“ALEXANDRU IOAN CUZA” UNIVERSITY OF IAȘI

FACULTY OF COMPUTER SCIENCE

MASTER THESIS

Building web applications with Behavior Driven Development

proposed by

Emanuel-Vasile Berea

Session: *February, 2017*

Scientific coordinator

Asist. Dr. Vasile Alaiba

“ALEXANDRU IOAN CUZA” UNIVERSITY OF IAȘI

FACULTY OF COMPUTER SCEINCE

Building web applications with Behavior Driven Development

Emanuel-Vasile Berea

Session: *February, 2017*

Scientific coordinator

Asist. Dr. Vasile Alaiba

DECLARAŢIE PRIVIND ORIGINALITATE ŞI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de disertație cu titlul „Building web applications with BDD methodology” este scrisă de mine şi nu a mai fost prezentată niciodată la o altă facultate sau instituţie de învăţământ superior din ţară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi deţin referinţa precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alţi autori deţine referinţa precisă;
- codul sursă, imagini etc. preluate din proiecte open-source sau alte surse sunt utilizate cu
- respectarea drepturilor de autor şi deţin referinţe precise;
- rezumarea ideilor altor autori precizează referinţa precisă la textul original.

Iaşi, februarie 2017

Absolvent Emanuel-Vasile Berea

____

(semnătura în original)

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca Lucrarea de disertație cu titlul „Building web applications with BDD methodology”, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea Alexandru Ioan Cuza Iaşi să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iaşi, *februarie 2017*

Absolvent *Emanuel-Vasile Berea*

____

(semnătura în original)

Summary

1. Introduction

1.1 Motivation

In our modern ages, more and more systems are controlled by software, being very hard to find a domain which was not transformed by IT industry. Depending on the hardware environment that they run, these systems can vary from simple software applications (like a PC math calculator for example) to more complex ones, that have a big impact on our lives, like air traffic control or medical applications. This means that it is very important to create bug-free applications that work as intended by project owner. IT companies are facing the same challenges on every project that they develop: as the software system becomes larger and more complex, it becomes harder to manage the resources involved (which can consist in people like developers, designers, testers, business analysts or hardware resources like servers, etc.) and deliver the project with a good quality, in a small amount of time and for a lower price. That is why it becomes mandatory to develop software in a systematic way and use software engineering principles and its sub-disciplines: software requirements, software design, software development process, software quality, etc. Very important is to choose the right software development process based on the project requirements and the resources involved, so it is crucial to review and take into consideration all the software development methodologies and their strong points and weaknesses. This paper will focus on a BDD – Behavior Driven Development, a methodology that is not often used but it has some strong points that will be discussed in greater detail in the next chapters.

BDD stands for Behavior Driven Development, and even if sometimes it is defined as a software development process that emerged from TDD (test-driven development)[4], it makes more sense to consider it a set of practices which incorporates, builds on, and enhances ideas from other methodologies, such as TDD, DDD, Agile and Lean[1b]. The goal of this paper is to offer a presentation of the BDD process and its usages when developing a web application, while also presenting the BDD tools that we can use.

The reason I choose to approach this subject, is that I saw many times that non-technical stakeholders that are involved in software projects have difficulties in understanding application behavior because of the technical vocabulary. Also, many times it is hard to validate the requirements and be sure that the developers are building exactly what the project owner wants. That is why, based on my short experience working with BDD, I consider that it is a methodology that needs more attention and it is worth studying.

1.2 Context

There are a lot of software development methodologies out there, each one of them having its own advantages and disadvantages. Many times, it is hard to choose the appropriate one for a specific project. This requires a good analysis and an evaluation for all the project characteristics and business needs. Some of the most widely known software development methodologies are Waterfall, Prototyping, V-Model, Spiral, Lean and Agile, where Agile and its practices (XP, Scrum, Kanban, TDD, BDD, DDD, etc.) tend to be more used today.

We can see that the main two directions that are approached when it comes to software development processes are the classical Waterfall methodologies, for more enterprise and not so dynamic projects and the new Agile methods for projects where the business needs change very often. In the following lines, we will make a short review for each of them.

Waterfall is a linear methodology, in which progress is a sequential (non-iterative) process which follows through the phases of conception, initiation, analysis, design, construction, testing, production/implementation and maintenance. This methodology focuses on planning, time schedules, target dates, budgets and implementation of an entire system at one time. The lifecycle of the project is strictly controlled by using extensive written documentation, as well as through formal reviews and approval/signoff by the project owners. This approach is used predominantly by large, traditional enterprises doing large software projects. It appreciates simplicity — define requirements and produce results.

Agile is a software development methodology that describes a set of principles that is used by other methodologies, that emphasize on adaptive planning, evolutionary development, early delivery, continuous improvement, and flexible response to change. The Agile principles were first time described in 2001, in the *Manifesto for Agile Software Development*[1] (Agile Manifesto), which also defined the core values of Agile:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

As we can see from those promoted values, Agile encourages face to face communication over written documentation which minimizes miscommunication and

[1] The twelve principles of Agile software can be found at this URL: http://agilemanifesto.org/

misunderstandings. Therefore, Agile is more recommended when the project requires regular, ongoing development of upgrades and updates.

Here we can see the contrast between the two development methodologies. In waterfall, the coding doesn't begin until design of the entire software application is complete, and similarly testing doesn't begin until coding is complete. In the agile methods, the work is done in iterative steps, to allow more flexibility for changes. If an application is developed and designed via a traditional method, there could be the case of unforeseen problems that could appear after the development phase has begun. In case the work is done iteratively, the project team will be able to evaluate the requirements with each iteration, and therefore determine at each step which is the needed implementation to create a valid product.

In the software industry, we often see projects that are failing to deliver the functionality and value that business owner needs. According to a 2011 study "*CHAOS Report*" done by *Standish Group*[1a], 42% of the project were delivered late, ran over budget, or failed to deliver all the requested features, while 21% of the projects were canceled entirely. Another report based on *Scott Ambler's* survey about IT projects, found a failure rate of 30-50%, even if it has more relaxed definition of success. Based on this we see that a lot of effort is wasted, which translated in software that does not solve the problem that it was created for, money and time wasted. This implies that it is very important to manage carefully our resources and try to find the best solution when choosing the software development process. In future chapters, we will look at what BDD offers us to fix these problems.

1.3 Functional requirements

As stated in the previous subchapters, this thesis will emphasize on the BDD part when building a web application in PHP. It will be mandatory to use a BDD framework and apply the all the BDD principles. Next, we will express the requirements of our application.

1.3.1 Description

The project will consist of a web application that will help one or more faculties to manage their admission contest. The main goal is to facilitate and make easier the interaction between the faculty staff that will be managing the admission, and the students who wants to attend the admission. This means that the application will be focused on two types of users: the faculty staff user and the student. There will also be an admin user which will be handling the creations of faculties and faculty staff users. School staff users will be will be assigned to faculties and they will be able to create new admission contests and enter the admission details, like the number of budget/fee available slots. They will also be able to accept students to the admission and generate report and publish data when the admission is over. Student users will be able to register their account for free, by using public available registration form. Their account will then be verified by an admin user and marked that it has correct data and that the registered person is eligible to become a student. Then as a student you can attend an admission contest that is organized by a faculty. After a faculty staff user accepted the student, he will be enrolled in the admission process. At the end of the admission, the student will be notified via email and he will be able to visualize the results and his status (accepted/rejected).

If we summarize the elements described above, we can organize the requirements in the next 5 feature groups:

- Features about account management (login/register/edit profile/forgot password)
- Features regarding the admin area (for managing all the application entities)
- Features regarding the admission contests
- Features about admission reports and data exportation in various open formats
- Features available to the student account

The development will take place in BDD style by stating the behavioral specifications first via a Gherkin[2] syntax. More details will be provided in the next chapters.

1.3.2 Usage domains

This project is targeting the educational sector from a country or a community. The application will be used for the most part by the educational institutions, more exactly by their secretary or staff that will be handling the admission contest. It will also be used by the students who want to attend the admission. The application will allow a central group of admin users to be able to add more *faculties* and *faculty staff persons* who will be able to manage the admissions. So, we can imagine that the application could be ordered by the education ministry of a country. This is a perfect way to imagine the case where the project owner is the ministry and how BDD will be used in order make the development smoother and help focus on features that bring business value and engage high collaborations between all parties involved.

1.4 Technical approach

The web application will use the following technologies:

[2] Gherkin is a business readable, domain specific language used by BDD frameworks to express application features and usage scenarios

- **Server-side:**
  - **PHP**: server-side scripting language for the web
  - **Symfony**: web application framework written in PHP
  - **Doctrine**: several PHP libraries primarily focused on database storage and object mapping
  - **Twig**: templating engine for PHP
- **Client-side:**
  - **JavaScript**: client-side scripting language
  - **jQuery**: JavaScript library for DOM handling and AJAX interaction
  - **Bootstrap**: front-end framework
- **Testing:**
  - **PHPUnit**: unit testing framework for PHP
  - **Behat**: open-source Behavior-Driven Development framework for PHP
  - **Mink**: open-source browser controller/emulator for web applications, written in PHP 5.3
  - **Selenium**: test automation tool for web applications, which can replace repetitive tasks that usually are done by manual testing

1.5 Risks

The main risks that could appear during the writing of this thesis and the development of the web application could be the following:

- Lack of documentation for the software libraries that were chosen for developing the web application
- There could be difficulties when trying to integrate/configure the Behat BDD framework, with Symfony
- Misunderstandings of the BDD principles, and not applying them correctly in the development phase

After a quick research, it appears that we found sufficient documentation related to BDD and the software libraries, which consist in both online sources and books. Regarding the integration between Behat and Symfony, it seems that there already is a Behat plugin which will make the configuration easier.

2. Contributions

The goal of this paper is to offer a presentation of BDD process and its usages when developing a web application. We will cover the BDD principles, advantages vs. disadvantages, comparison with other software development processes and provide an example using Behat framework on a PHP/Symfony project. The final purpose of the PHP project is just to exemplify how BDD can benefit us and present the main PHP based tools that we can use.

The work that was put into developing this project as a whole - academic paper and practical application, can be separated in two parts. First, there was the work that was done for writing the academic paper, which implied research work, like reading books and online articles about BDD. All the informational resources that were used can be found in the bibliography. The work also included a lot of reading for summarizing and extracting main ideas, to organize the information about BDD and to be make it easier to read. The first chapter is presenting the motivation and current landscape of software development methodologies and why BDD can be important to be studied. We also described the practical project, evaluating risks and giving details about its main features and the technology stack that will be used. In the third chapter, we presented theoretical fundamentals about all the concepts used for writing the paper and for the development of the practical project. We covered topics from BDD basics to information about the used technologies like Behat framework and Symfony. In the fourth chapter, we presented how we analyzed and modeled the application, we created use case scenarios and diagrams to understand better what we have to build. In the fifth chapter, we presented step by step how we did the implementation, while in the last chapter we presented our conclusions. The second part of work, consists in developing the practical project, which consist in a web application developed in Symfony and Behat. For this development phase, it was a lot of work needed to read documentation to configure and setup the frameworks into the project. Also, to use the Gherkin language and to develop by the BDD principles it took a while to accommodate and understand.

In the next chapters, we will present all the characteristics of BDD, and make a comparison with the other development processes and emphasize which are the strong points and when we should use BDD. We will also present what languages, tools and frameworks can be used to implement BDD, by creating a PHP web application as an example.

3. Theoretical fundamentals

In this chapter, we will present the theoretical aspects regarding BDD (Behavior-driven development) and the development of web applications using its principles. We will also present the main technologies and software frameworks used in the by the practical project that we proposed.

3.1 BDD introduction

Behavior-driven development is a set of software engineering practices, which are focused on providing a common language for all the persons implied in the development of a software projects (developer, designer, tester, business analyst, project manager, project owner, etc.), that facilitate more effective communication between them, having the final purpose to focus on features that matters for the business and deliver more valuable, higher quality software faster.

BDD was invented and described by Dan North at the beginning of 2000s and in his article "*Introducing BDD*"[5], while he tried to apply and improve TDD in his software projects. It is important to note that BDD does not represent a new software development methodology, but it is a set of practices that have emerged from TDD (Test-driven development). Even if it takes a lot of principles from TDD and DDD (Domain-driven design), BDD also takes and enhance ideas from other methodologies such as Scrum, Kanban, Lean, etc. So, BDD will not replace any existing development process, but it can be used as an extra development layer.

The main steps in a Test-Driven Development are:

1. Write a test
2. Run the test (which will fail)
3. Write code to make the test pass
4. Refactor code
5. Restart with step 1

While having some advantages, many times it is difficult for development teams to adopt TDD. The main problems that TDD practitioners are facing are:

- Many times, it is hard to know where to start.
- It makes the developers to become too detail-focused and losing the knowledge of the main business goal that they need to reach.

