Universitatea Tehnică a Moldovei

Application for Individual Monitoring of Study Timetable

Aplicație pentru monitorizarea individuală a orarului de studii

Studentă: Gușan Gina

Conducător: lector universitar,

Balan Mihaela

Chişinău 2016

Abstract

The thesis named **Application for Individual Monitoring of Study Timetable**, presented by student Guşan Gina as a Bachelor project, was developed at the Technical University of Moldova. It is written in English and contains 67 pages, 10 tables, 21 figures, 16 listings and 13 references. The thesis consists of a list of figures, list of tables, introduction, four chapters, conclusion, and references list.

The main objective of the current document is to solve a real problem in the local academic structures. It involves processing the common semester timetables and generating individual schedules for each type of user. There are three types of users defined in the application: teachers, students and administrators. The activities for each of them are defined as follows: the administrators are in charge of introducing the timetables in the system at the beginning of a new semester. There are more administrators allowed into the system, so that each of them could be responsible for a certain amount of data to be inserted. The students access the application via a registration form, which allows logging in the system. During registration, the student will be asked to select the membership group, guarantying this way the generation of the proper timetable for he/she. Teachers accounts, on the other side, are created by the administrators for a good structuring of product's components.

As first attempt, the product is proposed as a prototype that would represents a small change for the academic infrastructures, while the results are consistently improved. The application was built to be easy to implement in the university system, reliable at use and implies minimal effort to maintain. The system is independent of universities, meaning that it can be used by any local university for its own purposes.

The four chapters that compose the current report include the analysis of the encountered problem, the possible solutions for it and some additional hints for new features implementation. There are several addressed points of view that describe the current proposed solution. A careful review was performed, in order to emphasize all the strong sides of the applications and the components that need additional attention. Also, considering the current functionality provided by the application, some future directions for development were established. In order to ensure a plausible business plan for the stage when the product is released on the market, an economical research was accomplished. Analyzing the product from economical point of view offered a general idea about several financial indicators, which are crucial for the lifetime of the application.

This document is intended for readers with technical background.

Rezumat

Teza de licență cu numele **Aplicație pentru monitorizarea individuală a orarului de studii**, prezentată de studenta Gușan Gina în calitate de proiect de Licență, a fost dezvoltată în cadrul Universității Tehnice din Moldova. Aceasta este scrisă în limba engleză și conține 67 pagini, 10 tabele, 21 figuri, 16 listări de cod și 13 referințe. Teza consistă dintr-o listă de figuri, o listă de tabele, introducere, patru capitole, concluzie și bibliografie.

Obiectivul principal al lucrării curente este să rezolve o problemă reală apărută în sistemul academic local. Aceasta implică procesarea orarului semestrial general și crearea unor orare individuale pentru fiecare tip de utilizator. În cadrul aplicației sunt definiți trei tipuri de utilizatori: profesori, studenți și administratori. Activitățile pentru fiecare în parte sunt definite astfel: administratorii sunt responsabili de introducerea orarelor în sistem la începutul fiecărui semestru. Aplicația permite existența mai multor administratori, astfel încît fiecare dintre ei să fie responsabili de un anumit volum de date introduse. Studenții accesează aplicația prin forma de înregistrare, care permite logarea în sistem. În timpul înregistrării, studentul va fi rugat să selecteze grupa sa de apartenență, garantînd astfel generarea corectă a orarului pentru el/ea. Din alt punct de vedere, conturile profesorilor sunt create de administratori pentru o mai bună structurare a componentelor ce fac parte din produs.

Ca o primă încercare, produsul este propus ca un prototip care ar reprezenta o schimbare minoră în infrastructura academică, pe cînd rezultatele ar fi în mod vizibil îmbunătățite. Aplicația a fost dezvolatată în așa mod încît să fie ușor implementabilă în sistemul universitar, sigură la utilizare, iar mentenanța - să fie ușor realizabilă. Sistemul este independent de universitate, acest lucru însemnînd că poate fi utilizat de orice universitate locală, pentru scopurile personale ale acesteia.

Cele patru capitole care compun raportul curent includ analiza problemei întîmpinate, soluții posibile pentru aceasta și cîteva sugestii adiționale pentru implementarea unor noi caracteristici ale aplicației. Deasemenea, aplicația este analizată din cîteva puncte de vedere distincte și concrete, menite să descrie soluția curent propusă. O revizie minuțioasă a fost îndeplinită pentru a sublinia toate punctele forte ale aplicației și, totodată, componentele care necesită atenție adițională. Deasemenea, luînd în considerație funcționalitatea curentă a aplicației, cîteva direcții pentru viitoarea dezvoltare au fost stabilite. Pentru a asigura un business plan plauzibil pentru momentul în care produsul va fi lansat pe piață, o investigare economică a fost realizată. Analizînd produsul din punct de vedere economic, a fost posibila structurarea unei idei generală asupra cîtorva indicatori financiari, care sunt cruciali pentru durata de viață a proiectului.

Acest document este destinat cititorilor specializați în domeniul tehnic.

Table of contents

List of Tables

List of Figures

Listings

Introduction

As a retrospective on student's life now and then, it becomes clear that things have changed during the time. Nowadays, living became comparable more intense than a couple of decades ago, generally thanks to the revolution of technologies. People are dynamic, innovations appear everyday and it is necessary to keep track on them. At a certain moment, all this agitation can result in a loss of control when speaking about the management of personal schedules. A common problem encountered in universities is the fact that students become easily distracted. During an usual academic day, a student has to face several courses with minimal breaks between them. Usually, in the attempt of focusing on other activities that result in more than the admitted limit of student's break time, it is not seized how the activities in cause are overlapping with the studying schedule. At this point, the student's productivity in the boundaries of the university become affected. So, here appears the need of having more control over the university's timetable by avoiding physical presence for checking courses status and replacing it with something else.

At the moment, the only schedule available for reminding which is the flow of the courses for a given day is usually situated in the main hall of the university and generally looks like a a printed version of an Excel document. This fact becomes quite inconvenient, since our country is making big efforts for growing up from manual systems to the computerized ones. As a quick fix for avoiding such a situation, it is necessary to make studying schedules easier to reach by developing an application that would guarantee access to it via smartphones. Also, given the fact that internet connection via Wi-Fi on the university's territories is a prerogative nowadays, a web application would be the most suitable solution.

It is worth mentioning that such systems have been developed and implemented by abroad countries for a good period of time already, so it's time for Republic of Moldova to make some changes in the current academic system, too. The truth is that the systems mentioned above are even more complex from functional and structural point of view. They are build as an entire component having automatic generation of studying schedule, detailed description of courses, online assignments, course materials and others. Such a platform is quite similar with the application known by local students as Moodle, just offering more flexibility and area of activity for its users.

The transition to such a complex platform could have unwanted consequences, that's why it's preferable to try some kind of prototype for a better observation and understanding of local student's needs. The product proposed in this document represents that prototype mentioned above, that has as main goal to test the current target group, which is represented by teachers and students and evaluate their attitude regarding the system. It is trivial to understand if the people involved are willing to accept this changes because that would eventually lead to a bigger step of introducing a complex platform, as one described above.

inClass is the product designed for solving a real problem in the local academic structures. It involves processing the common semester timetables and generating individual schedules for each type of user. There are three types of users defined in the application: teachers, students and administrators. The administrators are the persons responsible for introducing the raw data into the system, while the logic of the application is in charge with the rest. Given the fact that there are

more administrators allowed into the system, each of them would be responsible for a certain amount of data to be inserted. The system is independent of universities, meaning that it can be used by any local university for its own purposes. All the entities are created by administrators, so a lot of space is allowed to decide the format of usage for the application.

For students, to access the application, it is necessary to complete a registration form and log in with those credentials after the process is completed. During registration, the student will be asked to select the membership group, guarantying this way the generation of the proper timetable for he/she. Teachers accounts, on the other side, are created by the administrator in charge in order to avoid disorders in the system. After completing registration, the admin is responsible of securely hand out the credentials to the teacher for which the account was created. It is important to mention that each introduced course has a label specified, which is the teacher's name. So, the timetable for teachers is created depending on the courses having the proper labels. Also, it is well known the fact that one teacher can have courses with students from different faculties, that's why the product in discussion is set up to display the information about the faculties where courses take place, despite the other details. For general description purpose, the system is able to display customized information about courses, depending on the type of user. More particular features will be discussed in the next chapters. This was the general description of the product. Detailed information is available in the further pages.

The current thesis consists of 4 chapters. Chapter 1 explains the system analysis, together with the general overview about the final product. A market research has been done, in order to define advantages/disadvantages of using the application and provide a better understanding over the resulting system. Chapter 2 contains a more complex description of the system, from architectural point of view. The system is examined from different approaches, using UML diagrams. Chapter 3 contains a brief explanation regarding the technologies used for the implementation process and how those technologies were linked to the application. Also, the system's basic features are discussed from technical point of view, motivating the choice for that specific implementation. Chapter 4 represents a review over the economical position of the product. It is presented an estimation of costs for developing the project and some economical prediction for the future use.

1 Description and Analysis of the Domain

Nowadays, we are living in the era of technological progress, where machines get to do the most part of the jobs people once used to do entirely. It is vital to understand and sustain the every day life improvements human kind gets while creating software, instead of using the ancient approach of doing everything individually. Sooner people get that, faster they reach the point when machines are included into the daily routine to complete the tasks that, for a person, would take much more resources. It is the most efficient way for people to save time and money while introducing software for optimization purpose. This means that there is an worldwide tendency for developing smart systems which would eventually replace the unnecessary extra effort made by one or more people involved in a certain process by swapping it with functional products. Here we reach the moment where an informational system becomes more suitable for small, repetitive jobs, in order to ensure some more free hands to take care of complex tasks.

As mentioned above, passing from traditional methods of building environmental infrastructures to the computerized ones it is not an easy task and the approach has to be a well-defined from the start. Big changes don't always fit perfectly, bringing with them a lot of problems to deal with. That's why, it is important to determine from the beginning the correct manner for defining new standards. Locally speaking, it has been seized a continuously growing interest in determining people to work with software, which is encouraging the technological progress to spread evenly among all the population. The young generation deals better with this phenomenon, while a real challenge is faced to middle age generation. Still, this temporary phase shouldn't be treated as a problem, but as a matter of accommodation.

Diving deeper into the subject, the local educational system gets involved in the discussion. As a matter of fact, the academic structures should be the first ones involved in the transition from standard methodologies to the progressive ones. It is already observed a constant aspiration and sympathy for new technologies which implies the reorganization of the academic structures. This means that people already reached the moment when are ready to bring software products in universities in order to optimize the routine processes which they are responsible for accomplishing. This is a big step that has to be adjusted properly to the university's needs.

inClass is the software product that comes with a project in development and a prototype to fix an existing problem in universities. At the same time, it represents a small change for the academic infrastructure, given the fact that is easy to implement, use and maintain, while the results are consistently improved. So, such a product brings with it a lot of benefits for its users:

- Helps in improving user's time management;
- Brings additional flexibility by saving user's time during breaks;
- It is practical in use and easy to handle;
- Can be easily accessed only by having a smartphone and internet connection.

1.1 The workflow of the system

In order to be able to access the inClass application, a user should create an account first. Given the fact that there are two types of user, the registration modality differs for each of them. The registration for students is a traditional one, by reaching the domain of the application and request a new account. At the registration stage, the student will be asked to identify himself/herself by specifying their study group. The student will be able to select its own group, ensuring this way a proper generation of his/her schedule. After log in, the student will be able to visualize his/her current timetable, depending on the current day. The application compares the general current day from the set up date and matches it with the label "Day", that every course has specified at creation. As mentioned above, there is a third-party user, the administrator, whose task is to populate the database with records, which are the courses, in our case and take care of the maintenance of the database during the current semester. If any changes appear in the general timetable, the administrator is able to introduce all the changes by editing the courses in cause.

Speaking about the other type of user, the teacher, there is a different procedure defined for him/her in order to get an account. For security reasons, the accounts for teachers are created by the admin. He is the one that directly generates the teacher's accounts and is also responsible for the secure credentials sharing with the teachers in discussion. The decision of admin taking care of teacher's accounts was made assuming that the courses are linked with the teacher. Because of that, the courses' moment of creation is dependent on the teacher's moment of account creation. So, for avoiding other conflicts, the system was developed based on the assumption that the teachers already exist in the database when the courses are created.

Another moment that gives additional credit to the discussed product is the fact that inClass is a flexible application, developed for a general purpose. Meaning that, even if it was first designed to forehead a specific problem appeared at a specific university, it's still independent from usability point of view. So, if any other university is desiring to implement in their administrative infrastructure this product, there are no problems with that, because the application permits multiple records of different universities. The only moment that should be taken in consideration is the database capacity, that should have proper dimensions for data insertions. At this point, the database population is handled by systematic updates, which assume that at every semester end, all the records from the database are destroyed (because the timetables also change) and new records are created when a new semester begins. Assuming that the project is in its early development stage, there are no archives of old data in the system. so, once a course is not valid anymore, it is simply erased. Still, this can not be considered as a flaw of the system, since once passed a semester, the old record are of no actual use.

1.2 Similar products on the market

The idea of creating the discussed application came from the possibility of using and testing another existing product on the market during some time. Assuming that, in order to create a good software, it is necessary to make a market research and find similar solutions to the problems that are being solved by that software, making a connection between the original system and the missing one from the local academic infrastructure was quite obvious. The initial product, which served

as inspiration, was implemented as a general platform for one of the foreign faculties in Europe. inClass can be considered as a child of the original application because it covers only a part of its initial functionality. If to discuss the original platform, it covered almost completely the academic infrastructure of the mentioned faculty. Also, another thing that should be taken in consideration is the fact that the foreign faculty had different rules for timetable creation, implying student's decision of the courses to be attended. All the courses and their description was available on the platform and contained details about those courses like subjects to be discussed, practical assignments, all kinds of feedback, like received marks during the semester. So, once a student was decided about the courses to attend, the system generated his/her individual timetable, according to the subscriptions the student made.

Making a parallel to the local universities, students don't get the chance to choose what the want to learn from a proposed list of subject because the curriculum is already established by the Ministry of Education. That's why, transposing the original platform for the local needs, the task gets easier with the fixed curriculum. Yet, the rest of the application can be considered a prototype of the original system.

Another local example, that was already implemented in the academic structures is the Moodle learning platform. Since the beginning, Moodle was designed to provide educators, administrators and learners with a single robust, secure and integrated system to create personalized learning environments.Moodle is also web-based and so can be accessed from anywhere in the world. With a default mobile-compatible interface, the content on the Moodle platform is easily accessible and consistent across different web browsers and devices.It has a simple interface, drag-and-drop features and well-documented resources along with ongoing usability improvements. It is a good solution for course content integration, but not for dealing with course management. Still, the project has an open-source approach, meaning that it is continually being reviewed and improved on to suit the current and evolving needs of its users, [1].

For future implementation, using Moodle's API for integration with the currently discussed product would constitute an even more powerful tool for academic activities. It would cover all the basic functionality needed to overpass the obsolete current methodology for teaching and learning purposes.

1.3 The development environment

A framework is a program, set of programs and code library that writes most of the application for the developer. While using a framework, the main job is to write the parts of the application that make it do the specific things the developer wants. When writing a Rails application, leaving aside the configuration and other housekeeping chores,there are 3 primary tasks to perform:

– Describe and model the application's domain - The domain is the universe of the application. In this case, the domain represents an application that would allow the monitoring of personal study timetable. So here has to be established what's in it, what entities exist in this universe and how the items in it relate to each other. This is equivalent to modeling a database structure to keep the entities and their relationship.

– Specify what can happen in this domain - The domain model is static; it has to become dynamic. Students and teachers can register fin order to view their timetables. So, it is necessary to identify all the possible scenarios or actions that the elements of the domain can participate in.

– Choose and design the publicly available views of the domain − At this point, thinking in Web-browser terms can start. Once it's decided that the domain has students and teachers, and that they can register for viewing timetables, some components like a home page, a registration page, and a log in page can be envisioned. Each of these pages, or views, shows the user how things stand at a certain point.

Based on the above three tasks to be accomplished, Ruby on Rails framework was chosen for the development of the discussed application, assuming that it suites the best from implementation point of view.

1.3.1 Back-end implementation: Ruby on Rails

For implementing the idea of the inClass project, Ruby language was chosen, Ruby being a general-purpose programming language, best known for its use in web programming. It is known among programmers for a brief, uncluttered syntax that doesn't require a lot of extra punctuation. Compared to Java, Ruby is streamlined, with less code required to create basic structures such as data fields. Ruby's key advantage is RubyGems, the package manager that makes it easy to create software libraries (gems) that extend Ruby. RubyGems provides a simple system to install gems. Anyone can upload a gem to the central RubyGems website, making the gem immediately available for installation and building complex websites by anyone, [2].

The application was built on Rails framework,which is also written in Ruby and is a web application framework. Rails is a MVC framework, providing default structures for a database, a web service, and web pages. It encourages and facilitates the use of HTML, CSS and JavaScript for display and user interfacing. In a default configuration, a model in the Ruby on Rails framework maps to a table in a database and to a Ruby file. For example, the model class Student is defined in the file 'student.rb' in the app/models directory, and linked to the table 'students' in the database.

