

COMENIUS UNIVERSITY IN BRATISLAVA  
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

WEB TEACHING TOOL FOR LEARNING AGILE  
PROGRAMMING  
BACHELOR THESIS

2019  
TAMARA SAVKOVA

COMENIUS UNIVERSITY IN BRATISLAVA  
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

WEB TEACHING TOOL FOR LEARNING AGILE  
PROGRAMMING  
BACHELOR THESIS

Program of Study: Aplikovaná informatika  
Field of Study: 2511 Aplikovaná informatika  
Department: Department of Applied Informatics  
Supervisor: Ing. František Gyárfáš, CSc.

Bratislava, 2019  
Tamara Savkova



Univerzita Komenského v Bratislave  
Fakulta matematiky, fyziky a informatiky

## ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Tamara Savkova  
**Študijný program:** aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)  
**Študijný odbor:** aplikovaná informatika  
**Typ záverečnej práce:** bakalárska  
**Jazyk záverečnej práce:** anglický  
**Sekundárny jazyk:** slovenský

**Názov:** Web teaching tool for learning agile programming  
*Webová výuka agilného programovania*

**Anotácia:** Cieľom bakalárskej práce je navrhnúť a implementovať interaktívne webové prostredie pre výuku vybraných agilných metód programovania. Poskytne používateľovi praktické úlohy pre testmi riadené programovanie, refaktorizačné cvičenia a cvičenia na prácu so zdedeným kódom. Vývojové prostredie pre riešenie cvičení bude webová aplikácia, vytvorený kód spolu s testmi bude zbiehaný vo virtuálnom prostredí na serveri a výsledky testov budú prezentované používateľovi. Aplikácia bude zahŕňať základné nástroje na administráciu systému: administráciu používateľov, správu cvičení a ich výsledkov a verzionovanie zdrojových súborov. Systém bude realizovaný pomocou technológií/nástrojov: JavaScript (jQuery, Bootstrap), MySQL, HTML5, CSS, kompilátory vybraných jazykov, knižnice pre testing, virtuálny server.

**Vedúci:** Ing. František Gyarfaš, CSc.  
**Katedra:** FMFI.KAI - Katedra aplikovanej informatiky  
**Vedúci katedry:** prof. Ing. Igor Farkaš, Dr.  
**Dátum zadania:** 04.10.2018

**Dátum schválenia:** 15.10.2018

doc. RNDr. Damas Gruska, PhD.  
garant študijného programu

.....  
študent

.....  
vedúci práce

Hereby I declare that I wrote this thesis by myself with the use of referenced literature, under the supervision of my thesis supervisor.

Tamara Savkova

I would like to thank my supervisor Ing. František Gyárfáš, CSc. for his help during work on the thesis.

## Abstract

The aim of the bachelor thesis is to design and implement an interactive web environment for teaching of selected agile programming methods. The resulting web environment is a web application, which may be used for acquaintance with theoretical side of agile programming methods, and also for trying out these methods in a form of courses with getting feedback instantly. Application provides creating and solving practical tasks on test-driven development, on refactoring and working with legacy code. The web application provides solving the exercises within the development environment, the generated code along with the tests is run in the virtual environment on the server, and the tests results are presented to the user. The application includes tools for managing the course items and source code version control.

**Keywords:** web application, test-driven development, unit-testing, legacy code, clean code, refactoring, educational software.

## Abstrakt

Cieľom bakalárskej práce je navrhnúť a implementovať interaktívne webové prostredie pre výuku vybraných agilných metód programovania. Vývojové prostredie je webová aplikácia, ktorá môže byť použitá na zoznámenie sa s teoretickou stranou agilných metód programovania a zároveň na vyskúšanie daných metód v rámci kurzov so získaním okamžitej spätnej väzby. Poskytuje používateľovi možnosť vytvárania a riešenia praktických úloh pre testmi riadené programovanie, refaktorizáciu a prácu so zdedeným kódom. Vývojové prostredie je súčasťou webovej aplikácie, vytvorený kód spolu s testmi je zbieraný vo virtuálnom prostredí na serveri a výsledky testov sú prezentované používateľovi. Aplikácia zahŕňa základne nástroje na správu prvkov kurzu a verzionovanie zdrojových súborov.

**Kľúčové slová:** webová aplikácia, testami riadený vývoj, jednotkové testy, zdedený kód, čistý kód, refaktorizácia, edukačný softvér.

# Contents

<b>Introduction</b>	<b>1</b>
<b>1 Sources</b>	<b>2</b>
1.1 Agile software development . . . . .	2
1.1.1 Extreme programming . . . . .	3
1.1.2 Methods of XP . . . . .	3
1.1.3 Practical use of XP . . . . .	6
1.2 Existing solutions . . . . .	7
1.2.1 Similar Bachelor and Master theses . . . . .	7
1.2.2 Similar web applications . . . . .	8
1.2.3 freeCodeCamp . . . . .	8
1.3 Technologies to be used . . . . .	9
1.3.1 Client side . . . . .	9
1.3.2 Server side . . . . .	10
<b>2 Concept</b>	<b>12</b>
2.1 Courses overview . . . . .	12
2.1.1 Courses structure . . . . .	12
2.1.2 Shown and hidden versions . . . . .	12
2.1.3 Development environment . . . . .	13
2.1.4 Version control . . . . .	14
2.1.5 Solution estimation . . . . .	14
2.1.6 Exercise types . . . . .	15
2.2 Architecture . . . . .	16
2.3 User interface . . . . .	17
2.3.1 Courses . . . . .	17
2.3.2 Exercise creation and editing . . . . .	17
2.3.3 Exercise solving . . . . .	19
2.3.4 Run output . . . . .	19
2.3.5 Exercise Toolbar . . . . .	20
2.4 Data structure . . . . .	20



2.5	Use cases . . . . .	20
2.6	Technical requirements . . . . .	20
2.6.1	Security . . . . .	22
<b>3</b>	<b>Solution</b>	<b>23</b>
3.1	Front-end . . . . .	23
3.1.1	Angular framework . . . . .	23
3.1.2	External modules . . . . .	24
3.1.3	Custom modules . . . . .	25
3.2	Back-end . . . . .	33
3.2.1	Virtual Sandbox . . . . .	33
3.2.2	Estimating module . . . . .	34
3.3	Database . . . . .	34
<b>4</b>	<b>Sample Course</b>	<b>37</b>
4.1	Sample course overview . . . . .	37
4.1.1	Legacy program overview . . . . .	37
4.1.2	Lesson 1: Debugging a legacy program . . . . .	38
4.1.3	Lesson 2: Adding new features to the legacy program . . . . .	38
4.1.4	Lesson 3: Refactoring the legacy program . . . . .	39
	<b>Conclusion</b>	<b>42</b>

# List of Figures

1.1	Development environment at Codecademy . . . . .	9
2.1	Course detail sketch . . . . .	13
2.2	Code editor sketch . . . . .	14
2.3	Component diagram . . . . .	18
2.4	Entity relationship model . . . . .	21
2.5	Use case diagram . . . . .	22
3.1	Quill rich text editor component . . . . .	24
3.2	Monaco code editor . . . . .	25
3.3	Courses list . . . . .	26
3.4	Course detail . . . . .	27
3.5	Manage component . . . . .	28
3.6	Filename change dialog . . . . .	29
3.7	Editor element on exercise create or editing . . . . .	30
3.8	Editor element . . . . .	31
3.9	Version control . . . . .	31
3.10	Exercises description . . . . .	32
3.11	Exercises toolbar . . . . .	32
3.12	Relational model of system data . . . . .	36

# Introduction

The agile programming principles were stated 20 years ago by authors of Agile manifesto, but they were used by software developers years before it happened. Experienced programmers develop own skills while doing mistakes and failures when looking for the best development approach out of numerous possibilities. Such skills are sound with agile programming principles, which, however, seem to be counter intuitive at first sight. Agile methods bring long-term benefits. Using the agile methods while working on the project and seeing its advantages with one's own eyes is a way more convincing. For that reason, it is important to learn agile programming methods. Only theory-based learning is not complete enough for learning agile programming. It requires a lot of practice, especially on projects development.

The term 'agile' is more associated with project management rather than its development. This approach on management is known for accepting the fact that everything changes, and for tending to adapt to new circumstances, gain from them. Some of the agile development methods are widely used and concerned to be compulsory, the others are mostly taken as extreme and experimental. But all of them make development process efficient, deliberate and responsible. The benefits of these methods are more obvious when it is time to bring changes to the code, and there is no need to 'edit and pray'.

The world is full of things, which become stronger when something makes them change. They are not only naturally evolved systems, but also artificial. This feature got name 'antifragile' and was described by Nassim Taleb, author of the series of books on it [32].

The aim of the Bachelor thesis is to illustrate the point that 'agile' is 'antifragile' with use of a learning environment. It may be used for acquaintance with the theoretical side of the agile programming methods, and also for creating space to try out the methods and get feedback instantly. This environment is presented in a web application. Concept chapter describes the application on a design level, when Solution chapter focuses on issues of implementation. The application is accompanies with a course, which covers the features suggested by the application and the agile methods. Its content, purpose and used methodologies are presented in Sample course chapter.

# Chapter 1

## Sources

The chapter on sources is devoted to the description of the theory, used for the thesis realisation, overview of the existing solutions of similar problems and introduction of the used technologies.

### 1.1 Agile software development

The aim of the web application is to introduce and teach how to use the selected methodologies of agile programming. They are based on principles of Agile software development, stated by authors of Manifesto for Agile Software Development [10].

This approach advocates adaptive planning, evolutionary development, empirical knowledge, and continual improvement, and it encourages rapid and flexible response to change [1]. Detailed planning of a large amount of the software process at once for a long span of time is a fragile approach and leads to further growing number of problems, when there is a need of change. The larger the system is, the more difficult it may be to add new features, and the more difficult it is to fix bugs, and the more resources it all requires. The agile methods are rather adaptive than predictive. So they welcome change, they are able to adapt and even thrive on change [17, From Nothing, to Monumental, to Agile].

Agile approach provides this with use of iterative development. It means a frequent production of working versions of the final system that have a subset of the required features. These working systems are short on functionality, but should otherwise be faithful to the demands of the final system. They should be fully integrated and carefully tested as a final delivery. The other feature is a feedback mechanism to estimate the situation [17, Controlling an Unpredictable Process - Iterations].

The overview on the methods, on how may they affect the process of software development and how they are represented in the project are covered in the following sections.

### 1.1.1 Extreme programming

One of the most efficient, time-proven and known frameworks of Agile software development is Extreme programming, also known as XP. It has already been proven to be very successful at many companies of all different sizes and industries world wide [34].

One of the authors of Manifesto for Agile Software Development American software engineer Kent Beck is a contributor to XP [9]. This method focuses on practical rather than management aspects of the development. The aim of the XP methodology is to improve software quality and responsiveness to changing customer requirements.

The name “Extreme” is taken from the idea that the beneficial elements of traditional software development are taken to “extreme” levels [9, Preface].

As soon as XP is a framework of Agile web development, it includes iterations and feedbacks. This approach assumes three levels of plans, the first one for the next a few months, the second one for the next iteration and the last one for the next task within an iteration. The plans are expected to be constantly modified during the whole development process. Unit testing and acceptance coverage is a very important part of every stage of an XP project. XP provides feedback throughout the whole project on many levels and in many ways [34, Planning/Feedback loops].

Also, while planning, new functionality can be added only when it is necessarily required. Otherwise, most of the unnecessary functionality would be removed later, so it would slow down the process and require excessive resources. The design should be simple, so the program would be ready for unexpected changes [33].

The web application, which is the result of the work on the bachelor thesis, is designed to encourage the creation of the projects to solve them in such a way, so students would develop a program in iterations and would receive a feedback. Each iteration has exercises, which lead through the work on the project. A sample project is an example of such a task, the exercises are grouped into three iterations. It covers the process of development an adaptive to changes program out of a fragile one, while learning the main XP methods. The sample project is described in detail in Chapter 4.

### 1.1.2 Methods of XP

XP is built on the best practices, they are combined in a way that they complement and control each other [9, Foreword]. The methods of XP, which were or may be used in the result web application, will be introduced in the following subsections.

## Test-Driven Development (TDD)

The technique of Test-Driven Development, also known as TDD, was developed by already mentioned Kent Beck, one of the authors of Manifesto for Agile Software Development [8].

This practice supports work in short development cycles. It includes writing of test cases before the implementation to state the requirements on the program. Well written tests provide continuous feedback for the programmer, which is also essential for Agile development [27]. TDD is also used when modifying legacy code. Together with other techniques it makes work with legacy code efficient [15, Test-Driven Development (TDD)].

