others have already made the effort to address the problem, and the solution awaits in the form of a gem. Gems can be found in Github[2], RubyGems[3] and The Ruby Toolbox.[4]

## 1.4 Purpose of this Thesis

In this thesis, we document the creation of an e-class platform by using RoR, how gems can be used to assist in development, and also demostrate the use of other technologies, to compliment what RoR offers. In the next part, it showcases the end result, and the functionality the application offers. Following, it goes in to detail about the code, the gems and how to properly use them.

# 2. THE ROLES AND FUNCTIONALITY OF E-CLASS

## 2.1 The Administrators

Administrators have a crucial role in the application, although their functionality is fairly limited. Administrators have the ability to:

**Manage users:** Have an overview of all users, and manage their roles. Since only administrators can access the user management page, a master account exists, whose role cannot be changed, to ensure there is always at least one administrator.

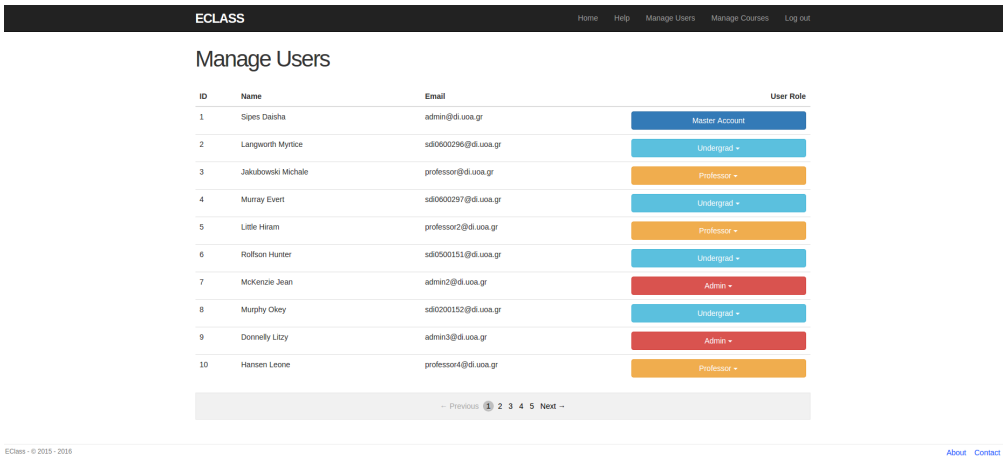


**Fig. 1: Admininstrator - User management**

**Create Courses:** Create new courses. Administrators must set the course's name and code, and assign a professor in charge of it. A drop-down menu with all professors is provided for convenience and error avoidance.

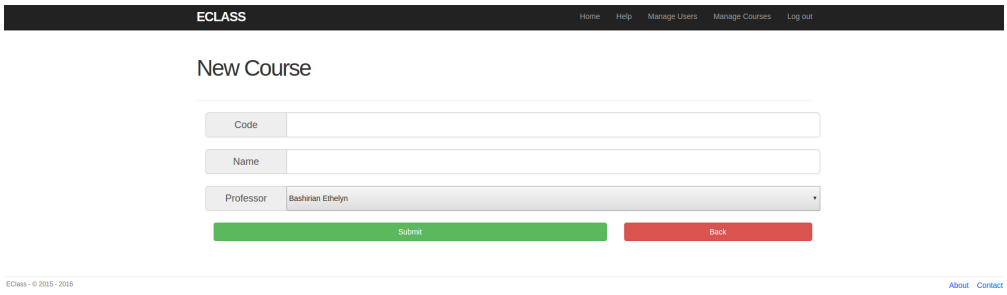


**Fig. 2: Admininstrator - Create new course**

**Edit and Delete Courses:** Changes need to be made from time to time. A course might need to have some attribute altered, or even be deleted. For that reason,

administators have the option to edit or delete each course.

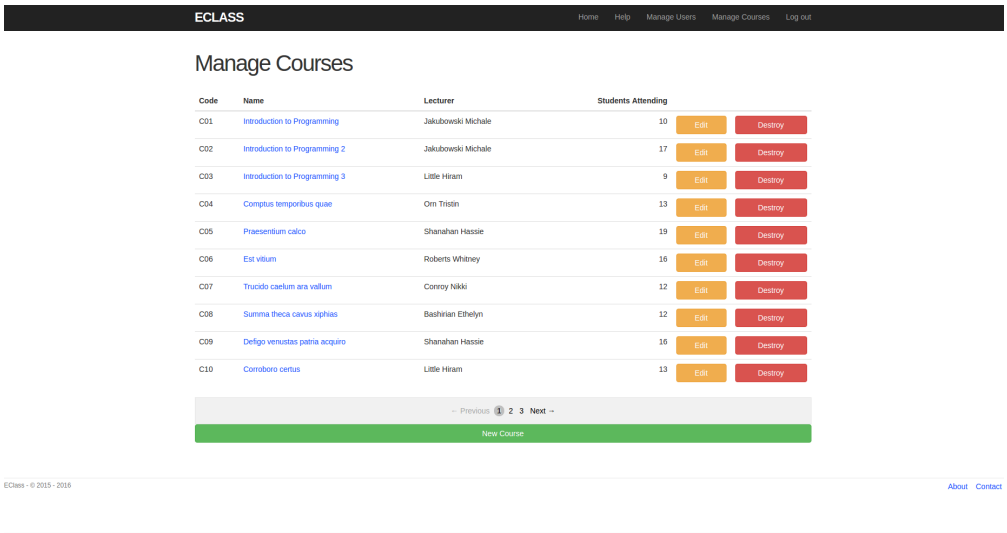


**Fig. 3: Admininstrator - Course index**

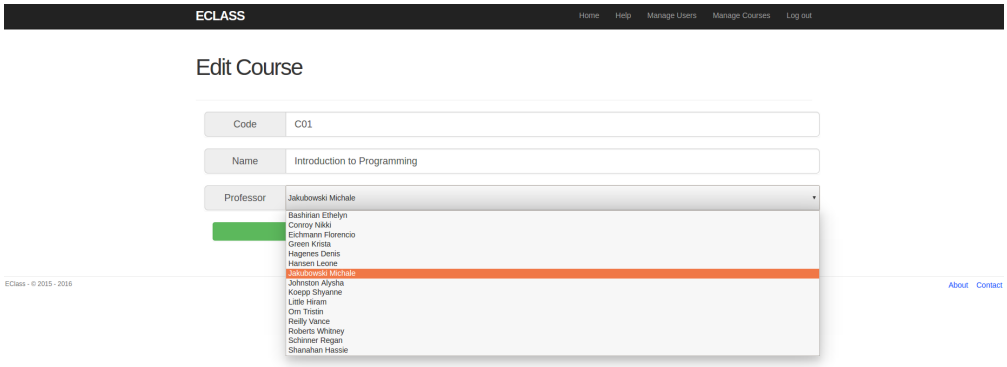


**Fig. 4: Admininstrator - Edit course**

## 2.2 Professors

Professors have a variety of tools available in order to manage their courses. These include:

**Edit Course Description:** Access to an index page containing only the courses they manage. Moreover, when viewing the main course index, a visual cue is in place to indicate their courses.