- As the project size grows and it becomes more complex, it will acquire a large number of unit tests, which becomes very hard to maintain.
- Test methods do not have an intuitive naming that states what should be expected from them. This will make hard to understand what the code does and repair the tests in case they broke. Also, these a test method is tightly coupled on the method it tests, which means that in case the method needs to be refactored, also the test method will need refactoring.

One advantage that BDD brings over TDD is that TDD tests are tightly coupled to a method from code and the focus is on the method, while BDD is focusing on the overall behavior.

Next, we will compare and present the differences between a classic development process like Waterfall and the BDD practices. Let's assume that we need to build the faculty admission application that we presented in the first chapter. The project owner will be the faculty staff, which will provide the developing team the needed details about what they want from the new application. The developing team will be compound from developers, designers, testers and business analysts. If the used software development methodology is Waterfall, then the steps will look like this:

1. The faculty staff discusses with the business analysts and tells him what features the application needs and how they should work.
2. The business analyst translates the information gathered after the discussions with the faculty staff into requirements for the developers to implement. The requirements are written in English in a word document.
3. The developers translate the requirements into code and unit test – which will be written in a programming language.
4. The tester translates the requirements into test cases – which will be used to check if the requirements meets the implementation.
5. Documentation engineers then translate the working software features back into plain English documentation.

Below is a diagram which represents the steps presented previously and the flow between them:

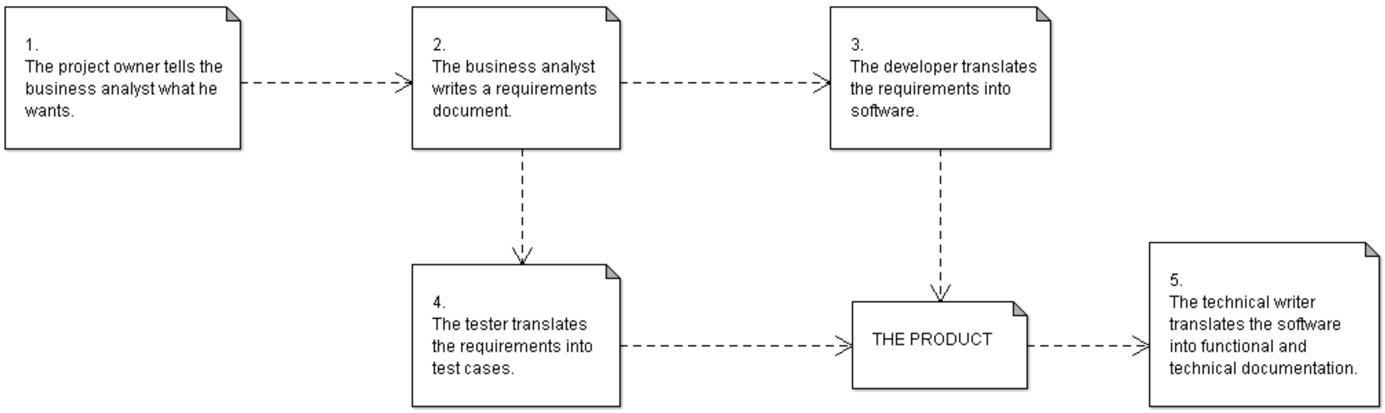

1.
The project owner tells the business analyst what he wants.
2.
The business analyst writes a requirements document.
3.
The developer translates the requirements into software.
4.
The tester translates the requirements into test cases.
THE PRODUCT
5.
The technical writer translates the software into functional and technical documentation.

Fig. 1: Diagram presenting the classical approach in a methodology like Waterfall

As we can see there are multiple points in those steps where information can be lost or misunderstood in translation. It is very possible that the implementation will not match what was required or that the final documentation will not be similar with the initial requirements.

In case we will use a BDD approach, the steps will be the following:

1. Similar with the Waterfall approach the business analyst will talk with the faculty staff, but this time the discussion will be focused on concrete examples, in order to eliminate the possible misunderstandings.
2. Before the work on writing requirements starts, the business analyst will have a meeting with the developers and testers that will be working on the project, and together they will discuss and translate key examples of the features that need to be implemented into a set of requirements, written in a special English-language format, called Gherkin.
3. The developer will use a BDD framework, which will transform the Gherkin requirements into a set of automated test that can be run against the application, and help objectively determine if a feature is implemented correctly or not.
4. The testers will use the results of the automated tests as a starting point for manual exploratory tests.
5. The automated tests will have a double role, they will act as low-level technical documentation, and provide up-to-date examples of how the system works. Later the reports can be reviewed and understand quickly which features have been implemented and which is the current status.

The next diagram represents the BDD steps and the flow between them:

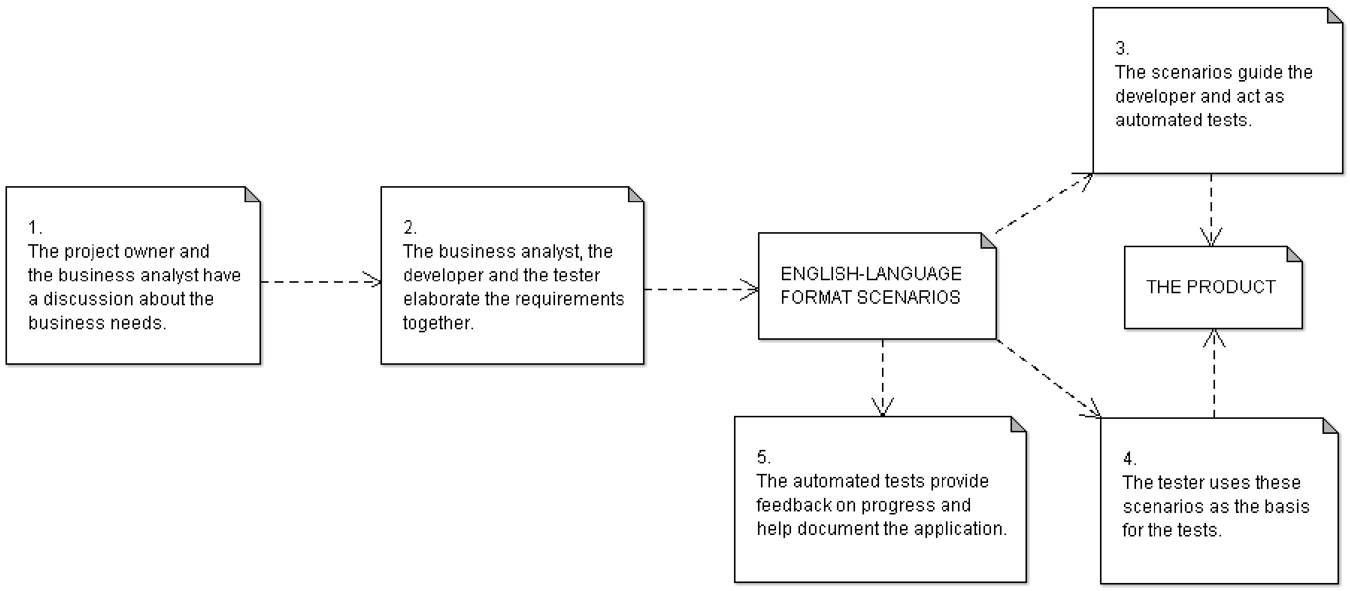

1.
The project owner and the business analyst have a discussion about the business needs.
2.
The business analyst, the developer and the tester elaborate the requirements together.
ENGLISH-LANGUAGE FORMAT SCENARIOS
3.
The scenarios guide the developer and act as automated tests.
THE PRODUCT
4.
The tester uses these scenarios as the basis for the tests.
5.
The automated tests provide feedback on progress and help document the application.

Fig. 2: Diagram presenting the BDD approach for the development cycle

The main differences between the two approaches, is that the second one makes heavy use of discussions around concrete examples, to reduce the amount of information lost in translation. Each step relies on the specifications written in Gherkin, that are based on the concrete examples discussed between the business owner and business analyst.

3.2 BDD principles and practices

To understand better the BDD principles and where they come from, it is important wo understand what problems BDD is trying to solve. The two most common problems in software engineering are: "*Not building the software right*" (which refers to failure that can appear because of technical and quality issues – like poorly designed or written code) and "*Not building the right software*" (which refers to not building the features that the business owner wants or the ones that brings him business value). The core BDD principles which overcomes these main problems and which are making BDD successful will be explained in the next subchapters. The information provided is adapted and interpreted from the book "*BDD in Action*" book by *John Smart*[1c].

3.2.1 Focus on features that deliver real business value

One of the main problems that developers can have, is the uncertainty about the requirements and the permanent changes that are need to be done as the project evolves. This means that it is crucial to focus on features that bring value to the business. The features represent a piece of functionality which will help the business to achieve its goals. For our application, some example of features that can help the faculty staff achieve the goals could be: "*Allow student to register online to eliminate the lines at the faculty offices.*", "*Allow the faculty staff to approve students that have registered.*", "*Allow faculty staff to generate the admission results and publish them in various open formats.*"

Another important rule of this principle is that the developing team does not have to set the requirements once and for all, but we should engage in discussions as the project develops, which in the end will help to build common understanding of what feature the developers should create. Also, it is important to not only accept and implement a list of features, but to engage with the project owner and understand the main business goals.

3.2.2 Work together to specify features

Using BDD implies that there will be a tight collaboration between the development team members, but also with the project stakeholders and the end users. When a team practices BDD, they will accumulate shared knowledge of the business environment and therefore they will have a higher engagement and attachment on the project. When the team works on the requirements to specify features, it will engage all team members not only the business analyst, which is a big plus.

3.2.3 Embrace uncertainty

In every software project, it is hard to foresee everything from the beginning, no matter how much time was spent planning. That is why BDD does not want to force the team to finish writing the documentation and lock the specifications at the beginning of the project, but instead it assumes that the understanding of the requirements will evolve during the life of the project. Also, it is important to get feedback from the project owners periodically – not only at the end of the project, so any misunderstanding and wrong implementation are eliminated. This means that features that are build should be sent to get feedback as soon as possible.

3.2.4 Illustrate features with concrete examples

When adopting BDD, a team should work tighter with the projects owners and the end users to write stories and scenarios of what is wanted from that feature to do. The team will have to define a set of concrete examples that will illustrate the needed result from the feature. Using examples, it is very important in BDD, because they are very effective in communicating precise requirements, which cannot be accomplish with natural language. Examples are also useful for clarifying misunderstand requirements.

3.2.5 Don’t write automated tests, write executable specification

The examples that are written by all the team members, will act as a base for the specifications that will be used to develop the system. Those specifications will represent both acceptance criteria and guidelines for developers. Most of the times, these acceptance criteria are transformed into automated test and executable specifications. An executable specification is an automated test that shows and verifies how an application delivers a particular business requirement. These automated tests have double role: acceptance tests and regression tests - as they can be run as part of the build process and also be run every time a change is made to the application.

3.2.6 Don’t write unit tests, write low-level specifications

BDD adopts an *outside-in* approach to write more maintainable code. The development will start from the acceptance criteria and then with writing the functionalities needed for the test to pass. Another important element is that the features have to contribute to the main goal and that not code should be written unless it contributes to make the acceptance test pass. When writing code, a BDD developer will have to think about the implementation as low-level executable specification and not just some unit tests for a class.