A controller is a server-side component of Rails that responds to external requests from the web server to the application, by determining which view file to render. The controller may also have to query one or more models directly for information and pass these on to the view. A controller may provide one or more actions. In Ruby on Rails, an action is typically a basic unit that describes how to respond to a specific external web-browser request. Also for the controller/action to be accessible for external web requests, it is necessary a corresponding route to be mapped to it. A view in the default configuration of Rails is an .erb file, evaluated and converted to HTML at run-time, [3].

inClass application was built on Ruby on Rails because this environments offers all the suitable components and tools for achieving an optimal result in a minimal period of time. Several RubyGems were used in order to reach additional functionality of the final product. One of the main RubyGems used in the project was Devise, which is a flexible authentication component for Rails, bringing a

complete MVC solution based on Rails engine, allowing to have multiple models signed in at the same time. Having the possibility to use the RubyGems made possible the inclusion of Hierapolis theme as front-end on the administration side of the application.

1.3.2 Model-View-Controller architecture inside Rails

The MVC principle divides the work of an application into three separate, but closely cooperating subsystems. In Figure 1.1 is represented the schematic diagram of Rails components interacting according to the MVC pattern:

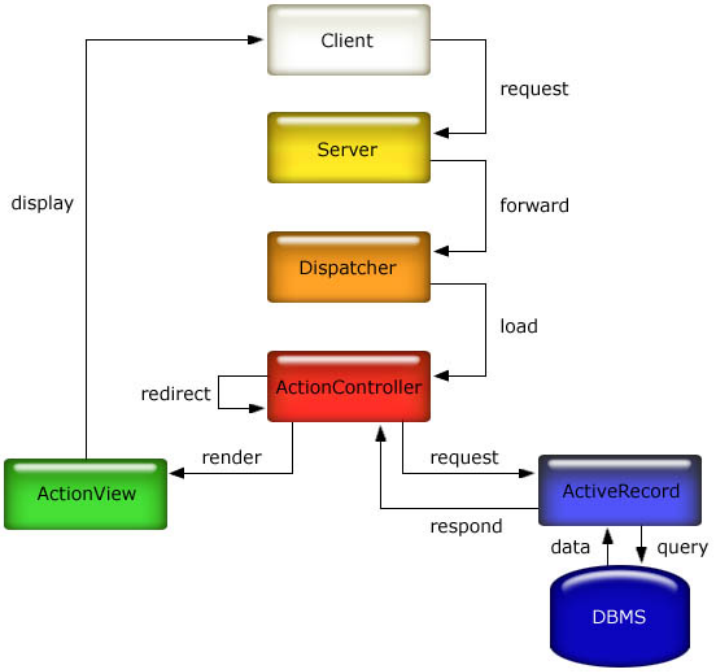

Figure 1.1– MVC pattern integration with Rails, [4]

The Model maintains the relationship between the objects and the database, while handling validation, association, transactions, and more. This subsystem is implemented in Active Record library, which provides an interface and binding between the tables in a relational database and the Ruby program code that manipulates database records. Ruby method names are automatically generated from the field names of database tables.

The View is a presentation of data in a particular format, triggered by a controller's decision to present the data. They are script-based template systems, which are easy to integrate with JavaScript technology. This subsystem is implemented in Action View library, which is an Embedded Ruby (ERb) based system for defining presentation templates for data presentation. Every web connection to a Rails application results in the displaying of a view.

The Controller is the facility within the application that directs traffic, on the one hand, querying the models for specific data and on the other hand, organizing that data (searching, sorting, messaging it) into a form that fits the needs of a given view. This subsystem is implemented in Action Controller, which is a data broker sitting between Active Record (the database interface) and Action View (the presentation engine), [5].

1.3.3 Handling Models with Active Record

Active Record is the **M** in MVC - the model - which is the layer of the system responsible for representing business data and logic. Active Record facilitates the creation and use of business objects whose data requires persistent storage to a database. It is an implementation of the Active Record pattern which itself is a description of an Object Relational Mapping system, [6].

Object-Relational Mapping, commonly referred to as its abbreviation ORM, is a technique that connects the rich objects of an application to tables in a relational database management system. Using ORM, the properties and relationships of the objects in an application can be easily stored and retrieved from a database without writing SQL statements directly and with less overall database access code[7].

Active Record gives several mechanisms, the most important being the ability to:

- Represent models and their data;
- Represent associations between these models;
- Represent inheritance hierarchies through related models;
- Validate models before they get persisted to the database;
- Perform database operations in an object-oriented fashion, [8].

When writing applications using other programming languages or frameworks, it may be necessary to write a lot of configuration code. This is particularly true for ORM frameworks in general. However, if the conventions adopted by Rails are followed, there is no need to write configuration when creating Active Record models. The idea is that if the application is configured in the very same way most of the time, then this should be the default way. Thus, explicit configuration would be needed only in those cases where a developer can't follow the standard convention, [9].

1.3.4 Front-end implementation: Bootstrap, CSS and JavaScript

For the front-end development of the application, Bootstrap was the main tool to be used. Bootstrap is a free and open-source front-end web framework for designing websites and web applications. It contains HTML and CSS based design templates for typography, forms, buttons, navigation and other interface components, as well as JavaScript extensions. Bootstrap includes a responsive, mobile first fluid grid system that appropriately scales up to 12 columns as the device or viewport size increases. It includes predefined classes for easy layout options, as well as powerful mixins for generating more semantic layouts. In addition to the regular HTML elements, Bootstrap contains other commonly used interface elements. These include buttons with advanced features (grouping of buttons or buttons with drop-down option, horizontal and vertical tabs, navigation, pagination, etc.), labels, advanced typographic capabilities, thumbnails, warning messages and a progress bar. The components are implemented as CSS classes, which must be applied to certain HTML elements in a page, [10].

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language. Although, most often, it is used to set the visual style of web pages written in HTML, for the inClass project, the CSS files were related to files written in HAML (a revised version of HTML). CSS is designed primarily to enable the separation of document content from document presentation, including aspects such as the layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics and reduce complexity and repetition in the structural content, [11].

Using sprinkles of JavaScript throughout a Rails application is almost inevitable if it is necessary to isolate code to only run specified pages. JavaScript can also make requests to the server, and parse the response. It also has the ability to update information on the page. Combining these two powers, a JavaScript writer can make a web page that can update just parts of itself, without needing to get the full page data from the server, [12]. This is exactly the case where JavaScript code was used in the currently discussed application.

2 Analysis of the application from architectural point of view

Modeling an information system implies a variety of activities, who have as primary descriptor the parallel handling of large information volumes. Earlier, in the requirements phase, it was built a conceptual model of the problem that identifies what objects or entities are involved, what they look like (by defining their attributes), and how they relate to one another. Now, it is the moment to describe the entire software structure, including all the interfaces, classes, interaction between users and system and more important - the interaction between system components. In order to characterize the inClass system, a collection of models were used:

- Use Case model. It is the representation of a user's interaction with the implemented system. Also it shows the different types of users and the various ways of interaction with the designed system.
- Activity model. This type of models are aimed to display the overall flow of control in system, by modeling the computational and organizational processes (work-flows).
- Sequence model. This is an interaction diagram, that shows how processes operate with one another and in what order. It is a collection of interactions arranged in time sequence.
- Database model. It is a static structure diagram, which emphases which are the entities where the data are stored in the system and what type of relations exist between them. The same diagram contains all the attributes of each table and the connections between them (primary and foreign keys).
- Deployment model. It shows what hardware and software components were used to setup the whole environment for designed system, [13].

This is the chapter where the system is analyzed using the above enumerated types of diagrams, in order to get a more clear image about the system and its components interaction. Each diagram is designed to face the product from a concrete point of view, so that it becomes straightforward how the system is built, works and which is the approach for future development.

2.1 System requirements

Speaking about the life cycle of a program, one of the basic levels to be taken into account is requirements specification towards the system. Covering this point helps the developer to determine which functionality the system is going to have for the future implementation. Usually, the requirements suffer changes during the development period, still it is vital to establish the basic ones since the beginning. As a matter of fact, versioning the system happens because testing the initial one rises some question marks, so the next product version comes with some revised requirements. One of the most important things that the discussed application should be able to do is displaying the current academic timetables for a given type of user. Also, the veracity of the data should be taken into account, because the system needs maintenance with database records being in conformity with the current academic situation.

Another moment of attention should be paid to the chosen framework and language for development of the system. Considering the environmental conditions, the best option would be creating a web application, assuming that the main target dispose of the necessary instruments for reaching the application: internet connection and suitable mobile phones. As a generalization, the system is expected to accomplish the following requirements:

- Allow different account types;
- Create access boundaries for each type of user;
- Create a database model that emphases the encountered real-life situation;
- Permit creation of different types of entities, according to the database model;
- Limit the scenario of account creation for teachers by delegating the task to the admin;
- Allow student registration by invoking the specification of group of appurtenance
- Permit editing current existing entities;
- Display current timetable depending on individual profile;
- Filter the courses by the corresponding labels for each user;
- Display data according to current day of the week for both teacher and student;
- Admit only one logged user per computer;
- Visualize statistical information about user's activity for admins.

This are the first stage requirements established for the application, with possible future modification depending on the development directions. Future possible requirements can be linked by integration using the API of an already existing platform for enlarging the offered services. More specific details would imply courses visualization, together with the possibility of online access to regular tests or exams and grades assignment directly on the platform. Other directions for development could imply developing student's and teacher's profiles by creating additional features for them.

At this moment the two types of users are limited only to data visualization. Creating additional options like daily reviews on the academic situation, some time management tools for course tasks organizing, interaction between the two types of users via announcements page would be some of the additional requirements for the second version of the system.

2.2 Representation of the system via Use Case Diagrams

The first diagram to be discussed presents the interaction of the administrator as actor in the system. For parallel visual observation, see Figure 2.1. All the Use Case diagrams have as prerequisite the fact that the user, no matter which type is it, must be logged in. If there is no account created,

registration is required. The primal activity the administrator can perform with a fresh database is to create one of the global entities of the system, a new university. In order to create one, it is necessary to access the "Universities" option in the menu. After clicking on it, the page containing all the available universities will be rendered. At the moment, there are no universities in the system, that's why the list is empty, Besides the list of universities, there is an "Add University" button, which suggestively exhorts the user to click it. After clicking, the form for creating new universities will be rendered bellow. This particular form contains the Edit Box, where the name of the university should be typed in and the "Submit" button, for accomplishing the procedure. Basically, this is how the "Create a new University" use case can be described.

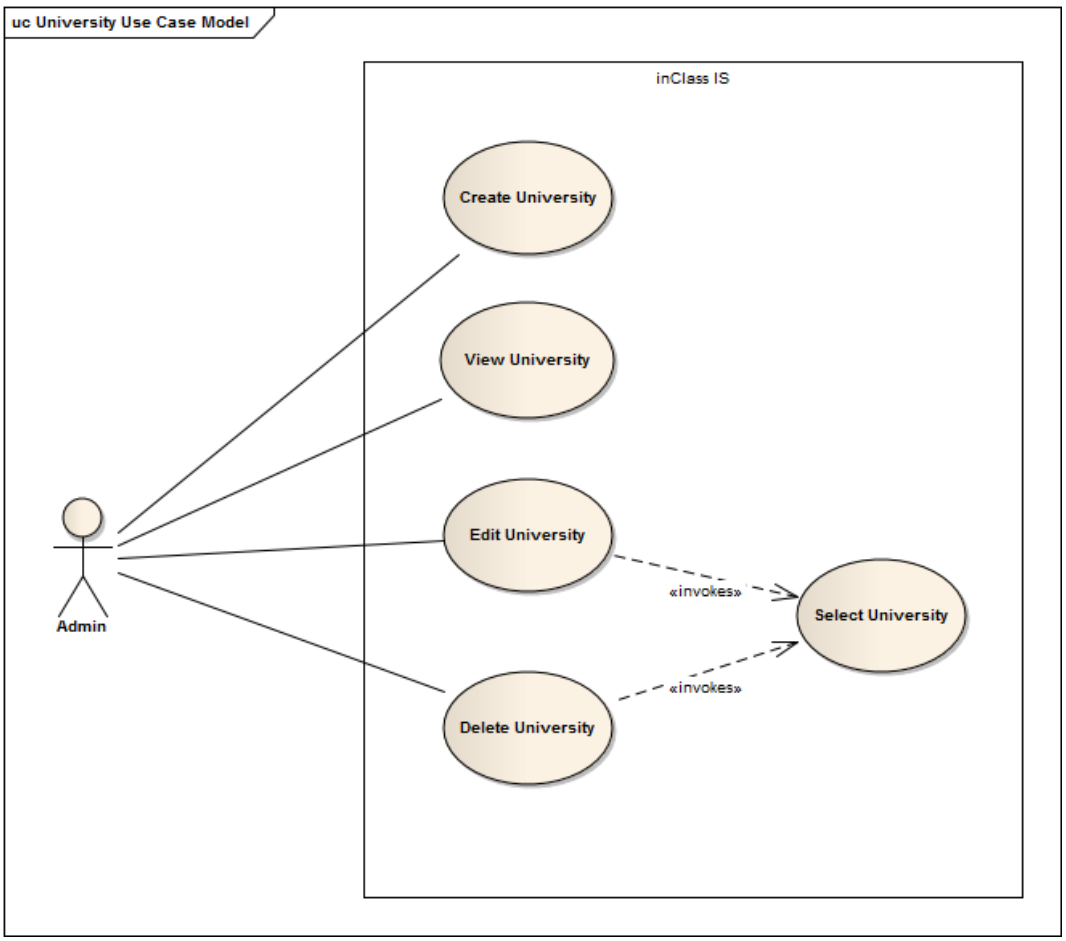

Figure 2.1– Use case diagram for Universities

After submitting the form, the university becomes immediately visible on the same page, in the mentioned list of universities. Besides the name of the university, on the right side, two buttons will become available for each new created entity. One of the buttons is for editing the current name of the selected university, the other one is for deletion. As a remark, should be mentioned that, as global entities in the database, deleting a university implies all its content being recursively destroyed from the database. In order to edit the University, it is sufficient to click on the "Edit University" button and the edit form will be rendered. The form contains an Edit Box with current name in it and the "Submit" button. The name of the university becomes accessible for retyping and pressing "Submit" button will make the final changes in the database record.

The next diagram shows the interaction between the same administrator with the system, by creating another type of entity. Assuming that at least one university already exists in the database, now it is possible to add components to it. For that, it is necessary to click on the University title for entering the general University page. It is quite similar with the index page of the universities, the only difference being the fact that here two types of components can be created: new faculties and new teachers. The Figure 2.2 shows the available use cases for dealing with faculties in the system. In order to create a new faculty, the admin has to enter the university page first. As no records are available at the moment, 2 empty lists will be displayed. The first one, visualizing faculties comes with an "Add faculty" button on the top. Clicking on the button will render the form for creating new faculties, which is similar with the one for creating universities. It also contains the Edit Box, where the name of the faculty should be typed in and the "Submit" button for data salvation.

Adding new faculties will populate the list from the same page, so each of the new entries has the options foe editing and deletion available. For editing the faculty, it is necessary to click the "Edit Faculty" button and rewrite data from the rendered form. By clicking the "Delete Faculty" button, the record will be permanently erased from the database, and recursively will be removed all the child components from the respective faculty. For visualization, there are two available options for this type of entities. One of them is via access from the "Universities" option in the menu, which ensures the user also with data manipulation (edit, delete), while accessing the "Faculties" option from the menu will give access only to visualize the existing data, ordered chronologically, depending on the last updates made to the faculties.