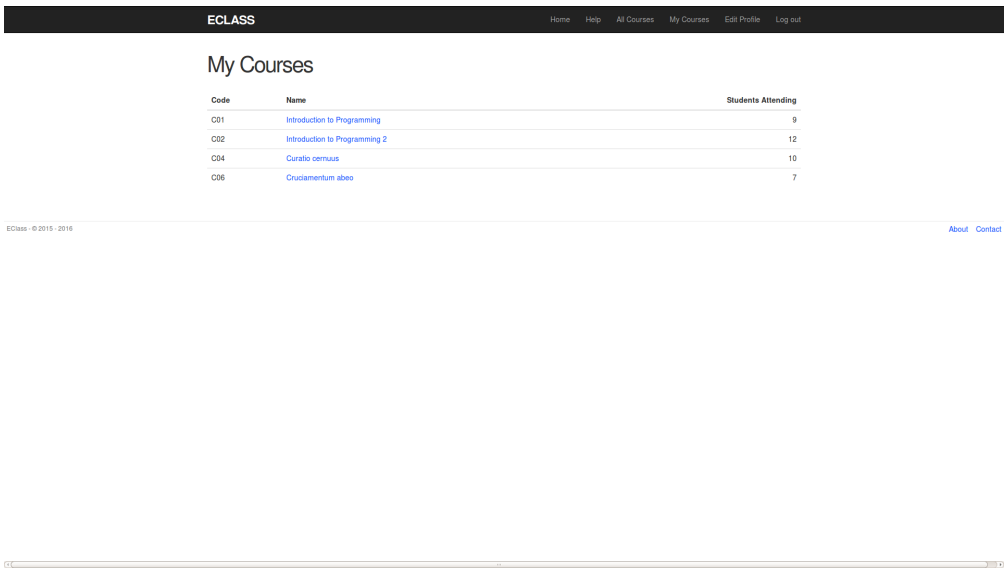


**Fig. 5: Professor - Personalized courses**

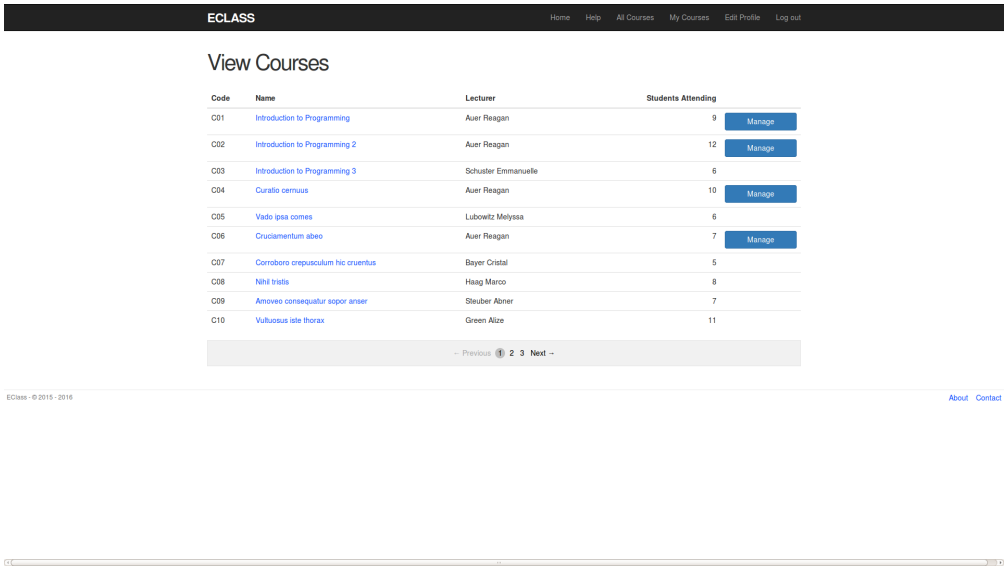


**Fig. 6: Professor - Course index**

**Edit Course Description:** The ability to edit the description of the courses they manage, to better reflect the objective of each course. They can accomplish that by clicking on the description while viewing the course.

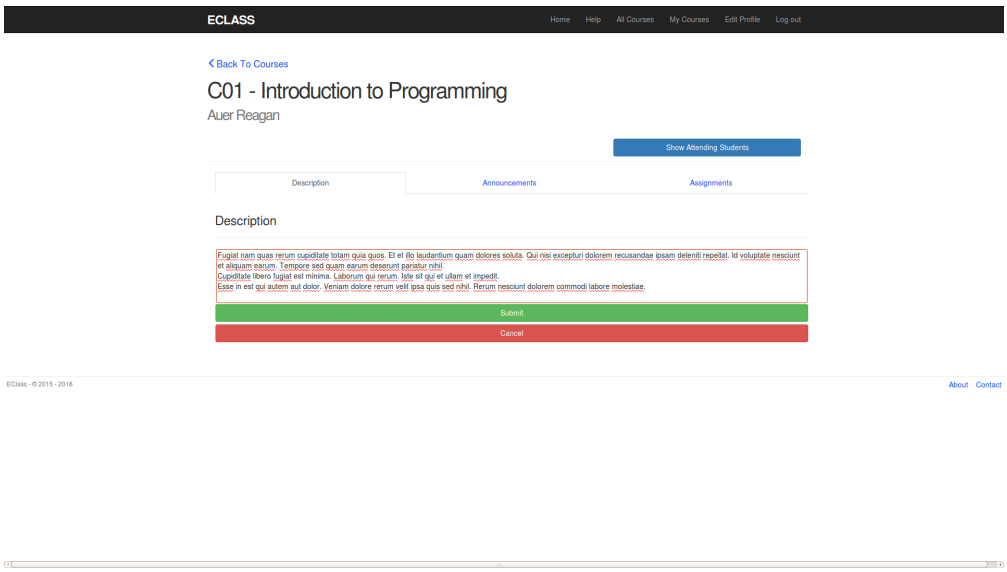


**Fig. 7: Professor - Edit course description**

**Create New Announcements:** Adding a new announcement for the students can be done seamlessly, directly from the course page.

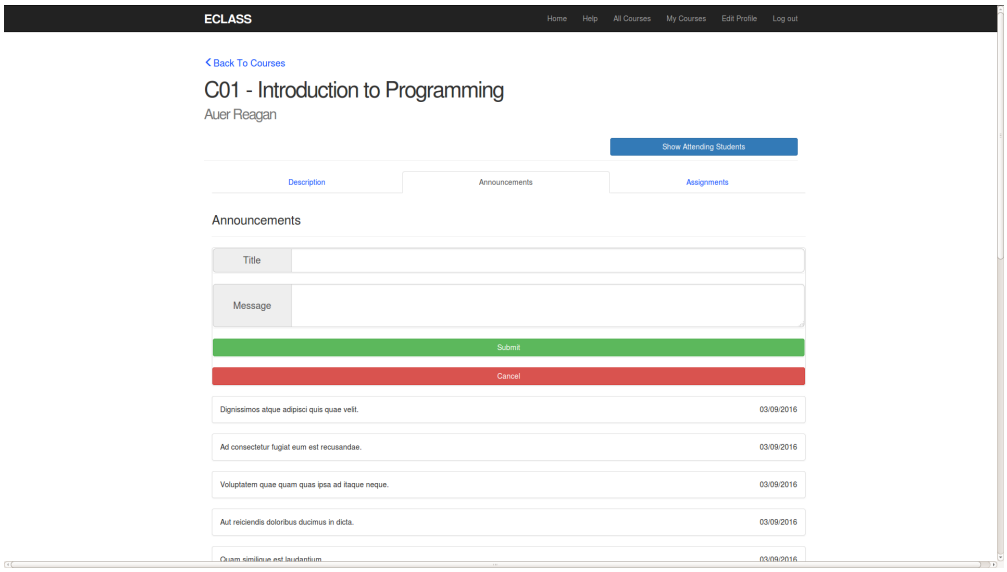


**Fig. 8: Professor - Create course announcement**

**Edit Course Announcements:** Announcements can be edited, just like descriptions. All it takes is for the course professor to click on the announcement title or message, and they can edit it in place.

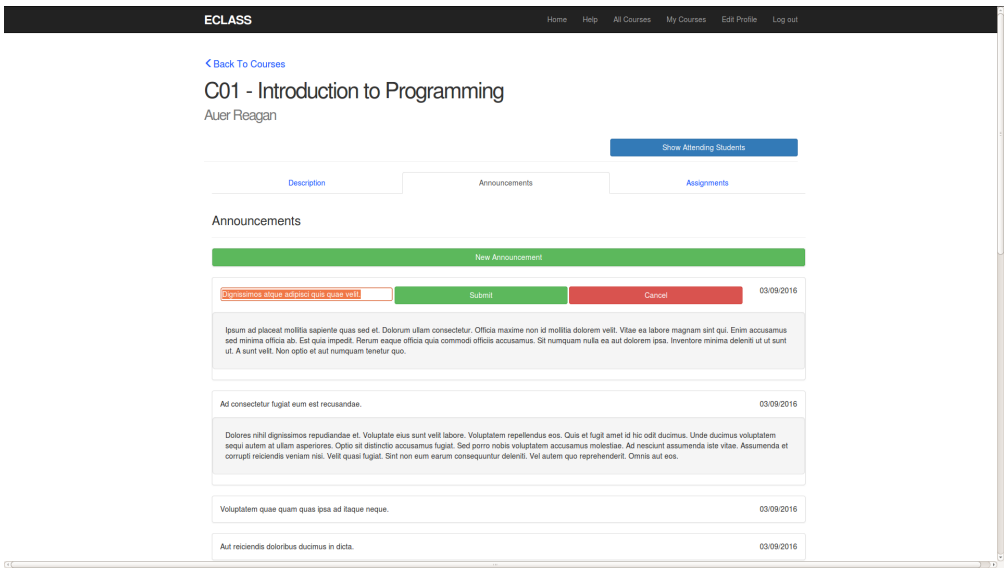


**Fig. 9: Professor - Edit course announcement title**

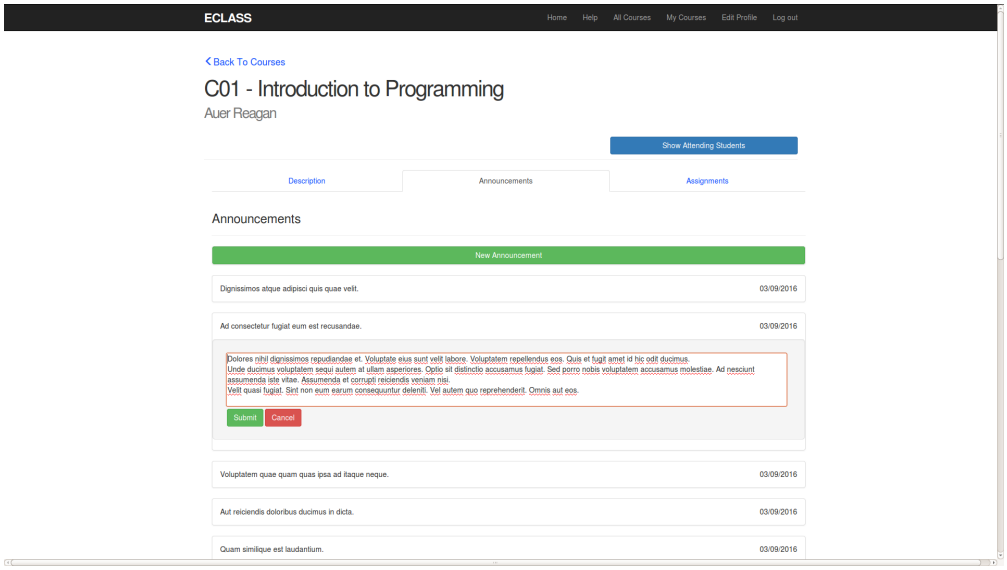


**Fig. 10: Professor - Edit course announcement message**

**Create Course Assignment:** Professors have the ability to create assignments for their students, and the choice to add accompanying documents, available to download, for assistance. Moreover, they set a deadline. Once the deadline is over, the students can no longer upload their homework.