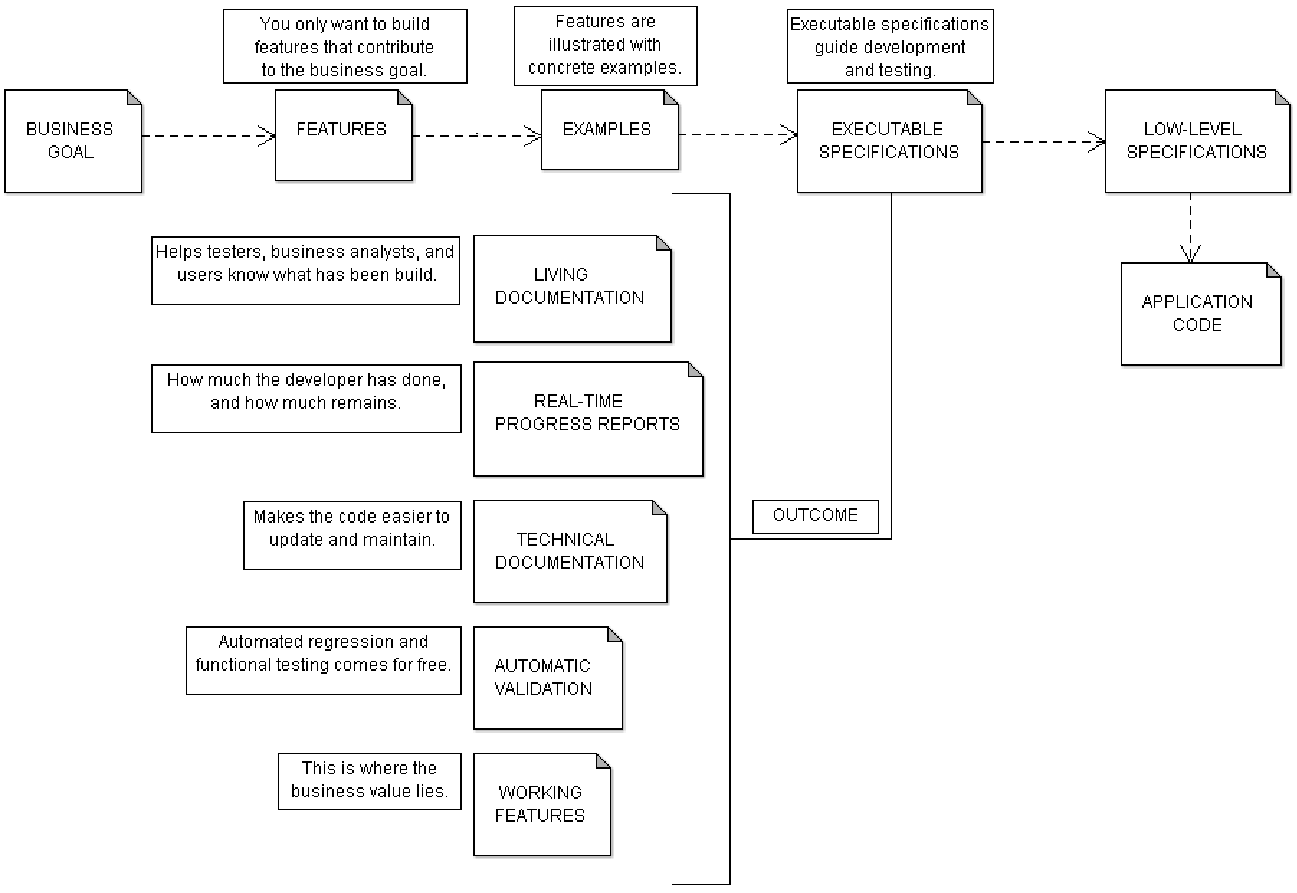
3.2.7 Deliver living documentation

The executable specification will be run to generate technical reports which becomes product documentation. The documentation will be always up to date and requires no maintenance. It also can be consulted easily to see how features works by future teams who will be working on the project.

3.2.8 Use living documentation to support maintenance work

The advantages of living documentation and executable specification will benefit the teams who will do maintenance work. The maintenance work will imply writing new executable specification or modifying existing ones. When a developer updates the application code, it could cause the tests to fail. There could be two reasons for this: the broken executable specification may not reflect the new requirements, which means that it could be deleted, or the new code has introduced a bug which affected an old feature. Living documentation will also be very helpful for iterative projects, where at each iteration the developer can view the current state of the project.

Below is a diagram which shows all the BDD practices and the relations between them:



BUSINESS GOAL
You only want to build features that contribute to the business goal.
FEATURES
Features are illustrated with concrete examples.
EXAMPLES
Executable specifications guide development and testing.
EXECUTABLE SPECIFICATIONS
LOW-LEVEL SPECIFICATIONS
APPLICATION CODE
Helps testers, business analysts, and users know what has been build.
LIVING DOCUMENTATION
How much the developer has done, and how much remains.
REAL-TIME PROGRESS REPORTS
Makes the code easier to update and maintain.
TECHNICAL DOCUMENTATION
Automated regression and functional testing comes for free.
AUTOMATIC VALIDATION
This is where the business value lies.
WORKING FEATURES
OUTCOME

Fig. 3: Diagram presenting the main activities and outcomes of BDD

3.3 BDD advantages and disadvantages

The main BDD advantages are:

- **Reduce waste**: BDD is focused on discovering and developing features that brings high business value, so any feature that does not bring value will be ignored.
- **Reduce costs**: Because of the reduced waste, it means that the development process will be focused just on the features that matters, so any additional cost is reduced.
- **Easier and safer changes**: The requirements changes will be easier to implement because of the living documentation and safer because of the executable specification, which will act as automated acceptance and unit tests.

- **Faster releases**: The releases will become more frequent as the large number of automated tests will eliminate some of the manual testing done by the testers. The testers will have instead more time to focus on other more complicated tests and scenarios.

The main BDD disadvantages are:

- **BDD requires high business engagement and collaboration**: BDD practices are very focused on discussions and collaboration between all parties involved. In case the business owners are unwilling or unable to engage in conversions and give feedback in time, then it will be hard to take advantage of BDD.
- **BDD works best in an Agile or iterative context**: One of the BDD principles assume that it is very hard to know all the requirements from the beginning and that we don't have to define them all in the first phase of a project, but instead the knowledge of the stakeholders will evolve during the lifetime of a project. That is why it is considered that BDD is well suited to be used in conjunction with Agile or iterative methodologies.
- **BDD does work not well in a silo**: In some cases, for example large projects developed by corporations, which delegates the work to multiple remote teams, will be harder for them to enable the high collaboration between teams.
- **Poorly written tests can lead to higher test-maintenance costs**: For some complex applications, it will require experience to design and write the automated acceptance tests. In case the application will acquire many poorly written tests, they will become hard to maintain.

3.4 Behat

Behat is a BDD framework that was heavily inspired by Ruby's[3] Cucumber project[4]. Starting from version 3.0, Behat is considered the official Cucumber implementation in PHP

[3] Ruby is a dynamic, open-source programming language: https://www.ruby-lang.org/
[4] Cucumber is a BDD framework for Ruby programming language: https://cucumber.io/

and it is part of one big family of BDD tools. So, many of the concepts that are described in the next subchapters will be valid for other Cucumber projects.

3.4.1 Behat introduction

Behat is an open-source Behavior-Driven Development framework for PHP, that helps delivering software by following BDD principles and focus on continuous communication, deliberate discovery of features that bring business value and test-automation. As any BDD framework, Behat is focused on communication and building the right system from the beginning, instead of proving that system was build right as other methodologies do. Behat[6] was built by the PHP community, and its codebase heavily uses Symfony components. Behat is very extensible, every part of the framework can be enhanced or replaced via the extension system. Also, the community developed many important extensions like: MinkExtension[9] and Symfony2Extension[10] which integrates Behat with Mink[8] and Symfony.

Behat can be installed in a PHP project via Composer[5]. The main configuration file for Behat is called *behat.yml* and it is placed in the project root. The file will define where the **.feature* files and the context classes will be stored. The file will also contain the configuration for Behat extensions are enabled.

3.4.2 Features and scenarios

Behat features are represented by **.feature* files which conventionally contains a simple Gherkin feature. A feature starts with a line which starts with the keyword "*Feature:*" and will contain one or more scenarios. A feature definition will contain some text that will be ignored by the parser, which will have the role to describe the feature, the business value that it brings and who will benefit from it. We can write anything that we want until the first scenario.

Every scenario consists of a list of Steps, which must start with one of the keywords "*Given*", "*When*", "*Then*", "*But*" or "*And*". An important note is that Behat treat all the same, but as developers we shouldn't, because the keywords have different semantical meaning.

3.4.3 Gherkin

Gherkin is a business readable, domain specific language, which has a similar format with plain English and that lets you describe the software's behavior without giving details how the implementation should be done. The two main purposes for Gherkin are to provide

[5] Composer is dependency manager tool for PHP https://getcomposer.org/

documentation and automated tests. Gherkin files have **.feature* extension and the source code contains the description of a single feature. The language was created as part of the Cucumber project - which was initially a BDD framework for Ruby, but now it represents a family of framework available for many languages.

Gherkin's syntax is a line-oriented language, being similar with YAML[6], Python or Ruby, either spaces or tabs can be used for indentation. The lines will start for the most part with a keyword, like: "*Feature*", "*Scenario*", "*Given*", "*When*", "*Then*" and "*And*". Line comments are starting with "*#*" character, everything after it being ignored. The Gherkin parser divides the file input into features, scenarios and steps[11]. When the file is run via a BDD framework, each step will be matched via regular expressions to a PHP function (in our case - because we develop in PHP), which will execute a certain action that will pass or fail.

This is a Gherkin source code file, taken from our project *login.feature* file:

```
Feature: Login
  In order to have access to all application features and use them
  As a guest user
  I need to be able to login

  Background:
    Given I am on "/"

  Scenario: As a guest user, I want to login into my account
    When I follow "Login"
    Then I should be on "/login"
    And I should see "Login"
    And I should see "Forgot your password?"
    When I fill in "username" with "emi.berea+student-1@gmail.com"
    And I fill in "password" with "12345"
    And I press "Log in"
    Then I should be on "/"
    And I should see "Edu Admission"
```

Table 1: Example of a *Gherkin* file

First line starts the feature, lines 2–4 are unparsed text, which is expected to describe the business value of this feature. Line 6 starts a background, while line 9 starts a scenario. Lines 10–18 are the steps for the scenario. Line 15 starts next scenario and so on.

3.4.4 Writing scenarios and Gherkin steps

To write good scenarios, we will need to take count of the semantical meaning of the Gherkin keywords:

[6] YAML is an acronym for "YAML Ain't Markup Language" and represents a human friendly data serialization standard for all programming languages http://yaml.org/

- **given**: the purpose of this step is to be a precondition and put the system in a known state. Common examples are: set database ("*Given the database is clean.*") or authenticate user ("*Given I am logged in as 'user1'*").
- **when**: describes the key action the user performs. Common examples: interact with webpages, fill forms, call CLI[7] processes.
- **then**: the purpose of this step is the observe outcomes. The observations should be related to the business value/benefit stated in the feature description. The observations should inspect the output of the system (user interface, message, command output).
- **and** & **but**: in case a scenario contains several consecutive "*Given*", "*When*" or "*Then*" steps, they can be linked via "*And*" or "*But*". Note that Behat interprets "*And*" and "*But*" same as the other, meaning it doesn't differentiate between them.
- **background**: they are a special case of scenarios, which allows to add context to the other scenarios from a feature. The difference between a simple scenario and a background is that the background is run before each scenario, but after the "*@BeforeScenario*" hook.
- **scenario outlines**: they represent a construct which allows to run the same scenario multiple times with different input values. This is useful because without this feature, we must copy the same scenario multiple times for each set of arguments, which will generate duplicate code. The values are specified as a text table under a "*Examples:*" keyword.
- **tables**: they have the same syntax as the scenario outlines, but they are used for specifying a larger data set - usually as input to a Given or as expected output from a Then.
- **tags**: they are a syntactic construct that allows to organize feature and scenarios. They are put on a separate line before the "*Feature:*" or "*Scenario:*" declaration.

3.4.5 Mink PHP

Mink is an open-source browser controller/emulator for web applications, written in PHP 5.3. Mink is a wrapper library which removes API differences between different browser emulators providing different drivers for every browser emulator and providing you with the easy way:

[7] CLI is acronym for Command Line Interface

- to control the browser
- traverse pages
- manipulate page elements
- interact with pages

A driver is like a layer between Mink API and a certain browser emulator, and it represents a PHP library that can be installed with Composer. Mink does not have any driver installed by default, but instead we should install what we need. The most used Mink Drives are:

- GoutteDriver - behat/mink-goutte-driver
- Selenium2Driver - behat/mink-selenium2-driver
- BrowserKitDriver - behat/mink-browserkit-driver
- ZombieDriver - behat/mink-zombie-driver
- SeleniumDriver - behat/mink-selenium-driver
- SahiDriver - behat/mink-sahi-driver
- WUnitDriver - behat/mink-wunit-driver

There are two different categories of browser emulators:

- **Headless browser emulators**: they are simple pure HTTP specification implementations, like *Goutte*. Those browser emulators send a real HTTP requests against an application and parse the response content. They are very simple to run and configure, because this type of emulators can be written in any available programming language and can be run through console on servers without GUI[8]. Headless emulators have both advantages and disadvantages. Advantages are simplicity, speed and ability to run it without the need of a real browser. But this type of browsers has one big disadvantage, they have no JS/AJAX support. So, you can't test your rich GUI web applications with headless browsers.
- **Browser controllers**: they simulate user interactions on browser and can retrieve actual information from current browser page. *Selenium* and *Sahi* are the two most famous browser controllers. The main advantage of browser controllers usage is the support for JS/AJAX interactions on page. The

[8] GUI acronym for Graphical User Interface

disadvantage is that such browser emulators require the installed browser, extra configuration and are usually much slower than headless counterparts.

3.5 The web

We will start by presenting what the web is, reviewing its main characteristics, and after that we will define what a web application is.

