TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Computer Engineering

IAG70LT

Jan Toodre 153157 IASM

LEARNING ENVIRONMENT FOR PROGRAMMING IN C

Master Thesis

Supervisor: Vladimir Viies PhD
Co-Supervisor: Lembit Jürimägi MSc

Tallinn 2017

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutitehnika instituut

IAG70LT

Jan Toodre 153157 IASM

PROGRAMMEERIMISKEELE C ÕPPEKESKKOND

Magistritöö

Juhendaja: Vladimir Viies PhD
Kaasjuhendaja: Lembit Jürimägi MSc

Tallinn 2017

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication. All works and major viewpoints of the other authors, data from other sources of literature and elsewhere used for writing this paper have been referenced.

Author: Jan Toodre

May 31, 2017

Abstract

The goal of this thesis is to provide solutions on how to create a learning environment which can be used to ease the process of teaching the C programming language.

The thesis will provide technologies and tools best suited for such an application. The key components of this environment are virtualized environment for students to practice their programming skills, automated grading and means of getting almost immediate results. To achieve such goals, multiple technologies are explained in greater detail. After that, the process of developing such a system and how people can continue contributing to it is described. As a result of this thesis and environment development, the productivity of professors will be increased vastly by allowing them to shift their focus from grading to teaching.

The thesis is in English and contains 56 pages of text, 4 chapters, 13 figures, 6 tables.

Annotatsioon

Lõputöö eesmärk on pakkuda välja lahendusi probleemidele, mis esinevad programmeerimise õpetamisel. Peamisteks probleemideks on erinevad arenduskeskkonnad iga tudengi arvutis ja aeg, mis kulub õppejõududel tööde kontrollimiseks. Esimene probleem avaldub siis, kui tudengid soovivad teha töid oma arvutites, kuid igal tudengil on kasutada erinev operatsioonisüsteem, mis toob kaasa ühildusprobleemid, kohustades õppejõude keskenduma eripärasustele tudengite töödes. Võttes kasutusele keskkonna, kus kõik saavad kasutada ühtseid tööriistu, on võimalik välistada süsteemispetsiifiliste probleemide teket ning tagada õppejõududele parem võimalus õpetamisele keskenduda.

Teise põhiprobleemi, hindamise ajamahukuse, lahendamiseks on vaja luua süsteem, mille abil on võimalik tudengeid hinnata ühtsete kriteeriumide alusel. Sellise ühtse süsteemi leidmine ja rakendamine on oluline, kuna see annab võimaluse rutiinsed tegevused automatiseerida, hoides nii kokku õppejõudude aega muude tegevuste jaoks.

Lisaks käsitleb lõputöö tehnoloogiaid, mida oleks võimalik kasutada antud süsteemi arendamiseks. Kasutades töös välja toodud tööriistu, pakub autor välja protsessi süsteemi arendamiseks, alustades projekti ülesehitusest, panustajatest ja soovituslikest töövõtetest ning lõpetades stiili juhendiga.

Lähtudes lõputöö autori kogemustest, on võimalik leida, et sellise süsteemi loomine on vajalik selleks, et olulisel määral parandada õppejõu töö efektiivsust. Olemasolevad lahendused ei vasta täpselt nõuetele ning ei ole kohandatavad, seega tuleb süsteem ise välja arendada.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 56 leheküljel, 4 peatükki, 13 joonist, 6 tabelit.

Glossary of Terms and Abbreviations

API	Application Programming Interface - Interface with clearly defined methods for communication between software components
CI	Continuous Integration
CD	Continuous Delivery
CDN	Content Delivery Network
CLI	Command Line Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
ES	ECMAScript - JavaScript standard
GCC	GNU C Compiler
Git	VCS - Tracking changes in files
HTML	Hypertext Markup Language
Hypervisor	also known as virtual machine monitor
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
MOOC	Massive Open Online Course
npm	node package manager
OS	Operating System
POC	Proof of Concept
RESTful	Representational state transfer
Schema	Data structure description
SEO	Search Engine Optimization
Slack	Popular Instant Messaging application
SQL	Structured Query Language
SSH	Secure Shell - Cryptographic network protocol for operating with network services
Stack Exchange	Questions & Answers community
TA	Teachers Assistant
TUT	Tallinn University of Technology
VCS	Version Control Software
webhook	method of altering the behaviour of web application with custom callbacks

Contents

List of Figures

List of Tables

Introduction

Learning computer programming is a thing everyone should do [4]. Not only does doing it help one tackle all kinds of problems, but by learning to program one learns to separate problems into smaller pieces, making them easier to grasp; thus, making the quest to find a solution a faster and easier process. In addition to that, should one be exposed to multiple fellow programmers' codes to the same problem, one will see that there are multiple ways to approach a certain task.