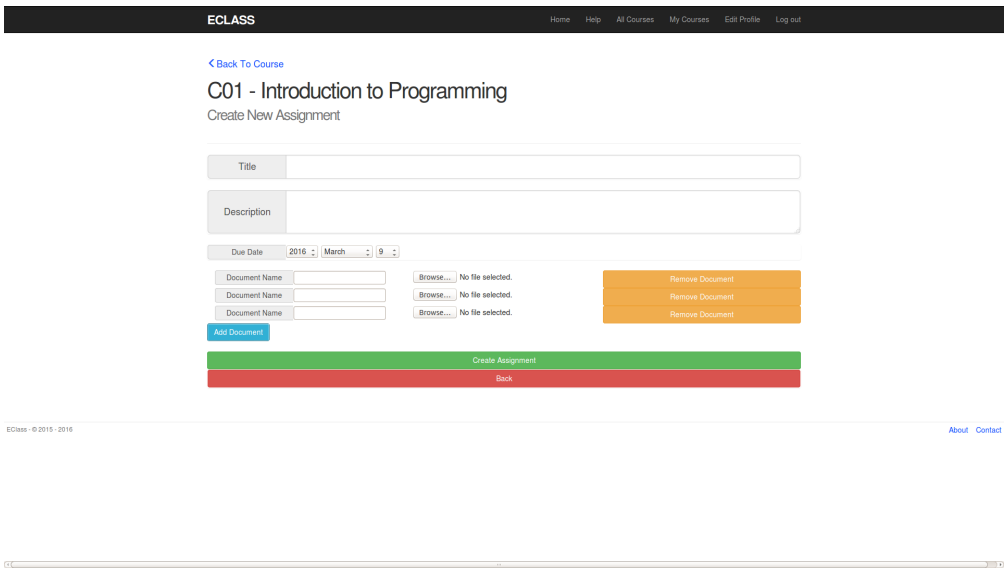


**Fig. 11: Professor - Create course assignment**

**Edit Course Assignment:** Assignments can be edited in two ways. The description can be edited in place in the assignment page, and documents can be added or deleted. There is a dedicated edit page in order to change the title and deadline of the assignment.

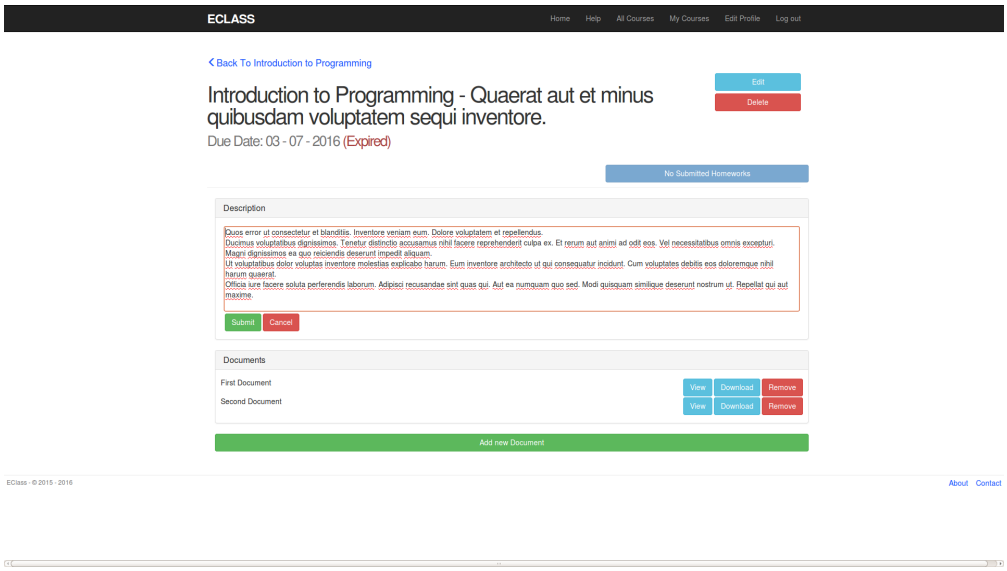


**Fig. 12: Professor - Edit course assignment - Description only**

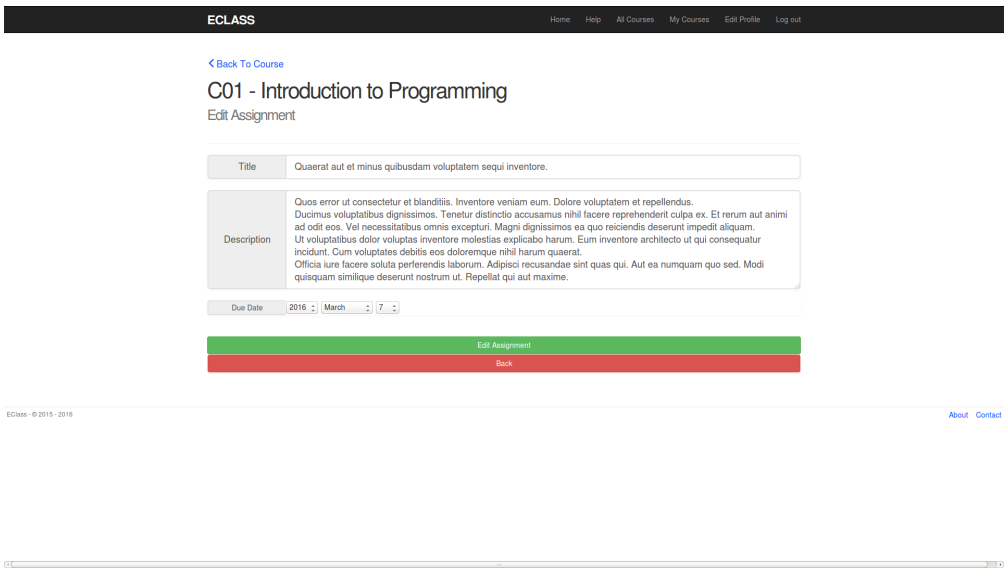


**Fig. 13: Professor - Edit course assignment**

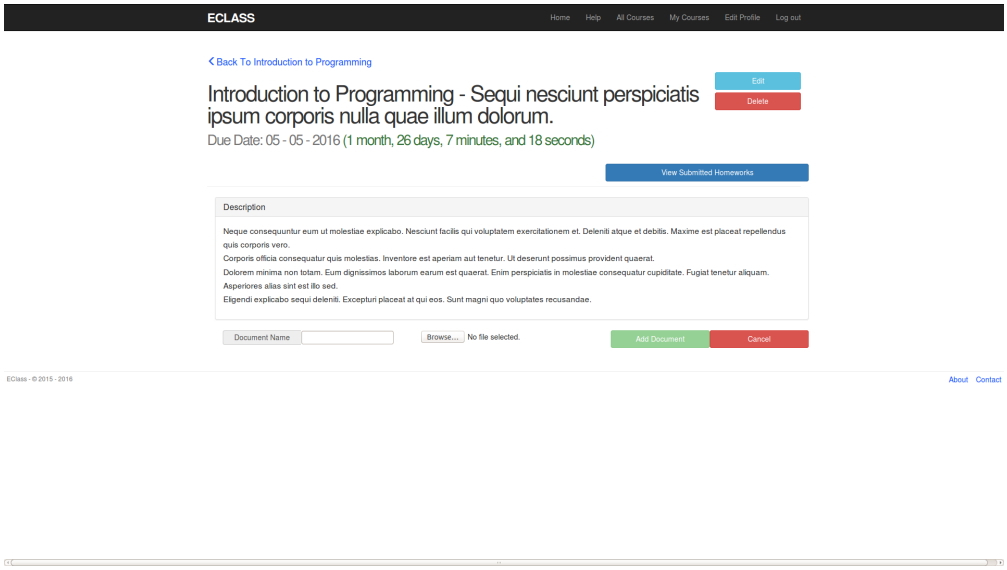


**Fig. 14: Professor - add and delete documents**

**View and Download Homeworks:** Professors have access to a page where they can download homeworks submitted by students. As shown in 12 and 14, the link is only enabled if at least a homework has been submitted.