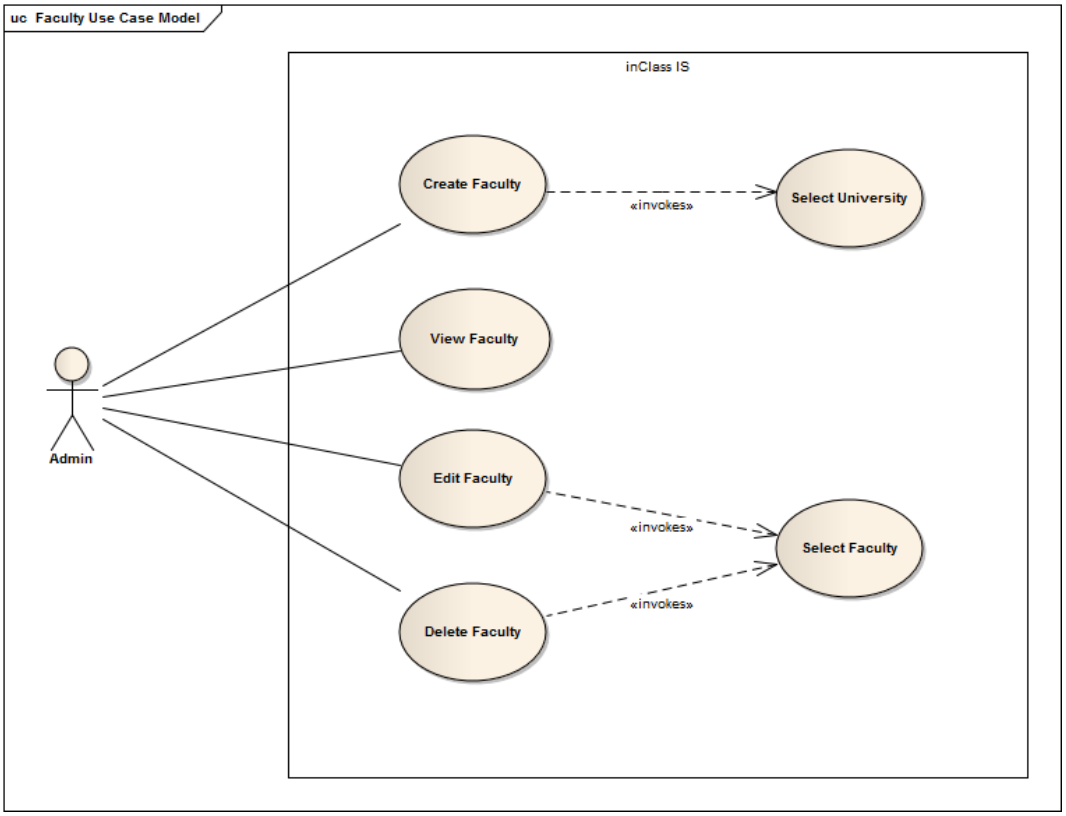

Figure 2.2– Use case diagram for Faculties

Another illustration of a use case diagram is bringing the timetable model as one of the basic components of the system. It is created with the purpose of structuring the courses after a certain label. The Timetable is a child component of the Group. For every group, a single timetable can be created and it reflects the current ongoing semester. This means, that after each semester, when the timetables change, it is necessary to update the records by removing the current timetable and add a new one. The database is restricted to keeping one timetable per group, that's why creating a new timetable differs from the above described activities. In order to create a new timetable, a Group show page should be reached. Here, the standard "Add timetable" button is available. By clicking on it, the form for creating a new timetable is rendered. It contains a selection field, where only one option is available, the current semester for the group it belongs. By clicking the "Submit button", the record will be added in the database. This entities also have attached edit and delete buttons. The only difference is the editing process, that makes possible the partial update of the timetable's name by updating the name of the implicated group, in case that group suffered prior changes. Also, at this point, 2 more actors appear into the scenario for data visualization.More details will be discussed in the next paragraph. The Figure 2.3 presents the diagram discussed above:

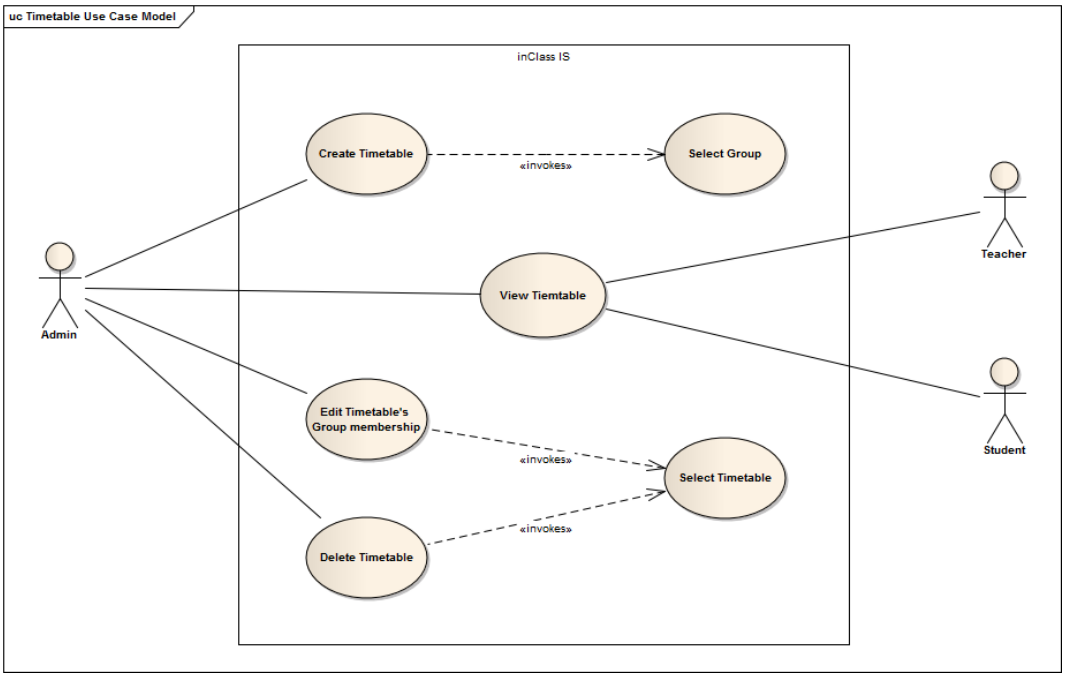

Figure 2.3– Use case diagram for Timetables

Reaching this point means that, in general lines, the system was completely analyzed via use case diagrams. The courses are the most low-level entities, which are basically the reason why the inClass application was developed. For parallel observation, see the Figure 2.4. In order to be able to create a new course, a timetable show page should be involved. After entering it, the same template is used. An "Add Course" button is immediately available on the top of the page and by clicking it, the application redirects to a specific form for course creation. It contains several fields like course name, starting hour, teacher who is in charge for it, room where it takes place and day when it happens. A part of the fields are rendered as Edit Boxes and the admin is free to enter whatever data he/she wants and there are some selection fields, where the options are limited. After all the

fields are completed, the "Submit" button saves the new record into the database.

For every day of the week, a different table is generated. Also, the tables are ordered, starting with the first academic day of the week, Monday and finishing with the last one - Friday. Depending on the specified starting hour of the course, the records in the table are also sorted in ascending order. When the courses are created, three types of users in the system can visualize these data. The only difference is the type of accounts from which the courses are accessed and displayed. For editing the courses only the admin has permission, and the process flow is similar with the other discussed ones. For every course, edit and delete buttons are available and clicking those buttons have the same outcome as the ones previously addressed.

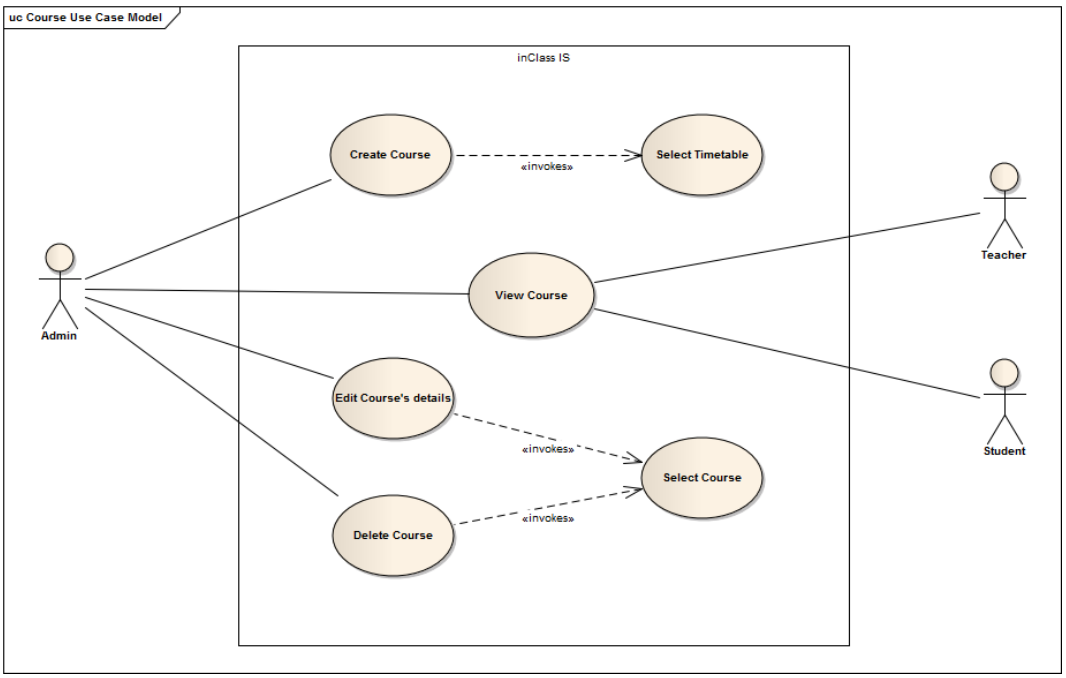

Figure 2.4– Use case diagram for Courses

2.3 System Analysis using Activity Diagrams

In the figure Figure 2.5, illustrated bellow, is represented the flow of steps needed to be done in order to add a new Group record into the database. The user who has the rights to perform such operations in the system is the administrator. The first thing to be done in his/her case is to log-in. After reaching the application, the admin is ready to start the interaction with the system. To achieve the proposed goal, several primal entities need to be created or checked if exist. In this particular case, it is assumed the entities are missing and have to be created first. The process begins with accessing the "Universities" option from the menu. It is clear that the university that would contain such a group doesn't exist, so it must be added.

The admin clicks on "Add University" button and starts introducing the name of the new recorded university. After submission, the new university becomes available as an element in the current list of universities in case the process was successfully accomplished. If not, an error message is displayed and the whole process starts from the beginning. Assuming the new record was created, clicking on the name of the university redirects the admin to the page where is shown the content of the current university. At the moment, the content is empty, so the admin proceeds to creating a new

faculty. The process for adding a faculty is equivalent with the one of creating an university. After reaching the show page of the faculty, the admin is ready to create a new specialty. The process, one more time, coincides with the ones discussed above. At last, having a new specialty created, makes possible the event of adding the new group. Again, the admin presses on the "Add Group" button and enters the name of the group into the rendered form. If the submission of the new record is positive, the activity flow ends successfully. If something went wrong, an error message is displayed and the admin is one more time redirected to the specialty's show page. Here, the creation process starts all over.

Another interesting situation appears when the administrator tries to create a new timetable, but is not sure if the group for which the timetable should be created exists. In this case, he/she has two options. For time minimization purpose, the admin accesses, first of all, the "Groups" option in the menu. He/she navigates through the list of available groups and tries to find the one needed. If the group exists, the admin simply enters it and adds the new timetable by the same procedure discussed till now. If the group doesn't exist, a different path has to be covered. The admin chooses the option "Universities" from menu and starts navigating through faculties and specialties until reaches the list of all available groups. Here he/she adds a new one. Assuming the creation was successful, now it is possible to enter the group's show page. Here, the process of adding a new timetable is performed by accessing the form for creating new records of this type. If the process ends correctly, the new timetable becomes ready to be used. If not, the cyclic flow of steps are resumed. In the Figure 2.6 the activity flow is represented graphically.

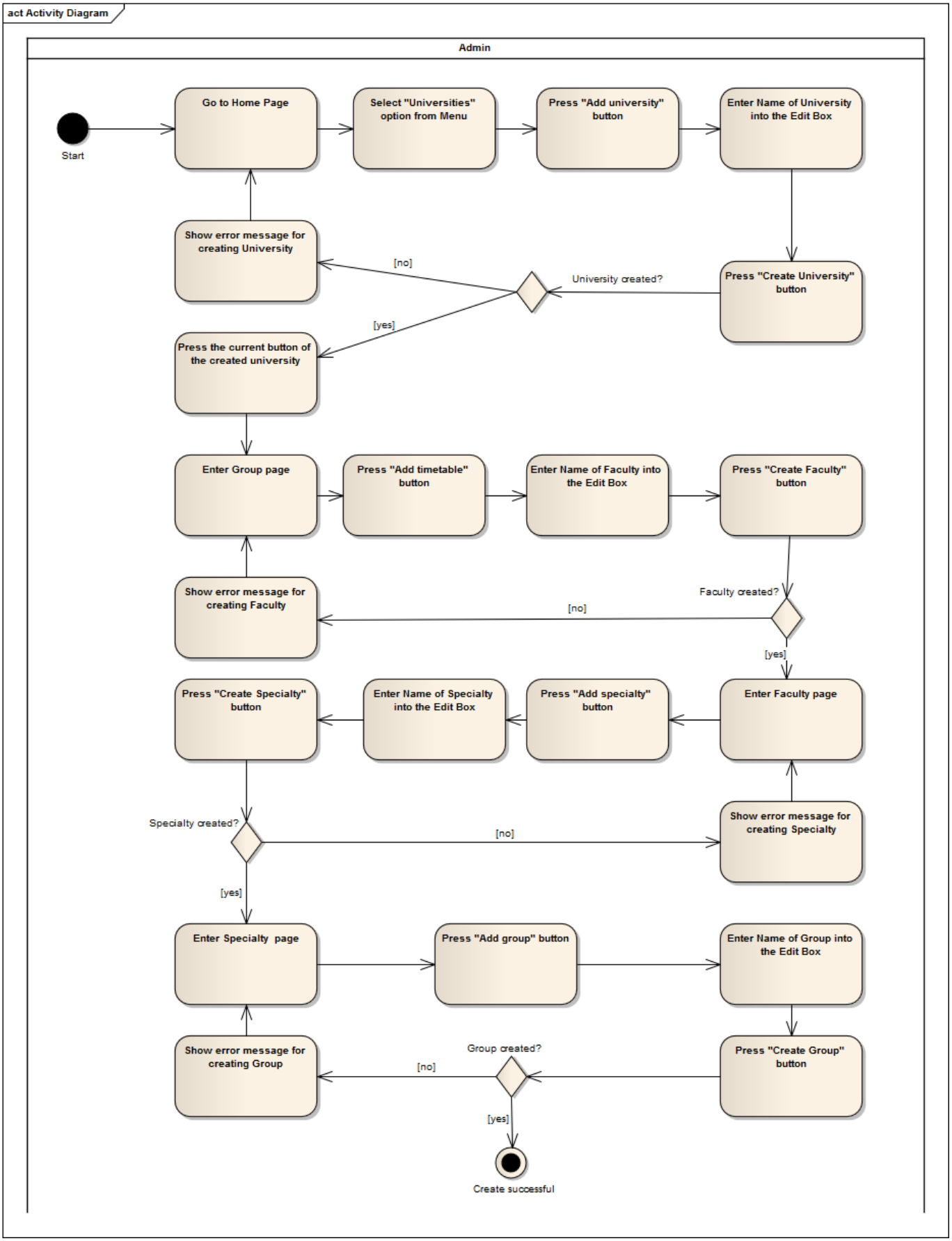

Figure 2.5 – Activity diagram for adding new Group

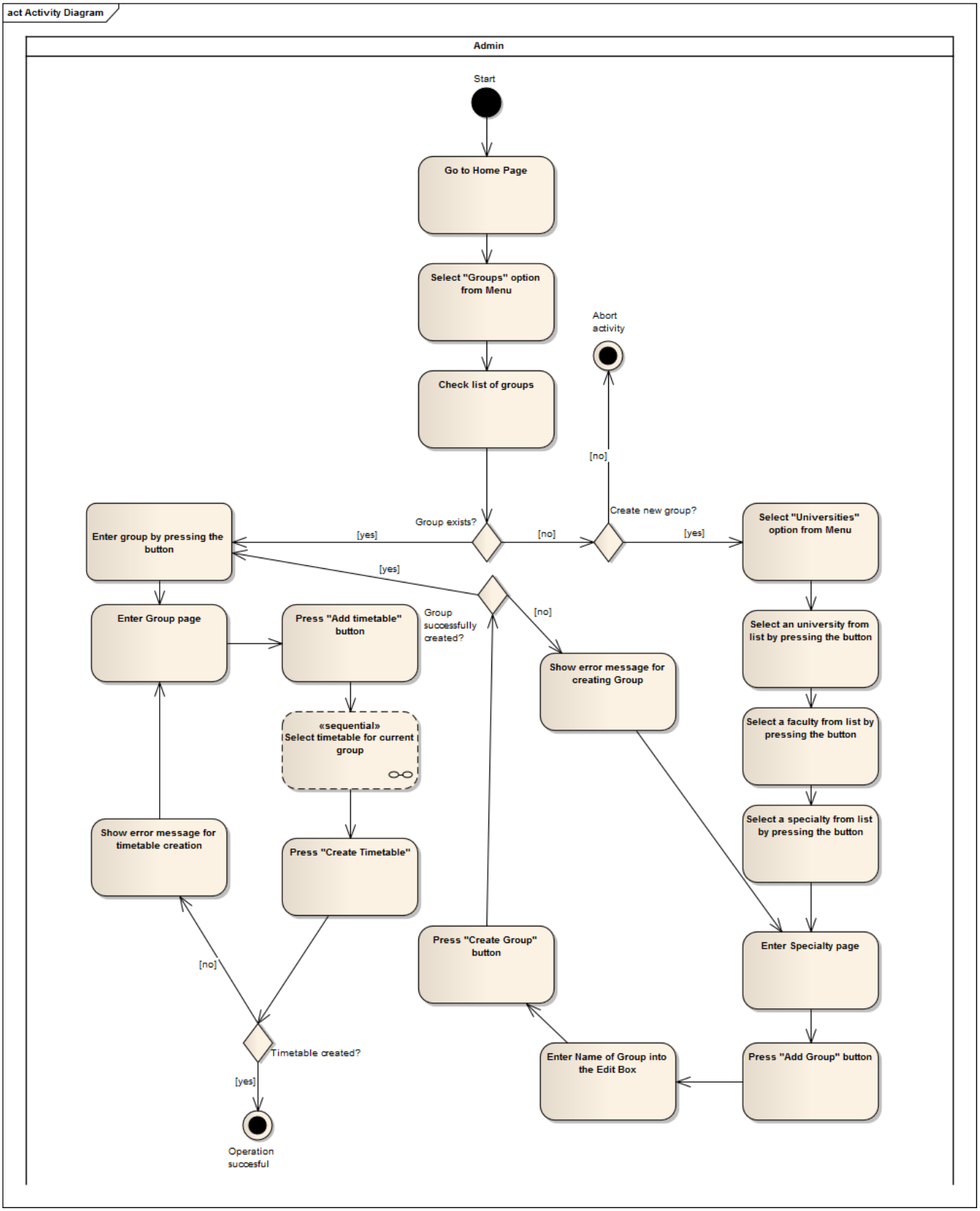

Figure 2.6– Activity diagram for adding new Timetable

2.4 Process interaction analysis using Sequence Diagram

The sequence diagram is the one that shows how the smallest structural components, the processes, interact one with each other. As example, the action of editing an existing faculty was represented using this type of diagrams. The Figure 2.7 illustrates the UML description of the proposed activity, using the Enterprise Architect environment.