The key feature of the web application is that it forces using TDD to solve some of the tasks. A student would program within the provided IDE with editors for the source code and for the tests to be run on it. These tests are run within the web application and a student would get a feedback.

The sample project shows how work with legacy code can be improved with use of TDD techniques. The first iteration provides an exercise on finding mistakes in a legacy program. The aim is to find bugs in the legacy program. But the source code is not available, only a general structure of the code. The user should write his own tests to find the bugs and fix them. In the next exercise student may see and modify source code and he should fix those mistakes. He may use the tests from previous exercise, what brings elements of TDD.

## Refactoring

Code refactoring is the process when a software system is changed in such a way that it does not alter the external behaviour of the code, thus improves its internal structure. The idea of factoring is different from general code cleanup, it is less risky and invasive. Series of small structural modifications take place. Moreover, the tests provide a confidence that code remains correct [15]. As a result, the code is cleaned up and the chances of introducing bugs reduce. Refactoring is about improving the code, which has already been written. It is important to create a good design firstly and then to write code. But refactoring is the opposite of this practice, because it deals with bad, chaotic design to rework it into a good one [18, Preface].

Refactoring is a common part of an efficient work with legacy code, when there is a need in a design improvement. Such a software change would make its structure more maintainable, while the behaviour would be kept [15].

Also there are cases, when refactoring is not recommended yet. For example, when there is a need to rewrite the whole code from the beginning instead. It may happen, when the code works mostly incorrectly. It is also recommended to avoid refactoring,

when the deadline is close, because refactoring may remain unfinished. It would lead to a raise in cost of maintenance as well as the code, which was not refactored at all [14, Preface].

The web application also would make practising on refactoring possible. Refactoring a legacy code exercises are introduced in the sample project. Since refactoring should not be made when there are many mistakes in the code, the refactoring exercises take place in the third, last iteration. At this moment, after the previous two iterations are completed and passed all the tests, the code would be correct. Each of the numerous exercises is about a single step of the refactoring. Their title and description would have references to the book "Clean Code" by Robert C. Martin [23]. It is one of the mostly recommended books for software development, and for a good reason. The code should maintain correct during and after refactoring, and it can be provided by tests, written by the user and teacher.

### **Mature optimisation**

Optimisation is similar to refactoring, they both keep the functionality of the code. Although they have different goals. Refactoring changes the program structure, so it is easier to maintain. Optimisation makes changes in the resources, used by the program, e.g. time or memory [15]. Refactoring makes code readable and understandable, while optimisation of the code to achieve the needed performance often makes code harder to understand [18].

Optimisation may not always be performed optimally and save the resources as a result. Donald Knuth, an American computer scientist and professor at Stanford University, mentioned this issue in his paper "Structured Programming With go to Statements" [21]. He wrote that the programmers waste too much time on worrying about the speed of not crucial software parts, what makes negative impact, when maintenance and debugging are considered. So he concludes, that premature optimisation is the root of the all evil. As soon as optimisation may reduce readability and make the system more complex, it may be harder to maintain and debug it. Because of that, it is realised at the end of the development. This practice is sound with Agile and XP principle of developing only the necessary parts and only when they are needed.

### **Pair programming**

Pair programming is a technique, when the code for a single task is written by two people. The first programmer, the driver, has control of the computer and actively implements the program, and the other, the observer, provides feedback with reviewing the written code and thinks strategically about the work direction [35]. Feedback is also provided during pair programming.

It would be possible to work on assignments of the web application in pairs.

### User story

Ron Jeffries, one of the authors of the Manifesto for Agile Software Development [10], proposed a “Three Cs” formula for user story creation. The first “C” is for Card, on which user stories are written. User story is a customer’s functional requirement. It does not contain all the information that makes it up, but it has the needed information to identify it. There may be notes of cost and priority of the requirement. The second “C” is for the Conversation between the stakeholders, which particularly takes place during the planning. The third one is for the Confirmation, which is provided by an acceptance test, telling if the aims of the conversation have been reached [19].

In XP, it may be used as an instrument for stating the requirements and providing a feedback, also the tests may be based on user stories. In the web application, the user stories would be used to formulate the requirements needed to fulfil to accomplish an exercise.

### 1.1.3 Practical use of XP

According to the 12<sup>th</sup> annual State of Agile survey, which was taken in 2018, only 1% of respondents’ organisations use XP [2]. Another Agile framework, Scrum, is used by the majority of 56%. However, hybrid of another Agile framework, Scrum, and XP, is used by 6% of companies. This hybrid, known as ScrumXP, combines practices of Scrum project management and programming practices of XP.

Extreme programming is not that popular practice because of several reasons. The following reasons are associated with the mentioned in the previous section techniques.

It may seem irrational, even impossible to write tests before the code exists, which is the idea of TDD. Even writing tests in general is not a common practice. As mentioned previously in a corresponding subsection, the benefits are real.

As about pair programming, it is counter intuitive as well. But when realised correctly, the software quality may be increased within a lesser time than if two programmers worked separately. On one hand, programmer working by himself gets distracted more often, pair programming simplifies testing, getting feedback and code review, programmers feel more confidence in their. But on the other hand, it cannot work for every programmer or company. Most of the developers, especially the experts, prefer working on their own [13, Two by Two].

The aim of the web application is to explain such an approach, show examples of use and to teach how to get benefits from it.



## 1.2 Existing solutions

The web application would be presented in a form of an interactive online course. Its interface and control would be inspired by the existing platforms, introduced in the following subsections.

### 1.2.1 Similar Bachelor and Master theses

This section will introduce and compare the resulting applications of work on Bachelor and Master theses at faculty of Mathematics, Physics and Informatics of Comenius University in Bratislava. The results of the introduced Bachelor and Master theses were created for practising single agile programming methods.

Bachelor thesis with title Web learning programming using TDD technology was written by Tomáš Bočinec in 2018 (original title is *Webová výuka programovania pomocou metodológie TDD*). The resulting web application is a good example of a functional learning environment. It leads through a sequence of steps for test-driven development, which is one of the agile programming methods.

Bachelor thesis Web learning programming in C++ using Unit testing (original title: *Webová výuka programovania v C++ pomocou jednotkového testovania*) was written by Viliam Vakerman. The resulting development environment was created for unit testing learning, which is a key method and is common for all agile programming methods.

Another Bachelor thesis Web environment for learning code refactoring (original title: *Vývojové prostredie pre učenie sa refaktORIZÁCIE kódu*) was written by Ivan Latták. The techniques of refactoring are compulsory to get all the benefits of agile programming.

The web application for learning Agile programming methods uses similar mechanism to compile the provided source code and tests, run the tests and return the result. All these resulting web applications share the client-server architecture.

The resulting application of this thesis is aimed to teach agile programming methods, which include TDD, refactoring, unit testing and more other techniques. As a result, the web application suggests different, more specific exercises structures and tools for creating and solving them.

The learning process has a form of courses. This approach enables making the exercises sequential, solved step by step, and grouped into lessons by their concept. Also agile methods are useful for working on projects, so the suggested course structure created a good framework for working on a project while learning.

As soon as methodologies include techniques of working with legacy code, the application enables two new features. The first one is used on exercise creation. An

exercise author may create such a version of source code, test of file, which would be shown to the student when he starts working on exercise. This version may contain a legacy code for further work on it. The second feature is loading the content of the submitted result files to the development environment. Application also enables use of files, which may be used by tests.

More modern, efficient and widely used technologies would be used. Also the used technologies made it possible to develop a large application rapidly. Their details are introduced in the next section 1.3 Technologies to be used.

## 1.2.2 Similar web applications

### Codecademy

Codecademy is an education company, which positions itself as an alternative to classical education. Web application they provide is a result of rethinking of current approach on education [12, About].

There are courses on Web Development, Programming and Data Science. In fact the exercises focus on programming languages syntax, the knowledge of which depends mostly on practising. Basic principles behind it, important for the beginners as well, are left without an attention. These principles are mentioned and learned at many universities, which are criticised by Codecademy. However, the courses are great for the ones, whose aim is to gain knowledge of a programming language itself without any learning material above. Sample course of web application for learning Agile programming methods includes explanation of the used methods, their purpose and benefits.

Codecademy popularises TDD, and offers a course on it for an enrolment payment. Also the web offers code reviewing and receiving a feedback, but Codecademy does not provide this functionality directly and the other service Git Hub is used for it.

Codecademy web page and development environment are very user-friendly and pleasing to the eye. The interface of the web application for learning Agile programming methods would be designed to produce the same impression. It is shown on the screenshot: 1.1. The environment is divided into three panes. The left contains the exercise content, the middle one is an editor, and the right one shows an output in the built-in web browser.

## 1.2.3 freeCodeCamp

Another online courses interactive platform is freeCodeCamp. Moreover, it is a community, which helps to learn programming for free and to get experience by contributing to open source projects used by nonprofit organisations. They offer practising with

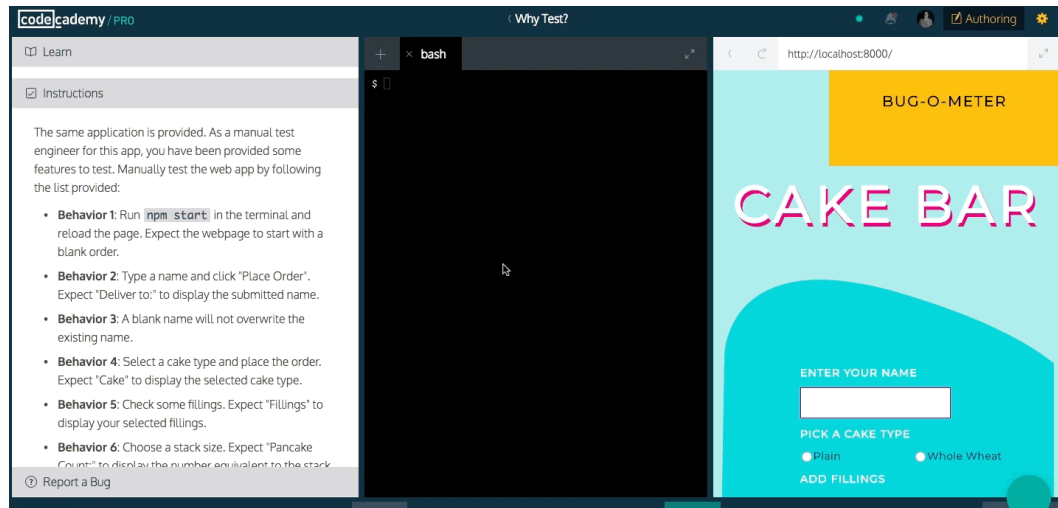


Figure 1.1: Development environment at Codecademy

completing coding challenges and work on the projects. Also joining the study group in the students' city to code in-person is encouraged. On the opposite to Codecademy, freeCodeCamp does not position itself as a high education alternative. They encourage to study to earn a degree and to pursue the courses they provide [16].

Some of the courses take form of projects, this approach would be used in the web application for learning Agile programming methods. The reason is that it depicts the principles and enables practising in the best way.

Like in Codecademy, the web does not provide a functionality of code review of the projects directly. The other service Git Hub is used for it.

They introduce a term User Story while working at the projects to introduce the project requirements. In the web application for learning Agile programming methods the user stories would be used to formulate the requirements needed to accomplish an exercise.

## 1.3 Technologies to be used

The technologies, which are used in the web application for learning Agile programming methods, are introduced and analysed in the following subsections.

### 1.3.1 Client side

This section introduces the technologies, used to create an application on the client side.

## Angular

The client side application uses Angular. It is a structural open-source web application framework led by the Angular Team at Google. Angular enables development of dynamic, modular and reusable dynamical web applications [6].

The Angular framework is built entirely in TypeScript, and it is used as a primary language. As soon as browsers cannot execute TypeScript directly, the code must be converted into JavaScript with tsc compiler [3, TypeScript Configuration].

Angular Material library was used to create consistent design, its usage accelerated the application development and made it possible to focus on its logic.

## HttpClient

The front-end of the application communicates with back-end services over the HTTP protocol. The web browsers support APIs for making HTTP requests of two different types: the XMLHttpRequest interface and the fetch() API. Angular offers the HttpClient, which is a simplified HTTP API that rests on the XMLHttpRequest interface exposed by browsers. Also it includes testability features, typed request and response objects, request and response interception, Observable APIs, and streamlined error handling [3, HttpClient]. Angular makes use of observables as an interface to handle a variety of common asynchronous operations. The observables provide support for passing messages between publishers and subscribers [3, Observables & RxJS].