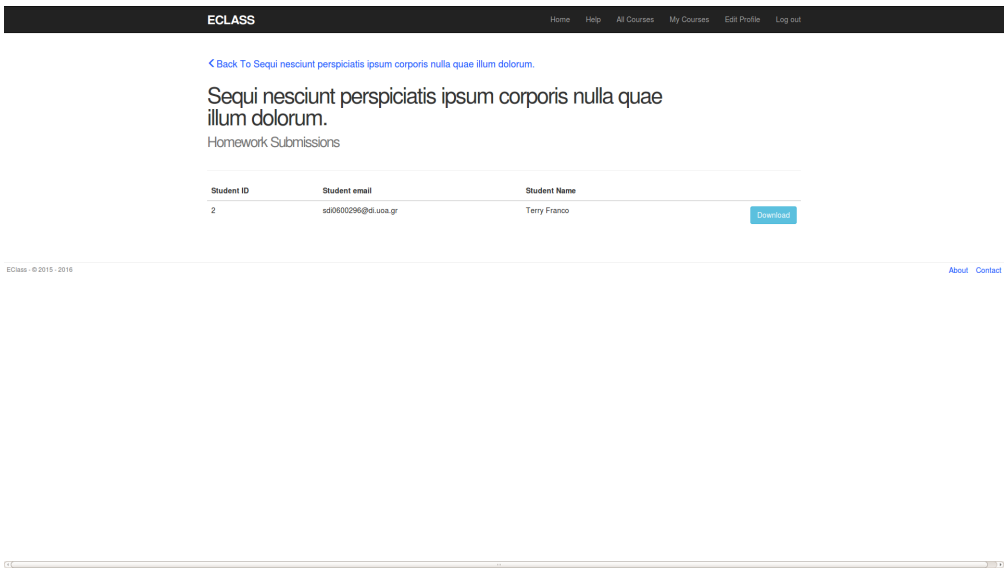


**Fig. 15: Professor - View homeworks**

**View Attending Students:** Lastly, professors can have an overview of all users attending each of their courses, their names and emails.

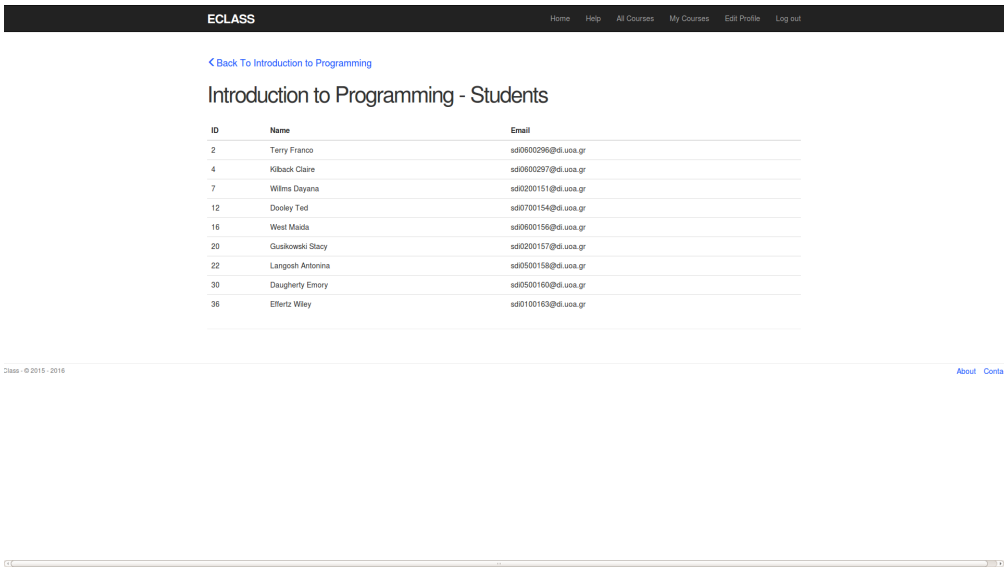


**Fig. 16: Professor - students attending course**

## 2.3 Undergraduate Students

Students differ from the other two categories, in the sense that they are mostly recepients of information instead of providers. However, they do enjoy unique features, like:

**An Index of Courses they Attend:** Besides the index of all available courses, shared by all roles, students also have a personalised page which includes only the courses they are enrolled.

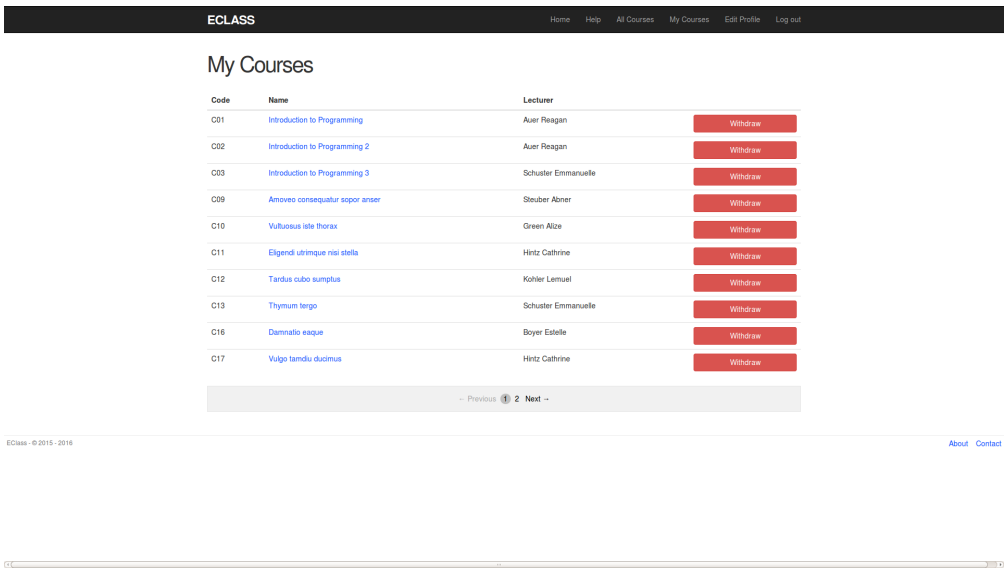


**Fig. 17: Student - My courses**

**Quick Enrolling and Withdrawing:** Students have the ability to easily enroll to and withdraw from courses, from every page associated with them. They can accomplish that in the course index page, the personalised course page 17, as well as each individual course page. 20

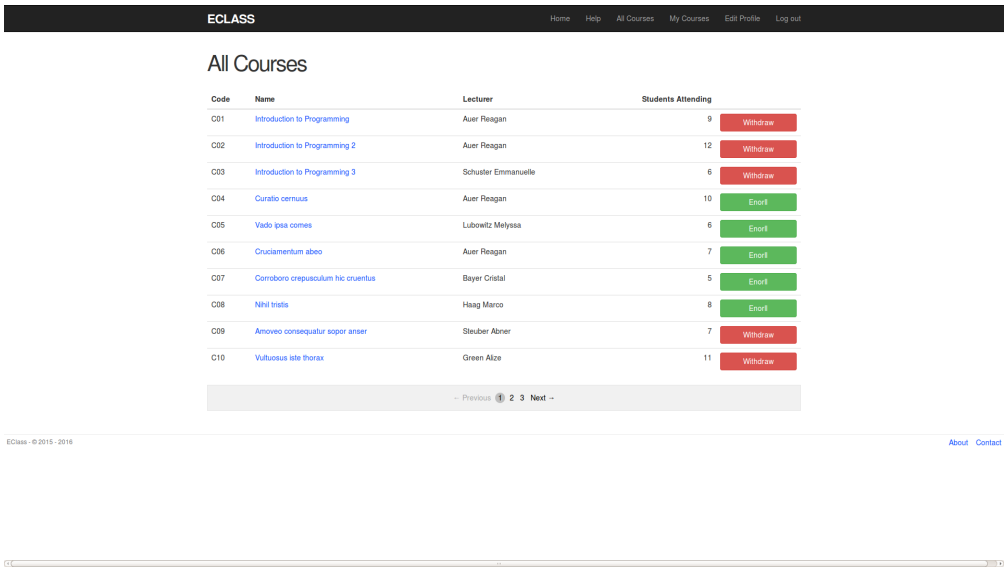


**Fig. 18: Student - Courses**

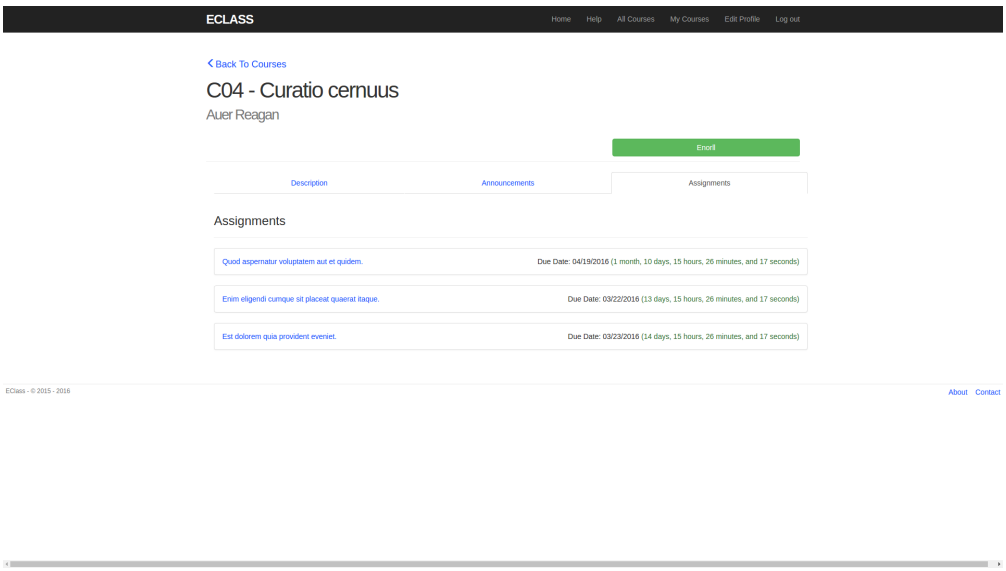


**Fig. 19: Student - Enroll from individual course**

**Visual Assignment Deadline Overview:** Students can check their assignment deadlines at a glance, both from the assignment index in the course page, and the assignment page itself.