The web, also known as WWW (World Wide Web) is an internet service, which represents some informational systems, that we can access it without knowing the differences between the data sources. The three main key elements of the web are the hypertext, URIs and the HTTP protocol. The hypertext is a type of electronic documents that contains links (hyperlinks to other hypertext documents). The URIs (Uniform Resource Identifier) represents a character string, which follows a certain format and is uniquely identifying a web resource. The identification can be made based on the name (URN – Uniform Resource Name) or on the location (URL – Uniform Resource Locator). HTTP (HyperText Transfer Protocol) is a protocol based on the client-server model, which allows the access on the web content. The main goals of the web are to ensure the independence against the device and the software layer and to provide scalability and multimedia content.

A web application is represented by a series of interconnected web pages, which has the content generated dynamically in the server-side, with the purpose of offering a specific functionality to its users. In our case, the application will be an ERP (Enterprise Resource Planning), which will allow schools and universities to organize their admission contest.

3.6 Development environment

To obtain the paper goal and create a web application using BDD principles, we used a series of technologies and tools that are specific to the web. We will describe in detail what technologies we have used, and the reason behind it together with the role that it had in our application.

3.6.1 The LAMP stack

LAMP is an acronym used for naming a set of open-source software components, which are used to host and run web applications. The main four components from the LAMP stack, that are needed to run a web application, are: an operating system, a web server, a database server and a programming language, which is needed to generate dynamic content on the server-side. Very often, those components are represented by Linux (operating system), Apache (web server), MySQL (database server) and PHP (scripting language)[13]. These are the components that were used initially, but there a lot of other variations, where each element of the stack can be replaced by a similar software. For example, Apache can be replaced by Nginx or lighttpd, the database server can be replaced by MariaDB or PosgreSQL (in which case the stack will be named LAPP) and the scripting language could be replaced by Pearl, Python and more recently by Ruby and node.js.

Next, we will present the main characteristics of each component that we used:

- **Linux** is a UNIX based operating system for computers, that first appeared at the beginning of the 1990 and which is open-source developed and distributed. Linux is available to the end users through its distributions, so we can say that Linux is a family of operating systems. Its main component, which all distributions are using it, is the *Linux kernel*[9]. While there are versions for both desktop PCs and servers, the second one is where Linux is more popular and used. For this project, we used the Ubuntu distribution, installed on a virtual machine.
- **Apache** is an open-source web server. Its role is to receive and respond to HTTP request, usually at the port 80. So, this component is relying on one of the three main components of the web, the HTTP protocol, which we described in a previous subchapter.
- **MySQL** is an open-source relational DBMS (Database Management System), being one of the most popular and largely used today. MySQL is available for many operating systems (such as Windows, Linux, OS X, Solaris, etc.) and compared with other DBMS, it is mainly used for web applications and optimized for them.
- **PHP** is a server-side scripting language, which is mostly used in the LAMP stack as the first option in front of Perl and Python. More details about PHP will be given in one of the next subchapters.

3.6.2 Composer

Composer is a command line tool, which manager software libraries that are written in PHP. Besides being a package manager (which allows to install and organize libraries in a standardized way), Composer is also a dependency manager, which detects if a library depends on another library and it installs automatically all the dependency tree. The tool downloads the libraries in a folder called *vendor/* inside the project root folder and automatically autoloads the libraries and make their API accessible through PHP namespaces to our custom source code.

[9] Linux kernel - https://www.kernel.org/

Composer was launched in march 2012, its features being inspired from **node.js**'s package manger *npm*[10] and **Ruby**'s *bundler*[11]. The main repository where libraries that can be installed with Composer is called *Packagist*[12].

The most used commands are:

- *composer install* – installs all the libraries according to *composer.lock*
- *composer update* – updates all libraries to the highest version available as configured
- *composer require <vendor/package>* - installs a new library and adds it automatically in *composer.json*, where *<vendor/package>* is the name of the package
- *composer update <vendor/package>* - installs or updates certain library

In our project, we used Composer to manage the Symfony application and install its libraries and bundles, including the BDD framework Behat.

3.7 PHP programming language

3.7.1 What is PHP?

PHP is a server-side scripting language, which was specially designed for building dynamic web pages. PHP is available for installation on every major operating system and it is used as the first option for server-side language of the LAMP stack, being more popular than Pearl and Python. The name PHP is a recursive acronym, which stands for "**P**HP: **H**ypertext **P**reprocessor" and that was used since the third version, which during the first two versions PHP stood for "**P**ersonal **H**ome **P**age".

3.7.2 Syntax

PHP has a syntax very similar with C, to which they added features needed for developing web applications. The main control instructions are as expected: if, switch, for, which, do while. PHP code can be included in HTML files or it can be used in "*.php*" files which contain only PHP code. In order to be executed by the interpreter, the code must be

[10] npm – Node Package Manager https://www.npmjs.com/
[11] Bundler – package manager for Ruby http://bundler.io/
[12] Packagist – repository for composer https://packagist.org/

included inside PHP tags: "*<?php // code here ?>*". PHP is a weak typed language (also known as dynamically typed), meaning that the variable typed is checked at runtime and the programmer does not have to specify the variable type, because it will be guessed by the interpreter, while in many cases it also deal with the data type conversions. PHP also offers a lot of predefined functions: data type handling and conversion, math, strings, arrays, date and time, resource access and file manipulation, URL and database processing, etc.

3.7.3 PHP frameworks

To facilitate the rapid development of web applications, the PHP community developed a series of web frameworks. The most popular and used frameworks today are: CodeIgniter, CakePHP, Laravel, Symfony, Yii, Zend, etc. In the next section of this chapter, we will present Symfony, the framework that we chose for developing the practical project for this paper regarding BDD.

3.8 Symfony framework

3.8.1 Introduction

Symfony is an open-source web framework, written in PHP. Symfony represents a set of software components that can be quickly integrated in any project and a series of development methodologies that assures the stability and maintenance of the applications. Symfony was developed initially by *SensioLabs*, a web agency from France, and it was launched initially in October 2005. Currently the latest version on the Symfony 2 branch is 2.8, while on the Symfony 3 branch is 3.2, as of February 2017. The framework is supported by large community of programmer, which are contributing to the main GitHub repositories and the other libraries.

3.8.2 Folder structure

A clean installation of Symfony2 framework, will generate the following folder structure:

- ***app/***: directory which contains configuration files, in YAML format for the most part
- ***src/***: directory which contains the custom application code

- ***vendor/***: directory which contains the third-party code, installed via Composer, including Symfony framework and all other libraries
- ***web/***: directory which should be configure as web server root and accessible via the browser

As a note, we presented the Symfony2 directory structure, as we are using the Symfony2 for developing this project, which has a little different structure compared to Symfony3. The *web/* directory will contain public front-end resources (such as CSS, JavaScript and images) and the front controller scripts that correspond to each Symfony environment ("*prod*, "*dev*" and "*test*"). Symfony uses the *Front Controller* pattern (that is specific to web applications), which consists in having a single-entry point into the application via the URL. The *app/* directory contains beside configuration files, the *AppKernel* class, which wraps a HTTP request received from the client and propagates it to the framework routing system. Also in this class are registered the bundles, from which the application is created.

3.8.3 Symfony components

A Symfony component is a standalone software library, written in PHP, which implement a specific functionality that is needed to develop a web application. These components represent the foundation of the Symfony full-stack framework. Version 1.0 of the framework could be used just as full-stack, while version 2.0 introduced the concept of components, making the framework more flexible. This means that somebody could use just some of the components for a project, being able to install them separately. So, we have the option to use Symfony full-stack (with all its components) or we can use just the components that we need, being free to implement the architecture of our project based on our requirements and business needs. Some of the most important components are the following:

- **Form** – provides support for creation, processing and reuse for HTML forms
- **HttpFoundation** – OOP interface for working with HTTP protocol
- **HttpKernel** – provides tools for creating a web framework based on HTTP
- **DependencyInjection** – provides methods to standardize and centralize the creation of objects
- **EventDispatcher** – provides the ability that the application modules can communicate more easy, via the events and listeners
- **Routing** – maps an HTTP request to a set of configuration variables

- **Security** – provides a security system over the resources, through authentication and ACL[13]
- **Templating** – provides tool necessary to build a templating system
- **Yaml** – parses YAML files and converts them into PHP arrays

Many of the Symfony components are used by other frameworks (such as Laravel) and CMS (such as Drupal, Sylius, etc.), a complete list of open-source project that are using Symfony components can be found on the Symfony website[16].

3.8.4 Symfony bundles

A Symfony full-stack application is compound from a group of bundles. A bundle is a directory which contains code which implements one module of the application, which provides a single important functionality. In Symfony almost everything should be included inside a bundle. Bundle can be classified in the following types:

- Bundles from the Symfony full-stack Standard Distribution, which completes and links the components, that were described in the previous subchapter
- Third-party bundles, which are found in the *vendor/* directory and which can be installed via Composer. They are offering some base functionalities that are already implemented and which can be override by the programmer, to fit the requirements of the application. Usually we are not allowed to modify the code of those bundles (as they are from *vendor/* forlder), but we can override them in our own bundles
- Bundles from the *src/* folder, which contains our custom code specific to our application.

The directory structure that a bundle can contain, is the following:

```
• Acme/...
    o HelloBundle
        ▪ HelloBundle.php
        ▪ Controller/          - controller classes
        ▪ Resources/           - resources sub-directors
            • meta/
                o LICENSE
            • config/          - configuration files
            • doc/             - documentation files
                o index.rst
            • translations/    - translation files
            • views/           - views
            • public/          - public accessible files
        ▪ Tests                - test files
```

[13] ACL – Access Control Lists

Inside *src/*, the first directory usually should be a vendor name, such as an acronym or the name of the company that is developing the application. This can be useful in case there are multiple companies that are working on the same project. Then this directory, can contain other nested folder, but the last one must be suffixed with "*Bundle*" – which in fact will represent the bundle. The folder hierarchy used inside *src/*, will also be used as PHP namespaces. A bundle will also have a name assigned, which is formed through the concatenation of the names of all upper folders from *src/*. For example, in our case, the name for the bundle will be: *AcmeHelloBundle*.

Also inside a bundle we can have other directories, based on the directories that it contains. More details can be seen in next table:

Type	**Directory**	**Details**
Commands	`Command/`	`Symfony commands`
Controllers	`Controller/`	`Controller classes`
Service Container Extensions	`DependencyInjection/`	`Configuration files for services`
Event Listeners	`EventListener/`	`Classes which defines events and listeners`
Configuration	`Resources/config/`	`Configuration files in XML or YAML format`
Web Resources	`Resources/public/`	`Publicly accessible files (CSS, JS, images, etc.)`
Translation files	`Resources/translations/`	`Translation files`
Templates	`Resources/views/`	`Views - HTML and Twig files`
Unit and Functional Tests	`Tests/`	`Test files, usually written in PHPUnit`[14]

Table 2: Optional directories in a Symfony bundle

The main Symfony full-stack bundles are:

- **FrameworkBundle** – links and correlates all the Symfony components described in a previous subchapter
- **SecurityBundle** – provides security for resource access
- **TwigBundle** – provides integration between Symfony and Twig[17] templating engine

[14] PHPUnit: unit testing framework for PHP language https://phpunit.de/

- **WebProfilerBundle** – provides debugging information for developers
- **SwiftMailerBundle** – wrapper over the SwiftMailer PHP library[18], that provides operations with emails
- **MonologBundle** – wrapper over monolog PHP library[19], which provides logging
- **AsseticBundle** – management for public front-end resources
- **DoctrineBundle** – provides support for working with Doctrine ORM[20] and databases

3.8.5 Symfony distributions

Because of its decoupled architecture which is based on components and bundles, it means that the developers can create Symfony distributions, that are formed from a selection of components and bundles in order to offer a functionality needed by a specific category of applications. The base distribution that is suited for most part of the project and which we also used in our project is *Symfony Full Stack* (also known as *Symfony Standard Edition*) which contains all base components and bundles. Other distributions that have added or removed bundles in order to offer new functionalities are: *Hello World Edition*, *Symfony CMF Standard Edition*, *Symfony REST Edition*, etc.

3.8.6 Particularities

One of the main characteristics of Symfony framework is its decoupled architecture, given by the components and bundles. As a good practice guideline for projects that are using Symfony, it is recommended to organize the application as a set of decoupled services, which can be called from the other modules of the application. By following this practice, we will have to put the logic inside service classes, instead of putting them directly in controller classes. This principle is called "*fat model – slim controller*", which states that the purpose of a controller is just to receive a request, call a service with the request data, get the result from the service and then send back the final response back to the web client. So, Symfony does not uses a classical architectural pattern like MVC[15], but it has an architecture based on services, being a request – response framework.

Another element that makes Symfony special compared to other PHP frameworks is the usage of open-source libraries in the full-stack distribution, along with its components. Here we are referring to Doctrine ORM, Twig templating engine and SwiftMailer library.