## Sass

User interface is implemented using HTML5 and CSS3. Sass language is used for writing the style sheets. It is a preprocessor scripting language, which is compiled into CSS. The CSS preprocessors are responsible for recompiling the existing code into the format compliant to the CSS standard. Sass provides such a functionality as multilevel nesting, what improves readability of the code and reduces its volume. It also allows to declare variables. CSS3 also has such a feature, but it is not supported by all the browsers, e.g. Internet Explorer. Mixin insertion solves the problem of a need to duplicate the code. Also Sass enables expanding, inheriting and importing the styles [29].

### 1.3.2 Server side

The server side Java application uses Spring Framework 5.0. It is responsible for business logic of core functionality of the web application, which is compilation of the solution source code and tests, running these tests and returning the result.

## **Maven**

Maven is a software project management and comprehension tool, used by the application on the server side. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. Maven automates such a tasks as downloading dependencies, putting additional jar files on a project classpath, compiling source code, running tests, packaging compiled code into deployable artifacts such as JAR, WAR and ZIP files, and deploying these artifacts to an application server [24]. Such an automatising assures avoiding the mistakes, which may occur when building the software manually.

## **Spring**

The Spring Framework is an open source developed by Pivotal Software application framework and inversion of control container for the Java platform. It provides comprehensive infrastructure support for developing Java applications.

It contains modules, which make development faster and simpler. Spring Boot is an extension of the Spring Framework, a convention-over-configuration solution. In other words, it makes it easy to create applications, which can “just run” [30].

## **Database technologies**

Spring Data is used to provide Spring-based programming model to access the data and relational database PostgreSQL. Spring Data JPA module is used to implement JPA based repositories [31]. Spring Data JPA is a framework that handles most of the complexity of JDBC-based database access and object-relational mappings. Hibernate framework implements the JPA and provides mapping of the application domain objects to the relational database tables. It adds a layer on top of JPA and provides a no-code implementation of the repository pattern and the creation of database queries from method names. [20, Part VI, Persistence]. The CRUD functionality is provided with creating interfaces for every entity, which extend CrudRepository interface. Using this tool made it possible to focus on business logic rather than on setting up the data layer access.

## **Testing technologies**

Java source code and tests are within the web application. The server side application would receive a request to compile the provided source code and tests. JUnit 4 framework is used to write the tests, which is very convenient and intuitive in use.

The compilation and running the code would take place on a virtual machine for security reasons.

# Chapter 2

## Concept

This chapter is devoted to specification and concept of the resulting web application. The application enables creation and management of online courses and interactive solving the exercises. In this chapter, author of the materials would be further called *teacher* and the solver would be called *student*.

### 2.1 Courses overview

Providing the materials for practising agile programming methods is realised in a form of courses. This section contains description of their concept. Concept of a course detail page is illustrated on sketch 2.1 Course detail sketch, its elements are described in detail the following sections.

#### 2.1.1 Courses structure

The learning materials have a structure of courses, and each contains lessons with exercises. The course detail page contains a general information for every course element. Application provides tools for creating and editing course elements, they contain a title and description, and lessons and exercises may be reordered. Buttons “Manage lessons” “and Manage exercises” at course detail page lead to the pages, which provide this functionality. There are five types of exercises, they are specified in detail in section 2.1.6 Exercise types. Exercises have different states, depending on progress. Expansion panel is used to show lessons. Each course can be expanded to show the containing exercises. There is an exercise type and progress shown for each.

#### 2.1.2 Shown and hidden versions

The application provides creation two versions of source code, tests and files. Shown version is shown to the student, when he opens an exercise. Hidden version is not

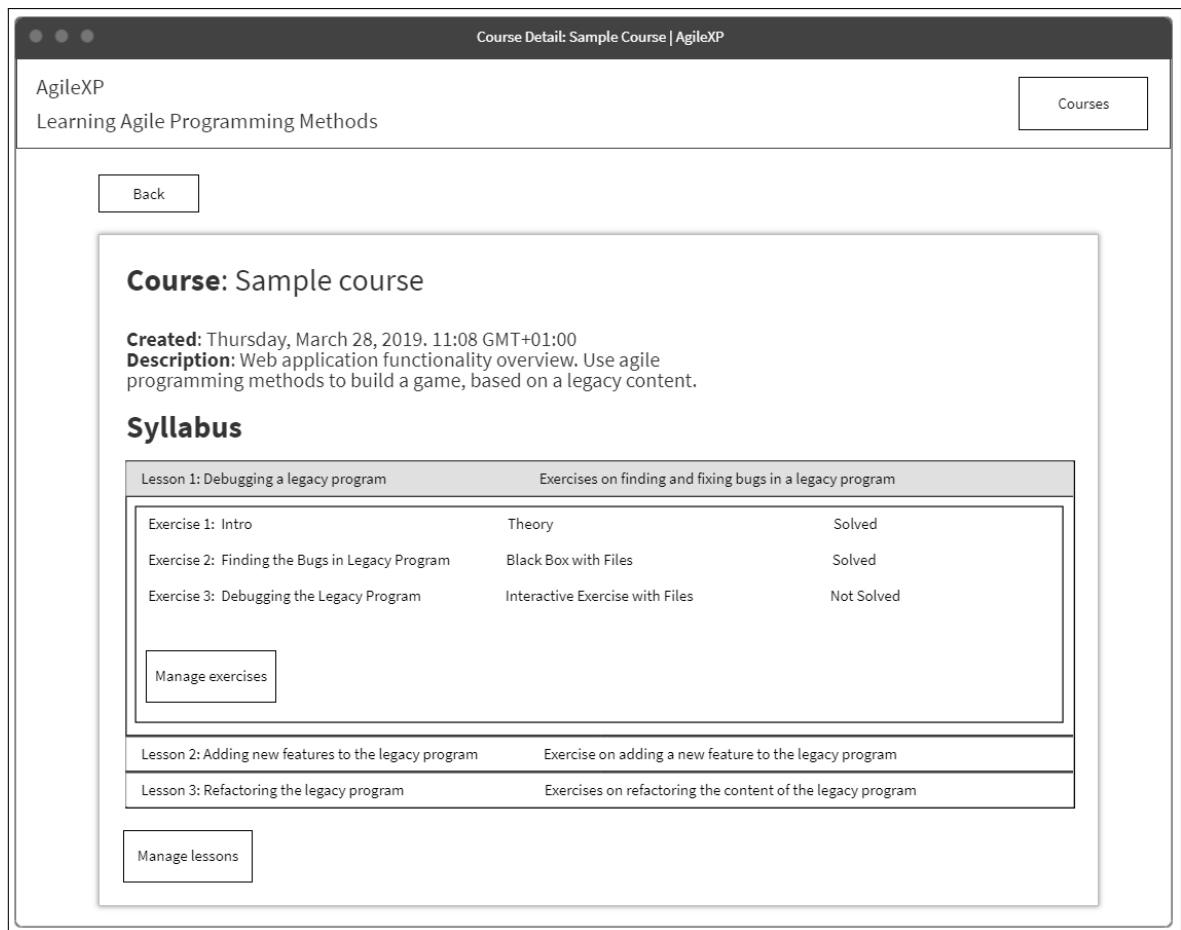


Figure 2.1: Course detail sketch

available to him, it is used to running the tests. In most of the exercises, both of the versions are used, but some of them contain only one.

Introducing two versions may be used to show basic code structure to the student, for providing legacy code, or simply for recalling the syntax. On exercise creation, teacher can simply duplicate the hidden version to fill the shown one, but also to create two different.

### 2.1.3 Development environment

The web application has convenient development environment, and it is close to conventional IDEs. It consists of code editors with multiple tabs, it is easy to add and remove these tabs. The environment enables code execution and providing a feedback. The application supports use of Java 8 programming language in exercises. Testing framework JUnit 4 is used for testing. As soon as the programming language is Java, the feedback includes not only tests results, but also possible compilation errors. A concept of a code editor is illustrated at sketch 2.2 Code editor sketch.



Figure 2.2: Code editor sketch

### 2.1.4 Version control

One of the requirements for the resulting application is version control of the student's solutions. The web application stores and shows source code, tests and files, which are submitted and run by student.

When solving an exercise, the student can view previous versions and copy their files content to the editor. He can access the files from any exercise within a current course. As soon as there is a very large amount of solutions, they all would not be loaded at once, but user may load them page by page. A concept of a version control display is illustrated at sketch of the editor 2.2 Code editor sketch.

This feature is useful for refactoring or working with legacy code, e.g. when student needs to extend tests from previous solution with new ones.

### 2.1.5 Solution estimation

Solution estimations hold the solution reference and time of its creation. The result is represented with three elements: estimation text, estimation value and a solved flag. Structure of estimation text is different for each exercise type, and is described in section 2.1.6 Exercise types. Estimation value is a percentage of progress, and a solved flag tells if it was solved.



### 2.1.6 Exercise types

Environment, which supports only work with source code and estimates solutions automatically, is not enough for practising various agile programming methods. There are five types of exercises to provide different ways of practising agile programming methods separately or in combinations. Their concepts, purpose and usage are described in the following sections.

#### Theory

Exercise type *Theory* is the basic one, it is aimed to provide information to the student. Such an exercise may be used mainly to introduce or conclude a current course or lesson, or to make an overview on legacy code.

It provides only a text area and is marked as solved at the moment, when is viewed by the student. On exercise creation and update, the teacher can use text editor with ability to format, e.g. to provide example source code. The formatted text is provided to the student.

#### Interactive Exercise

*Interactive Exercise* type provides tools for programming. On exercise creation, the teacher is required to fill title and description of exercise. Then he uses editors to bring in a shown version of source code. Next editors are designed for tests. Teacher may use them to fill hidden tests to control the solving progress. Also he may provide shown tests, e.g. to recall the JUnit 4 syntax.

On students side, an exercise of this type contains an exercise description and two editors. The first one is for source code, and the second one is for tests. The student may use the environment to solve an exercise, and he may run his code withing the web application and get feedback. The provided feedback consists of two parts. The first one tells about compilation and tests results for the students code from the development environment. The second one is based on the teacher's code, which is not be available for the student.

#### Interactive Exercise with Files

This exercise type provides the same features as previous one, and expands its tools with working with files. One more text editor contains files, and there are teachers' and students' versions as well. These files may be used for the purpose of testing.

### Black Box

The next exercise type is named *Black Box*, what means a system or a function, for which only its input and output is known. This term describes the main feature of this type, since the student may not see source code, but may send input and get some form of output from the program. This tool is useful for practising unit testing and test-driven development. On working on this exercise, the student may try to find mistakes in the hidden program with the use of tests.

To create an exercise of this type, teacher should fill its title and description. The second should contain code structure or functions names, or any kind of information to give the student needed knowledge about the hidden code. So he would be able to call necessary functions from his tests. Except filling an exercise title and description on exercise creation, the teacher should enter controlling source code, which would be hidden from the student, it should be compilable and contain several logical mistakes. The application would provide information for the teacher on how to design the source code, so it would be used correctly. The tests, which would be shown to the student, would be filled as well. The purpose of these hidden tests is the same as for such a tests in exercise type *Interactive Exercise*.

To pass such an exercise, the student should write tests, which would not be passed by the program. The student would run his tests for the hidden program, and would get feedback, telling the number of found mistakes and their total number.

Estimation process was designed in such a way, so only correct solutions can pass. Incorrect solutions are ignored and not counted to the solution. Examples of such a solutions are the following: the student may write test, which would not be passed, but it would not find the mistake. It may happen, if this test would not pass a correct version of the hidden program as well. This solution should be ignored. Another example is writing several tests, so they would find the same mistake several times. Only one test would be counted for solution, the rest would be ignored.

### Black Box with Files

This exercise type extends previous one with ability to work with files, which are used for testing.

## 2.2 Architecture

The application uses client-server architecture, its components are illustrated on 2.3 Component diagram.

The topmost presentation level of the application is provided with Browser subsystem. It communicates with Server subsystem, which is responsible for business logic

and data access. Server subsystem consists of several components. Server accesses Database server, which belongs to Database subsystem, and an Estimation module.

## 2.3 User interface

This chapter is devoted to user interface concept.

### 2.3.1 Courses

The start page provides a list of the created courses and general information on them. With selecting one of the listed courses, the user may enter it. Also the user may manage the courses, e.g. create, edit or delete them, with the use of the button. This page leads to all of the application pages, and they contain “Back” button for easier navigation.

When a course is entered, a new page contains an information about it. There is an expansion panel below with lessons and exercises they include. Every lesson and exercise may be managed as well. Management includes editing, deleting, reordering and creation. The following sections are devoted to a detailed description of user interface for exercises creation, editing and solving.

### 2.3.2 Exercise creation and editing

Exercise creation and editing has the same structure, with difference in handling the user input and availability of initially defined values. The input fields change dynamically, depending on the exercise type of the exercise. It is possible to change exercise type without losing data the fields contain.