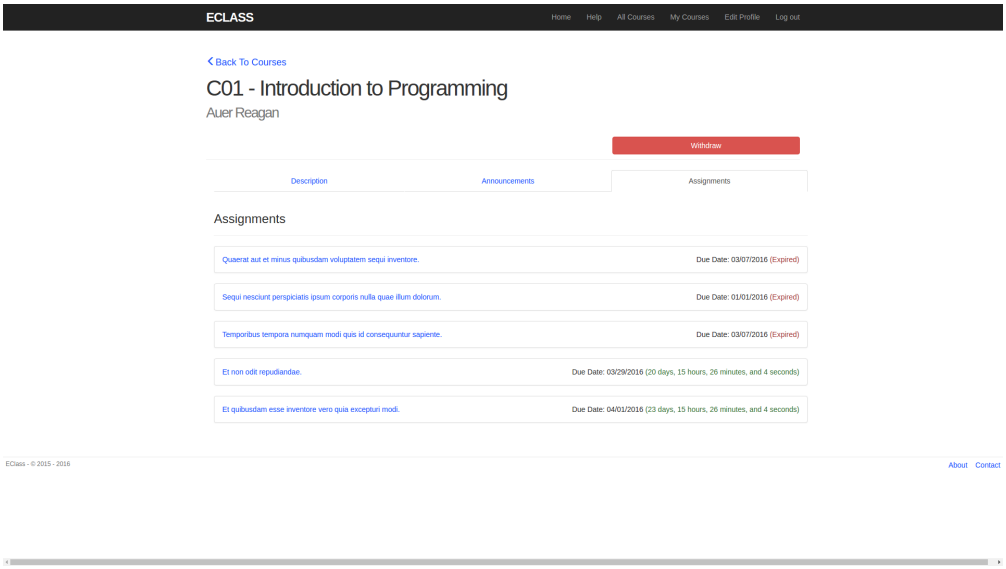


**Fig. 20: Student - Course assignments overview**

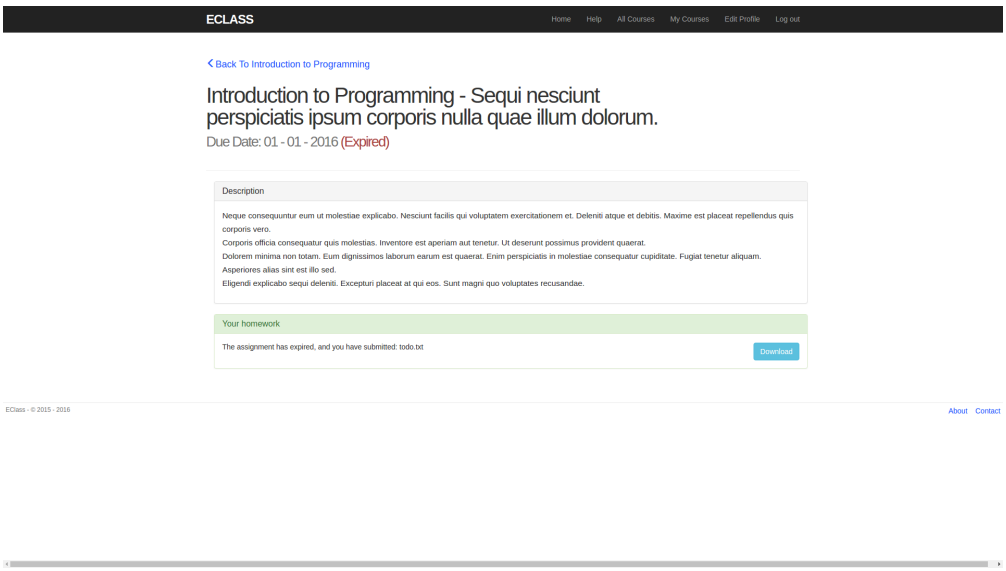


**Fig. 21: Student - View assignment**

**Homework Submission and Download:** The interface is different depending on whether the homework is submitted or not. A student is able to upload a different file, overwriting the original, as well as download the one he has uploaded. Once the deadline for the assignment is over, the student no longer has the ability to upload files 24, but is still able to download a file he has submitted 21.

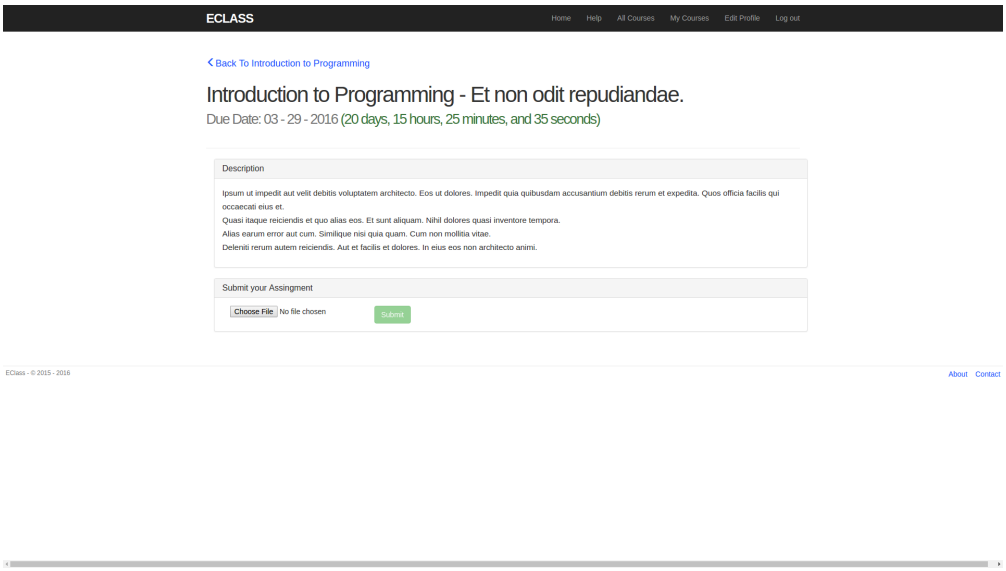


**Fig. 22: Student - Assignment not expired, homework not submitted**

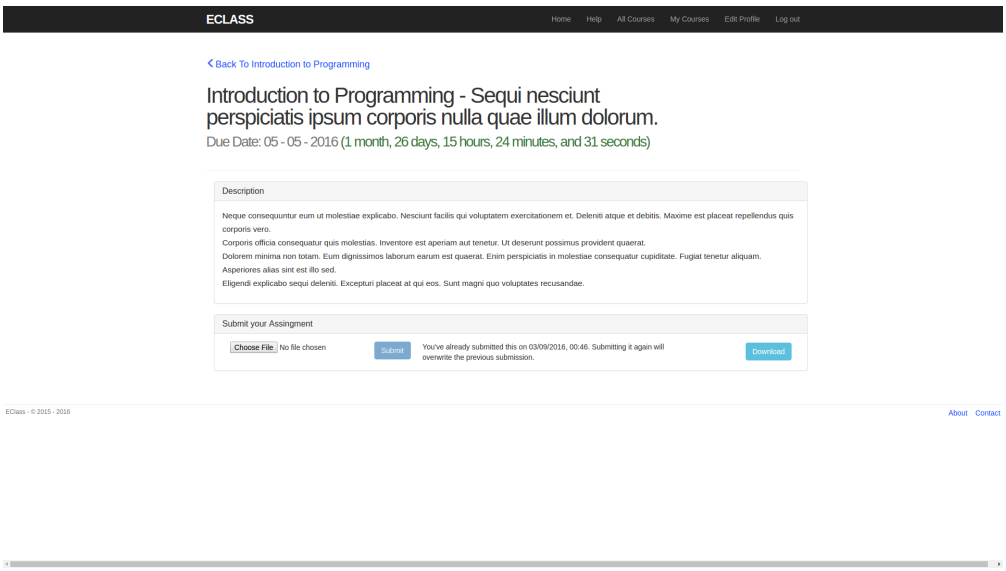


**Fig. 23: Student - Assignment not expired, homework submitted**

**View and Download Assisting Files:** Students are also able to view and download files associated with assignments through the assignment page.