[15] MVC – Model View Controller

We will present the Doctrine libraries in detail in the next subchapter, for now we will present only the integration with Symfony. To summarize Doctrine has the role to map the records from the database to PHP objects. This can be obtained through the Doctrine entities, which are PHP classes that contains fields and getter/setter methods, which will be mapped to the table columns from the database. This mapping is created through the metadata, which can be represented in many formats. We choose to use PHP annotations because they are easier to visualize and manage. Through these annotations, we can define the link between the object property and the table column, stating the data types or we can define relations between the entities (and of course between tables).

3.8.7 Symfony – integration with Doctrine ORM

Doctrine is a set of PHP libraries which consists of DBAL (Database Abstraction Layer) and ORM (Object-Relational Mapping). DBAL is an API[16], based on PDO (PHP Data Objects), which abstracts the communication between the application and the different RDBMS (MySQL, PostgreSQL, etc.), so we don't have to care about the different SQL dialects. ORM is a software component, which has the role to map or create a link between the data types available in an OOP language and the data types that are available in a RDBMS. Doctrine ORM is built on top of Doctrine DBAL, so it can be used with a large number of database servers. Like any ORM, the main advantage, that it brings to an application, is that the developers will not work directly with the database data types and they will not use SQL (Structured Query Language) directly in the code or other database specific elements, but they will be able to work with a complete object-oriented API, which will make much easier to work with databases. Also, Doctrine ORM brings new ways to interrogate a database, via a new SQL dialect called DQL (Doctrine Query Language), which allows using objects and their properties in the interrogation statements, instead of the tables and columns of the database. Doctrine can be used in any PHP project and it can be installed via Composer, but it is also installed from default in a number of PHP frameworks.

As we mentioned in the previous chapter, Doctrine is preinstalled in Symfony. Inside a Symfony bundle, we will have related to Doctrine two folders: */Entity* and */Repository*. In */Entity* we will have the entity classes, which will map the objects to the tables from the

[16] API – Application Programming Interface

database. The mapping can be written in multiple file formats, similar with the Symfony configuration files: PHP annotations, YAML or XML. In */Repository* directory will contain repository classes, which will contain PHP methods, which will contain DQL statements, that can be executed against the database.

4. Analysis and modeling

In this chapter, we will analyze the requirements and what we want to accomplish in this project, we will create usage scenarios, to see the application through the eyes of the final user. We will identify the main modules of the application, and we will see what problems could appear, and how we can find solutions for them. Also, we will model in detail the application modules based on the requirements.

4.1 Project requirements

Synthesizing what was mentioned in the first chapter, where we briefly described the project, we will build a web application, which goal will be to help schools and its attending students to communicate more easily and make the admission information more accessible to everyone. From now on we will refer to the three types of users of our application as: ***admin***, ***school staff user*** and ***student***.

Starting from the base requirements stated above, we can organize the requirements in the next 5 groups:

- **Features about account management:**
  - Login
    - all 3 user types should login via a "*/login*" URL
  - Register
    - *admin* and *school staff users* are registered via the admin area, by another admin user
    - a *super admin* user is created from the application installation script
    - student users will register themselves via a public link "*/register/student*"
  - View/update profile
    - each account will have a profile page
  - Change password
    - each account will be able to restore its password
- **Features regarding the admin area (this should be available for the admin user):**
  - create schools (faculty, high school, etc.)
  - create *school staff users* and assign then to schools

    - verify and accept the registered *students*
    - manage all the database entities (tables) via CRUD[17] forms
- **Features regarding the admission contests (this should be available for the school staff user):**
    - create new admission contest for the faculty that the user is assigned
    - the system should have a validation mechanism, so there are no duplicates admissions (that take place at the same time)
    - the admission will contain the date (month and year) when it takes place, the maximum number of budget/fee financed slots and a list with all the attending students
    - the admission will have a list of statuses like: open, processing results, closed
    - accept the students that have requested to attend an admission
    - a student that will want to attend will have a list of statuses like: requested and accepted
    - manage admission, run reports when the admission is finished and publish results
    - the school staff user will receive a notification when a student
- **Features about admission reports and data exportation in various open formats:**
    - after an admission has the status closed, the results should be published publicly (without requesting login via an account)
    - also, the results should be downloadable in various open formats, like XML and JSON
- **Features available to the student account:**
    - registration via a public link /register/student
    - accepted by the admin users
    - being able to visualize the open admissions and the closed admissions that he participated at
    - ability to attend an open admission and see when it was approved by a school staff user
    - a student that will want to attend will have a list of statuses like: requested and accepted
    - visualize the results when they are available

[17] CRUD is acronym for "create, read, update and delete" and represents the operations that can be done via HTTP or via database

- receive notification when his account is accepted by the admin and when the admission results are available
- visualize the history of all his participations to admission contests

Now that we defined the requirements as a detailed list, we have made out first step to understand what we must build. We can assume that this list was compiled after a discussion between the business analyst and the project owner. Next, we will start developing use scenarios, which will be done by all the team members of a project that follows BDD. This will help the development team to understand how the end user will interact with our application.

4.2 Use case scenarios

In this subchapter, we will elaborate some scenarios, which will help us understand the way our application will work and the way it will be used by the end user. To illustrate the usage scenarios, we will use UML diagrams. **UML** (**U**nified **M**odeling **L**anguage) is a general-purpose modeling language, used in the development of software products for visualizing the way software systems are designed and build. UML diagrams can be classified in 3 main types: structural, behavioral and interaction. To visualize the scenarios, we will use a type of behavioral diagrams named ***Use Case*** (or scenario) diagrams, which will capture the system requirements. The ***Use Case*** diagrams presents the system requirements from the end user point of view, by using as its main graphical elements: *actors* (external entities with which the systems interact), *use cases* (system functionalities) and *relations* (dependencies between the first 2 elements).

Next, we will present some scenarios, which will describe the way our application will be used and the way it will interact with the user.

4.2.1 Registration for the student user

1. Objective/Context

The user is accessing the application via a web browser and he will visualize the home page as an anonymous user, because he does not have an account yet. Here he can access the login or register pages. In case he clicks on the “Login” link, then the application will display a login form, where the user needs to enter his credentials to authenticate. Besides the login form, the user has the option to access the register page in case he does not have an account yet. In case he clicks on the “Register” link, then the application will display a registration form.

2. Scenario/Steps

1. The user completes the requested data (email, password, first name, last name and pin – personal identification number) in the registration form fields.

2. The application takes the form data, process it and then sends a conformation email to the user. He will access the URL that he received in the email, which will activate his account and then redirect the user to the application, where he will be automatically authenticated.

3. In the end the user will be able to access his account, but he will need to be approved by an admin user.

3. Extensions

If the data that was introduced by the user is invalid, then an error message will be displayed regarding the incorrect information and the user will be asked to provide again that data.

4. *Use Case* diagram

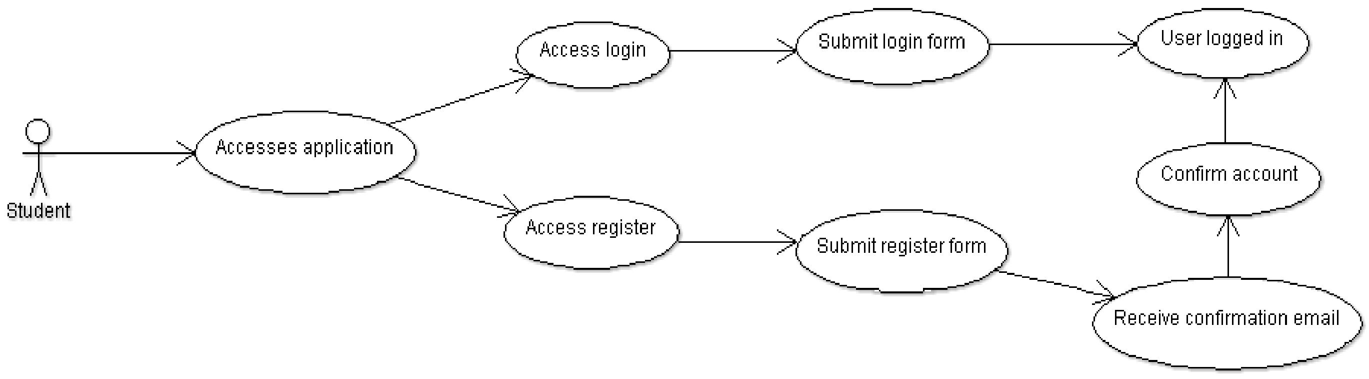


Fig. 4: Use Case diagram which represents the student login workflow

4.2.2 Admin user accepts newly registered student user

1. Objective/Context

The student user has his account enabled, but at this moment he cannot attend new admission contests because he must be approved by the admins first.

2. Scenario/Steps

1. After the student creates his account, a notification will be send to all the admin users, via email and the application notification system.

2. One of the admin users can review the notification and visualize the student profile to verify if the information provided by his profile is correct.

3. If everything is ok, then the admin can approve or disapprove the profile. On case the profile is approved, the student will receive a notification that he was accepted and that now he can attend admissions. Otherwise the student will receive a notification to review his profile.

3. Extensions

We can let the admin user to update the student profile instead, which the student will be just notified about the information that was corrected.

4. *Use Case* diagram

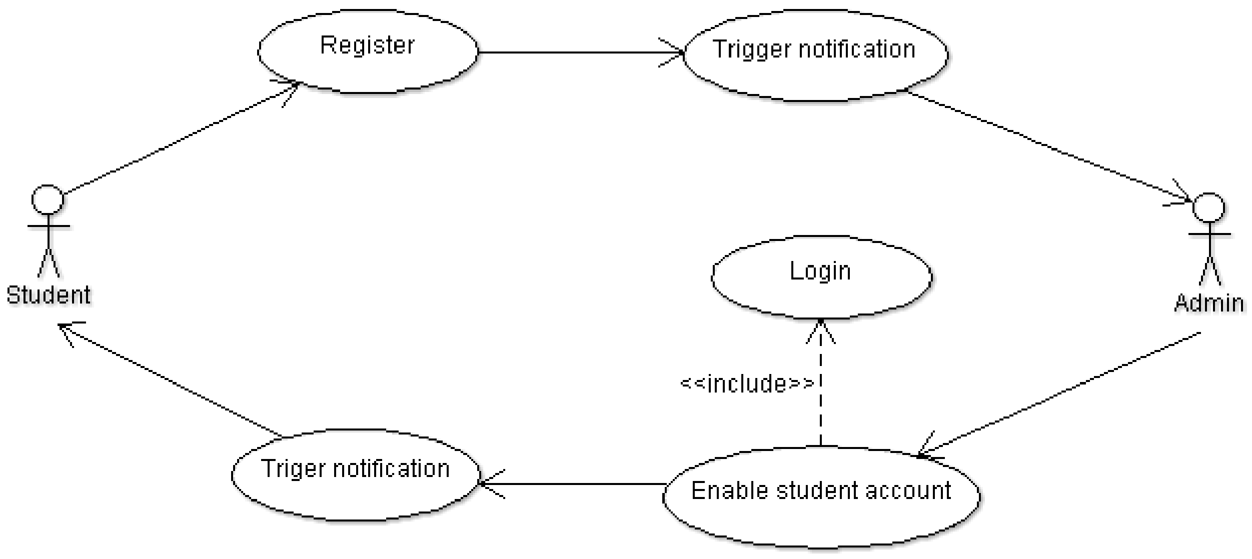


Fig. 5: Use Case diagram which represents the admin accept new student workflow

4.2.3 As an admin user create a school and assign to it school staff users

1. Objective/Context

The admin user is logged into the application and he is viewing the admin dashboard.

2. Scenario/Steps

1. The admin accesses the "Create school page". Then he completes and submits the form.

2. The page will redirect to the "Schools dashboard", where he can view a table with all the schools available in the system.

3. Then the admin will access the "Create school staff user". Then he completes and submits the form.

4. The page will redirect to the "School staff user dashboard", where he can view a table with all the school staff users available.

5. The newly created school staff user will receive an email with a confirmation link, which will redirect him to a page where he can enter his password and gain access to his account.

3. Extensions

The forms for creating school/staff user will display validation messages in case the entered data is not valid.