So, in order to edit a faculty, it is necessary to access the application, go to the show page of a certain university and click on the "Edit faculty" button. What happens next is the loading of the corresponding view, passing as parameters the `university_id` of the faculty and its own id. In Ruby, the forms are used just for representational and structural purpose, while the controllers are the entities that truly own the data.

Obviously, the view now calls the `Faculties Controller` with the `edit` method. In the controller, first, a call to the `set_university` method is made in order to find the parent university of the implicated faculty. The `find` function returns the `University` instance from the class, which has the specified `university_id`. Now, in a local variable `faculty`, is stored the `Faculty` instance found in the `University` instance according to the `faculty_id`. This local variable is then passed over to the view and the `edit` page is rendered, after the page is reloaded. Once the form is rendered, the updated name of the faculty is typed in the Edit Box.

When the `Submit` button is clicked, the application passes the new `params` through the view and back to controller, where it calls the `update` method. This method finds the `Faculty` object by its id and stores it locally. After that, a check is preformed (by calling update method) for establishing if the new changes were applied having the `faculty_params`. A boolean is returned as result which renders the university's home page if `true`, or renders the `edit` form again, if `false`.

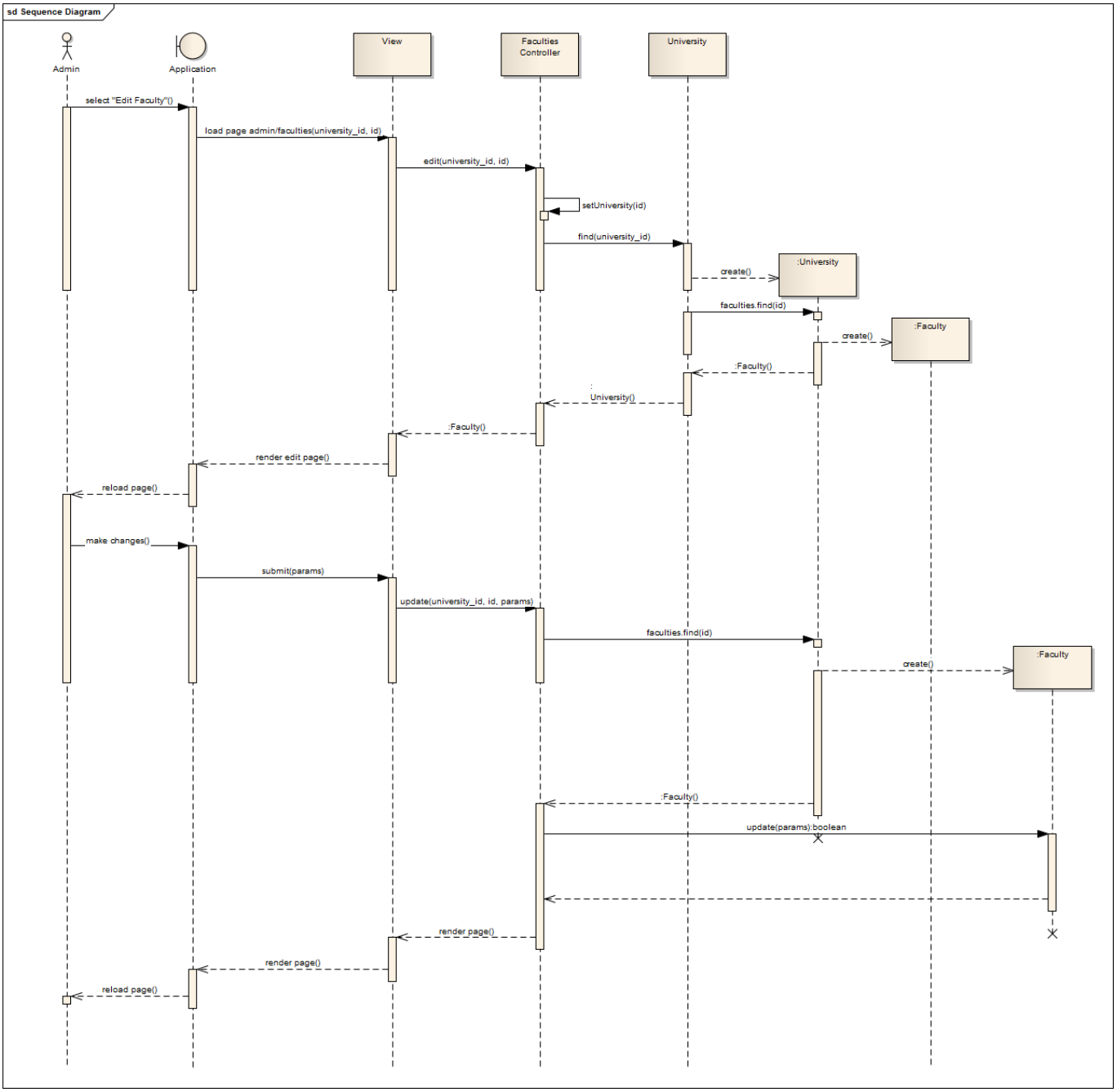

Figure 2.7– Sequence Diagram for editing an existing Faculty

2.5 The Database Model of the system

Modeling how a system should store data is one of the primary steps to be done in the designing stage. Having a proper configuration of the database model guarantees an effective development of the required functionality. In Figure 2.8 is represented the database model which stands on the base of the system in discussion. Having defined the purpose of the application and the limits of functions it provides, it is mandatory to define the models that will store the data and the relationships between them.

After analyzing the components of the system, it is clear that they have a tree structure. This means that the most general node is the University table. It contains several attributes like `title`, `creation date`, etc. and has as primary key the id it receives at creation. Following the Ruby convention, the `id` field is added automatically and is hidden in the schema. Still, it is there and developers know about that and this is the reason why all the tables have as primary keys these hidden fields.

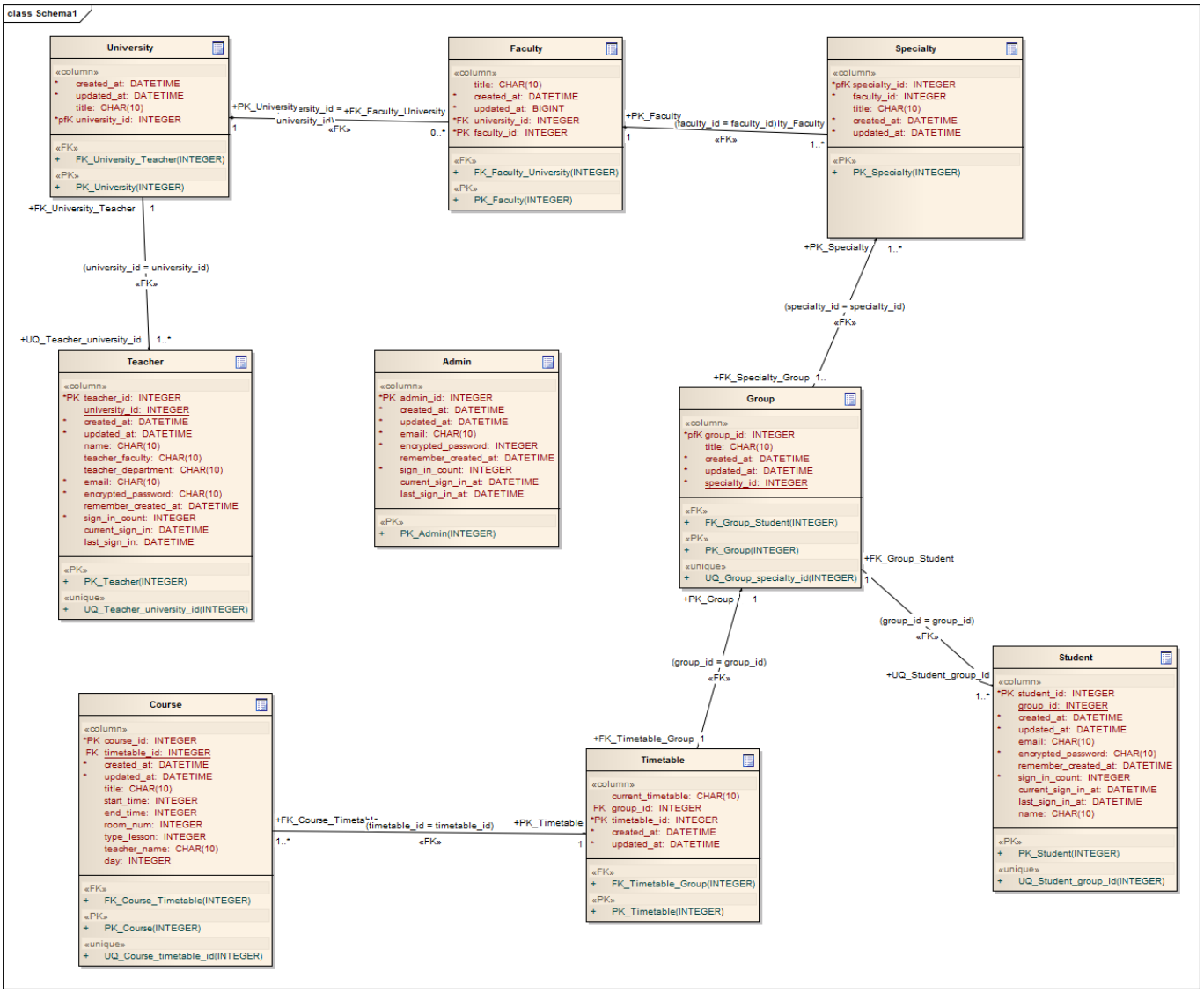

Figure 2.8 – Database model of the application

Now, applying the real - life model into the database structure, a university can have one or more faculties, meaning that the faculty has as foreign key the `university_id`, in order to create a relation between the two tables. The University is also linked via `university_id` with the Teacher table, as the teachers are treated as entities belonging to a certain university. This fact was decided so, assuming that a teacher may teach courses to different faculties. Similarly, assuming that a faculty can have one or more specialties, a `faculty_id` attribute is specified in the Specialty table, as foreign key to the Faculty table. Now, a relation between a specialty and a group is established, assuming that a specialty has many groups and a group can belong just to one specialty. The `specialty_id` attribute is contained in the Group table as foreign key to Specialty table.

From architectural point of view, it was established that the first version of the current application will contain only one timetable, reflecting the ongoing semester. That's why the Group can have only one Semester and the Semester can belong only to one Group. The `group_id` foreign key is contained in the Group table. Also a group should have one or more students, so another relation is established, the `group_id` being present in the Student table also as foreign key. Finally, every timetable should contain courses, so the Courses table is linked to Timetable via the `timetable_id` foreign key.

2.6 Final stage of product development: Deployment Diagram

At the deployment stage, it is assumed that the product is finished from development point of view and can be launched in production. Th Figure 2.9 materializes the components needed for release and the interaction between them that would guarantee a good functioning of the application as a complex system.

The general deployment process consists of several interrelated activities with possible transitions between them. It includes all the operations to prepare a system for assembly and transfer to the customer site. The resources required for deploying the current Ruby application involve 3 types of servers: an web server, an application server and a database server. The NginX web server was chosen for establishing an HTTPS connection with the PC via an web browser. Nginx uses an asynchronous event-driven approach to handling requests with modular event-driven architecture, providing more predictable performance under high loads. For the application server, Unicorn fits the best, assuming that is compatible with Ruby versions starting from 1.9.3 and is designed to serve fast clients on low-latency, high-bandwidth connections and take advantage of features in Unix/Unix-like kernels. Unicorn does not care if the application is thread-safe or not, workers all run within their own isolated address space and only serve one client at a time for maximum robustness. Combined with nginx and Ruby interpreter, any binary upgrades are possible without losing connections or dropping clients.

The database server was chosen to be MySQL, which is the world's most popular open source relational database management system. It is an engine that effectively helps deliver high performance to scalable database applications, which is expected the current application to become, once it's released. In time, most probably, the application is going to operate with parallel database servers, to improve performance of loading data, building indexes and evaluating queries.

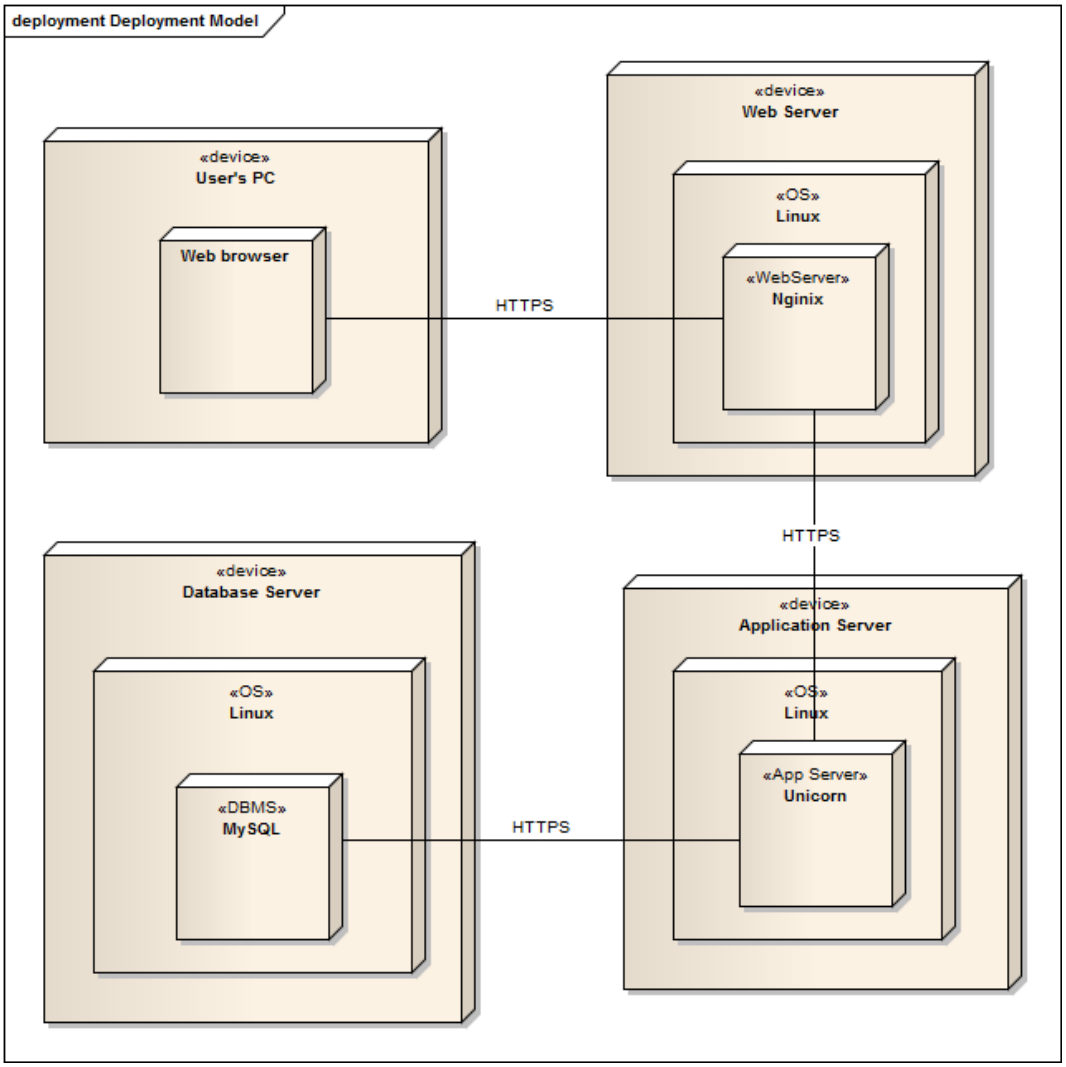

Figure 2.9– Deployment model of the system

3 Implementation and development methodologies

In this chapter, the inClass application is analyzed from the implementation point of view. As a quick remind, some general facts about the Rails framework will be mentioned, in order to understand better the listings presented bellow. Remember that Rails follows the MVC architectural pattern, which enforces a separation between "domain logic" (also called "business logic") from the input and presentation logic, associated with a graphical user interface (GUI). In the case of web applications, the "domain logic" typically consists of data models for things like users, articles, and products, and the GUI is just a web page in a web browser.

When interacting with a Rails application, a browser sends a request, which is received by a web server and passed on to a Rails controller, which is in charge of what to do next. In some cases, the controller will immediately render a view, which is a template that gets converted to HTML and sent back to the browser. More commonly for dynamic sites, the controller interacts with a model, which is a Ruby object that represents an element of the site (such as a user) and is in charge of communicating with the database. After invoking the model, the controller then renders the view and returns the complete web page to the browser as HTML. This is the complete interaction process between Rails components. Now, it is the moment to get started with some details about how the functionality of inClass application was implemented.