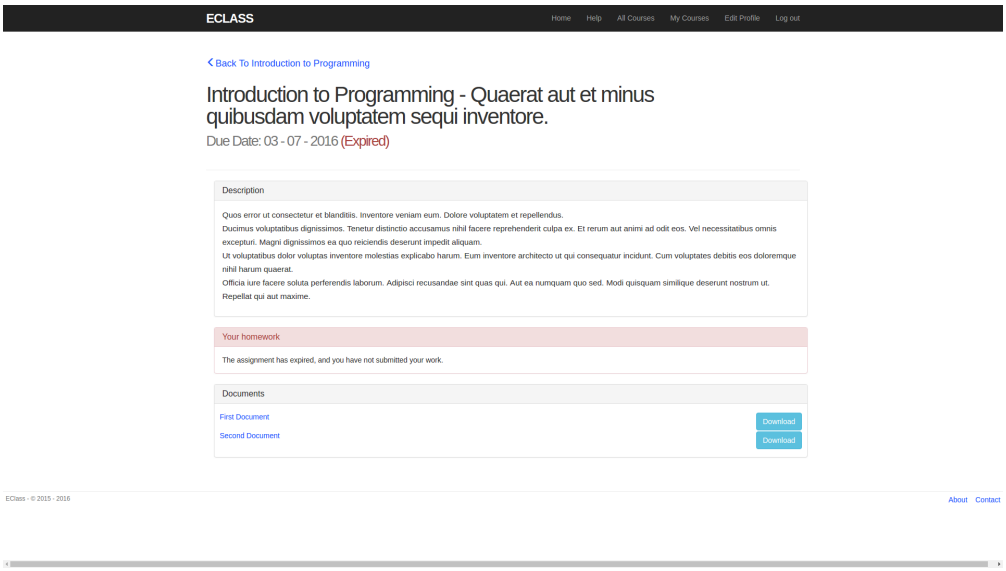


**Fig. 24: Student - Document download**

## 2.4 Shared Pages

Pages demonstrated above are similar, but offer different features according to the role of the user visiting them. Obviously, in some cases this is not needed, so pages like the log-in screen or the sign-up page are shared.

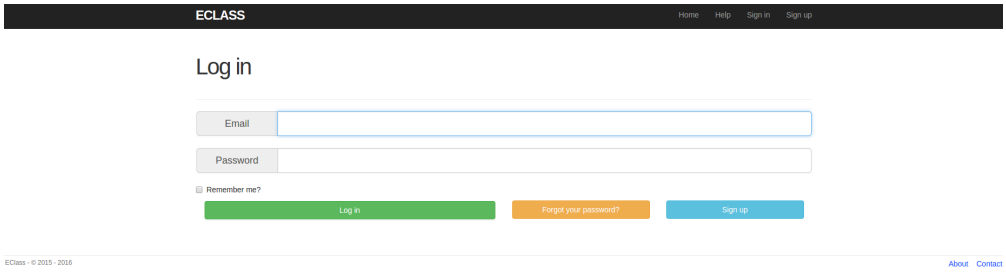


**Fig. 25: Log-in page**

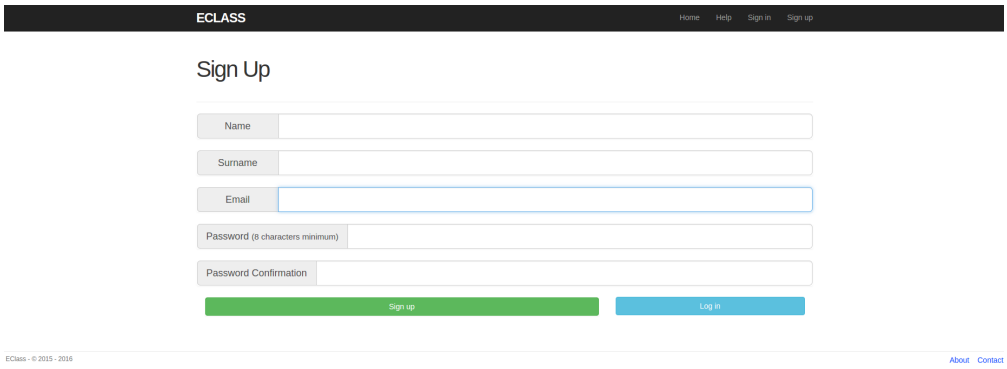


**Fig. 26: Sign-up page**

# 3. DEVELOPING THE APPLICATION

Following the presentation of the application and its features, we now analyse the process of developing it. By explaining how it came to be, a closer look is taken at RoR and what it has to offer. RoR is by design a MVC framework, therefore the next logical step is to examine the models, views and controllers, as well as the techniques, tools and gems used to tie it all together.

## 3.1 The Models

RoR uses Active Record(AR) as an Object Relational Mapping(ORM) framework.[5] This, in association with CoC, do most of the configuration automatically. Manual configuration only needs to be done when the standard convention cannot be followed. A closer look at the models of the application shall accentuate this point. Models that do not offer functionality not already discussed will be omitted.

### 3.1.1 User

```
class User < ActiveRecord::Base
  rolify
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable

  validates :name, :surname, presence: true

  after_create :assign_default_role

  has_many :homeworks, inverse_of: :user

  has_many :courses_teaching, :class_name => :Course, inverse_of: :lecturer,
↪    :foreign_key => "lecturer_id"

  has_many :student_attends_courses, inverse_of: :student, :foreign_key => "user_id"
  has_many :courses_attending, :class_name => :Course, through:
↪    :student_attends_courses, :foreign_key => "user_id"

  def assign_default_role
    add_role(:undergrad)
  end

  def is_master_acc?
    self.id == User.first.id
  end

end
```

**Listing 1: The User Model**

This is the file for the User model. Dissecting it line by line provides a better understanding.

- **class User** < ActiveRecord::Base

  This line contains the name of the model and shows the inheritance of the AR base.

- rolify
  devise :database_authenticatable, :registerable,
  :recoverable, :rememberable, :trackable, :validatable

  There are functions provided by gems, rolify [8] and devise[9], and will be discussed in detail later on.

- validates :name, :surname, presence: **true**

  With this simple function, it is ensured that every User record will contain a name and a surname. E-mail and password are not present, since they are handled by devise[9].

- after_create :assign_default_role

  `after_create` sets callback functions to be executed on the object, immediately after they are created. In this case, a role is assigned to the user as soon as they sign up.

- has_many :homeworks, inverse_of: :user

  By using has_many, model relationships are defined, specificaly a one-to-many relationship. `inverse_of:` is optional, but quite powerful. It offers bi-directional access to models, omitting the need for an SQL query to do so.

- has_many :courses_teaching, :class_name => :Course, inverse_of: :lecturer,
  ↪ :foreign_key => ”lecturer_id”

  This line of code is similar to the previous one. However, it offers 2 additional parameters. Since the first argument of has_many is not the name of a model, the model is provided by using `:class_name`. This enables us to use an alias, which accomplishes 2 goals. Firstly, it allows us to have more than one relationships between two models. In this case, the course has to belong to a single user, its lecturer, but also have many users, the students who attend it. Additionally, it increases code readability, since `course.lecturer` is more intuitive than `course.user`. The last argument, `:foreign_key`, specifies the column in the corresponding table, in this case the courses table, which includes the key to the user model.

- has_many :student_attends_courses, inverse_of: :student, :foreign_key =>
  ↪ ”user_id”
  has_many :courses_attending, :class_name => :Course, through:
  ↪ :student_attends_courses, :foreign_key => ”user_id”

Further highlighting the benefit of setting an alias, a second relationship between users and courses is set. This time it is a many-to-many relationship. Usually, this is accomplished by using has_and_belongs_to_many. Since it is not the only relationship between the two tables, a third table needs to be introduced, by using the through: argument. In order to increase the readability of the code, an alias is again used. By adding a slight complexity, merely an extra line, two different kinds of relationships can be set between the models, with an added bonus of "natural" code readability.

- ```
  def assign_default_role
    add_role(:undergrad)
  end

  def is_master_acc?
    self.id == User.first.id
  end
  ```

Two methods are defined for the User model. assign_default_role is the method called after a new instance of the model is created. All users are initially set to be students, to prevent unexpected behaviour in case a role is not assigned. The other function, is_master_acc? is used to verify that an administrator account will always exist, as discussed earlier. 1

### 3.1.2 Course

```
class Course < ActiveRecord::Base
  belongs_to :lecturer, :class_name => :User, inverse_of: :courses_teaching

  has_many :student_attends_courses, inverse_of: :course, :foreign_key => "course_id"
  has_many :students, :class_name => :User, through: :student_attends_courses,
  ↪    :foreign_key => "course_id"

  has_many :announcements, inverse_of: :course, :dependent => :destroy
  has_many :assignments, inverse_of: :course, :dependent => :destroy

  validates :code, :name, :lecturer, presence: true
end
```

**Listing 2: The Course Model**

The course model. Examing it fills up the blanks left from the user model 1 and gives us an overall understanding of relationships between models.

- belongs_to :lecturer, :class_name => :User, inverse_of: :courses_teaching

  belongs_to compliments the has_many used by the user model. belongs_to
```

is ambiguous, in the sense it is not clear whether it is a one-to-many or one-to-one relationship. When `inverse_of:` is set, the relationship is indicated by its value. A singural word means it is one-to-one, and plural is one-to-many. However, `inverse_of:` is optional, so when it is not user, the relationship is determined by the corresponding model — has_many means one-to-many, whereas has_one is used for one-to-one. Regardless of the kind of the relationship, `belongs_to` means that this model will hold the id of the object it belongs to. By conversion, this column is named model_name_id, but since an alias is used, `:class_name` is set, and `:foreign_key` is set to the corresponding model.

- ```
  has_many :student_attends_courses, inverse_of: :course, :foreign_key =>
  ↪ ”course_id”
  has_many :students, :class_name => :User, through: :student_attends_courses,
  ↪ :foreign_key => ”course_id”
  ```

  These expressions mirror the ones in the user model 1.

- ```
  has_many :announcements, inverse_of: :course, :dependent => :destroy
  has_many :assignments, inverse_of: :course, :dependent => :destroy
  ```

  Through the above associations, it is clear that the course model has one-to-many associations with the assignment and announcement models. It is safe to assume that both of their models include `belongs_to :course, inverse_of:` in them and a `course_id` column in their tables.
  The parameter `dependent => :destroy` makes sure than once a course is deleted, all announcements and assignments associated with it will be deleted as well.