Non-dynamic part contains fields for exercise title, description and exercise type selector. When an exercise type is chosen, the form changes and the corresponding editors appear. Every editor component has the same structure, and differs only with content and handling the user input.

There are three types of editor. The first one is a source code editor, used for exercises of type *Interactive Exercise* and *Interactive Exercise with Files*. The second one is tests editor and used for exercises of type *Black Box* and *Black Box with Files*. And the last one is text editor, used for exercises of type *Interactive Exercise with Files* and *Black Box with Files*.

The first type *Theory* contains only description of the exercise. Exercise type *Interactive Exercise* has editors for source code and tests, and exercise type *Interactive Exercise with Files* has one more editor for text files.

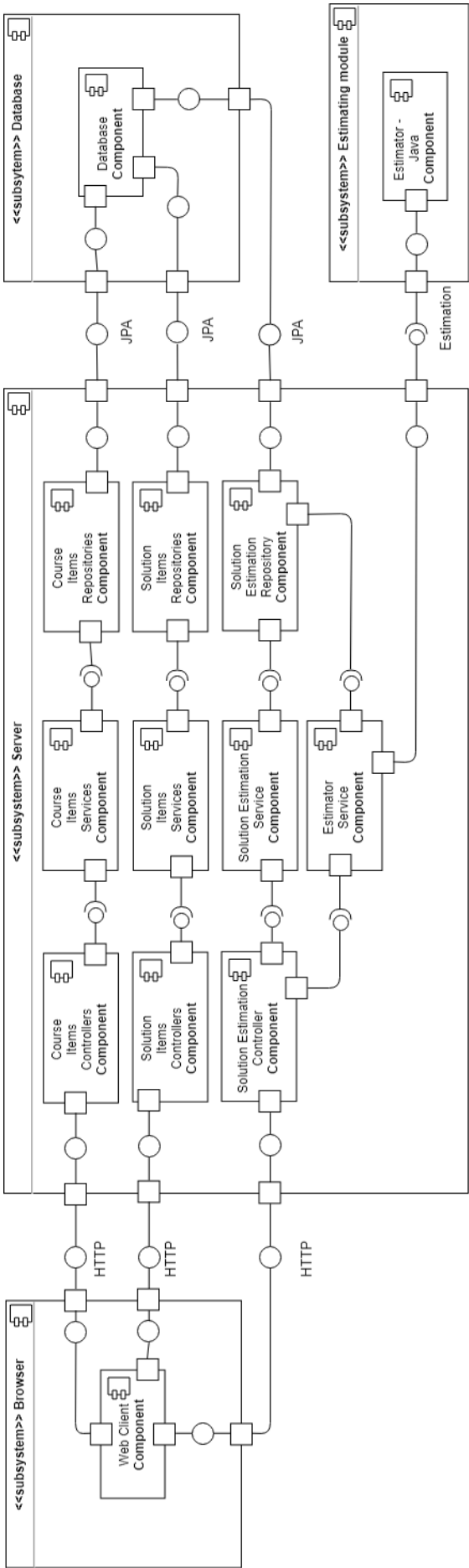


Figure 2.3: Component diagram

Application provides tool for convenient creation of two versions of source code, tests and text files. On exercise creation or update, there is an editor for a hidden version, and there is a select component below. The component contains two options for the shown version. The first option, “the same”, is used when an exercise author wants the shown version to be the same as hidden. And the second option, “custom”, is used to create or edit a different shown version, and a new editor element appears. Some of the exercise types requires only a hidden or shown version of source code or test, so there is no select component.

The submit button is enabled as soon as the form input is valid. On submit, there appears a notification telling if the data were saved successfully. When creating new exercise, there appears suggestion to create another one.

### 2.3.3 Exercise solving

When solving an exercise, its view is determined by its type. Every exercise type has view and functionality according to the concept, described in a section 2.1.6 Exercise types.

Every exercise contains a component for displaying a description. The description content is created with use of rich text editor. Editor components from the previous section are used for exercise solving as well.

Every editor component is extended with version control component, as it required at the concept section 2.1.4 Version control. It enables resetting the editor content and loading content from a submitted version.

### 2.3.4 Run output

On solving exercise of any type, except *Thesis*, there is available a “Run” button for estimation a current result. When an exercise is submitted for the first time, an output area appears. During the estimation process, it contains “Loading...” message. When feedback is ready, it is printed out within the output. Feedback contains compilation and testing result for both public and private versions, and represented as percentage solving progress.

Estimation runs two times for exercise types *Interactive Exercise* and *Interactive Exercise with Files*. First time for public version (running student’s tests on his sources) and second time for private version (running teacher’s tests on student’s sources). The progress is represented as percentage and means how much tests were passed out of their total number.

Testing result for exercise types *Black Box* and *Black Box with Files* contains only number of revealed mistakes and total number of them. The progress is represented as percentage, corresponding to the found mistakes number.

### 2.3.5 Exercise Toolbar

The pages for solving the exercises contain two equal toolbars, one on the top and second on the bottom of the page. They contain the mentioned “Back” button, navigation within the lesson exercises, and there is a flag representing the progress on the solving current exercise. Toolbar view changes dynamically on an exercise change.

## 2.4 Data structure

This section describes data model of the system for the resulting web application. It keeps account of created courses, lessons they contain, and exercises with used code and text files. Also it holds information about submitted solutions, used files and results. Data structure of the application is illustrated on diagram 2.4.

The diagram shows a hierarchical structure of the courses, where each courses (**courses**) may contain several lessons (**lessons**), and each lesson contains exercises (**exercises**). These tables contain information about name, description and creation time. Lesson and exercises items have indexes, which defines in which order they should be solved.

Every exercises has a type (**exercise types**). The ones, which have type *Black Box (with Files)*, have a stored bugs number. Exercises have various types of content, each with different purpose, and they are subsets of **exercise content**.

To store data about solutions, a table **solutions** was created for referencing on a corresponding exercise, and it is referenced from tables **solution estimation** and **solution content**. There are different types of solution files, and they are subsets of **solution content**.

## 2.5 Use cases

The users of the application are divided into two groups, teachers and students. In fact, the application does not separate these groups, so any user can be an author and a solver. Use case diagram illustrates how these two groups interact with the system.

## 2.6 Technical requirements

This section is devoted to the requirements for the resulting web application and their purpose.

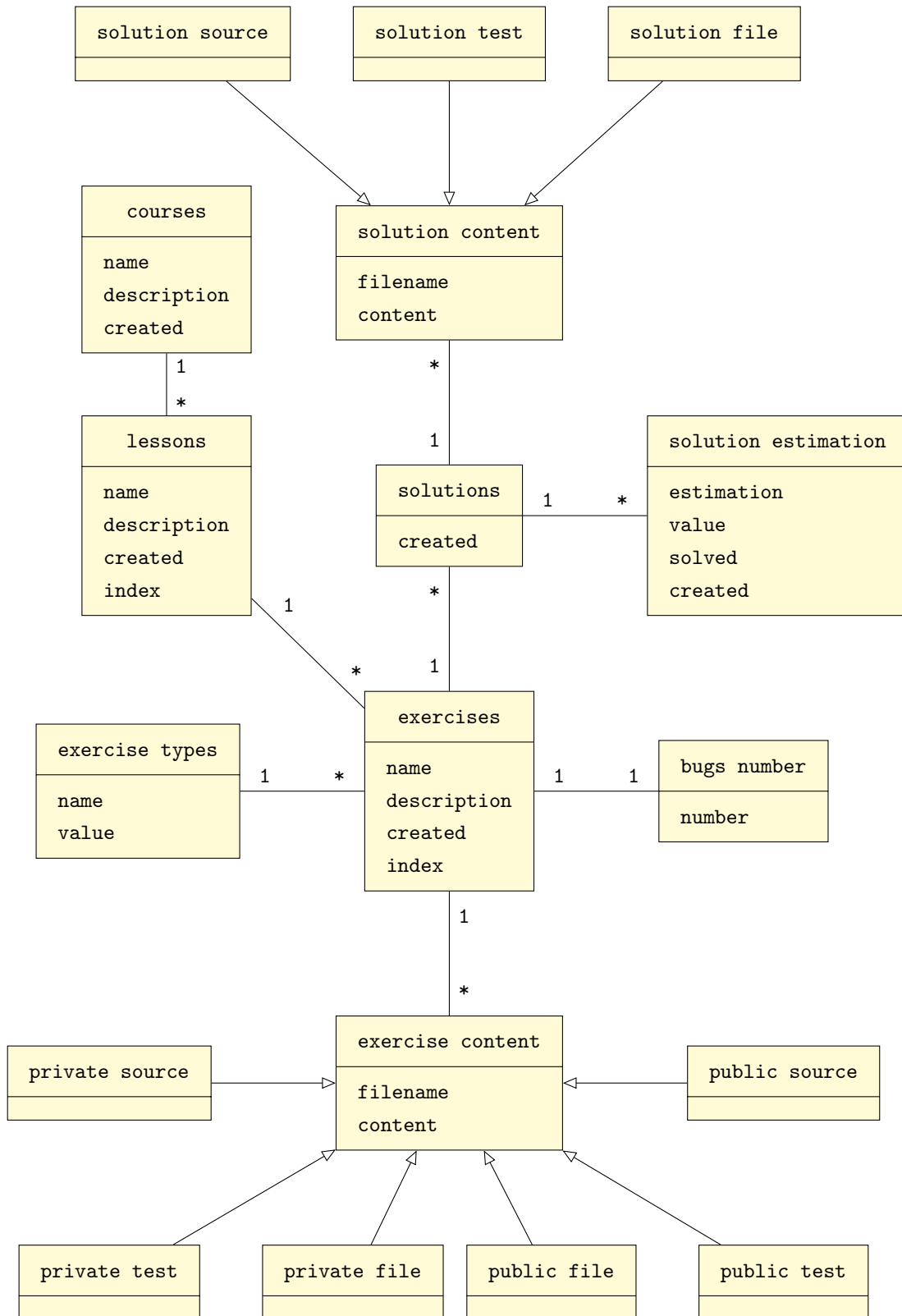


Figure 2.4: Entity relationship model





# Chapter 3

## Solution

This chapter is devoted to the implementation of the web application, which is the result of the bachelor thesis. It was realised according to the concept introduced in the previous chapter 2 Concept to meet the stated requirements.

This application uses full-stack web framework as soon as it supports front-end interface, back-end services and databases development. The application uses client-server model, where front-end corresponds to the client side, and back-end is considered as server side. The following sections describe implementation of every technologies stack component.

### 3.1 Front-end

This section is devoted to the implementation of application, which runs on client side and provides interaction with him.

#### 3.1.1 Angular framework

Front-end uses Angular, a platform and framework for building dynamic applications on the client side. Its primary programming language is TypeScript. Like any Angular application, the result application depends on features and functionality provided by libraries that are available as npm packages, and npm package manager was used to download and install them [6, Node.js].

##### **Angular modules**

Angular provides modules, which provide components implementing common interaction patterns according to the Material Design specification [5, Components]. Components such as Toolbar, Table, Paginator, Dialog, Expansion Panel etc. were used for creating fast, modern and consistent user interface.

## Angular Reactive Forms module

Angular Reactive Forms module was used to create complex dynamic and nested forms. They allow storing inputs from the user or generate multiple nested forms with the same structure. They were used to create and edit courses, lessons and exercises, and also to solve the latter. As a result, it is possible to change the form view as soon as different exercise type is selected. Also they are used to add, remove and rename files at the code editors on exercise creation.

Accuracy and completeness of the input is improved with the form validation, provided by the Form Group. The required fields change dynamically with the form itself.

### 3.1.2 External modules

Except Angular modules, application uses two external modules, which provide editors. Functionality and use of them is described in the following sections.

#### Quill module

The first one is based on Quill rich text editor. It was chosen because its component provides useful text formatting features, e.g. source code, quotations and headers. There is a view of a used quill editor at screenshot 3.1. This component is used to create and edit exercises description.

The code editor is configured to store user input as HTML-string. The editor uses Angular DomSanitizer to sanitise these values, so it also meets a requirement of security.