3.1 Project configuration

Assuming that the Ruby on Rails environment was already installed on the local machine, creating a new project is done by typing the command from Listing 1 in the Terminal:

```
$ rails new student
```

Listing 1 – Create new Rails project

This command creates a Rails application, called Student in a `student` directory and install the gem dependencies in `Gemfile`. The `student` directory has a number of auto-generated files and folders that make up the structure of a Rails application. Most of the work happened in the `app` folder. Table 3.1 presents a basic overview of each of the files and folders that Rails creates by default and their functions.

At this moment, a functional Rails application, called `Student` already exists on the local computer. To access it, it is necessary to start a web server on the development machine. This can be done by running the following command in the Terminal pointing to the `student` directory:

```
$ bin/rails server
```

Listing 2 – Navigate to localhost

Table 3.1– Auto-generated files in a Rails project

File/Folder	Purpose
app/	Contains controllers, models, views, helpers, mailers and assets for the application.
bin/	Contains the rails script that starts the app.
config/	Configurations for the application's routes, database, etc.
config.ru	Rack configuration for Rack based servers used to start the application.
db/	Contains the current database schema, as well as the database migrations.
Gemfile	Allows to specify what gem dependencies are needed for the Rails application.
lib/	Extended modules for the application.
log/	Application log files.
public/	The only folder seen by the world as-is. Contains static files and compiled assets.
Rakefile	This file locates and loads tasks that can be run from the command line.
test/	Unit tests, fixtures, and other test tools.
tmp/	Temporary files (like cache, pid, and session files).
vendor/	A place for all third-party code (vendored gems).

To see the application in action, it is needed to navigate to `http://localhost:3000` via a web browser. Given the fact that the project is still empty, the Rails default information page is displayed. The Figure 3.1 illustrates how the page actually looks like:

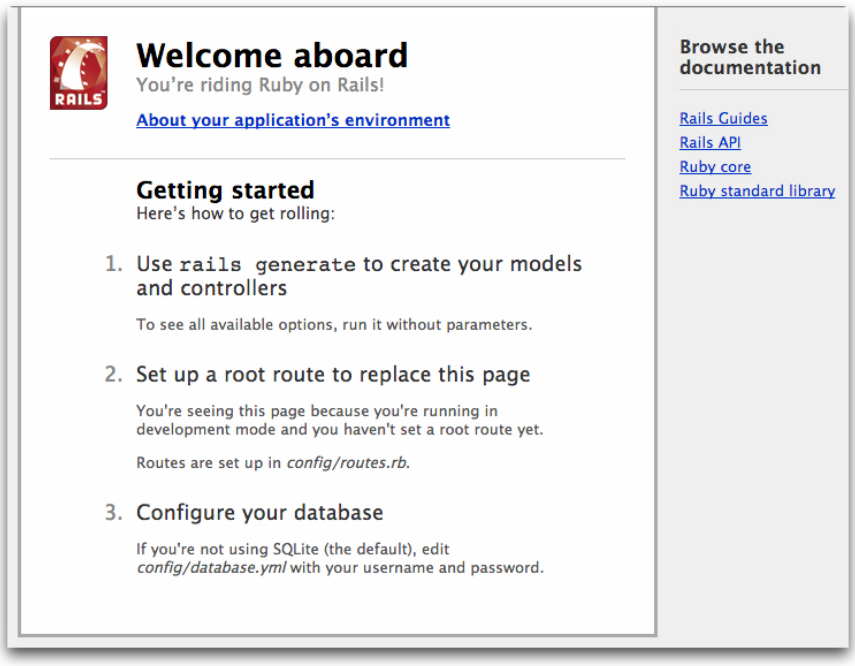

Figure 3.1– Rails Welcome page

The "Welcome aboard" page is the basic test for a new Rails application: it makes sure that the development software is configured correctly enough to serve a page. Hitting `Ctrl+C` in the terminal window where the server runs will stop the web server.

3.2 Back-end implementation of the application

To get Rails store, display and change data, it is needed to create minimum a model, a controller and a view. As example will serve the development of the system's `University` component. Given the fact that this entity is supposed to allow storing instances, a `University` model should be created. To do this, the command in Listing 3 has to be executed in Terminal:

```
bin/rails generate model University
```

Listing 3 – Generate model University

According to Active Record conventions, models in Rails use a singular name, and their corresponding database tables use a plural name. After running `bin/rails generate model`, a database migration file will be created inside the `db/migrate` directory. Migrations are Ruby classes that are designed to make it simple to create and modify database tables. Rails uses rake commands to run migrations, and it's possible to undo a migration after it's been applied to a database. Migration filenames include a timestamp to ensure that they're processed in the order that they were created.

The Listing 4 offers a look inside the recently generated migration. It can be observed that a series of attributes were added to the `universities` table in the database, which is mapped to the `University` model. Active Record is smart enough to automatically map column names to model attributes, which means there is no need to declare attributes inside Rails models, as that will be done automatically by Active Record.

```
1 class CreateUniversities < ActiveRecord::Migration
2   def change
3     create_table :universities do |t|
4       t.timestamps null: false
5       t.string :title
6     end
7   end
8 end
```

Listing 4 – Migration for Creating University

The above migration creates a method named `change` which will be called when running this migration. The action defined in this method is also reversible, which means Rails knows how to reverse the change made by this migration, in case of reversing later. Running this migration will create an `universities` table with one string column. It also creates a timestamp field to allow Rails track university creation and update times.At this point, a rake command has to be used to run the migration and create the `universities` table, together with its attributes:

```
$ bin/rake db:migrate
```

Listing 5 – Add University table in the Database

The procedure of generating the database is equivalent for the rest of the components in the application. The same steps were executed for creating faculties, specialties, timetables, groups,

courses, students, teachers and admins. Completing this phase means the successful modeling of the data layer, resulting in a reliable structure for storing records.

The project contains several special models, which have Devise authentication attached. As mentioned above, Devise is a gem for Rails, bringing a high-level solution that takes care of many different aspects for implementation. It presents controllers, views, mailers, and routes. While it's simple to issue a couple of commands and have Devise up and running, it is highly customizable. Devise has very thorough documentation and a large community that produces a boatload of useful extensions. The gem comes with handful modules, allowing to choose only the required ones. As seen in the application's schema and model (Listings 6 and 7), after Devise was added to the Teacher component, several modules were also generated to support password recovery, e-mail confirmation, account lock out, and many others. Bellow, an illustration of the application's schema, regarding the Teacher entity, is presented:

```
ActiveRecord::Schema.define(version: 20160514113821) do

  create_table "teachers", force: :cascade do |t|
    t.datetime "created_at",                          null: false
    t.datetime "updated_at",                          null: false
    t.string   "name"
    t.string   "teacher_faculty"
    t.string   "teacher_department"
    t.integer  "university_id"
    t.string   "email",                  default: "", null: false
    t.string   "encrypted_password",     default: "", null: false
    t.string   "reset_password_token"
    t.datetime "reset_password_sent_at"
    t.datetime "remember_created_at"
    t.integer  "sign_in_count",          default: 0,  null: false
    t.datetime "current_sign_in_at"
    t.datetime "last_sign_in_at"
    t.string   "current_sign_in_ip"
    t.string   "last_sign_in_ip"
  end

  add_index "teachers", ["email"], name: "index_teachers_on_email", unique: true
  add_index "teachers", ["reset_password_token"], name: "
    index_teachers_on_reset_password_token", unique: true

end
```

Listing 6 – Teacher Schema using Devise

Also, several changes were generated in the Teacher model file, regarding the possible options that can be implemented, using Devise. The original code implied only database relations between tables, assuming that a Teacher can belong only to one University and can have only one timetable assigned. After Devise integration, the model became suitable for authentication, as seen bellow:

```
class Teacher < ActiveRecord::Base
    # Include default devise modules. Others available are:
    # :confirmable, :lockable, :timeoutable and :omniauthable
    devise :database_authenticatable, :recoverable, :rememberable, :trackable, :
   validatable
  belongs_to :university
  has_one :timetable
end
```

Listing 7– Teacher Model using Devise

In the discussed project, the Admin model, the Teacher model and the Student model have the authentication resource provided by Devise. Once the database is set up, controllers need to be created. As example, in the Listing 8 is presented the command that is used to generate a new controller:

```
bin/rails generate controller Universities
```

Listing 8– Generate controller University

The fresh University Controller handles taking the input data from the URL (or the request body in the case of POST and PUT) and then interacts with the Model, as well as setting up data for the view portion. By default, Rails emphasizes a "RESTful" approach; that is to say, it uses the given HTTP verb (for example, GET, POST, PUT, DELETE) and well-organized urls to represent resources, leading to a general structure where code goes for operations on an object. It provides tools to handle basic CRUD (Create, Read, Update, Delete) actions on your data. In Rails terminology, the actions are typically:

- index - An action that returns a listing of all resources; for example GET /universities will hit the index action on the Universities Controller, listing the known universities.
- show - An action that performs a read operation for a single resource; for example GET /universities/1 will hit the show on the Universities Controller, showing the details of the University with id 1.
- new - The action to show a new object form, in our case, the new University form; for example GET /universities/new will hit the new action on the Universities Controller and show the new University form.
- create - A post action to take the form data from the new action and to attempt to create a record; for example POST /universities will hit the create action on the Universities Controller, typically with some associated form data.
- edit - Will show the form to edit a specific resource; for example GET /universities/1/edit will hit the edit action on the Universities Controller and show a form to edit the URL with id 1.
- update - Will attempt to update the given resource; for example PUT /universities/1 will hit the update action on the URLs Controller and attempt to update the University with id 1.

– destroy - Will attempt to destroy a given resource; for example DELETE /universities/1 will hit the destroy action on the Universities Controller and attempt to destroy the University with id 1.

So, a resourceful route provides a mapping between HTTP verbs and URLs to controller actions. By the convention mentioned above, each action also maps to particular CRUD operations in a database. A single entry in the routing file such as `resources :universities` creates seven different routes in the application, all mapping to the University Controller:

Table 3.2– Routes mapping the actions from University Controller

HTTP Verb	**Path**	**Controller#Action**	**Used for**
GET	/universities	`universities#index`	display a list of all universities
GET	/universities/new	`universities#new`	return HTML form for creating an university
POST	/universities	`universities#create`	create a new university
GET	/universities/:id	`universities#show`	display a specific university
GET	/universities/:id/edit	`universities#edit`	return HTML form for editing an university
PATCH/PUT	/universities/:id	`universities#update`	update a specific university
DELETE	/universities/:id	`universities#destroy`	delete a specific university

For better understanding, in Listing 9 are presented all the resources and their routes that were created for the project. Can be observed the multiple namespaces used for each category of user. The namespaces help divide the application's functionality, depending on the resources allowed for a given user:

```
1  Rails.application.routes.draw do
2
3    devise_for :scholars, :controllers => {:sessions => 'scholar/sessions'}
4    devise_for :teachers, :controllers => {:sessions => 'teacher/sessions'}
5    devise_for :admins
6
7    namespace :admin do
8      get '/' => 'home#index'
9      resources :teachers, only: [:index, :show]
10     resources :faculties, only: [:index, :show]
11     resources :specialties, only: [:index, :show]
12     resources :groups, only: [:index, :show]
13     resources :universities do
14       resources :teachers, except: [:index, :show]
15       resources :faculties, except: [:index, :show] do
16         resources :specialties, except: [:index, :show] do
17           resources :groups, except: [:index, :show] do
18             resources :timetables do
19               resources :courses
20             end
21           end
22         end
```

```
      end
    end
  end

  namespace :teacher do
    get '/' => 'home#index'
    resources :timetables
  end

  namespace :scholar do
    get '/' => 'home#index'
    resources :timetables
    get 'profile' => 'profiles#show', as: :profile
    get 'profile/edit' => 'profiles#edit', as: :edit
    match 'profile/update' => 'profiles#update', as: :update, via: [:put, :patch]
  end

  root 'home#index'
```

Listing 9 – Application's resources routes

Given the fact that the current University Controller needs all the actions implemented in order to be able to create, edit, show and destroy an instance, the content of the corresponding file looks like this:

```
class Admin::UniversitiesController < AdminsController
  def index
    @universities = University.all
  end

  def show
    @university = University.find(params[:id])
  end

  def create
    @university = University.new(university_params)
    if @university.save
      redirect_to admin_universities_path(@university)
    end
  end

  def new
    @university = University.new
    respond_to :html, :js
  end

  def edit
    @university = University.find(params[:id])
    respond_to :html, :js
```

```
  end

  def update
    @university = University.find(params[:id])
    if @university.update university_params
      redirect_to :action => 'index', :id => @university
    else
      render :edit
    end
  end

  def destroy
    @university = University.delete(params[:id])
    redirect_to :action => :index
  end

  private
  def university_params
    params.require(:university).permit(:title)
  end
end
```

Listing 10 – Content of University Controller

The rest of the controllers (Faculty Controller, Specialty Controller, Group Controller, Timetable Controller, Course Controller, Student Controller, Teacher Controller, Admin Controller) follow the same principle of creation and action implementation, depending on the needs it has to accomplish. Besides the controllers that need to be created, there is still one that wasn't mentioned until now. This is the Application Controller, which is generated at the creation moment of the Student application. Given the fact that the system operates with different namespaces, for better structuring of the project, some configuration where needed to be specified in this controller. Detailed information is presented in Listing 11:

```
class ApplicationController < ActionController::Base
  before_action :configure_permitted_parameters, if: :devise_controller?
  protect_from_forgery with: :exception

  protected
  def after_sign_in_path_for (resource)
    if resource.class == Admin
      stored_location_for(resource) || admin_path
    elsif resource.class == Teacher
      stored_location_for(resource) || teacher_path
    elsif resource.class == Scholar
      stored_location_for(resource) || scholar_path
    end
  end

```

```
16  def after_sign_up_path_for (resource)
17    if resource.class == Admin
18      stored_location_for(resource) || admin_path
19    elsif resource.class == Teacher
20      stored_location_for(resource) || teacher_path
21    elsif resource.class == Scholar
22      stored_location_for(resource) || scholar_path
23    end
24  end
25
26  def configure_permitted_parameters
27    devise_parameter_sanitizer.for(:sign_up){|u| u.permit(:email, :password, :
     password_confirmation, :group_id)}
28  end
```

Listing 11– Content of Application Controller

These three methods are responsible for proper resource rendering, depending on the type of user. The methods are implemented for log in and log out activity. The third method is responsible for student's registration, specifying the permitted parameters. The first two params are Devise - related, while the third one was manually included for ensuring the connection between student and its membership group.

3.3 Front-end implementation of the application

A Rails View is an ERb (Embedded Ruby) program that shares data with controllers through mutually accessible variables.If the app/views directory of the Student application is accessed, one subdirectory for each of the created controllers can be seen. Each of these subdirectories is created automatically when the same-named controller is created with the generate script. Each method defined in the controller needs to have a corresponding `erb` file, with the same name as the method, to display the data that the method is collecting (except for the delete method, where no view is needed). In the current project, another gem was used in order to transform all `erb` files in `haml` files, which simplifies things for the coding process. So, for the University Controller, 4 views were generated. The first one to be analyzed is the `new` view. It contains the form for creating a new University instance, as shown in Listing 12. Actually, the views can not be considered as front-end components, but the fragments of code presented in this section are combined with front-end elements, as the system's development is currently finished.

```
1 = form_for [:admin, @university] do |f|
2   = f.text_field :title, :class => "input-form-edit"
3   = f.submit :class => "btn btn-default add-btn-default"
```

Listing 12– Form for creating a new University

Assuming that the University is defined as having only one attribute - title - in the database, the form contains only the text field, where the name can be typed in. The other element is the

"Submit" button, for saving the new instance. It can be observed that these two elements are using two CSS classes for the front-end part.

These classes are located in the `student/app/assets/stylesheet/layout/admin` directory, following the convention about structuring front-end files. Listing 13 is displaying the content of the "input-form-edit" and "btn btn-default add-btn-default" classes.

```
.input-form-edit{
  margin-bottom: 15px;
  height: 40px;
  font-size: 16px;
  font-family: $font_lato;
}

.btn{
  &.btn-default{
    font-family: $font_lato;

  }
  &.add-btn-default{
    margin-top: 15px;
  }

  &.select-btn-default{
    margin-top: 15px;
  }
```

Listing 13– CSS classes for the University instance

The `.btn` class consists of 3 children classes, which are applied to different types of buttons in the application. This technique permits code reuse by avoiding the creation of separate classes and keeps up with the DRY(Don't Repeat Yourself!) principle during the system implementation. For rendering the `new` form in the current page, JavaScript language was used:

```
var newUniversity = $('#new-university')
newUniversity.html('<%=j render partial: "new" %>')
```

Listing 14– Render form in current page using Javascript

The `#new-university` variable is an `id` used in the University's show page where the form has to be rendered. Its position indicates the specific location in the page where the form has to be inserted. The `edit` view is similar with the `new` one, that's why it is skipped.