- ```
  validates :code, :name, :lecturer, presence: true
  ```

  The validation has one noticable difference to the one found in User 1. It validates the presence of `:lecturer`, although the attribute is named `:lecturer_id`. This is actually crucial, because not only it validates that the field is not empty, but also that the id belongs to an existing record.

### 3.1.3 StudentAttendsCourse

```
class StudentAttendsCourse < ActiveRecord::Base
  belongs_to :courses_attending, :class_name => :Course, inverse_of:
  ↪ :student_attends_courses, :foreign_key => ”course_id”
  belongs_to :student, :class_name => :User, inverse_of: :student_attends_courses,
  ↪ :foreign_key => ”user_id”
end
```

**Listing 3: The StudentAttendsCourse Model**
```

The missing link to the many-to-many relationship between courses and the students who attend them. This model belongs to both a student and a course, and through it the two other models are connected. `:class_name`, `:inverse_of` and `:foreign_key` are set according to the values mentioned before. 1 2

### 3.1.4 Announcement

```
class Announcement < ActiveRecord::Base
  belongs_to :course, inverse_of: :announcements
  validates  :title, :message, :course, presence: true
  default_scope { order('updated_at DESC') }
end
```

**Listing 4: The Announcement Model**

The announcement model. The last function in the model, `default_scope`[10]. This method takes a block of code as an argument and adds a scope for all operations in the model. In this example, it is used in conjuction with `order('updated_at␣DESC')`, which orders announcements in descending order, according to their last update, thus ensuring that the most recently updated announcements always appear first.

### 3.1.5 Assignment

```
class Assignment < ActiveRecord::Base
  belongs_to :course, inverse_of: :assignments
  has_many :documents, inverse_of: :assignment, :dependent => :destroy
  has_many :homeworks, inverse_of: :assignment, :dependent => :destroy
  accepts_nested_attributes_for :documents
  validates :title, :description, :due_date, :course, presence: true
  validates_associated :documents
end
```

**Listing 5: The Assignment Model**

The assignment model contains a set of methods related to models that belong to it.

- accepts_nested_attributes_for :documents

  This function enables the management of the documents model through its parent assignment. This is extremely useful because it allows the addition of documents to the assignment directly when creating it, as well as updating documents through the assignment.

- validates_associated :documents

  This method acts like `validates` does, but for the associated object. It completes the functionality of `accepts_nested_attributes_for`, by validating the attributes of the associated record.

### 3.1.6 Document

```
class Document < ActiveRecord::Base
  belongs_to :assignment, inverse_of: :documents
  mount_uploader :doc, DocumentUploader
  validates :name, :doc, :assignment, presence: true
end
```

In the document uploader, the `mount_uploader` appears. This method is provided by the carrierwave gem.[11]

## 3.2 The Controllers

Controllers are responsible for transferring data between the end user and the application. The routing from each request to the appropriate controller is done in a single file, “routes.rb”. Controllers in RoR prepare the information and make it available to for the views, and ara capable on responding differently to different kinds of requests. Below, selected bits and pieces from controllers are presented. Due to the CoC nature of RoR, a significant part of the functionality is identical among controllers, so it makes little sense to showcase them all.

### 3.2.1 The Course Controller

- `class CoursesController < ApplicationController`

  All controllers in RoR inherit the ApplicationController[6], which in turn inherits ActionController[7]. This allows for a centalised class to configure application security.

- `before_action :set_course, only: [:show, :edit, :update, :destroy, :description, :attending_students]`

  Like in the models, `before_action` sets methods to be called before the actual method is called. In this example, the method `set_course` is executed exclusively — unless called directly of course — before the actions between the brackets.

  ```
  def set_course
    params[:id] = params[:course_id] if params[:id].nil?
    @course = Course.find(params[:id])
  end
  ```

  This conforms with the DRY nature of RoR, since instead of having the code above, or a call of the method, separately in all the methods in the bracket, the methods requiring it are neatly congregated in the `before_action`.

```
def create
  @course = Course.new(course_params)

  respond_to do |format|
    if @course.save
      format.html { redirect_to @course, notice: 'Course was successfully created.'
↪     }
      format.json { render :show, status: :created, location: @course }
    else
      get_professors
      format.html { render :new }
      format.json { render json: @course.errors, status: :unprocessable_entity }
    end
  end
end
```

Examining the method to create a new course, leads to interesting findings. Right away, another method, called `course_params` is passed as an argument in the `Course.new` method.

```
def course_params
  params.require(:course).permit(:code, :name, :lecturer_id, :description)
end
```

The `course_params` sanitizes the parameters passed by the request, and only allows the proper ones to go through. The next step is to attempt to save the newly created course. Depending on the outcome, one of two might happen. Either the course is successfully saved, and the controller responds with the corresponding format of the request, or it fails and the appropriate action is taken. In this specific case, the function `get_professors` is called, and then a proper response, depending on the format is given.

```
def get_professors
  professors = User.with_role :professor
  @professors = []
  professors.each do |a|
    @professors << ["#{a.surname} #{a.name}", a.id]
  end
  @professors.sort! { |a, b| a[0] <=> b[0]}
end
```

The `get_professors` method is used to create an array which contains information about all available professors. That array is used in the drop-down menu of the course form, as displayed earlier. 4

```
• def create
    @course = Course.new(course_params)

    respond_to do |format|
      if @course.save
        format.html { redirect_to @course, notice: 'Course was successfully created.'
↪       }
        format.json { render :show, status: :created, location: @course }
      else
        get_professors
        format.html { render :new }
        format.json { render json: @course.errors, status: :unprocessable_entity }
      end
    end
  end
```

This method is a good example of the aliases discussed in the models section 1. The `is_professor?` and `is_undergrad?` methods are provided by rolify[8]. Depending on the type of user making the request, the controller is able to get either the courses the professor is teacher, or the ones the undergradute student attends. Then, the `paginate` method, provided by the will_paginate gem[12], takes care of the pagination in the view page, as showcased before 5 17. `render` method renders the desired view. By conversion, Rails renders the view that shares the name with the controller. In case this is not the desired effect, it has to be instructed to do otherwise.

### 3.2.2 The Assignment Controller

```
def assignment_params
  params.require(:assignment).permit(:title, :description, :due_date, :course_id,
↪   documents_attributes: [:name, :doc, :_destroy])
end
```

The `assignment_params` method is interesting, since it has already been established that assignment may include documents when they are created 11.
The `documents_attributes` array and its contents is what permits the attributes of the documents to pass the sanitazation process, and be created.

## 3.3 The Views and Helpers

According to the MVC paradigm, views are used to present the information to the user. However, in order to not break the paradigm, no logic should be implemented in the views. To avoid convoluted code in views and controllers, or a version of the view for every possible outcome, RoR uses helpers and partials. Helpers may contain logic and

are called from the view, maintaining the MVC pattern and avoiding that obstacle. Views mainly consist of the following parts, glued together by the helpers.

### 3.3.1 The Views

**HTML Pages:** Html pages are the main component of the views. However, they hardly ever are plain html pages. RoR by default uses Embedded Ruby(ERB) pages. Other options include HTML Abstraction Markup Language(HAML) [13] and Slim [14]. This project uses ERB files, with the exception of the assignment form 11, which utilizes the Cocoon gem [15]. The gem example was in HAML, so the form was created in it as well for variety's sake.

**Partial Pages:** As the name implies, those are not whole pages, but html snippets, fitting like pieces of a puzzle where they are required. Besides not being able to stand alone, partials are the same as the HTML pages described above. Their name is required to start with an underscore(_).

```
<span style="color: <%= color %>">
  <%= time_left %>
</span>
```

**JavaScript Files:** Not to be confused with JavaScript files used holding the usual JavaScript functions — these are found in the assets/javascripts folder, and often written in CoffeeScript [20]. The JavaScript files associated with views are mostly rendered as a response, made by the controller, to an Asynchronous JavaScript and XML(AJAX) request. JavaScript files can also contain ERB.

```
$('<%= "#course_#{@course.id}_att" %>').html('<%= "#{@attending}" %>');
$('<%= "#course_#{@course.id}" %>').html("<%= escape_javascript(render :partial
↪   => 'withdraw', locals: {course: @course}) %>")
```

### 3.3.2 The Helpers

Helpers are ruby methods, used to implement logic for the views. Helpers might return a result to the view, render a partial, maybe even do nothing. The following snippet is used to determine whether it is the course professor viewing the assignment, so the edit and delete buttons should be displayed, and render that partial if that is true.

```
def edit_and_delete_button(course, id)
  render partial: "edit_and_delete_assignment" if is_course_professor?(course, id)
end
```

## 3.4 The Gems

Gems, Ruby's open-source, community-backed libraries, are extensively used in Rails. It is highly likely that a problem that occurs or a feature that needs to be implemented

already exist in a gem. Following is a sample of the gems used in this application.