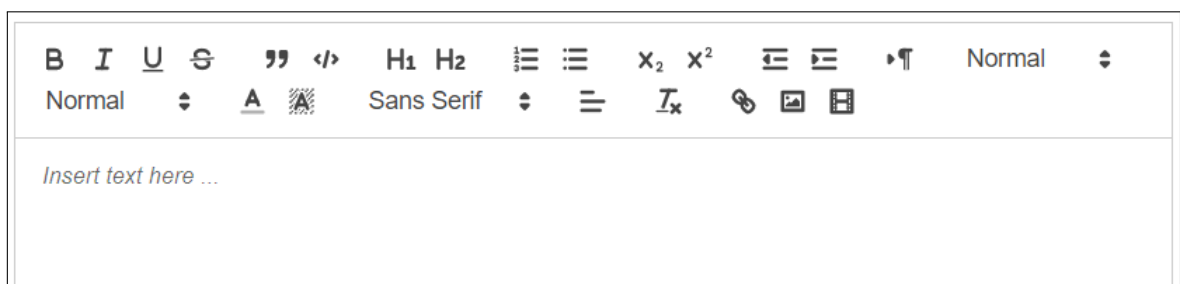


Figure 3.1: Quill rich text editor component

#### Monaco module

The second module is based on Monaco code editor, which powers a high productivity Visual Studio Code editor. Its component was chosen because it has numerous tools to create a powerful development environment. The editor enables highlighting of the Java code, searching, suggestions, minimap, so the requirement of providing a close to

conventional IDE was met. There is an example of use of Monaco editor module at screenshot 3.2.



Figure 3.2: Monaco code editor

### 3.1.3 Custom modules

Angular provides tools for making program modular. There are three modules in the application: courses, lessons and exercises. They contain components, routing modules, service providers. Components of these modules present data and define the view together with HTML templates. Routing modules enable navigation from one view to the next as users perform application tasks. The services are used for encapsulation of data access. The service providers handle sending requests to the server and retrieving the responses with use of HttpClient [6, Architecture]. The modules, components, routes and services were generated with use of Angular CLI, a command line interface.

The created components are used to display pages or sections. They are reusable, so some of them are used for different purposes, e.g. to create and edit a course, to manage a course and a lesson. Also they are used to display repetitive sections to avoid duplicates, e.g. show three same editor elements with different content.

#### App module

App module is not quite custom as soon as it is the root module and every Angular application must have it. This module defines features, used by the whole application, e.g. the main navigation toolbar. This module imports Courses module and Not Found Module, described in the following sections.

## Courses module

This module contains components for viewing and management the courses. It contains four components. Two of them, Manage and Upsert, are shared with imported Lessons and Exercise modules. The rest two components deal with courses only.

Courses module contains course model to hold data and service providers to send and receive them from the server.

**Courses list component** This module is defined in App Module as a start page of the application. This component displays a list of courses, with general information. The table view was implemented with use of Angular Table module. Angular Paginator module provided component for pagination to limit the shown items number.

There is an option of management the courses, which leads to Manage component. Example is shown on screenshot 3.3.

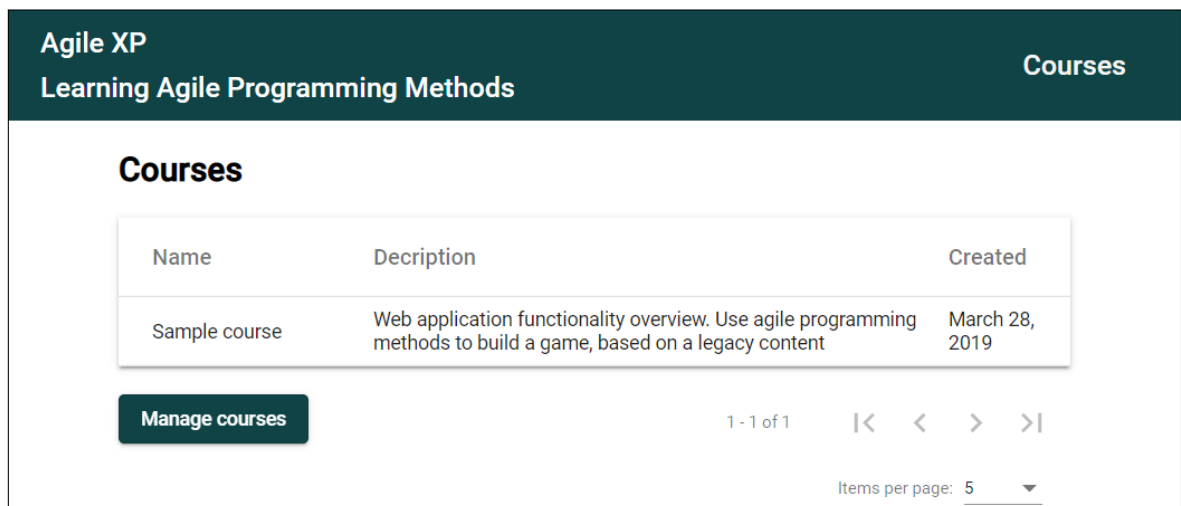


Figure 3.3: Courses list

**Course detail component** This component provides detailed information about one of the courses, chosen from the courses list. It provides a complete overview to acquaint with the course. There is an extension panel, implemented with use of Angular Extension Panel module, under the general information on the course, to list the course lessons, and to list the exercises from the lesson on click.

Course detail component lead to Manage component to manage both lessons and exercises.

Example is shown on screenshot 3.4.

**Manage component** The previous components references to this module, as it was defined in the router of the Course module. This component provides tools for creating,

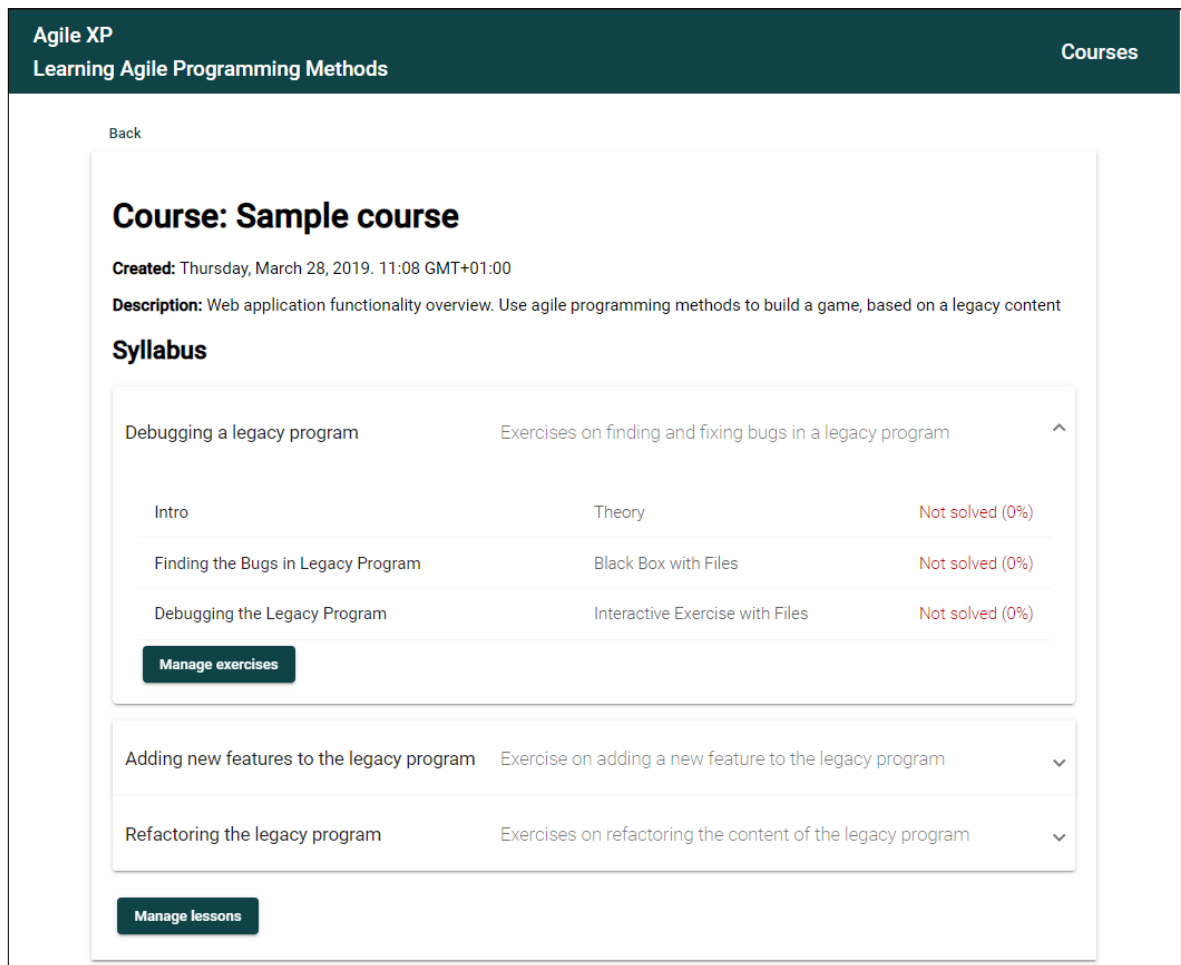


Figure 3.4: Course detail

editing, deleting, reordering the courses, lessons and exercises. The reason of this merge is that management has the same structure and mostly the same functionality for three manageable items.

The Manage component contains abstract class to define general functionality, like creating, editing and deleting. Three classes for managing courses, lessons and exercises, inherit from it and use a common HTML template and styles. Lessons and exercises management is extended with reordering feature, implemented with use of Angular Drag and Drop module.

Both create and edit buttons lead to Updert module, which is also used for different purposes.

Example is shown on screenshot 3.5.

**Upsert component** The name of this module is a reference to a database management feature, which means a combination of *insert* and *update* [28, ON CONFLICT Clause]. Upsert module is used to creation and editing the courses and lessons.

The reason is the same as for Manage module. Exercises creation and editing is

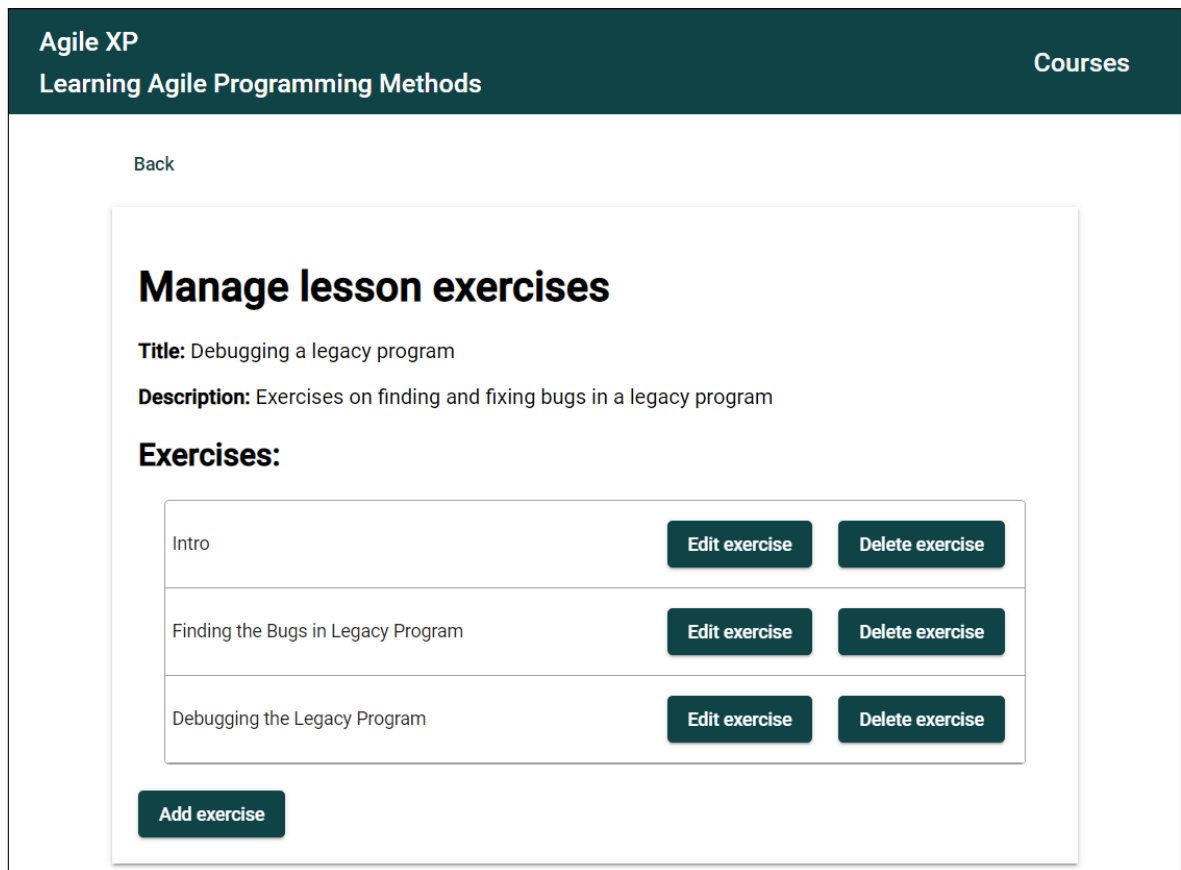


Figure 3.5: Manage component

not that simple, to it required another module.