In addition to the reason of finding creative solutions to problems, there are many other reasons for programming. For example, the ability to automate tedious tasks or finding a way to do them in a more efficient way. Since people need some variance in their activities, they tend to be more prone to errors when solving very repetitive tasks. However, today we have computers available which are perfect for performing such tasks. For example analysing log files with specific fields of data - creating a simple Python script could save a lot of copy-paste effort and process very massive input files within seconds. In addition, computers do not require salary, sleeping time and food - resources that most people cannot do without.

The author of this thesis is working as a teacher's assistant at the time of writing, where a few problems that require immediate solving have arisen. Firstly, grading tasks is repetitive and time consuming for professors, but can easily be automated. Secondly, there is a lack of a uniform set of tools; thus, each new tool used brings a new set of problems to be dealt with. The main goal of this thesis is to solve those two problems by creating an automated testing environment and learning environment, in which the used tools are the same.

The thesis will offer an overview of the possible tools, frameworks and concepts used for the development of the learning environment. The goal of the output of this thesis is to provide guidelines and suggestions on how to create such an infrastructure to automate grading and provide software development environment for students.

1. Learning Programming

Learning Programming in Tallinn University of Technology (TUT) will give students an overview of algorithms and programming in C. Even though creating algorithms and creative problem solving are the main topics in IAG programming courses, teaching algorithms is out of the scope of this thesis and as a result will not be discussed any further. In the process of learning C, students are exposed to multiple core programming concepts. First experience will be the famous *Hello World!* program, after which variables, conditions, loops, functions, and so forth are introduced. Each of those concepts need to be practised, which results in creating multiple programs with simple and predictable outputs. However, grading these tasks can be quite demanding for professors- not because of the student competencies, but because of time. It is essential to have professors dedicate their time to teaching, not repetitive tasks which will eventually drain them. Therefore, the creation of learning environment must be undertaken.

1.1. Background and motivation

Being a teacher's assistant in programming courses IAG0581 and IAG0582 has exposed the author to the world of being a teacher, including the woes that come with the profession. The generic workflow of teaching programming in TUT can be summarised as follows:

1. Create materials for the lab;
2. Create examples for the lab;
3. Create tasks for the lab and set the deadline and means of submitting work;
4. After task's due date, grade submissions;
5. Insert results into Google Sheets for the students to see.

This will cycle throughout the whole semester. Some of these tasks can be done once and later on reused and, if necessary, modified. However, what cannot be reused in order to be more efficient or save time is grading the tasks and publishing the results. This has to be done for every student and every task, which is time-consuming and repetitive. These steps can and should be automated as much as possible. As an illustrative example, we can assume that an experienced professor takes 5 minutes to check a student's task - this includes finding the student's e-mail address with the source code, downloading, opening, investigating, compiling, running, testing, grading the task and sometimes checking for plagiarism. Calculating the total time spent on merely grading could be done with equation 1, where *i* is the number of students, *t* is time and *n* is the number of tests.

$$t_{total} = \sum_{i=1}^{i} t_{grading} * n_{tests} \tag{1}$$

Assuming that the professor has 100 students each semester and hands out tasks once every two weeks - this will result in approximately 66.6 hours of time spent on grading per semester. The author finds that spending this much time on grading can quickly become draining and tedious for the professor, especially if the same time could instead be directed towards creating new tasks, materials and focusing on teaching. This in turn would result in a happier professor.

1.2. Classroom Environment

TUT ICT building has several computer classes, which are equipped with SUSE Linux Enterprise Edition. The good thing about having Enterprise Edition of Linux is the long term support and long life cycles. However, there is also a downside to it - license fees and outdated third party software packages. The former prevents the environment to be set up by students - it is unreasonable to require the students to buy expensive software for a programming course. The latter, however, will prevent students from using the latest software. As an example, IDE Code::Blocks latest available version supported in that environment is multiple major releases behind from the latest Code::Blocks release. These are just some of the reasons why setting up a uniform environment for students and professors is preferable.

As it was mentioned before, students tend to use their own computers instead of classroom ones if possible. This will result in various problems, starting with compatibility issues. The bare minimum software required for C programming is available in classroom computers. A comparison of tools used in various students' computers and school's computers is given in Table 1.