**Bootstrap-sass:** Boostrap-sass provides a port of Bootstrap 3 for RoR. [16]

**Devise:** Devise offers out-of-the-box user authentication and management. [9]

**Rolify:** Rolify assists with role management and scoping. [8]

**CarrierWave:** CarrierWave is a classier solution for file uploading. [11]

**Cocoon:** Cocoon makes nested form handling seamless. [15]

**Will Paginate:** Will Paginate offers pagination with a single method. [12]

**Best in Place:** Best in Place offers a highly customizable option for editing records in place by using JSON. [17]

**jQuery Turbolinks:** jQuery Turbolinks is used to address an issue to jQuery functions, caused by turbolinks. [18]

**Faker:** Faker generates genuine-looking data. It is ideal for seeding the database tables, which is perfect for developing an application. [19]

## 3.5 How it all ties together - A Workflow

In the previous sections, each component of the application was analysed in detail. To fully understand how it all ties together, an example is in order. The example consists of a step-by-step examination of an administrator changes the role of a user. In the image below, the administrator is changing the role of the user from undergrad to professor.

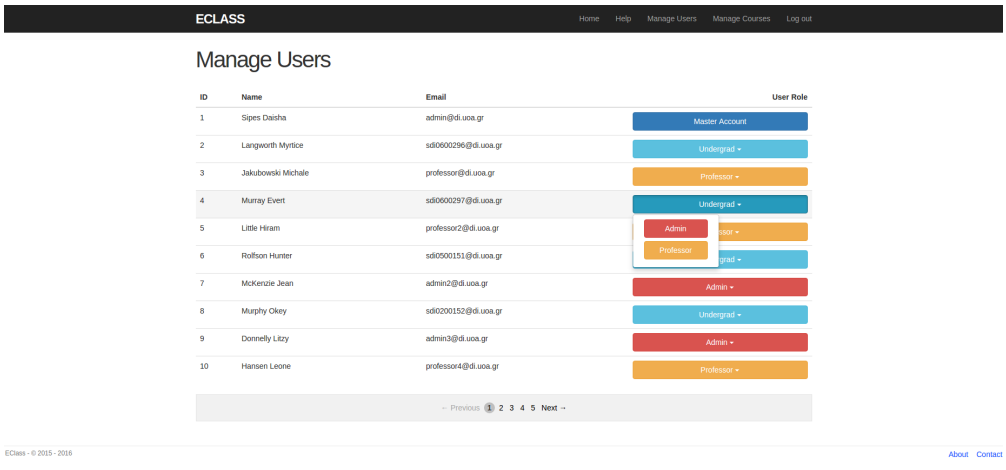


**Fig. 27: Administrator changing the role of a user.**

The HTML behind the “Professor” button

```
<form action=”/users/4/give_role?rol=professor” accept-charset=”UTF-8”
↪   data-remote=”true” method=”post”>
 <input name=”commit” value=”Professor” class=”btn btn-block btn-warning”
↪   type=”submit”>
</form>
```

According to the routes file,

```
resources :users, only: [:index] do
  post ”give_role”
end
```

that path is linked to the users controller, in the `give_role` method. The controller contains two `before_action` methods.

```
before_action :set_user, only: [:give_role]
before_action :set_hashes
```

The `set_hashes`, which are instance variables, containing the three roles as well as the colors associated with each.

```
  def set_hashes
    @colours = {
      ”admin” => ”danger”,
      ”professor” => ”warning”,
      ”undergrad” => ”info”
    }

    @drop_roles = [
      ”admin”,
      ”professor”,
      ”undergrad”
    ]
  end
```

The `set_user` method retrieves the user whose roles is about to change, and it removes the current role, unless it is the master account.

```
  def set_user
    @user = User.find(params[:user_id])
    @user.roles = [] unless @user.is_master_acc?
    @role = params[:rol]
  end
```

It checks that by calling the is_master_acc? method from the user model.

```
def is_master_acc?
  self.id == User.first.id
end
```

Once both the before_action methods are called, the give_role is executed. It calls the add_role method provided by the rolify gem [8], again checking if it is the master account.

```
def give_role
  @user.add_role(@role) unless @user.is_master_acc?
  respond_to do |format|
    format.js
  end
end
```

The method then responds to the .js format. This renders the .js file with the same name as the controller method.

```
$('<%= "#user_#{@user.id}" %>').empty();
$('<%= "#user_#{@user.id}" %>').html("<%= escape_javascript(render :partial => 'role',
↪   locals: {user: @user, role: @role}) %>")
```

In turn, that clears the div containing the dropdown menu, and renders the _role.html.erb partial,

```
<div class="btn-group btn-block">
  <button type="button" class='btn btn-<%= "#{@colours[role]}" %> btn-block
↪   dropdown-toggle' data-toggle="dropdown" aria-haspopup="true"
↪   aria-expanded="false">
    <%= role.capitalize %> <span class="caret"></span>
  </button>
  <ul class="dropdown-menu">
    <% @drop_roles.each do |r| %>
      <li><a href="#"><%= render partial: "give_role", locals: {user: user, role: r} unless r
↪   == role %></a></li>
    <% end %>
  </ul>
</div>
```

which creates the drop down menu, and calls

```
<%= form_tag(user_give_role_path(user, :rol => role), remote: true) do %>
  <%= submit_tag "#{role.capitalize}", class: "btn btn-block btn-#{@colours[role]}"%>
<% end %>
```

on each role besides the newly assigned one, to create the proper buttons.

# 4. IN CONCLUSION

In all crafting areas, it's easier to accomplish something if you have the tools of the trade. Developing web applications does not deviate from that course. This thesis aims to establish the Ruby on Rails framework as one of the best choices when it comes to web development, since it offers swift and agile development, exellent workflow and enjoys amazing community support. Through its DRY and CoC philosophies, it takes away all the chores from developing web applications, leaving programmers to focus on the fun parts.

# ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| MVC | Model-View-Controller |
| RoR | Ruby on Rails |
| DRY | Don’t Repeat Yourself |
| CoC | Convention Over Configuration |
| ORM | Object Relational Mapping |
| AR | Active Record |
| ERB | Embedded Ruby |
| HAML | HTML Abstraction Markup Language |
| AJAX | Asynchronous JavaScript and XML |

# REFERENCES


[1] “Ruby on Rails” *Ruby on Rails* [Online]. Available: http://rubyonrails.org. [Accessed: Mar 8, 2016]

[2] “GitHub” *GitHub* [Online]. Available: https://github.com. [Accessed: Mar 8, 2016]

[3] “Ruby Gems” *Ruby Gems* [Online]. Available: https://rubygems.org. [Accessed: Mar 8, 2016]

[4] “The Ruby Toolbox” *The Ruby Toolbox* [Online]. Available: https://www.ruby-toolbox.com. [Accessed: Mar 8, 2016]

[5] “Active Record Basics” *Active Record Basics* [Online]. Available: http://guides.rubyonrails.org/active_record_basics.html [Accessed: Mar 8, 2016]

[6] “Application Controller” *Application Controller* [Online]. Available: http://api.rubyonrails.org/classes/ActionController/Base.html [Accessed: Mar 8, 2016]

[7] “Action Controller Base” *Action Controller Base* [Online]. Available: http://api.rubyonrails.org/classes/ActionController/Base.html [Accessed: Mar 8, 2016]

[8] “Rolify” *Role management library with resource scoping* [Online]. Available: https://github.com/RolifyCommunity/rolify [Accessed: Mar 8, 2016]

[9] “Devise” *Flexible authentication solution for Rails with Warden* [Online]. Available: https://github.com/plataformatec/devise [Accessed: Mar 8, 2016]

[10] “Active Record Scoping Default Methods” *Active Record Scoping Default Methods* [Online]. Available: http://api.rubyonrails.org/classes/ActiveRecord/Scoping/Default/ClassMethods.html [Accessed: Mar 9, 2016]

[11] “CarrierWave” *Classier solution for file uploads for Rails, Sinatra and other Ruby web frameworks* [Online]. Available: https://github.com/carrierwaveuploader/carrierwave [Accessed: Mar 9, 2016]

[12] “https://github.com/mislav/will_paginate” *Pagination library for Rails, Sinatra, Merb, DataMapper, and more* [Online]. Available: https://github.com/mislav/will_paginate [Accessed: Mar 9, 2016]

[13] “HAML” *HTML Abstraction Markup Language* [Online]. Available: http://www.haml.info [Accessed: Mar 9, 2016]

[14] “Slim” *A fast, lightweight template engine for Ruby* [Online]. Available: http://slim-lang.com [Accessed: Mar 9, 2016]

[15] “Cocoon” *Dynamic nested forms using jQuery made easy* [Online]. Available: https://github.com/nathanvda/cocoon [Accessed: Mar 9, 2016]

[16] “Bootstrap-sass” *Official Sass port of Bootstrap 2 and 3* [Online]. Available: https://github.com/twbs/bootstrap-sass [Accessed: Mar 9, 2016]

[17] “Best_in_place” *A RESTful unobtrusive jQuery Inplace-Editor and a helper as a Rails Gem* [Online]. Available: https://github.com/bernat/best_in_place [Accessed: Mar 9, 2016]

[18] “jQuery Turbolinks” *jQuery plugin for drop-in fix binded events problem caused by Turbolinks* [Online]. Available: https://github.com/kossnocorp/jquery.turbolinks [Accessed: Mar 9, 2016]

[19] “Faker” *A library for generating fake data such as names, addresses, and phone numbers.* [Online]. Available: https://github.com/stympy/faker [Accessed: Mar 9, 2016]

[20] “CoffeeScript” *CoffeeScript is a little language that compiles into JavaScript.* [Online]. Available: https://http://coffeescript.org [Accessed: Mar 9, 2016]