### Lessons module

This module is imported by Courses module, so Lessons module can use its components. Lessons module also contains lesson model and services, and Courses module can access them. This module does not contain any component, and only uses shared components from the Courses module.

### Exercises module

Exercises module is the most complex. It contains several models and services, and various components for one or many purposes.

**Editor component** Exercise module has Exercise Upsert and Exercise Solve components, and each of them has editors. Each of them is implemented with Editor component. It uses tab view to represent several files. Content of any file may be manipulated with use of Monaco code editor. Angular Tab Module was used to create tabs, which contain file name and buttons for renaming and deletion. The last tab

contains a button for adding a new file. The default file name is “filename.java” or “filename.txt”, and it may be changed in a dialog window, shown on screenshot 3.6. Extension .java or .txt is added to a new file name from the input. Dialog component implemented the dialog with use of Angular Dialog module.

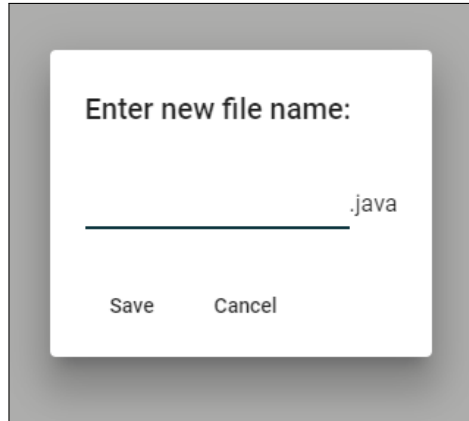


Figure 3.6: Filename change dialog

**Exercise Upsert component** Exercise creating and editing is implemented with the same component, which contains various nested components. Angular Reactive Forms were used to create a dynamic form to interact with the user and store his input. Content of fields, text and code editors from the nested components are bound to the form control and is processed on submit. This component also uses abstract component, inherited by creating and editing component.

Non-dynamic part contains fields for exercise title, description and exercise type selector. Angular Form Field component was used for title, Quill rich text editor component was used for filling the description, and Angular Select was used for selecting an exercise type. This part is implemented with Create Intro component.

When an exercise type is chosen, the form changes and the corresponding editors appear. The same component was used to place multiple editors with the same structure. The used form enables changing exercise type without losing data, and they would be available in the appeared editors.

Each editor is implemented with Editor component. View of the whole component is shown on screenshot 3.7.

Dynamics of the form was used to fulfil a required feature of convenient creation of two versions of source code, tests and text files. Upsert Editors component implements this feature. On exercise creation or update, there is an editor for a hidden version, and there is a select component below. The component contains two options for the shown version. The first option, “the same”, is used when an exercise author wants the shown version to be the same a hidden. As a result, the content of the editor would

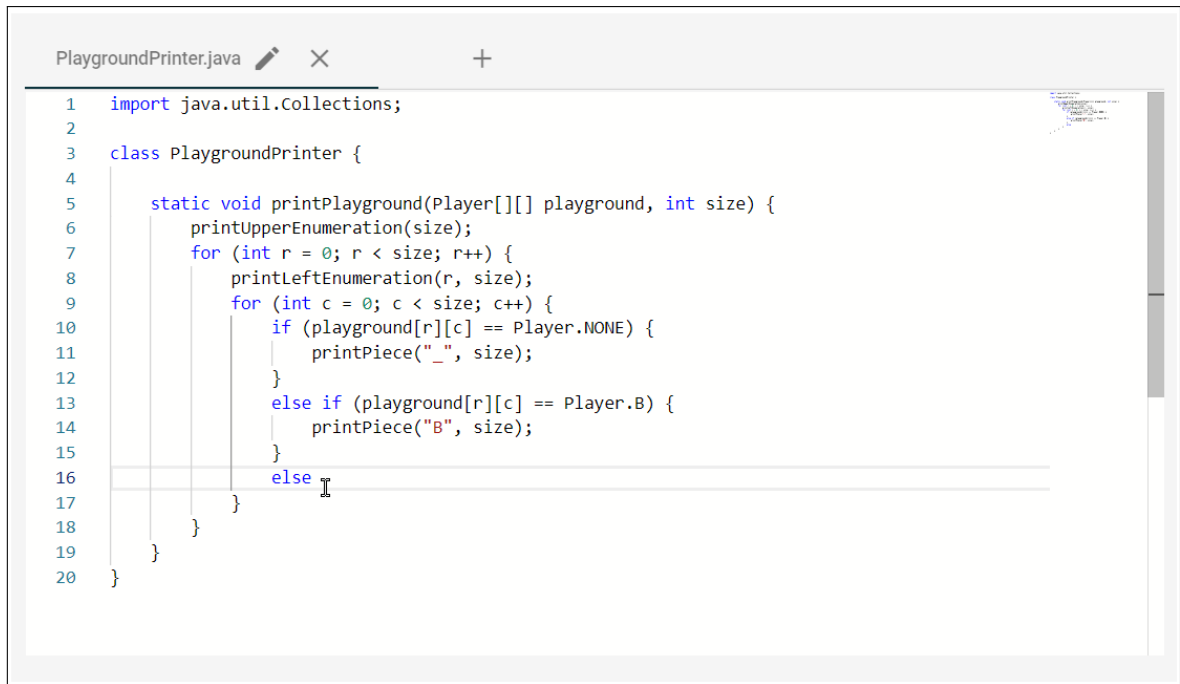


Figure 3.7: Editor element on exercise create or editing

be saved as both hidden and shown version. And the second option, “custom”, is used to create or edit a different shown version, and an new editor element appears. Some of the exercise types require only a hidden or shown version of source code or test, so there is no select component.

Create Submit component displays a submit button, which is enabled as soon as the form input is valid. On submit, there would appear a notification telling if the data were saved successfully. When cheating new exercise, there would be an suggestion to create another one.

Submitting of the exercise is realised with Exercise Creator and Editor services, called from the Submit Component. Both of them inherit from abstract Exercise Saver service.

**Exercise Solve component** This component has a complex structure too, but this component is used only for solving the exercises. Interaction with the user and storing his input was also implemented with use of dynamic forms.

When solving an exercise, its view is determined by its type. Every exercise type has view and functionality according to the concept, described in a section 2.1.6 Exercise types. Exercise Solve component determines the number, content of the editors and a nested form, to which an editor is bounded to.

Solve Editor component consists of an editor, which is implemented with code Editor component, and a version control section. A typical code editor is illustrated on screenshot 3.8. Editor component uses Monaco module, it is possible to add, rename



and remove files. Version control enables resetting the editor content and loading content from a submitted version. When an exercise is chosen, a dialog appears. It views eight most recent submitted solutions. User may require getting more from the server with “Load more button”, and use paginator to display them. Dialog for loading a solution from version control is illustrated on screenshot 3.9. Expansion panel was used; when a solution is chosen, preview of a submitted code is expanded.

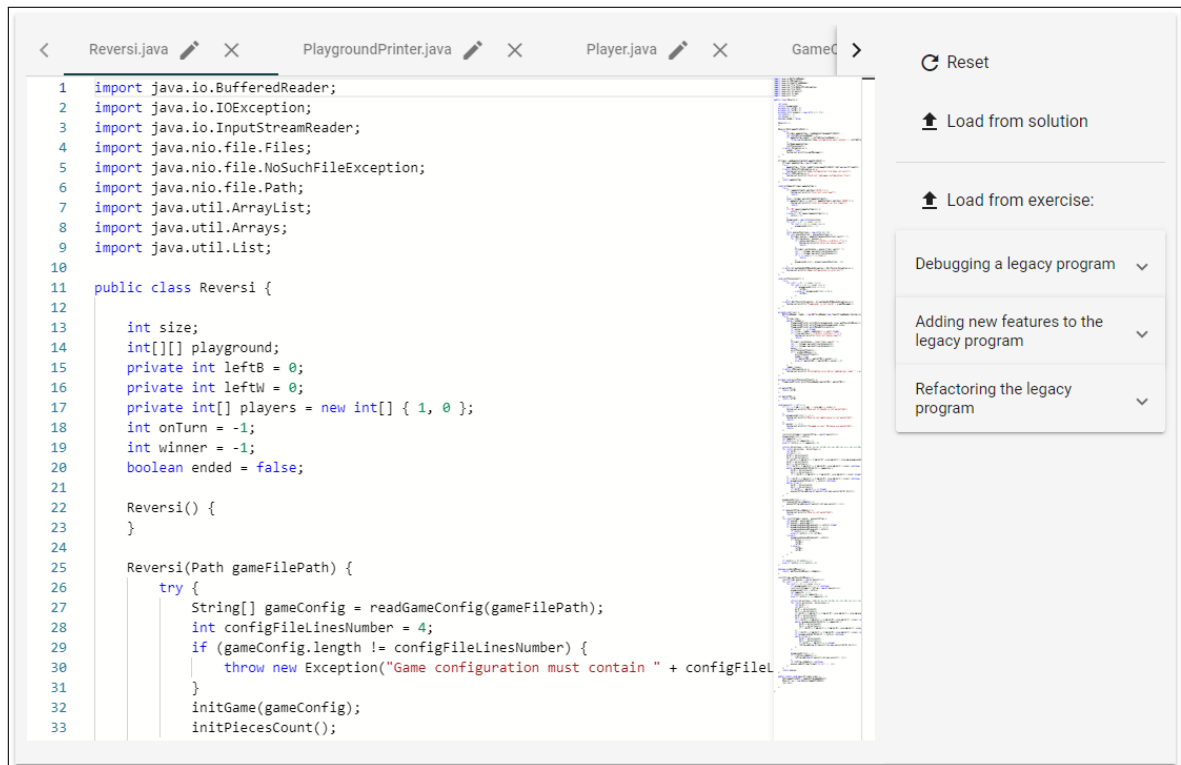


Figure 3.8: Editor element

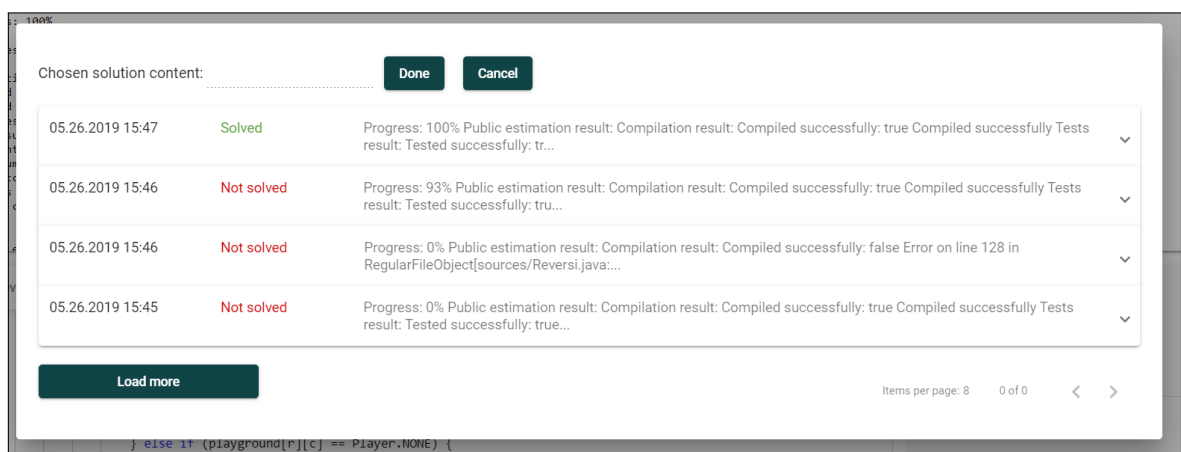
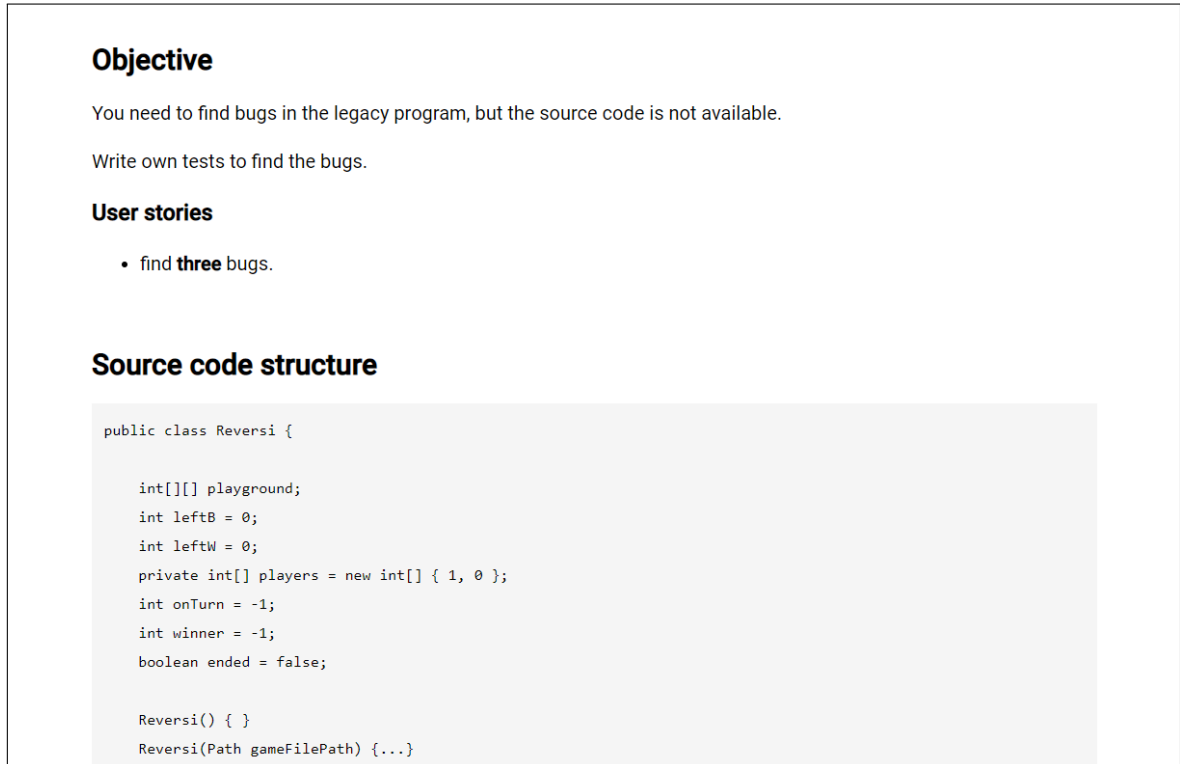