4. *Use Case* diagram

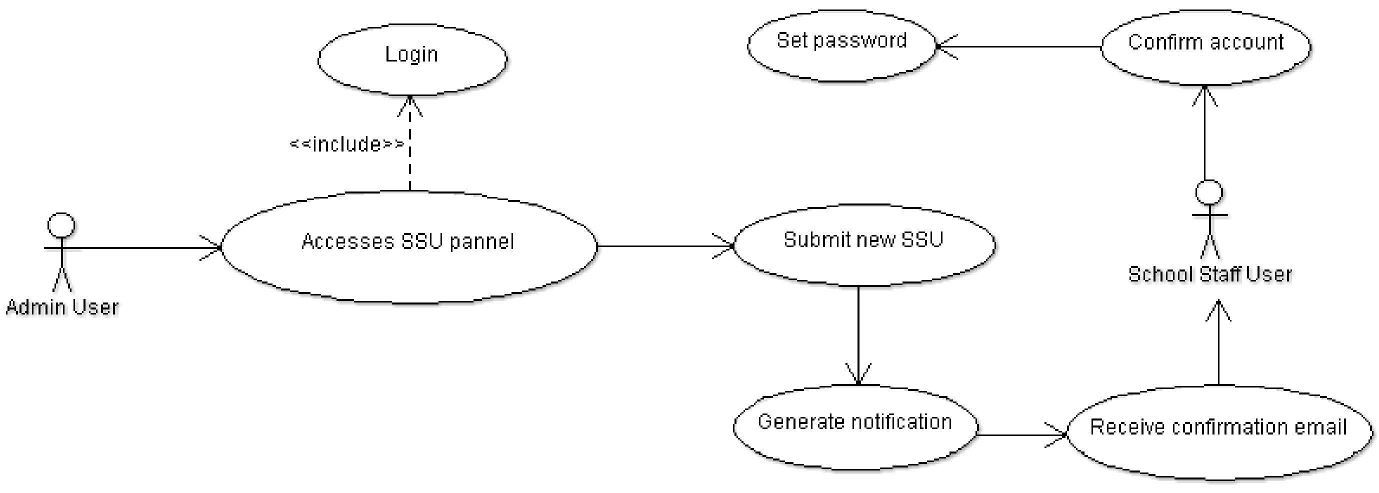


Fig. 6: Use Case diagram which represents the creation of a School Staff User

4.2.4 Create an admission contest as a school staff user

1. Objective/Context

The school staff user is logged into his account and accessed the “Create new admission” page.

2. Scenario/Steps

1. The school staff user will complete the form for an admission and he must complete the form with the following details: session date, budgets financed slots, fee slots, budget – free threshold and fee – rejected threshold.

2. Now the admission becomes available to students to attend.

3. A student attends the admission.

4. All the student staff users get notified, and one of them can accept the student.

5. The student will be notified that he was accepted, and the admission will appear in his account as attended.

3. Extensions

The school staff user will be able to talk with the student in case he has questions via an internal mail system.

4. *Use Case* diagram

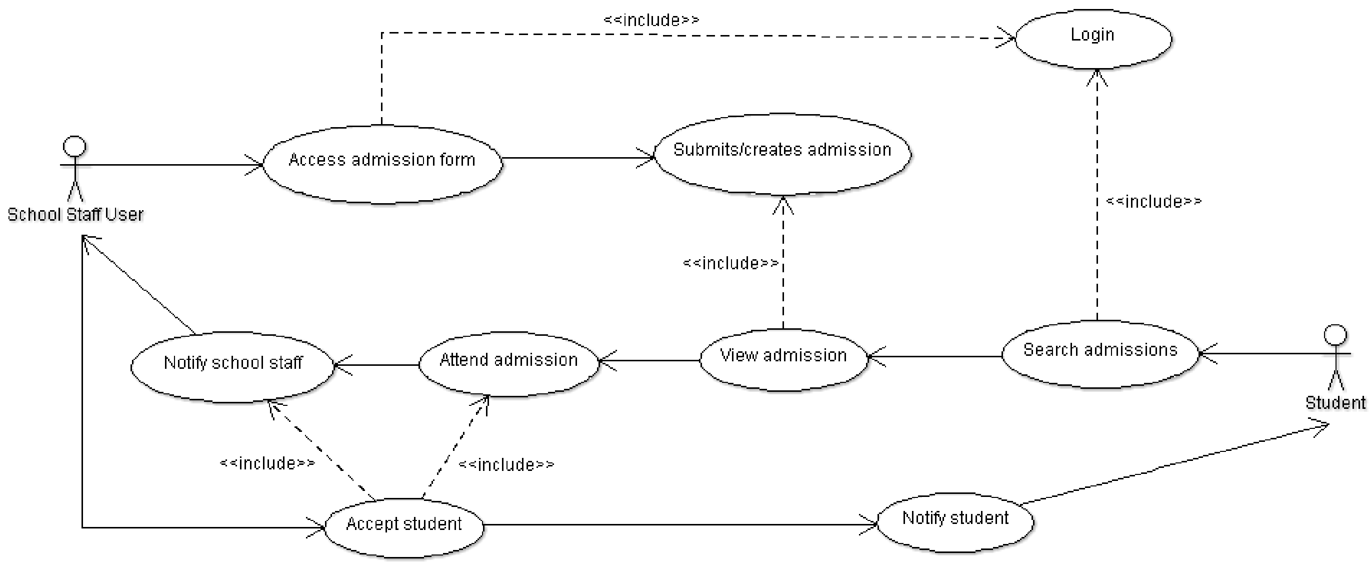


Fig. 7: Use Case diagram which represents the creation of an admission by a school staff user and the student attending workflow

4.2.5 Close an admission

1. Objective/Context

The admission period has ended.

2. Scenario/Steps

1. When the admission period has ended, the school staff user will receive a notification, and must mark the admission that is ready to be processed.

2. A cronjob command will take the admissions and compute the results for all users.

3. The results will be publicly available for visualization and download in various formats (XML and JSON).

4. The students will be notified via email by their results at the admission the attended.

3. Extensions

In the future, there can be added even more open formats like CSV for example.

4. *Use Case* diagram

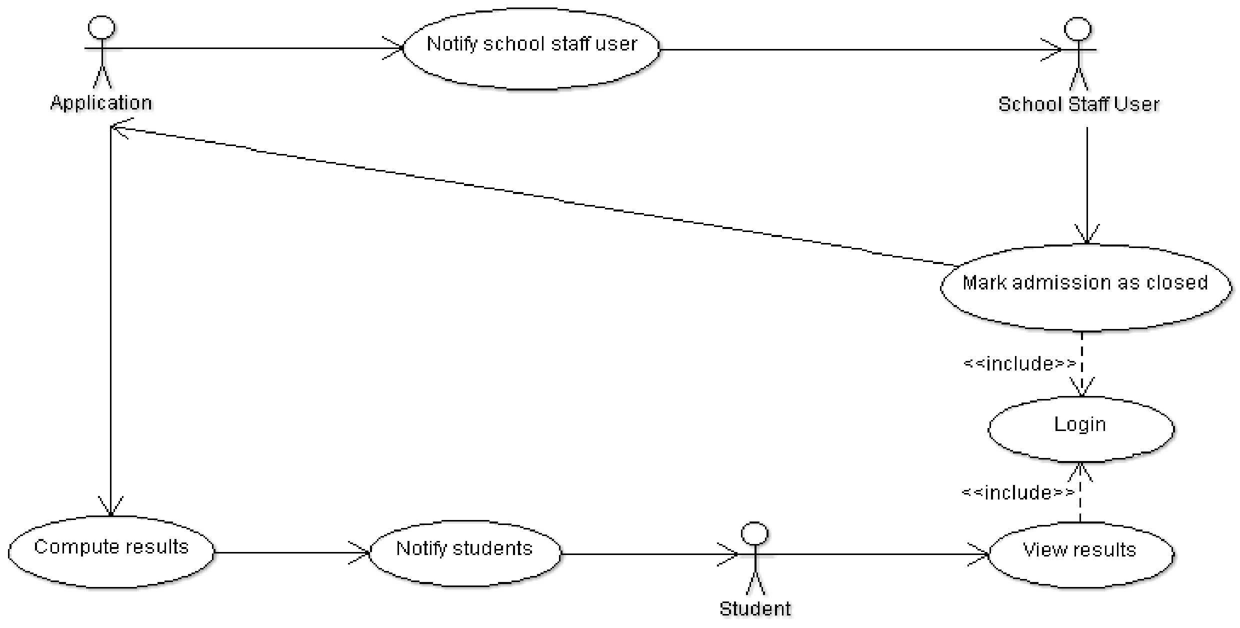


Fig. 8: Use Case diagram which represents the school staff user closing admission workflow

4.3 Writing Gherkin features

After we have created the *Use Case* diagrams, we can continue with the next step from the BDD modeling and write the scenarios in Gherkin. For our school admission project, we will imagine that this step is executed by the entire team. We will start by thinking and focusing on the features that bring the most business value from the imagined scenarios. We will describe real examples and try to transform them in specifications, that will be implemented. The next tables will contain examples for the student registration, the admin management area and the school staff user account which manages the admission.

```
Feature: Student register
  In order to be able to attend admissions
  As a student user
  I need to be able to register a student account

  Background:
    Given I am on "/"

  Scenario: As a student, I want to register a new student account
    When I follow "Register"
    Then I should be on "/register/student"
    And I should see "Student Register"
    And I should see "Terms and Conditions"
    When I fill in "eb_user_registration_student_email" with "emi.berea+student-
101@gmail.com"
    And I fill in "eb_user_registration_student_plainPassword_first" with "12345"
    And I fill in "eb_user_registration_student_plainPassword_second" with "12345"
    And I fill in "eb_user_registration_student_firstName" with "StudentFn101"
    And I fill in "eb_user_registration_student_lastName" with "StudentLn101"
    And I check "Terms and Conditions"
    And I press "Register"
    Then I should be on "/register/check-email"
    And I should see "The user has been created successfully"
    And I should see "An email has been sent to emi.berea+student-101@gmail.com"
    When I follow registration link "emi.berea+student-101@gmail.com"
    Then I should be on "/register/confirmed"
    And I should see "Congrats emi.berea+student-101@gmail.com, your account is now
activated."
```

Table 3: Example from the *student.feature* file which describes the behavior for the Student registration

```
Feature: Admin
  In order to be able to make the application to be used by schools
  As an admin user
  I need to be able to create schools and school staff users

  Background:
    Given I am on "/"
    When I follow "Login"
    Then I should be on "/login"
    And I should see "Login"
    And I should see "Forgot your password?"
    When I fill in "username" with "emi.berea+admin-1@gmail.com"
    And I fill in "password" with "12345"
    And I press "Log in"
    Then I should be on "/admin/dashboard"
    And I should see "Admin dashboard"

  Scenario: As an admin, I want to create a new school
    When I follow "Schools"
    Then I should be on "/admin/school/"
    And I should see "School List"
    And I should see "Add School"
    When I follow "Add School"
    Then I should be on "/admin/school/new"
    And I should see "School Creation"
    When I fill in "Name" with "Faculty of Computer Science"
    And I fill in "Country" with "Romania"
    And I fill in "City" with "Iasi"
    And I fill in "Address" with "street General Berthelot 16"
    And I select "Faculty" from "Type"
    And I press "Create"
    Then I should see "New school created!"
    And I should see "Faculty of Computer Science"
    And I should see "Back to the list"
    When I follow "Back to the list"
    Then I should be on "/admin/school/"
    And I should see "School List"
    And I should see "Faculty of Computer Science"
```

Table 4: Example from the *admin.feature* file which describes the behavior for the CRUD pannels

```
Feature: School Staff User admission creation
  In order to be able to facilitate the attendance process for students
  As a School Staff User
  I need to be able to create an admission and make it available online

  Background:
    Given I am on "/"
    When I follow "Login"
    Then I should be on "/login"
    And I should see "Login"
    And I should see "Forgot your password?"
    When I fill in "username" with "emi.berea+ssu-1@gmail.com"
    And I fill in "password" with "12345"
    And I press "Log in"
    Then I should be on "/"
    And I should see "Edu Admission - Main page"

  Scenario: As a Ssu, I want to create a new admission then I logout and login as a
student and attend the created admission
    When I follow "Admissions"
    Then I should be on "/school-staff/admission"
    And I should see "Admission List"
    And I should see "Add Admission"
    When I follow "Add Admission"
    Then I should be on "school-staff/admission/new"
    And I should see "Admission Creation"
    When I select "4" from "eb_core_admission_sessionDate_day"
    And I select "Jul" from "eb_core_admission_sessionDate_month"
    And I select "2017" from "eb_core_admission_sessionDate_year"
    And I fill in "Budget financed no" with "275"
    And I fill in "Fee payer no" with "90"
    And I press "Create"
    Then I should see "New admission created!"
    And I should see "04-07-2017"
    When I follow "Back to the list"
    Then I should be on "school-staff/admission"
    And I should see "Admission List"
    And I should see "04-07-2017"
```

Table 5: Example from the *admission.feature* file which describes the behavior for the admission workflow and its interaction with the student and the school staff

For the remaining application functionalities, the *Gherkin* files will follow a similar pattern, by describing step by step the scenarios, serving as specification for the implementation part, which will be detailed in the next chapter.