The next view to be discussed is the `index` view. For the design part of this page Bootstrap classes were used. For similar pages, the same concept was followed. The file presented bellow displays the page containing all the available current universities. Listing 15 comes with implementation details:

```
.container
  .col-md-9.col-md-offset-1
```

```
    .row
      %h1.text-center.heading-font
        All universities
      = link_to(new_admin_university_path, :method =>:get, :remote => true, :class => "icon-left add-button") do
        %span
          Add university
      #new-university
      %hr
      - @universities.order(:title).each do |u|
        %ul.action-block
          .col-md-6
            %li
              %b
                = link_to u.title, {:controller => :universities, :action => :show, :id => u.id, :method => :get}
          .col-md-3
            %li
              = link_to({:controller => :universities, :action => :edit, :id => u.id}, :class =>"icon-left edit-button", :method => :get, :remote => true) do
                %span
                  Edit university
              .edit-university{:id => "edit-university-#{u.id}"}
          .col-md-3
            %li
              = link_to({:controller => :universities, :action => :destroy, :id => u.id}, :class =>"icon-left delete-button", :method => :delete) do
                %span
                  Delete university
        %hr
```

Listing 15– Index view of the University instance

From the code listed above, it is seen that some additional CSS classes were used for buttons style, which implies icon positioning in the left-side of the actual button. The "icon-left", presented in Listing 16 class also has child classes, that have specific implementations for every type of button in the application.

```
.icon-left{

  &:before{
      content: "";
      width: 18px;
      height: 18px;
      float: left;
      display: inline-block;
  }

  &.delete-button{
```

```
12      &:before{
13        background-image: asset-url('delete.png');
14      }
15    }
16
17    &.edit-button{
18      &:before{
19        background-image: asset-url('edit.png');
20      }
21    }
22
23    &.add-button{
24      &:before{
25        background-image: asset-url('add.png');
26      }
27    }
28  }
```

Listing 16– CSS classes used for buttons design

Speaking about the front-end applied strategies, the `show` page of the University follows the same principle as the `index` page, that's why further details are not presented in the paper work.

3.4 User's experience handling the product

Another checkpoint in the development of an application is the user experience, described as the way a person feels about using a certain product. Performing such a procedure usually highlights some important aspects of the human-computer interaction and product ownership. It also creates some first impressions judging by the visual style of the product. For best results, the interaction with it should be as simple as possible. To increase the quality of the application, it is recommended to continuously work on requirements, based on information gathered from user's feedback. In order to meet this goal, a set of possible questions from the user's side were generated. These questions include all the basic functionality the users expect to face while interacting with the application. Giving simple answers to those questions, improve the general user's opinion about the product. Find the questions listed bellow:

- What are the prerequisites in order to be able to use the application?
- Who are the users of the application?
- What is the format for student registration?
- What is the format for teacher registration?
- When is the application ready for use?
- Who populates the database with records?
- Who is responsible for database maintenance?

– How to create a specific entity into the system?

– What is the structure of the system?

– What is the format of the course creation?

– What kind of information should be prepared in advance for database population?

– What is the validity period of the database records?

Thinking from user's point of view, a good product means a friendly user interface for interaction. inClass was thought as an easy to comprehend application, that would guarantee an easy access to the main components. That's why, for the first version, it was chosen to display data in the most accessible structure, via tabular representation.

3.4.1 Using the application as administrator

The system was designed to look like other familiar application, which would intuitively guide the users through its components. Some of the basic activities that the users can perform during a session with inClass are presented bellow as application screenshots. In Figure 3.2 is shown the first steps made by the administrator in order to access the system:

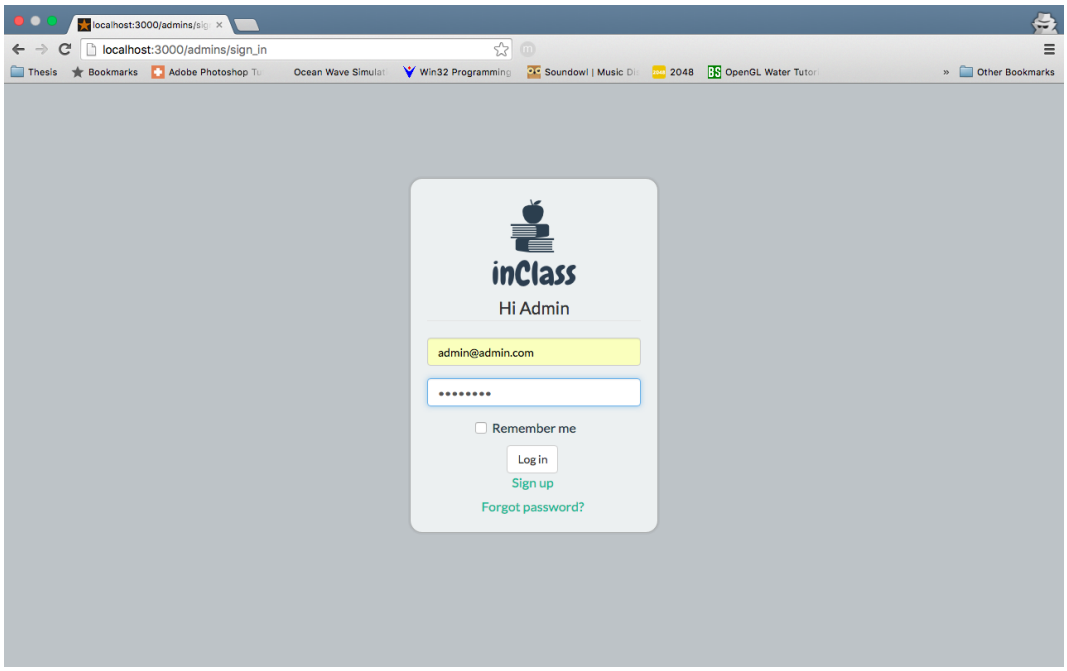

Figure 3.2– Admin log-in form

Given the fact that the application wasn't yet deployed, in order to reach it, the server has to be started from Terminal. After the necessary command was typed in, the administrator has to open a local browser and navigate to the address `http://localhost:3000`. Because the administrator side of the application is hidden from the standard users, the admin has to add to the url address a `/admin` missing piece of text for reaching the log in form.

After successful log-in, the admin can start creating the entities, which have a tree representation, where the main node is the university. The Figure 3.3 illustrates how new universities are created.

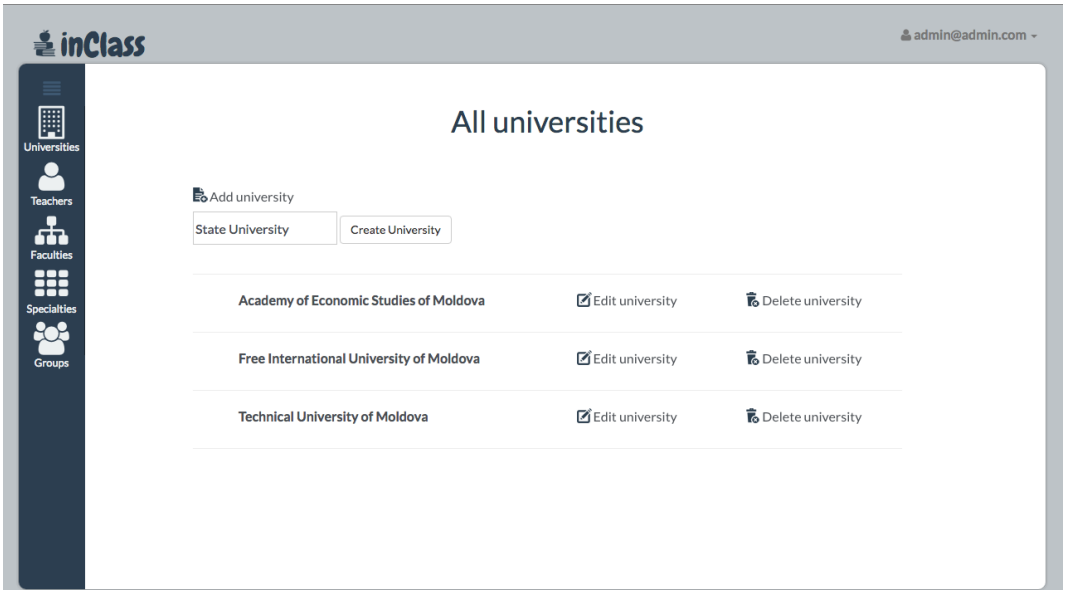

Figure 3.3– Adding new universities process

Also, as mentioned above, the administrator is responsible for registering teachers into the system. In Figure 3.4 is shown how a new teacher account is added:

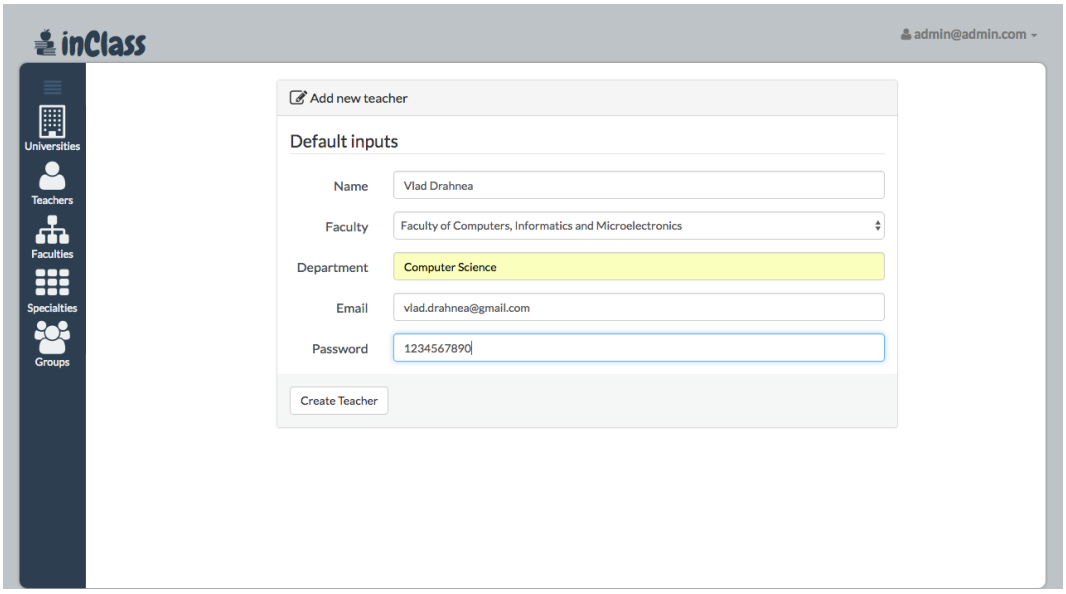

Figure 3.4– Teacher registration

The process of adding new faculties, specialties, groups and timetables is similar with the one of adding new universities, that's why additional screenshots were skipped. At this moment, the next process that is worth mentioning is the creation of courses, as parts of a timetable. This activity is represented in Figure 3.5:

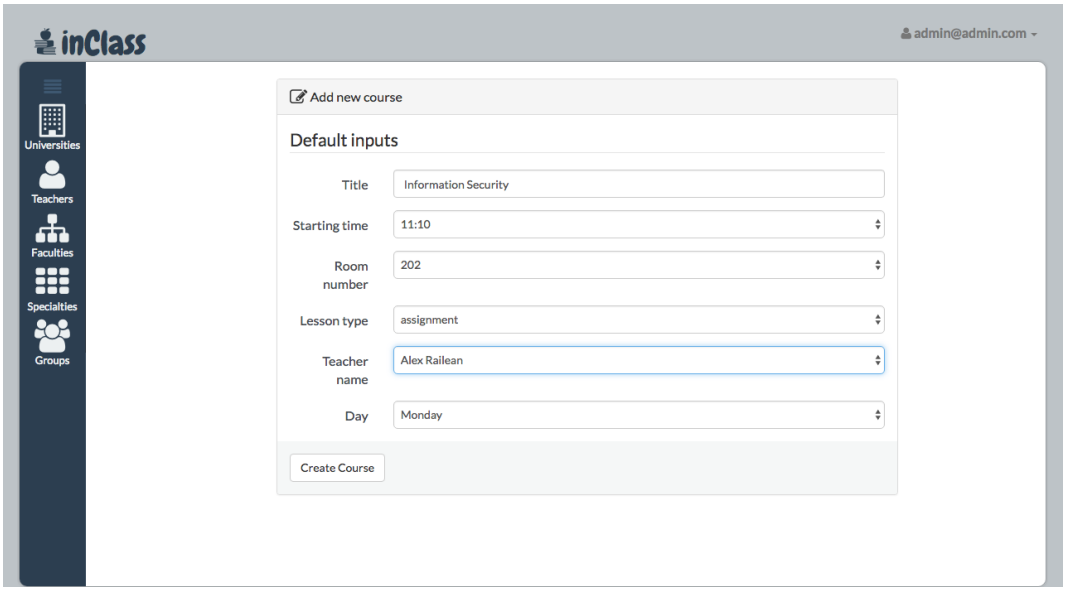

Figure 3.5 – Form for adding a new course

For accessibility purposes, the menu in the left-side of the application contains links to ease the access to the in-depth layers, like groups, timetables and eventually - courses. So, to check his/her last activity, the administrator can avoid traversing the long path in order to reach the courses by clicking the appropriate link in the menu. The records are displayed in ascending order according to the last activities performed on each of them.

As a final step, adding all the necessary courses, results in a completely digitized timetable, available for any student subscribed as member of the group for which the timetable was created. The Figure 3.6 illustrates how a timetable is displayed for the admin, and later - for students:

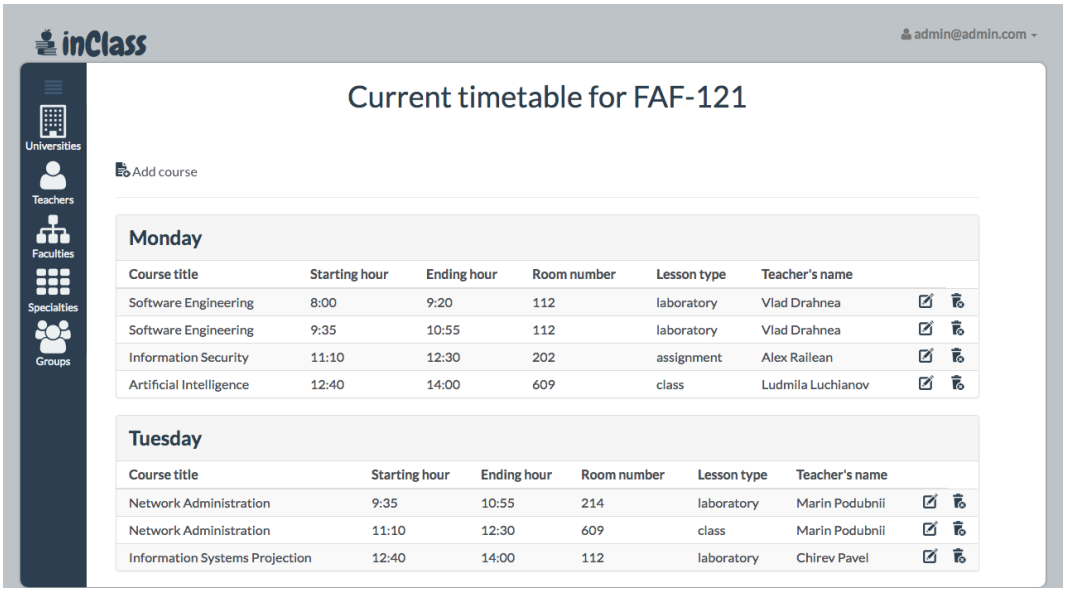

Figure 3.6 – Final representation of a timetable for admin

This is the schematic representation of the admin side of the application. For the design part, it is reminded the fact that Hierapolis theme was integrated. There still remains to discuss the student-teacher side of the application, where the design was created from scratches and modeled using Bootstrap and CSS - written classes for the front-end part.

3.4.2 Using the application as teacher/student

Assuming that the administrator has already created accounts for a specific group of teacher, let's log in the system with his/her credentials. In Figure 3.7 is presented the common home page, which stores the both possibilities of log in, for teachers and for students:

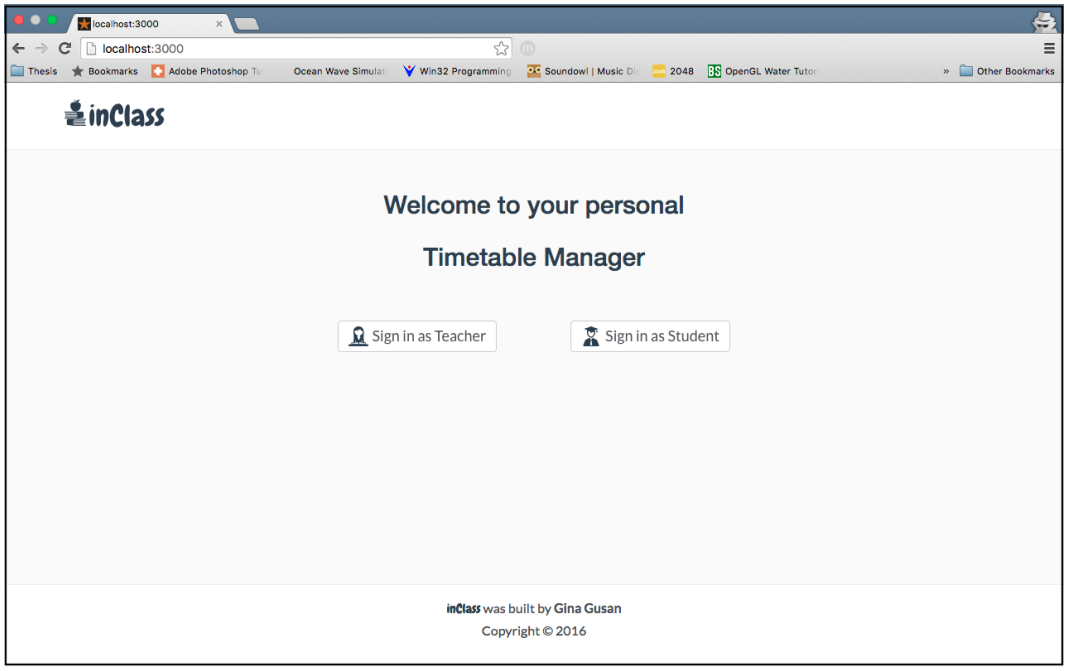

Figure 3.7– Home page of the application

After clicking the "Sign in as Teacher" button, the corresponding form will be rendered. The Figure 3.8 illustrates how the form actually looks like. It can be seen that, for having access in the system, every teacher has to receive a password that matches his email address.