Figure 3.9: Version control

Every exercise contains a Intro component for displaying a description of the exercise. The description content was created with use of rich text editor component of

Qill module, and its content is formatted. There is an example at screenshot of the exercise description 3.10.



**Objective**

You need to find bugs in the legacy program, but the source code is not available.

Write own tests to find the bugs.

**User stories**

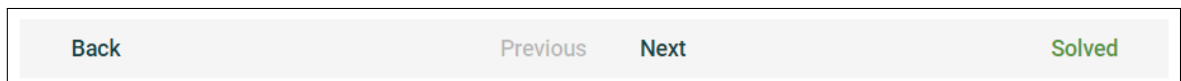
- find **three** bugs.

**Source code structure**

```
public class Reversi {  
  
    int[][] playground;  
    int leftB = 0;  
    int leftW = 0;  
    private int[] players = new int[] { 1, 0 };  
    int onTurn = -1;  
    int winner = -1;  
    boolean ended = false;  
  
    Reversi() { }  
    Reversi(Path gameFilePath) {...}
```

Figure 3.10: Exercises description

The pages for solving the exercises contain two equal toolbars, on the top and at the bottom of the page. They are implemented with Toolbar component with use of Angular Toolbar. Route module defines navigation back to Course Detail component with button, or navigation to previous or next exercise. Also there is a flag representing the progress on the solving current exercise. An example of the toolbar view is available at screenshot 3.11. They change dynamically on an exercise change and restrict from navigating to not available exercise.



Back Previous Next Solved

Figure 3.11: Exercises toolbar

Solve Run component is responsible for running the estimation of the submitted result. Firstly, when “Run” button is clicked, an output area appears and it contains “Loading...” message. Submitted source code, tests, files, solution itself and estimation should be saved to database, and they references, so they should be sent to the server in order. Sending requests to the server runs asynchronously with use async-await feature to make code more readable and call functions sequentially without making code hell

of nested calls and conditions. The received estimation is shown to the user in the output area.

### **Not found module**

This module is imported by the App Module, and is used for redirection to it if the routing modules does not contain a requested path.

## **3.2 Back-end**

This section describes implementation of application, which runs on server and is responsible for handling business logic and data storage.

The server side Java application uses Spring Framework, and is managed with Maven. HTTP requests are handled by Controller components. Service components are used to implement business logic on a separate layer. One of them, Estimator Service component, receives estimation from estimation module, which creates it with Estimator-Java Component. Every data entity has own Repository component, which extends CrudRepository to provide CRUD functionality for the entity class that is being managed.

### **3.2.1 Virtual Sandbox**

Virtual Sandbox is used to fulfil a security requirement, stated in 2.6.1 Security. The result application runs potentially harmful running of the code, provided by user, in a docker container. Compilation may be performed on server, and application would be still secure. But it would require dependencies, which may not be available on server. This section describes virtual sandbox usage.

When server gets a request to estimate the solution, it gets an object with Request Body annotation, which is mapped to a domain object. It is deserialised to a Java object, which contains objects describing the solution files. They are stored to a named by solution is temporary file, stored at an uploading directory. Storage package contains tools for managing files in this uploading directory.

Then Dockerfile with archived Java program are copied to the temporary solution folder. The next step is running the container. Docker Client plugin, provided by Spotify, contains tools for working with Docker. A class was created to use them, it takes care of building an image, container creation, starting it, getting an estimation file content and destroying the container. The result is parsed from JSON format, and based on it estimation object it created.

### 3.2.2 Estimating module

Estimating module is realised as Java program, which runs in Docker container. It uses Maven as well as a server application.

Server application stores Dockerfile and jar of the program. On beginning of the estimation process, these files are moved to a temporary directory for solution.

Instead of executing commands from the server application or defining them in Dockerfile, the estimator program runs in the Docker container. It brings such a benefit as modularity, because it becomes easier to add more supported for estimation programming languages. For writing the server application may use this or another such a module, depending on the program. Also it brings an opportunity to define the output of compilation, testing, error messages.

The estimator program contains two classes for estimating a solution. The first class is responsible for exercise type *Interactive exercise* and *Interactive exercise with Files*, and the second handles *Black Box* with *Black Box with Files*. The both classes extend abstract class *Estimator* and implement estimation function, which starts the process.

The result of the estimation is hold in an instance of *Estimation* class. It has an attribute *value*, which represents the progress percentage. For *Interactive exercise (with Files)* type, the progress means how much tests were passed out of their total amount. Result class of JUnit API collects and summarises information from running multiple tests, but it does not provide information on total tests number. However, this number can be received during the tests running with JUnit API. *TestResult* class was created to store the total tests number. Its constructor takes a *Result* instance, copies its attributes, and adds a necessary attribute. Finally, the test result with compilation results are stored in an instance of *Estimation* class.

The estimation result is created with converting the estimation object into JSON string with use of *Gson* plugin from Google. This output is written to a text file. Docker Client plugin enables getting an archive of a filesystem resource in a container, so the file with estimation can be read with the server application.

Another way to deliver the result to the server is printing it to the console, and Docker Client may receive it. This simple solution has one problem. When solution program prints to console, its output also would be received by the server. As a result, writing the result to the file is more efficient.

## 3.3 Database

The resulting web application uses PostgreSQL relational database management system. The entity-relationship model of the database system should have been trans-

formed into relation model for database creation and further implementation.

Most of the entities are in many-to-one, one-to-many or one-to-one relationship. These relationships were transformed with use of references and unique constraints.

The set of entities **exercise content** has the same attributes as the entities, which are in a subset relationship with it. Single Table strategy was used for transformation. As a result, all the subsets are stored in the same table **exercise content**, and are distinguished by **exercise content type** attribute. Hibernate Inheritance Mapping was used to implement this relation as inheritance, a Single Table inheritance strategy was used. The same approach was used on transformation from entity-relationship model to relation model for table **solution content** and subsets of its entities.

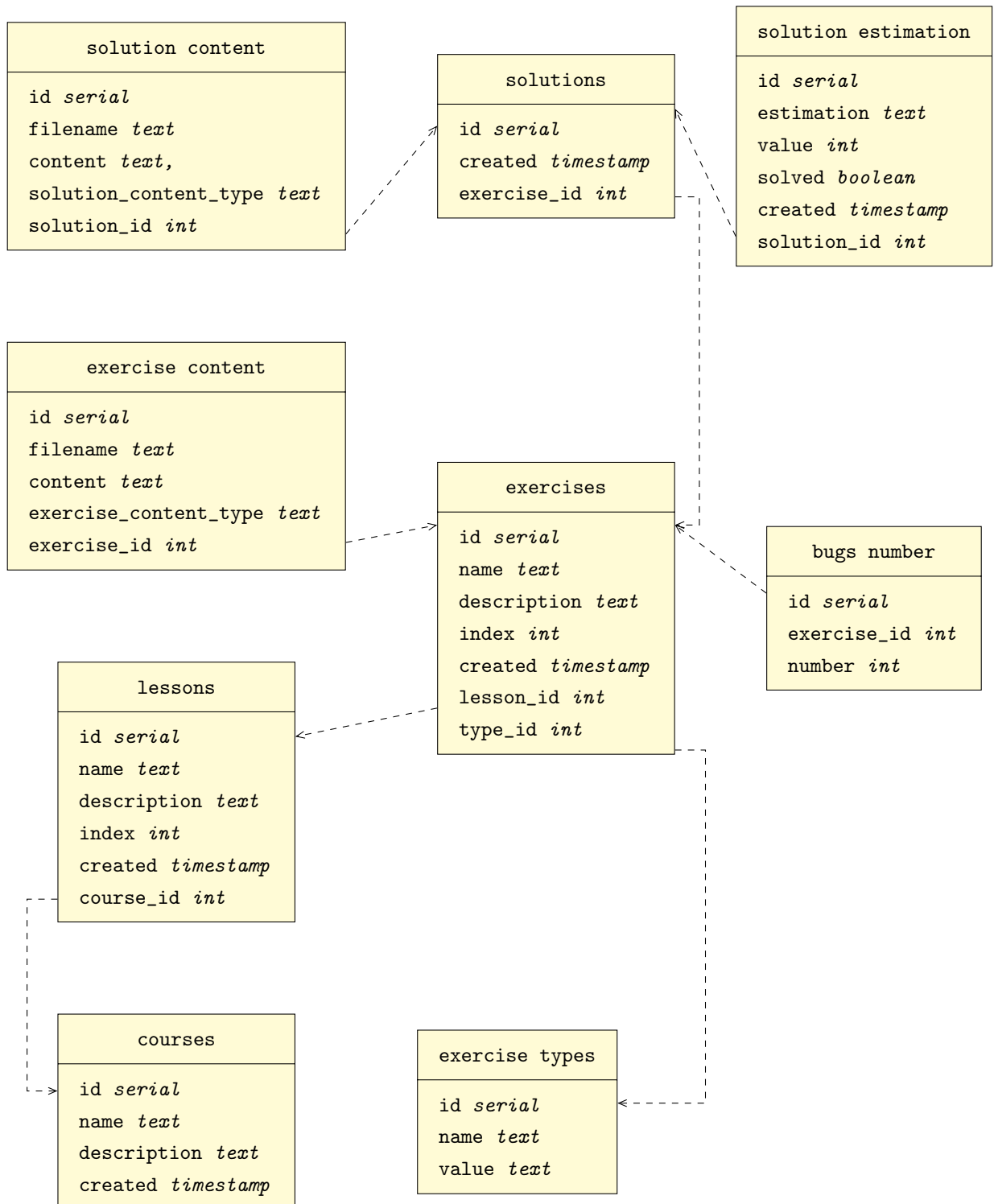


Figure 3.12: Relational model of system data

# Chapter 4

## Sample Course

In order to present the functionality and usage of web application, a sample course was created. Its exercises include all the exercises types and agile programming methods. The following section will lead through the content of the course.

### 4.1 Sample course overview

There are three lessons in the course, each represents an iteration of the work on the program. Every lesson contains exercises, which lead through the work on the project. The course is designed in such a way, so the user would learn agile programming methods and apply his skills while working on the project. Most of the exercises would cover multiple skills, like test-driven development, unit testing, refactoring and working with legacy code.

Section *Legacy program overview* gives more information about the program, which is worked on during the course. Further sections *Lesson 1 to 3* describes lessons content, purpose, and which agile programming methodologies do they introduce.

#### 4.1.1 Legacy program overview

The student would work on an interactive Reversi game, based on a legacy program. It is a console application for two players. Each Reversi piece has a black side and a white side. The state of the game is read from configuration files, which have the following structure: first row has "B" (black) or "W" (white), which means the player on turn; the second row contains positions of black pieces; and the third row contains positions of white pieces. When the game starts, program asks a player on turn to enter piece position to move on. The game ends when player on turn has no pieces to move on. The one, who has more pieces on the board, wins the game.