5. Implementation

5.1 Project setup

To setup a new Symfony project we used the recommended method, which is the *Symfony installer*[18] command-line tool. We choose to use version 2.8 of Symfony, instead of the new 3.* version, because 2.8 is more stable for now, and it integrates better with other third-party libraries that we want to use. The project is developed in a local LAMP environment which is hosted with VirtualBox[19].

Next, we will install Behat framework, and we will do that via the *Composer* package manager. In order to install Behat we added in the *composer.json* file the following lines:

```
"behat/behat": "^3.3",
"behat/symfony2-extension": "^2.1",
"behat/mink": "^1.7",
"behat/mink-extension": "^2.2",
"behat/mink-browserkit-driver": "^1.3",
"behat/mink-goutte-driver": "^1.2",
"behat/mink-selenium2-driver": "^1.3",
```

and then run the following command: ***composer update behat/behat*** and *Composer* will automatically install all dependencies.

Now that we have installed Behat, we need to configure and initialize it in our Symfony project. First, we create a *behat.yml* file in the root directory of the project with the following content:

```
default:
    extensions:
        Behat\Symfony2Extension: ~
        Behat\MinkExtension:
            base_url: http://fii-dissertation-edua.l/app_dev.php/
            sessions:
                default:
                    symfony2: ~
#                     selenium2: ~
    autoload:
        '': '%paths.base%/features' # %paths.base% is a special variable in
behat.yml that refers to the folder in which behat.yml is stored.
    suites:
        default:
            contexts: [Context\FeatureContext]
```

Table 6: Example from the *behat.yml* configuration file for Behat

And we run in the terminal from the application root directory the command: ***./bin/behat --init*** This generates a *features\Context* directory into the project root folder, which will contain a class named *FeatureContext*. We will modify this class declaration to extend *MinkContext*

[18] Symfony installer is described at this URL: http://symfony.com/doc/current/setup.html
[19] VirtualBox – free and open-source software that manages and runs virtual machines

which provide access to the Mink PHP library, so we can easily manipulate the browser and make interactions with the web page content.

5.2 Write and run first Behat feature

Now we will create our first feature for this project, and we will need to write a scenario for the account management functionalities, like login, register, etc. We will create a new file in the *features/* directory named *login.feature* in which we will copy the content of the scenarios that we wrote in chapter 4.3:

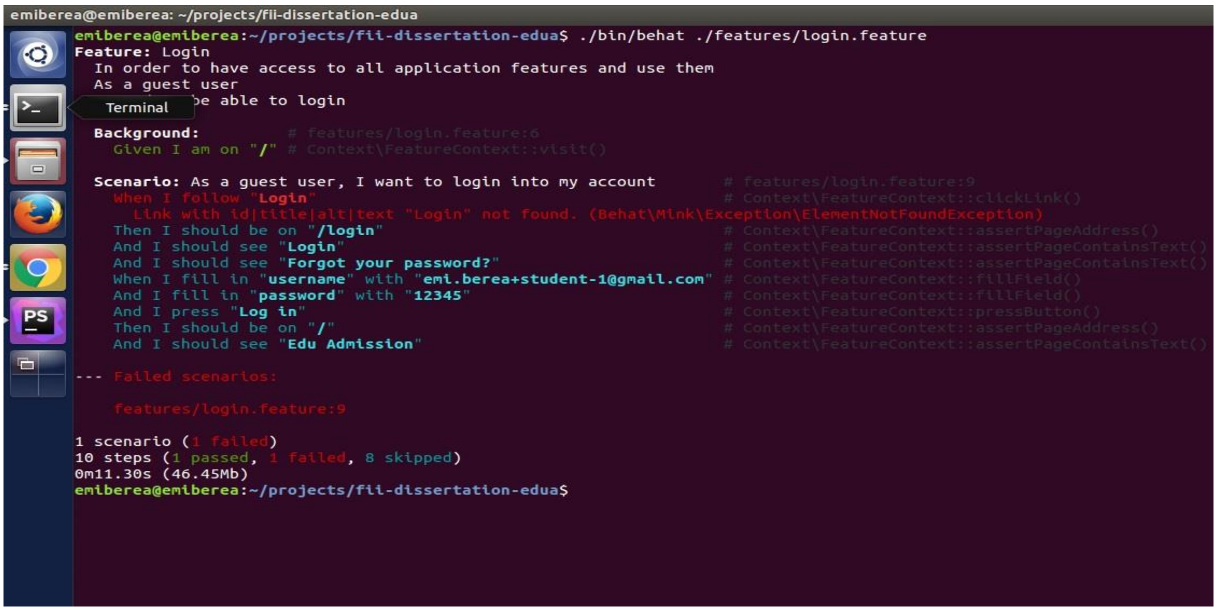
```
Feature: Login
  In order to have access to all application features and use them
  As a guest user
  I need to be able to login

  Background:
    Given I am on "/"

  Scenario: As a guest user, I want to login into my account
    When I follow "Login"
    Then I should be on "/login"
    And I should see "Login"
    And I should see "Forgot your password?"
    When I fill in "username" with "emi.berea+student-1@gmail.com"
    And I fill in "password" with "12345"
    And I press "Log in"
    Then I should be on "/"
    And I should see "Edu Admission"
```

Table 7: *login.feature* file which describes the behavior for the login page

Now we will run the *login.feature* file via the terminal command: "*./bin/behat ./features/login.feature*" and see that it has not passed – obviously because we didn't implement nothing yet.



```
emiberea@emiberea: ~/projects/fii-dissertation-edua
emiberea@emiberea:~/projects/fii-dissertation-edua$ ./bin/behat ./features/login.feature
Feature: Login
  In order to have access to all application features and use them
  As a guest user
        be able to login

  Background:          # features/login.feature:6
    Given I am on "/" # Context\FeatureContext::visit()

  Scenario: As a guest user, I want to login into my account        # features/login.feature:9
    When I follow "Login"                                          # Context\FeatureContext::clickLink()
      Link with id|title|alt|text "Login" not found. (Behat\Mink\Exception\ElementNotFoundException)
    Then I should be on "/login"                                   # Context\FeatureContext::assertPageAddress()
    And I should see "Login"                                       # Context\FeatureContext::assertPageContainsText()
    And I should see "Forgot your password?"                       # Context\FeatureContext::assertPageContainsText()
    When I fill in "username" with "emi.berea+student-1@gmail.com" # Context\FeatureContext::fillField()
    And I fill in "password" with "12345"                          # Context\FeatureContext::fillField()
    And I press "Log in"                                           # Context\FeatureContext::pressButton()
    Then I should be on "/"                                        # Context\FeatureContext::assertPageAddress()
    And I should see "Edu Admission"                               # Context\FeatureContext::assertPageContainsText()

--- Failed scenarios:

    features/login.feature:9

1 scenario (1 failed)
10 steps (1 passed, 1 failed, 8 skipped)
0m11.30s (46.45Mb)
emiberea@emiberea:~/projects/fii-dissertation-edua$
```

Fig. 9: screenshot which displays the terminal window where the *login.feature* file was executed

Now, our job is to implement what is needed to make the test pass. From what we see in the test, we will need an URL "*/login*" and web page that will display a login form. Also, we will need some database users with at least a username and password field. Because we are using Symfony framework, we can use one of its popular bundles which provides user management called *FOSUserBundle*[21]. The bundle will be installed via *Composer* (as any modern PHP library) and then configured inside Symfony. Next, we have to override and extend classes and controllers from *FOSUserBundle* in order to create our page. In the end, we created a fully functional page that looks like this:

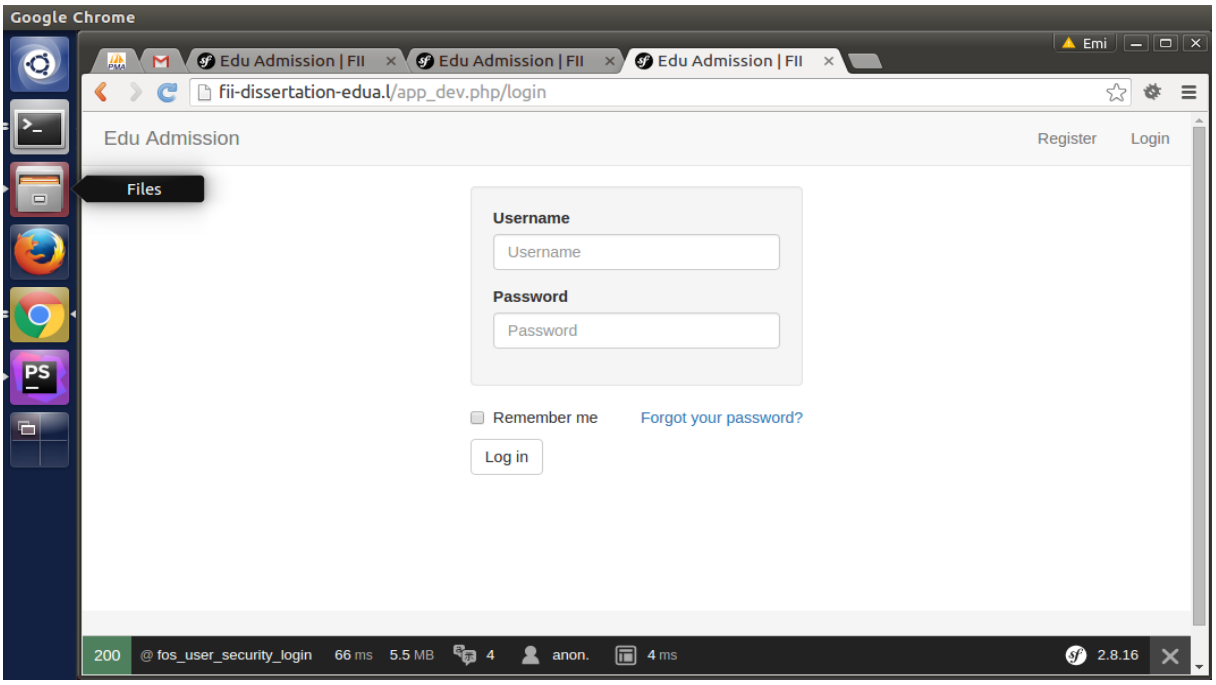
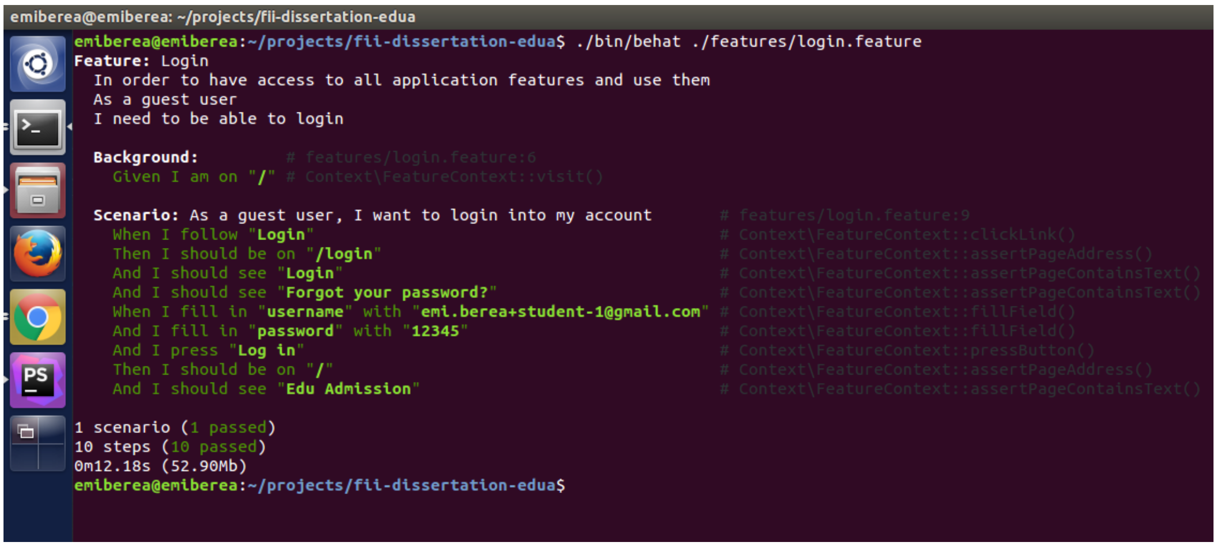


Fig. 10: screenshot which shows how the login page looks

Next, we can try to run the Behat tests again and see if it has passed:



```
emiberea@emiberea: ~/projects/fii-dissertation-edua
emiberea@emiberea:~/projects/fii-dissertation-edua$ ./bin/behat ./features/login.feature
Feature: Login
  In order to have access to all application features and use them
  As a guest user
  I need to be able to login

  Background:         # features/login.feature:6
    Given I am on "/" # Context\FeatureContext::visit()

  Scenario: As a guest user, I want to login into my account       # features/login.feature:9
    When I follow "Login"                                          # Context\FeatureContext::clickLink()
    Then I should be on "/login"                                   # Context\FeatureContext::assertPageAddress()
    And I should see "Login"                                       # Context\FeatureContext::assertPageContainsText()
    And I should see "Forgot your password?"                       # Context\FeatureContext::assertPageContainsText()
    When I fill in "username" with "emi.berea+student-1@gmail.com" # Context\FeatureContext::fillField()
    And I fill in "password" with "12345"                          # Context\FeatureContext::fillField()
    And I press "Log in"                                           # Context\FeatureContext::pressButton()
    Then I should be on "/"                                        # Context\FeatureContext::assertPageAddress()
    And I should see "Edu Admission"                               # Context\FeatureContext::assertPageContainsText()

1 scenario (1 passed)
10 steps (10 passed)
0m12.18s (52.90Mb)
emiberea@emiberea:~/projects/fii-dissertation-edua$
```

Fig. 11: screenshot which displays the terminal window where the *login.feature* file was executed successfully

We now have our first working feature of our application. In the next subchapter, we will continue with showing the other main features of the project and how BDD was used.