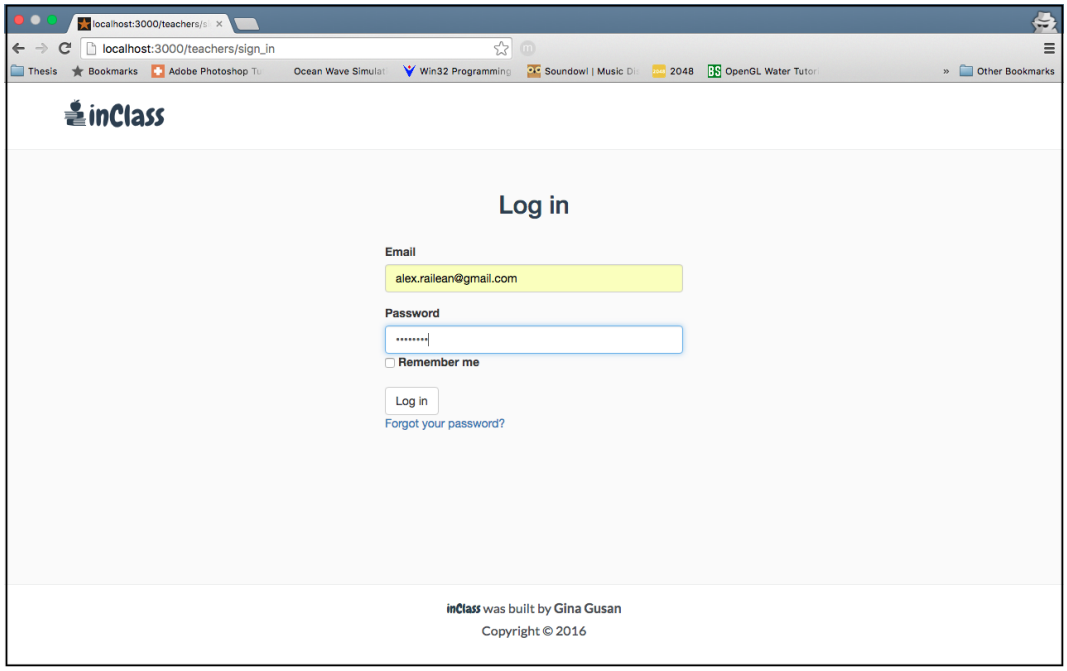

Figure 3.8– Teacher's log in form

As a matter of fact, the forms for teacher and student are similar, with one exception, that the student has one additional button that gives the possibility to register into the system. Each student is responsible for personal registration, where he/she specifies for what group he wants his timetable to be generated. The Figure 3.9 presents this distinctive situation:

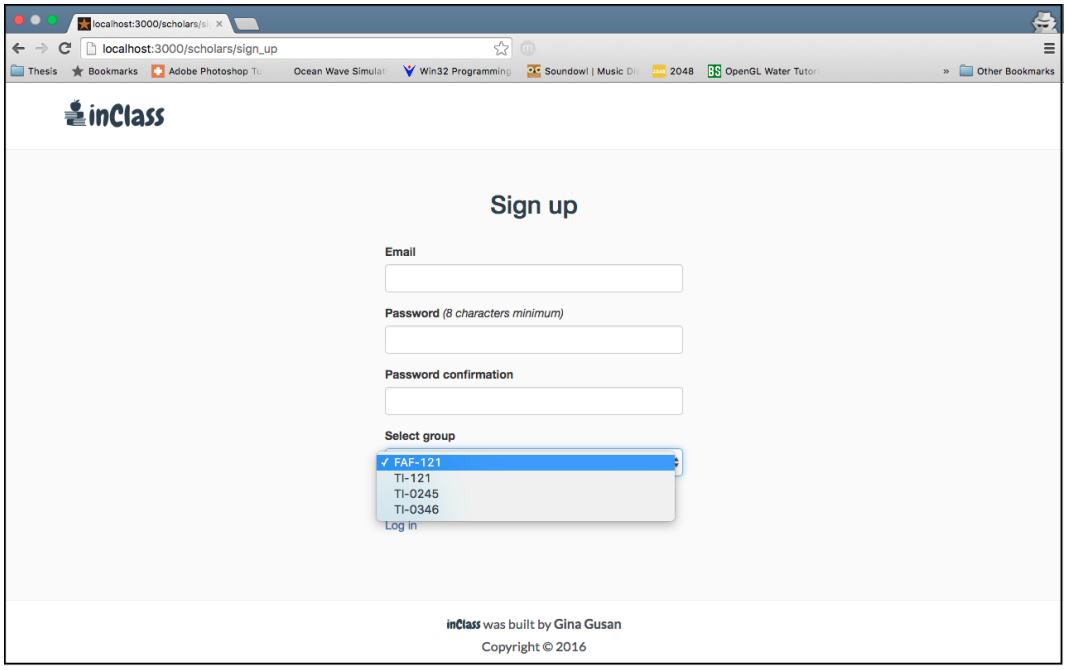

Figure 3.9 – Student's sign up form

After logging in the system, the teacher is redirected to his home page, which displays his current timetable. The application compares the system's current day and selects only the courses that have this day specified at the time of creation. Bellow, is presented the result of this selection, for the current teacher:

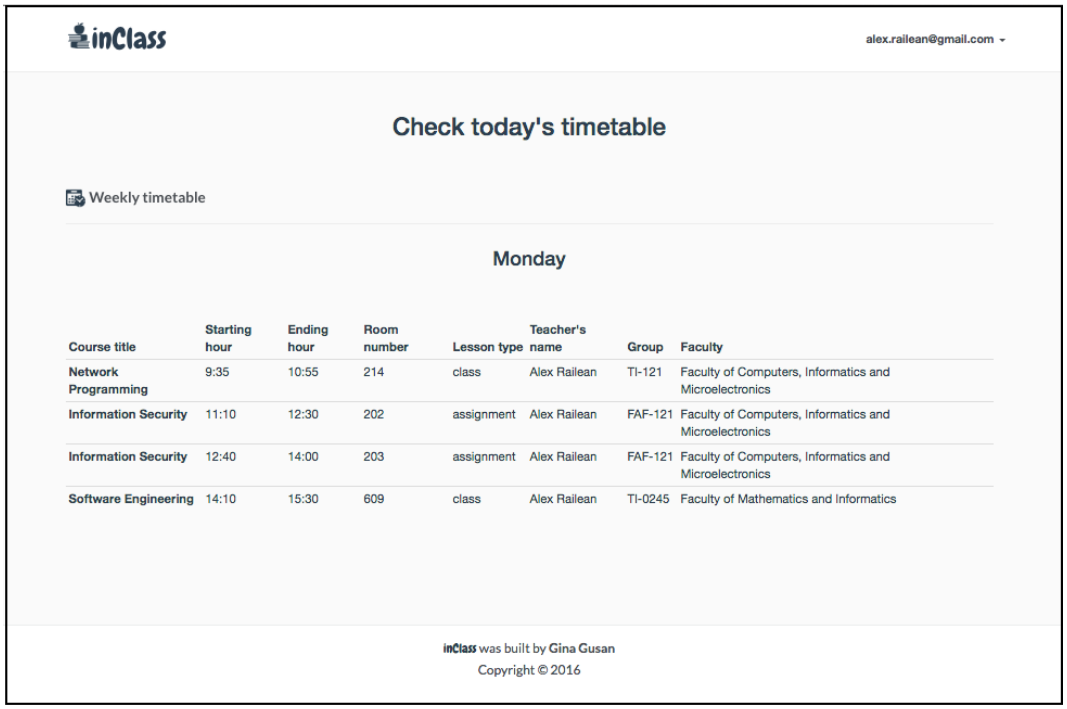

Figure 3.10 – Teacher's current timetable

The process is equivalent for the student as user of the system, the only difference being that the "Faculty" and "Group" columns were omitted from the listing, from obvious reasons. In Figure 3.10 can be observed the button "Weekly Timetable" that, when it is clicked, redirect the teacher to the courses full schedule, where he/she can visualize the daily courses grouped by day and sorted in ascending order from the first working day until the last one. The Figure 3.11 presents that situation:

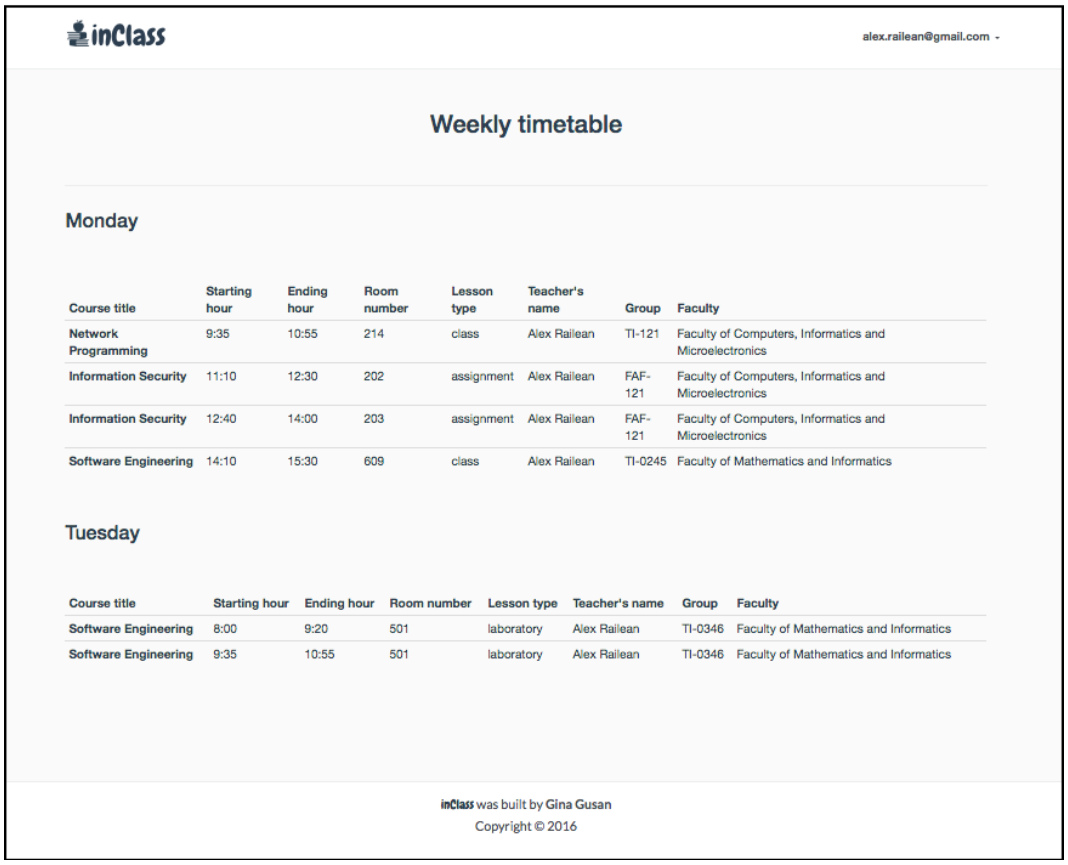

Figure 3.11 – Teacher's weekly timetable

At this point, the interaction between the system and each type of user is completed. Basically, all the functionality was covered and presented bellow. The next pending task to be implemented in the application's functionality is the sorting of timetable's courses by even and odd weeks. This particularity is encountered only at the Technical University of Moldova, that's why, in general lines the product is ready for release on the local market.

4 Economic Analysis

4.1 Project description

inClass is a project developed for a specific target group, which are people involved in academic structures, like universities. The main task of the application is to ease the access to the current studying timetable by generating the individual version for each user. There are three types of users involved: administrators (they are in charge of introducing data into the system), students and teachers (they are able to access the data from the system through account registration). Given that nowadays the technological progress is exponentially growing, the fact that inClass is a web application represents a major advantage for its users. As a matter of principle, universities already guarantee internet access on their territories, meaning that now it is possible to access inClass from a smartphone at any moment of time. Having it available online gives flexibility to teachers and students during classes and breaks. Instead of visiting the general timetable from the hole of the university, it is enough to navigate on the internet and be aware of where the next course should take place.

It is worth mentioning that there is no alternative application in our country that would offer such kind of service. Similar projects are already implemented in countries abroad and represent a big step for transferring the manual administration system to the computerized one. Having such an application integrated in the university's structure would bring benefits to all implied parts as it would be out of charge for users and easy to maintain for administrators. The only extra job that would allow avoiding the old habits is maintenance of the data by updating it once in a semester, when the timetable changes.

In order to proceed to the implementation of the system, it is necessary to think about a project budget, which offers a scientific and justifiable economical point of view over the system itself. In order to decide if the product is viable on the national economy, it is expected to be elaborated an initial overview of the costs and incomes after launching on the market. That's why, economic analysis is one of the first steps, meant to motivate, create and implement the given informational system, whose final result would be a general overview on the elaboration of the system. In this chapter, there are going to be analyzed all the expenses necessary to elaborate the project, starting from the materials/non-materials used for long term, time schedule establishment and indirect expenses. Then there will be computed the days necessary to elaborate the project, the team involved and the salary for each person. After computing these indicators and having a synopsis on the subject in question, will be much easier to predict the next steps to be done in order to have a profit.

4.2 Project time schedule

Given the fact that the system in discussion can be considered a complex one, it is expected to plan a time schedule first. For better results and efficient outcome, Agile software development will be applied. It promotes adaptive planning, evolutionary development, early delivery and continuous improvement. It also encourages rapid and flexible response to change. It is an iterative and incremental process that goes through 5 stages: planning, research, development, testing, and deployment. The benefit of using Agile is that a prototype is created in the early stage of development and all the future modifications can be treated as improvements for the base application.

4.2.1 SWOT Analysis

Performing the SWOT analysis over a system gives a brief overview about expectations or possible problems that can appear during the lifetime of the system.It is mandatory to identify the SWOTs because they can inform later steps in planning to achieve the expected objectives. In Table 4.1 it is represented the strategic planning method, used to evaluate Strength, Weaknesses, Opportunities and Threads that can involve the given system.

Table 4.1– SWOT analysis

Strengths	Weaknesses
– No competitors on local market; – Accessible and portable; – It is expected to be highly profitable; – Easy maintainable;	– Systematic database updates needed; – Require internet connection in order to access; – Not fully responsive on smartphones;
Opportunities	**Threats**
– Ensures with access 2 types of users (teachers, students); – Nice solution for personal time management;	– Heavy loads at each database update; – Account registration for teachers not fully secured;

Referring to the SWOT analysis as the basis of the promotion/development of any product, each company has the opportunity to foresee the possible profit as well as possible wastage. One of the most dangerous factors that can be prevented with SWOT is the risk of concurrency, which also has an important role in market development and the increase of product's quality. To predict other negative effects, any company must have a well-designed plan, like having monthly statistics, which

would contribute at diminishing economical threats.

4.2.2 Defining objectives

The main objective of the project is to improve the individual time management both for teachers and students. Also, as a new product on the national market, its purpose is to spread among universities as an administration tool for semester timetables. The application has several development targets, which would enrich the application with even more functionality for personal usage and monitoring of current study situation. As the application is provided as a free product, it is expected to gain users among all academical infrastructures.

4.2.3 Time schedule establishment

There are five main steps to be accomplished in order to reach a final result for the project in discussion:

- Planning (form some ideas about the design of the project, build some sketches and diagrams, determine amount of needed resources)
- Researching (analyze the market, the target group for which the project is intended, learn technologies for further use)
- Developing (create the actual system, adjust requirements to the used technologies and frameworks)
- Testing (perform debugging in order to ensure the well behavior of the system and integrity of functionality)
- Deployment (make the application available for users)

The key to a well-scheduled plan, that would lead to the in-time accomplishment of the project, is the proper subdivision of the workload, according to the given resources. Meaning that, every step enumerated above, should be scheduled for a finite period of time, judging by the level of difficulty implied. The planning period should be considered a flexible one, since at this moment some general ideas on the project have to be discussed. Changes are allowed at this step, since requirements are still established. For the researching period, it is important to be open to new ideas for consideration, in order to have a better outcome. The development period should be the most accurate one and should be divided in sprints, for better monitoring of the workflow. In the table bellow are represented all the general steps involving the development of the project. The following adnotations were used: PM – project manager, SA – system architect, SM – sales manager, D – developer.

Table 4.2– Time schedule

Nr	Activity Name	Duration (days)	People involved
1	Analyze requirements and activities schedule	7	PM, SA, SM, D
2	Perform market analysis	5	PM, SM
3	Establish functional features	14	PM, SA
4	Elaborate Use Case Diagrams	5	PM, SA, D
5	Implement database design	7	PM, SA, D
6	Implement website design	7	PM, SA, D
7	Create back-end functionality	28	D
8	Test back-end functionality	4	D
9	Create front-end functionality	21	D
10	Test entire project	7	PM, SA, D, SM
11	Validate results	3	PM, SA, D, SM
12	Write documentation	7	D
13	Deploy the system	2	PM, SA, D
14	Commercialize the product	3	SM
15	Total time to finish the system	120	

As a generalization, Table 4.2 represents a sketch for the first iteration of the project. Each activity was evaluated with a time stamp and responsible employees were assigned. For the completion of the purposed system including 14 activities was estimated a total amount of time including 120 working days. Bellow are indicated the amount of spent time for each individual:

- PM: 50 days;
- SA: 45 days;
- SM: 13 days;
- D: 101 days.