### 4.1.2 Lesson 1: Debugging a legacy program

The first lesson is called *Debugging a legacy program* and it contains three exercises, which step by step lead through looking for mistakes and fixing them.

As soon as the programmer does not know how legacy program works, even if he is the author, it becomes more difficult to fix mistakes and not produce more. And using tests becomes more crucial than usually. The purpose of the following exercises is to show how testing is useful for fixing program correctly and efficiently.

#### Exercise 1: Intro

The first exercise is introductory. It contains no assignments, but acquaints the user with general information about the course, its lessons, exercises and the legacy program.

#### Exercise 2: Finding the Bugs in Legacy Program

Work with legacy code starts from the second exercise. The user would get program, which contains three mistakes. As soon as this exercise is of type *Black Box with Files*, the user would be provided only with development editors for tests and files. He would not be able to see or change source code of the program.

Exercise is designed in such a way, so the user would practice test-driven development. The user would need to write tests to find the mistakes. On submit, the user would get a feedback, telling how much errors did he reveal and their total number.

The mistakes relate to marginal cases and incorrect user inputs. Original program works incorrectly on such an inputs: move on piece, which is not empty; move on not adjacent piece, what violates the rules; and move out of the bounds of the board.

#### Exercise 3: Debugging the Legacy Program

When user reveals mistakes in the previous exercise, these mistakes would be corrected in this one. It is recommended to use tests, which were submitted in the previous exercise, and web applications provides such a feature to make work with legacy code more convenient. This exercise is interactive, so user would solve it with changing the program files, running the modified program and get feedback on his solution.

### 4.1.3 Lesson 2: Adding new features to the legacy program

The second lesson is devoted to the most often kind of work with legacy code, which is adding a new feature. Adding new functionality may be as risky as fixing the mistakes, so proper testing is important too. The aim of this lesson is to show how unit testing can be helpful for making sure, that program remains functional. Also this lesson would introduce of safe and efficient methods of changing legacy code.



**Exercise 1: Board Size**

In this exercise the student should add a new feature, and to use tests to be sure that the program is solvable. The code uses magic constants, which represent a board size. The student should use a variable instead. When refactored, a new feature can be implemented with the use of sprout method. The configuration files contain one more line for configuring a board size, sprout methods would read them and store size to the variable.

**Exercise 2: Show Hints**

This exercise is aimed to acquaint with one of the techniques of safe legacy code modifying. Student should add a new feature of showing hints, and a new code should be isolated from the old one. Sprout technique should be used, so old code would not be modified, and it would be possible to test a new feature.

**4.1.4 Lesson 3: Refactoring the legacy program**

The last lesson is devoted to the most efficient way to tidy code and make it easier to aim three factors of code quality and cleanliness: readability, maintainability and extensibility. Refactoring should be done in the end and no other code changes should be done at the same time. Because of this, this iteration of programming is done within the last lesson of the course. Each of the following exercises is devoted to one of the techniques of refactoring and leads step by step through them. These techniques will be practiced on the legacy code from previous lessons, as soon as it is a good example of program with lack of refactoring, and it becomes a good proof of necessity of it. Exercise titles refer to related chapters of the book “*Clean Code: A Handbook of Agile Software Craftsmanship*” by Robert C. Martin. Also exercises contain citations from it, which describe the reason and idea behind the used techniques in the best way.

**Exercise 1: Constants versus Enums**

This step of refactoring is aimed to make the code more readable. Original legacy code uses integers to represent cells to tell if a black, white or no piece is placed on it. The task is to replace these repetitive numbers with constant variables. In this case it is more reasonable to prefer *enum* with name *Player* rather than *constant*, as soon as its meaning cannot be lost, because they belong to an enumeration that is named [23, Constants versus Enums].

**Exercise 2: No Duplication**

The next step of refactoring is described as ‘*the primary enemy of a well-designed system*’ by Robert C. Martin, and he explains why: “*It represents additional work, additional risk, and additional unnecessary complexity*”[23, No Duplication].”

Source code from the exercise contains repetitive code, and the task is to eliminate such a duplicitous with extracting new functions and use them. The most noticeable advantage of this change is that instead making changes in every occurrence of such a code, it is enough to change it once in one place.

**Exercise 3: Do One Thing**

The next exercise introduces an important principle of programming, which is applied to programming entities on many levels, e.g. lines of code, functions, classes, and even refactoring itself. The legacy code contains several cases of violating this principle[23, Do One Thing].

The aim of the exercise is to extract new functions from the others, which do more than ‘one thing’. As a result, work with code becomes more efficient. It becomes possible to test application on more levels, to use methods of changing legacy code is needed, and to understand what function does at one glance.

**Exercise 4: One Level of Abstraction per Function**

When the core steps of refactoring are completed, this exercise introduces often neglected step of refactoring, which improves code readability significantly.

This exercise suggests the same solution of extracting methods as the previous ones. As a result of this separation, it becomes clear which lines of code express an essential concept or a detail [23, One Level of Abstraction per Function].

**Exercise 5: Small!**

When a function is too big to fit into the screen, it makes it clear, that it should be refactored [23, Small!]. Previous refactoring steps lead not only to the mentioned benefits, but also reduce size of the functions. Most of them used extraction of one functions from another to aim the goals, but this exercise suggests one more approach.

The task is to change the structure of storing values, which represent the number of black and white pieces on the board. When using a map instead of two integers, one of the function significantly reduces, and readability is not only maintained, but also improved.

**Exercise 6: Error Handling I & Exercise 7: Error Handling II**

The last two exercises conclude the lesson with leading through handling the errors. They belong to the lesson on refactoring, as soon as it is important to write clean code to handle errors.

There are two kinds of errors, so work on them is separated into two exercises. Legacy code uses a harmful practice of printing messages to console and returning *void*, when any error occurs. It leads to unexpected program behaviour, which may be difficult to reveal, repeat and correct. The exercises suggest use of custom exceptions to catch errors efficiently.

# Conclusion

The main aim of the bachelor thesis was to develop the web application, which would introduce and provide environment for practising agile methods of programming. As soon as these methods are initially taken as counter intuitive, it is even more important to try them to understand why they are considered to be efficient and useful. An interactive form enables leading through every step, explaining it and providing feedback.

The resulting web application reached this goal. It is accompanied with a course, which not only illustrates the functionality, but also goes through the theoretical and practical aspects of agile programming methods.

## Further development

Besides the aimed functions of the application, it may be extended with features, which would enable using it at school or university classes, or as a massive open online course.

Adding an authentication feature would enable adding a feedback and discussions modules. The teachers would be able to track the visits and progress on the course. Also it would be possible for a teachers' team to collaborate on courses. The application enables free passing from one exercise to another for purpose of demonstration. With adding the authentication feature, the teachers would get an option to restrict this feature. A new role of administrator would appear, who would be responsible for registration and courses content management.

The application enables adding different modules for solution estimation. These modules may extend the supported languages and their versions, or suggest another approach of estimation.

The resulting application may become a core another one, which introduces not only aspects of agile programming, but also of agile management.

# Bibliography

- [1] Agile Alliance. What is Agile Software Development?, 2013. <https://www.agilealliance.org/agile101/>, last accessed on 26/05/2019.
- [2] The 12th annual State of Agile Report. <https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report>, last accessed on 26/05/2019.
- [3] *Angular Guide*. <https://angular.io/guide/>, last accessed on 26/05/2019.
- [4] *Angular Guide*. <https://angular.io/cli>, last accessed on 26/05/2019.
- [5] Angular Material: Components. <https://material.angular.io/components>, last accessed on 26/05/2019.
- [6] Angular: Getting started. <https://angular.io/guide/quickstart>, last accessed on 26/05/2019.
- [7] Angular: Reactive Forms. <https://angular.io/guide/reactive-forms>, last accessed on 26/05/2019.
- [8] Kent Beck. *Test Driven Development: By Example*. O'Reilly Media, 2002.
- [9] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change*. O'Reilly Media, 2004.
- [10] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for Agile Software Development. <http://agilemanifesto.org/>, last accessed on 26/05/2019, 2001.
- [11] Tomáš Bočinec. Webová výuka programovania pomocou metodológie TDD, 2018.
- [12] Codecademy web page: About. <https://www.codecademy.com/about>, last accessed on 26/05/2019.

- [13] Lee Copeland. Extreme Programming. *Computerworld*, 2001. <https://www.computerworld.com/article/2585634/app-development/extreme-programming.html>, last accessed on 26/05/2019.
- [14] Ward Cunningham. Ward Explains Debt Metaphor. <http://wiki.c2.com/?WardExplainsDebtMetaphor>, last accessed on 26/05/2019, 1992.
- [15] Michael C. Feathers. *Working Effectively with Legacy Code*. Prentice Hall PTR, 2005.
- [16] freeCodeCamp web page: About. <https://about.freecodecamp.org/>, last accessed on 26/05/2019.
- [17] Martin Fowler. The New Methodology. *Wuhan University Journal of Natural Sciences*, 2005. [https://www.researchgate.net/publication/225478666\\_The\\_New\\_Methodology](https://www.researchgate.net/publication/225478666_The_New_Methodology), last accessed on 26/05/2019.
- [18] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Reading, MA : Addison-Wesley, 1999.
- [19] Ron Jeffries. Essential XP: Card, Conversation, Confirmation. <http://ronjeffries.com/xprog/articles/expcardconversationconfirmation/>, last accessed on 26/05/2019.
- [20] *The Java EE 6 Tutorial*. <https://docs.oracle.com/javaee/6/tutorial/doc/bnbpy.html>, last accessed on 26/05/2019.
- [21] Donald E. Knuth. Structured Programming with go to Statements. *ACM Computing Surveys (CSUR)*, last accessed on 26/05/2019, 1974. <https://dl.acm.org/citation.cfm?id=356640>.
- [22] Daniel Krajč. Webové vývojové prostredie pre jazyk C++. Master's thesis, Univerzita Komenského v Bratislave. Fakulta matematiky, fyziky a informatiky. Kat-edra aplikovanej informatiky, 2010.
- [23] Robert C. Martin. *Clean Code. A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, 2009.
- [24] *Apache Maven Project: What is Maven?* <https://maven.apache.org/what-is-maven.html>, last accessed on 26/05/2019.
- [25] Data Validation: Sanitize. [https://www.owasp.org/index.php/Data\\_Validation#Sanitize](https://www.owasp.org/index.php/Data_Validation#Sanitize), last accessed on 26/05/2019.

- [26] Top 10-2017 A7-Cross-Site Scripting (XSS), 2017. [https://www.owasp.org/index.php/Top\\_10-2017\\_A7-Cross-Site\\_Scripting\\_\(XSS\)](https://www.owasp.org/index.php/Top_10-2017_A7-Cross-Site_Scripting_(XSS)), last accessed on 26/05/2019.
- [27] David Parsons, Ramesh Lal, and Manfred Lange. Test Driven Development: Advancing Knowledge by Conjecture and Confirmation. *Future Internet*, 2011. [https://www.researchgate.net/publication/220103002\\_Test\\_Driven\\_Development\\_Advancing\\_Knowledge\\_by\\_Conjecture\\_and\\_Confirmation](https://www.researchgate.net/publication/220103002_Test_Driven_Development_Advancing_Knowledge_by_Conjecture_and_Confirmation), last accessed on 26/05/2019.
- [28] PostgreSQL 9.5.17 Documentation: Insert. <https://www.postgresql.org/docs/9.5/sql-insert.html>, last accessed on 26/05/2019.
- [29] *Sass basics*. <https://sass-lang.com/guide>, last accessed on 26/05/2019.
- [30] *Spring Boot: Overview*. <http://spring.io/projects/spring-boot>, last accessed on 26/05/2019.
- [31] *Spring Data: Overview*. <http://spring.io/projects/spring-data>, last accessed on 26/05/2019.
- [32] Nassim Nicholas Taleb. *Antifragile: Things That Gain From Disorder*. Penguin Books, 2012.
- [33] Don Wells. Never Add Functionality Early. <http://www.extremeprogramming.org/rules/early.html>, last accessed on 26/05/2019, 1999.
- [34] Don C. Wells. Extreme Programming: A Gentle Introduction. [https://www.researchgate.net/publication/246155995\\_Extreme\\_Programming\\_A\\_Gentle\\_Introduction](https://www.researchgate.net/publication/246155995_Extreme_Programming_A_Gentle_Introduction), last accessed on 26/05/2019, 2003.
- [35] Laurie Williams. Integrating Pair Programming into a Software Development Process. *CSEET '01 Proceedings of the 14th Conference on Software Engineering Education and Training*, 2001. <https://dl.acm.org/citation.cfm?id=794899>, last accessed on 26/05/2019.