5.3 Developing project using Behat and following BDD principles

Now we will take each application feature and implement it one by one. We will start now with the Admin features. As an admin user, we should be able to manage all the database entities via CRUD forms. This implies having control over the creation of school staff users and students. They will also be able to send the access to the accounts via email links they will be able to create schools and assign the school staff to them. We will start our development from the following Gherkin file:

```
Feature: Admin
  In order to be able to make the application to be used by schools
  As an admin user
  I need to be able to create schools and school staff users

  Background:
    Given I am on "/"
    When I follow "Login"
    Then I should be on "/login"
    And I should see "Login"
    And I should see "Forgot your password?"
    When I fill in "username" with "emi.berea+admin-1@gmail.com"
    And I fill in "password" with "12345"
    And I press "Log in"
    Then I should be on "/admin/dashboard"
    And I should see "Admin dashboard"

  Scenario: As an admin, I want to create a new school
    When I follow "Schools"
    Then I should be on "/admin/school/"
    And I should see "School List"
    And I should see "Add School"
    When I follow "Add School"
    Then I should be on "/admin/school/new"
    And I should see "School Creation"
    When I fill in "Name" with "Faculty of Computer Science"
    And I fill in "Country" with "Romania"
    And I fill in "City" with "Iasi"
    And I fill in "Address" with "street General Berthelot 16"
    And I select "Faculty" from "Type"
    And I press "Create"
    Then I should see "New school created!"
    And I should see "Faculty of Computer Science"
    And I should see "Back to the list"
    When I follow "Back to the list"
    Then I should be on "/admin/school/"
    And I should see "School List"
    And I should see "Faculty of Computer Science"

  Scenario: As an admin, I want to create a new school staff user
    When I follow "School Staff Users"
    Then I should be on "/admin/school-staff/"
    And I should see "School Staff User List"
    And I should see "Add School Staff User"
    When I follow "Add School Staff User"
    Then I should be on "/admin/school-staff/new"
    And I should see "School Staff User Creation"
    When I fill in "Email" with "emi.berea+ssu-101@gmail.com"
    And I fill in "First name" with "SsuFn101"
    And I fill in "Last name" with "SsuLn101"
    And I fill in "Title" with "Mrs."
    And I fill in "Job title" with "Secretary"
    And I fill in "Academic degree" with "Bachelor"
    And I select "Facultatea de Informatica, Iasi" from "School"
    And I press "Create"
    Then I should see "New School Staff User created!"
    And I should see "emi.berea+ssu-101@gmail.com"
    And I should see "Back to the list"
    When I follow "Back to the list"
    Then I should be on "/admin/school-staff/"
    And I should see "School Staff User List"
    And I should see "emi.berea+ssu-101@gmail.com"

  Scenario: As an admin, I want to create a new student
    When I follow "Students"
    Then I should be on "/admin/student/"
    And I should see "Student List"
    And I should see "Add Student"
    When I follow "Add Student"
    Then I should be on "/admin/student/new"
    And I should see "Student Creation"
    When I fill in "Email" with "emi.berea+student-101@gmail.com"
    And I fill in "First name" with "StudentFn101"
    And I fill in "Last name" with "StudentLn101"
    And I fill in "Father initial" with "T"
    And I fill in "Pin" with "123456"
    And I fill in "City" with "Iasi"
    And I fill in "Address" with "street Carol I"
    And I fill in "High school" with "Moisil High School"
    And I fill in "Specialization" with "Computer Science"
    And I fill in "Baccalaureate average grade" with "8.5"
    And I fill in "Baccalaureate maximum grade" with "9.6"
    And I check "Enabled"
    And I check "Verified"
    And I press "Create"
    Then I should see "New student created!"
    And I should see "emi.berea+student-101@gmail.com"
    And I should see "Back to the list"
    When I follow "Back to the list"
    Then I should be on "/admin/student/"
    And I should see "Student List"
    And I should see "emi.berea+student-101@gmail.com"

  Scenario: As an admin, I want to assign a school staff user to a school
    When I follow "School Staff Users"
    Then I should be on "/admin/school-staff/"
    And I should see "School Staff User List"
    And I should see "emi.berea+ssu-1@gmail.com"
    When I follow "emi.berea+ssu-1@gmail.com"
    Then I should see "School Staff User"
    And I should see "Edit"
    When I follow "Edit"
    Then I should see "School Staff User Edit"
    When I select "Facultatea de Istorie, Iasi" from "School"
    And I press "Edit"
    Then I should see "School Staff User edited successfully!"
```

Table 8: Gherkin file related to admin features

First, we will create a new Symfony bundle called *EBAdminBundle* which will hold the code related to the admin area. We will create a CRUD controller for each entity that we want to manage: *StudentUser*, *SchoolStaffUser* and *Admission*. Then we will create the *Twig* templates and form builder classes. The controller class *StudentController* will look like this:

```php
<?php

namespace EB\AdminBundle\Controller;

use EB\AdminBundle\Form\Type\StudentType;
use EB\UserBundle\Entity\StudentUser;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;

/**
 * @Route("/admin/student")
 */
class StudentController extends Controller
{
    /**
     * Lists all studentUser entities.
     *
     * @Route("/", name="eb_admin_student_index")
     * @Method("GET")
     */
    public function indexAction()
    {
        $em = $this->getDoctrine()->getManager();

        $studentUsers = $em->getRepository('EBUserBundle:StudentUser')->findAll();

        return $this->render('EBAdminBundle:Student:index.html.twig', array(
            'studentUsers' => $studentUsers,
        ));
    }

    /**
     * Creates a new studentUser entity.
     *
     * @Route("/new", name="eb_admin_student_new")
     * @Method({"GET", "POST"})
     */
    public function newAction(Request $request)
    {
        $studentUser = new Studentuser();
        $form = $this->createForm(StudentType::class, $studentUser);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $studentUser->setPlainPassword(uniqid());
            $studentUser->setEnabled(true);
            $studentUser->addRole('ROLE_STUDENT');
            $studentUser->setPasswordRequestedAt(new \DateTime());
            $studentUser->setConfirmationToken($this-
>get('fos_user.util.token_generator')->generateToken());

            // send registration (reset password) email to the Student
            $this->get('eb_core.service.mailer')->sendEmail($studentUser-
>getEmail(), 'EBAdminBundle:Student/Email:newAccount.html.twig', [
                'studentUser' => $studentUser,
                'confirmationUrl' => $this->generateUrl('fos_user_resetting_reset',
['token' => $studentUser->getConfirmationToken()], true),
            ]);

            $em = $this->getDoctrine()->getManager();
            $em->persist($studentUser);
            $em->flush($studentUser);

            $this->addFlash('success', 'New student created!');

            return $this->redirectToRoute('eb_admin_student_show', array('id' =>
$studentUser->getId()));
```

```
        }

        return $this->render('EBAdminBundle:Student:new.html.twig', array(
            'studentUser' => $studentUser,
            'form' => $form->createView(),
        ));
    }

    // other methods
}
```

Table 9: *StudentController* from EBAdminBundle

Continuing this way, we will implement all the methods, until all the functionalities of the Student admin panel are done and the Behat tests will pass. Similarly, the other admin panels will be implemented.

5.4 Testing with Behat

Another utility of Behat is that it can be used on acceptance testing. Clients can verify if the initial requirements are fulfilled. The Gherkin syntax also helps to easily validate the project features. On regression testing Behat is useful because it can check if there were introduced bugs, during the process of adding new code/functionalities.

The main advantage of using BDD frameworks for test automation, beside the fact that it allows us to automate many testing tasks that would have need to be done manually, is that it allows to do the following types of tests:

- **Graphical user interface testing** - where we can verify the behaviour of the application from the end-user point of view, we can develop test scenarios, formed by small steps, which consists of user action, such as clicking a link/button, complete a form, etc. and after that we can verify that the respective step has produces the expected output.
- **Regression testing** - we can use Behat to seek to uncover software regressions, where software features that were previously working correctly, stops working as intended, usually regressions occur during the development process, as an unintended consequence of program changes. Typically, the Selenium tests can be rerun from time to time based on a schedule, to check if previous bugs have re-emerged.
- **Continuous integration** - also Behat can be used in conjunction with continuous integration servers to ensure web applications are automatically tested via scripts as opposed to manually each time a developer added something new to the project code base.

Conclusions

After all the work that was done for creating and documenting this project, we think that we manage to deliver a good documentation about BDD and the available tools. We consulted several books and online resources related to BDD in order to provide relevant information. In the first chapters, we learned that in the software engineering field, it is very important to do rigorous analysis of the project requirements and of the available development process. We saw that BDD is a set of practices that can be used to enhance any iterative development methodology. BDD is focusing on defining the features that bring the most value to the business via continuous communication between the stakeholders and it emphasize on writing scenarios and usage examples in a syntax similar to English language. The examples also represent executable specifications and living documentation that will help future development. We also saw that BDD methodologies and Behat framework offer a lot of improvements over other software development processes and have application in testing fields (acceptance, regression, automated). We noted that in the right environment BDD can bring advantages like: reduced waste, reduced costs, easier code changes and faster releases. While in some other not ideal contexts (like projects with low collaboration between parties, non-iterative contexts and others) it cannot be applied correctly and present some disadvantages. After all, we can say that BDD is important for everyone to study and try to use its practices in order to maximize the efficiency of the development process.

On the practical project side, we presented the modern tools that are available for the PHP language at this moment, starting with Symfony and Behat. Our "*EduA*" web application was a perfect playground to see BDD at work and learn how to apply each step of the BDD process. We learned about Symfony and its included libraries to have a starting point for the project. Then we continued with Behat and Mink to follow the BDD practices. From my experience with Symfony on other projects that were not following the BDD practices, I consider that BDD can shorten the time needed for development by 20-30%, depending how experienced are the developers. The application is offering the features that were described in this paper, but the system is still in its first form and to be fully used in real world, it needs some more work and features like: attachments of documents and online payments (in order to make the student registration fully online) and attractive design.

Through this written paper and the practical project, which consist in the web application described in this document, I think that I manage to offer a good presentation of the BDD landscape and a good starting point for everyone interesting in starting studying this topic.

References

[1] BDD in Action: Behavior-driven development for the whole software lifecycle 1st Edition, John Ferguson Smart, a: page 4, b: page 12, c: pages 16 - 30

[2] The Cucumber Book: Behaviour-Driven Development for Testers and Developers (Pragmatic Programmers) 1st Edition, Matt Wynne, Aslak Hellesoy

[3] Cucumber Recipes: Automate Anything with BDD Tools and Techniques (Pragmatic Programmers) 1st Edition, Ian Dees, Matt Wynne, Aslak Hellesoy

[4] https://en.wikipedia.org/wiki/Behavior-driven_development

[5] http://dannorth.net/introducing-bdd

[6] http://behat.org/en/latest/

[7] https://github.com/Behat/Behat

[8] http://mink.behat.org

[9] https://github.com/Behat/MinkExtension

[10] https://github.com/Behat/Symfony2Extension

[11] https://github.com/cucumber/cucumber/wiki/Gherkin

[12] http://www.bdd-in-action.com/

[13] https://en.wikipedia.org/wiki/LAMP_(software_bundle)

[14] http://php.net/

[15] http://symfony.com/doc/current/index.html

[16] http://symfony.com/projects

[17] http://twig.sensiolabs.org/

[18] http://swiftmailer.org/

[19] https://github.com/Seldaek/monolog

[20] http://www.doctrine-project.org/

[21] https://github.com/FriendsOfSymfony/FOSUserBundle

[22] https://jquery.com/

[23] http://getbootstrap.com/