4.3 Economic motivation

It is actual in economy to bring economical proofs for the IT projects, basing on the specific of the concurrences in economic relationships, which suppose a wide research space. In the conditions of a low degree of determination of the marketing environment, of high prices' volatility, decreased degree of prognoses depth, a common business-plan doesn't allow the exact foreseeing of the final results of the business. In this context, one of the basic instruments is choosing the methods, the right positions and index for the economical proofs. Realization of this goal conditions a large number of scientifically research, subordinated to the primary goal and formulated by means of the following objectives:

– Studying the theoretical and methodical aspects of the business-planning in the conditions of the concurrency on the market;

– Systematization, determining the methodology and specifying the index for the economical proof of the business-plans in IT;

– Study and analysis of the actual practice of economical proofs of the business-plans for IT in Republic of Moldova;

– Developing methodological concepts of the proofs of the decision of investment in the conditions of risk and incertitude;

– Studying the evaluation criteria of the business-projects' efficiency and elaboration of a mechanism of complex evaluation of these.

4.3.1 Tangible and intangible asset expenses

Expenses and initial budget are the ones that sharpen from the beginning the project itself, by imposing some limitations on the complexity of the system and defining its boundaries. In this section, an evaluation of the necessary amount of money will be computed, in order to ensure the correct administration of financial resources. At the beginning, in Table 4.3, will be listed all the tangible assets used for the project. Tangible assets are defined as any assets that have a physical form.

Table 4.3 – Tangible assets expenses

Material	**Specification**	**Measurement unit**	**Price per unit (MDL)**	**Quantity**	**Sum (MDL)**
MacBook Pro	i5 processor	Unit	24000	1	24000
				Total	24000

The Table 4.4 presents the intangible assets, used for the project. Intangible assets are all those that do not posses a physical form. In this case, the discussion is about the **software** needed to build the application. For the design of UML diagrams, *Enterprise* Architect was used.

Table 4.4 – Intangible asset expenses

Material	**Specification**	**Measurement unit**	**Price per unit (MDL)**	**Quantity**	**Sum (MDL)**
License	Enterprise Architect Desktop Edition License	Unit	1800	3	5400
License	Sublime Text 3	Unit	1200	3	3600
				Total	9000

Also, additional expenses represent the direct expenses, used for work facilitation during project development. These costs cannot be included in any of the previous tables, because their values aren't included directly into the budget of the project and they have to be mentioned out of the topic. The Table 4.5 emphases the direct expenses:

Table 4.5 – Direct expenses

Material	Specification	Measurement unit	Price per unit (MDL)	Quantity	Sum (MDL)
Whiteboard	Universal Dry Erase Board	Unit	700	1	700
Paper	A4	100 sheets	60	1	60
Pen	Blue pen	Unit	5	10	50
				Total	810

Given all the raw data, a total amount of direct expenses can be calculated:

$$T_e = 24000 + 9000 + 810 = 33810 \tag{4.1}$$

4.3.2 Salary expenses

In this compartment will be discussed the remuneration of each employee during the development of the project. It is assumed that each member of the team has the salary listed bellow:

– Project Manager – 500 MDL

– System Architect – 580 MDL

– Sales Manager – 350 MDL

– Developer – 400 MDL

After the closing of the project, some statistics regarding financial costs for employee remuneration can be presented in Table 4.6:

Table 4.6 – Salary expenses

Employee	Working days	Salary per day (MDL)	Salary fund (MDL)
Project Manager	50	500	25000
System Architect	45	580	26100
Sales Manager	13	350	4550
Developer	101	400	40400
		Total	96050

Besides salaries, the social service fund also retrieves a part of the money, constituting 23% of total salary. Also, there is the medical insurance fund, constituting $4,5\%$ of the same sum. The next step is to compute the social service fund, according to the relation (4.2) :

$$\begin{aligned} FS &= F_{re} \cdot T_{fs} \\ &= 96050 \cdot 0.23 \\ &= 22092, \end{aligned} \tag{4.2}$$

where FS is the salary expense, F_{re} is the salary expense fund and T_{fs} is the social service tax approved each year. The medical insurance fund is computed as:

$$\begin{aligned} MI &= F_{re} \cdot T_{mi} \\ &= 96050 \cdot 0.045 \\ &= 4322, \end{aligned} \tag{4.3}$$

where T_{mi} is the mandatory medical insurance tax approved each year by law of medical insurance.

Now, the total work expense fund is calculated as sum of the previous computed indicators:

$$\begin{aligned} WEF &= F_{re} + FS + MI \\ &= 96050 + 22092 + 4322 \\ &= 122464, \end{aligned} \tag{4.4}$$

where WEF is the work expense fund, FS is the social fund and MI is the medical insurance fund. The final indicator shows the total work expense fund.

4.4 Individual person salary

Having the total work expense fund computed, it is necessary to determine the net salary for the developer. Considering the developer's salary of 400 MDL per day and there is a totally 120 working days for accomplishing the project, so the gross salary that the developer gets is:

$$GS = 400 \cdot 120 = 48000, \tag{4.5}$$

where GS is the gross salary computed in MDL.

Social fund tax this year represents 6%, so the amount that should be tax paid in MDL represents

$$SF = 48000 \cdot 0.06 = 2880. \tag{4.6}$$

Medical insurance tax represents 3.5% and gives the following result

$$MIF = 48000 \cdot 0.045 = 2160. \tag{4.7}$$

In order to proceed with income tax computations, it is necessary to calculate the amount of taxed salary.

$$\begin{aligned} TS &= GS - SF - MIF - PE \\ &= 48000 - 2880 - 2160 - 10128- \\ &= 32832, \end{aligned} \tag{4.8}$$

where TS is the taxed salary, GS – gross salary, SF – social fund, PE – personal exemption, which this year is approved to be 10128.

The last but not the least thing to be computed is the total income tax, which is 7% for income

under 29640 MDL and 18% for income over 29640 MDL.

$$\begin{aligned} IT &= TS - ST \\ &= 29640 \cdot 0.07 + (32832 - 29640) \cdot 0.18 \\ &= 2074.8 + 574.6 = 2649.4, \end{aligned} \tag{4.9}$$

where IT is the income tax, TS – the taxed salary and ST – the salary tax.

With all this now it is possible to find out what's going to be the net income.

$$\begin{aligned} NS &= GS - IT - SF - MIF \\ &= 48000 - 2649.4 - 2880 - 2160 \\ &= 40310.6, \end{aligned} \tag{4.10}$$

where NS is the net salary, GS – gross salary, IT – income tax, SF – social fund, MIF – medical insurance fund.

4.4.1 Indirect expenses

Other expenses involved in the completion of the project involves production consumption. In the Table 4.7 are mentioned expenses like public transport, electricity, internet access and office water.

Table 4.7– Indirect expenses

Material	**Specification**	**Measurement unit**	**Price per unit (MDL)**	**Quantity**	**Sum (MDL)**
Internet	Moldtelecom	Pack	200.00	4	800
Transport	Public bus	Trip	2.00	240	480
Water	Apa-Canal Chisinau	m3	9.2	20	184
Electricity	Union Fenosa	KWh	2,16	500	1080
				Total	2544

4.4.2 Wear and depreciation

When speaking about economic analysis, it is vital to understand the influence of time over the product. Usually, it happens that a depreciation in value can appear, that's why it is a value that should be computed and take into account. As a tuple, with the depreciation, the wear will be calculated. Depression will be computed uniformly for the whole project duration, so that there are no accountancy issues. As a matter of fact, a business plan divided for 3 years should be compartmentalized into 3 uniform parts according to each year.

Normally wear is computed regarding to the type of asset. The computer mentioned above as tangible asset can be used for a period of 3 years. Licenses will last for a single year. Straight line depreciation will be applied. First step is to sum up tangible and intangible assets, while the salvage

costs of each of the items at the end of their period of use, has to be subtracted:

$$\begin{aligned} TAV &= \sum (AC - SV) \\ &= (25200 - 5000) + (2880 - 1000) \\ &= 22080, \end{aligned} \tag{4.11}$$

where TAV is the total assets value, AC – assets cost, SV – salvage value. In order to get the yearly wear, divide total asset value by the period of use of assets, being 3 years.

$$\begin{aligned} W_y &= TAV/T_{use} \\ &= 22080/3 \\ &= 7360, \end{aligned} \tag{4.12}$$

where W_y is the wear per year, TAV – total assets value, T_{use} – period of use. Relation (4.12) included tangible assets which will last for 5 years and intangible assets which last only one year. The initial value of assets in MDL was

$$\begin{aligned} W &= W_y/D_y \cdot T_p \\ &= 7360/365 \cdot 120 \\ &= 2420, \end{aligned} \tag{4.13}$$

4.4.3 Product cost

At this point, it is time to compute the product cost which includes direct and indirect expenses, salary expenses and wear expenses as shown in Table 4.8.

Table 4.8– Total Product Cost

Expense type	**Sum (MDL)**	**Percentage (%)**
Direct expenses	810	0.62
Intangible expenses	3600	2.78
Salary expenses	122464	94.7
Asset wear expenses	2420	1.87
Total product cost	**129294**	**100**

4.4.4 Economic indicators and results

At the moment, all the expenses for the development of the project were computed. Now is time to consider how the application should be included on the market. As mentioned above, the target group which is meaningful for the given project represent the academic infrastructures, including teachers and students. So, as starting investors, should be considered universities' administrations. There is no optimal price established for the initial project, so a indicator of 25% on top of the production cost will be used, in order to determine what the real cost the application should have.

$$\begin{aligned} GP &= C_{total}/N_{cs} + P_p \\ &= 129294/100 + 0.25 \cdot 1292.94 \\ &= 1616, \end{aligned} \tag{4.14}$$

where GP is the gross price, C_{total} – total product cost, N_{cs} – number of copies sold, P_p – chosen profit percentage. This is not the price of the end product, since it is necessary to add sales tax (VAT), which represents 20% and is added to the gross price.

$$\begin{aligned} P_{sale} &= GP + TX_{sales} \\ &= 1616 + 0.2 \cdot 1616 \\ &= 1932, \end{aligned} \tag{4.15}$$

where P_{sale} is the sale prices including VAT, GP – gross price, TX_{sales} – sales tax. The net income is computed by multiplying gross price and the number of expected copies to be sold, which will be

$$\begin{aligned} I_{net} &= GP \cdot N_{cs} \\ &= 1616 \cdot 100 \\ &= 161600, \end{aligned} \tag{4.16}$$

where I_{net} is the net income, GP – gross price, N_{cs} – number of copies sold. Moreover it is necessary to compute the gross and net profit. The indicators are GPr – gross profit and NPr – net profit.

$$\begin{aligned} GPr &= I_{net} - C_{production} \\ &= 161600 - 129294 \\ &= 32306 \\ NPr &= GPr - 12\% \\ &= 32306 - 12\% \\ &= 28429, \end{aligned} \tag{4.17}$$

where I_{net} is the net income, $C_{production}$ – cost of production. The profitability indicators are C_{profit}

– cost profitability, S_{profit} – sales profitability computed in MDL.

$$\begin{aligned} C_{profit} &= GPr/C_{production} \cdot 100\% \\ &= 32306/129294 \cdot 100\% \\ &= 24.9\% \\ S_{profit} &= GPr/I_{net} \cdot 100\% \\ &= 28429/161600 \cdot 100\% \\ &= 18.59\%. \end{aligned} \tag{4.18}$$

4.5 Economic conclusions

After analyzing the given project from economical point of view, several conclusions were made. First of all, making such a research gave the possibility to understand better which are the strengths and the weaknesses of the system and where additional attention should be paid in order to avoid unforeseeable external factors that could create unexpected problems.Several other indexes were computed such as direct and indirect expenses, tangible and intangible resources, worker remuneration, etc. As an observation, the biggest expense would be the salaries paid to employees and future improvements for maintaining the application up to date. Now, the next step is to find suitable clients, that would use the application and, as a result, would suggest different improvements for the good functioning of the system. Finding at least one university for implementation purpose would represent a big step into the future of the application, guaranteeing an economic profit and great perspectives for upcoming versions.

Conclusions

Elaborating the inClass project, several steps were accomplished first. Before the actual development was started, the main goal was to understand entirely the problem that needed to be solved, by providing an optimal solution to it. Nowadays, we are living in the era of technological progress, where machines get to do the most part of the jobs people once used to do entirely. That's why, it is vital to sustain the every day life improvements human kind gets while creating software, instead of using the ancient approach of doing everything individually. This is a thing that our country understood and during the last few years, more and more state structures are continuously developing their foundations, by providing reliable, modern technologies for the every day use.

The current application refers to a specific type of state structures, that are local universities. These entities are providing a high rate of technological progress during the last years, thing that encourages the stuff and also the students and teachers. Given the fact that it has been seized a continuously growing interest in determining people to work with software, now becomes much more realistic to reorganize the academic structures. This means that people already reached the moment when are ready to bring software products in universities in order to optimize the routine processes which they are responsible for accomplishing. This is a big step that has to be adjusted properly to the university's needs.

inClass is the software product that comes with a project in development and a prototype to fix an existing problem in universities. At the same time, it represents a small change for the academic infrastructure, given the fact that is easy to implement, use and maintain, while the results are consistently improved.The application was implemented as a web platform, assuming that universities don't have the problem of free connection to the internet anymore. So, in order to access the application, it is enough to own a smartphone and Wi-Fi. It represents the perfect solution for avoiding the general timetable, which is so uncomfortable to reach and use.

For a better representation of the application, a full analysis over the system was done first. It consists of UML diagrams, which have the purpose of visualizing the application from different points of view. After this schematic overview of the system is done, there remain no pending questions about the functionality provided by the application. Once things get clear, it is time to start implementing those functionalities. The technology used for developing the application is a powerful one. Ruby on Rails is one of the most high rated frameworks at the current moment, which allows the developer to implement various functionality with a minimum of written code. Also, it allows the use of gems, which simplify the task even more.

Finally, assuming that the project is finished, it is necessary to provide an user guide, for easing the tasks of the actual users. There are several moments that need to be paid attention to, moments that were clearly specified in the content of the current document. The main thing to be mentioned is that the application allows 3 types of users: administrator, teachers and student. In order to use the application it is necessary to register in the system. Registration is specific for each type of user.

At the moment, the provided solution is unique on the local market, thing that is convenient from economical point of view. There are necessary less expenses to take into account in order to finish the project successfully. The less expenses are, the less the market price will be, which makes

the software available for use for local universities. In terms of economical analysis, the numbers were pretty impressive. If computing all the possible expenses that could occur while developing the project with some more improvements, the team would also need to find sufficient clients, that would guarantee a low price of the product and more than 18 % as profit.

Also, is a good idea of coming with some future possible improvements for the application. The most recent improvement would definitely be the adjustments of the application to support sorting timetables by current weak, meaning that the even and odd weeks would be take in account. Of course, this is a requirement that is not valid for all universities, but it is still very important for the ones that have different schedules for even and odd weeks. Another improvement could be done for the student/teacher side of the application. At the current moment, the teachers and students are allowed only to visualize data. Assuming that the current project it is a prototype, this is acceptable. Still, the idea of having several tools available for checking the current status of a certain day or semester, by using progress bars and other visual components, would enlarge the boundaries of activity for these types of users. Another feature to be implemented could be the possibility of interaction between student and teachers, while discussing some updates of the courses via a special page for announcements. Many other improvements can be done once the universities get to the point when they implement the system as part of their infrastructures.

References

1 Moodle, *Effectiveness of moodle-enabled learning in private schools*, https://www.researchgate.net/publication/

2 Daniel Kehoe, What is Ruby on Rails?, *Why Ruby?*, 2013, http://railsapps.github.io/what-is-ruby-rails.html/

3 About Ruby, *Ruby On Rails : Making an App Part 1*, 2014, http://webtutplus.com/ruby-on-rails-making-a-app-part-1/

4 MVC pattern, *From zero to deploy*, https://www.railstutorial.org/book/beginning/

5 Michael Hartl, Rails: MVC-alike framework, *Learn Web Development with Rails*, 2014, https://www.railstutorial.org/book/

6 Active Record, *The Active Record Pattern*, http://guides.rubyonrails.org/active_record_basics.html/

7 Object Relational Mapping, *Object Relational Mapping in Rails terms*, http://guides.rubyonrails.org/active_record_basics.html/

8 Active Record mechanism, *Active Record as an ORM Framework*, http://guides.rubyonrails.org/active_record_basics.html/

9 Conventions in Rails, *Convention over Configuration in Active Record*,http://guides.rubyonrails.org/active_record_basics.html/

10 Bootstrap, *Bootstrap Components*, http://getbootstrap.com/components/

11 Nicolae Sfetcu, Cascading Style Sheets, *Web Design and Development*, 2003, http://books.google.md/books/?id=jAGSwb84057/

12 JavaScript, *Organizing JavaScript in Rails applications*, http://brandonhilkert.com/blog/organizing-javascript-in-rails-application-with-turbolinks/

13 UML diagrams, *UML Tutorial*, http://www.tutorialspoint.com/uml/index.htm/