Table 1. Comparison of classroom and students environments

Environment	School	Students
Operating System	SUSE Linux Enterprise Edition, Windows 10	Linux (variable), Mac OS X (variable), Windows (variable)
IDE	Code::Blocks, Geany, SciTE	Code::Blocks, Dev-C++, Geany, Visual Studio Code, Visual Studio, Xcode
Compilers	GCC	Apple LLVM Compiler, Clang, GCC, LLVM GCC, Visual-C++
Debugging tools	gdb, strace, Valgrind	ddd, gdb, instruments, strace, Valgrind, Visual Studio Debugger

Having too many occurrences of different software in students' computers is a problem - each tool and system has its own oddities. However, both of these environments have issues - on one end of the spectrum there is a vast variety of different software and on the other end there is possibly outdated software. To solve those problems, the author of this thesis urges that an unified environment should be created.

2. Learning Environment

The main goal is to develop a learning environment which has all the necessary tools for development and an automated grading system - the former meant to be distributed and used by students and latter for grading their work without having the professors investigate every student's work separately. In this section, the ways of achieving such results are explained in more detail.

2.1. Requirements

The main requirements for such an environment are:

- Uniform programming environment;
- Possibility to add new tasks;
- Automated grading;
- Instant results.

It is important to have the previously mentioned qualities in this environment. However, they do not necessarily have to be within one system. This leaves the option to achieve the goals by using parts of systems which are already available.

2.2. Existing Systems

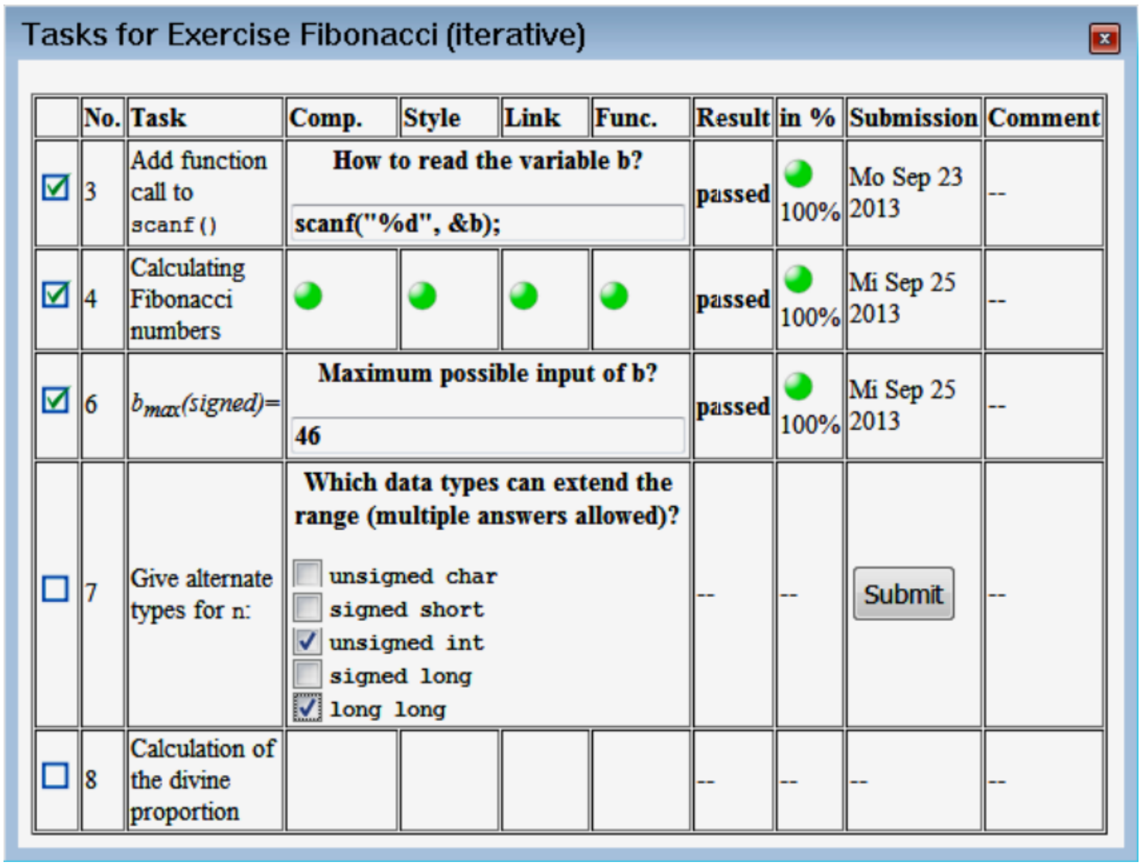
After some research, it appears that there is abundance of learning environments. These environments vary from general purpose e-learning environments such as Moodle and Blackboard to MOOC platforms such as edX, Udemy and Coursera. There are, of course more programming oriented environments such as Codecademy and HackerRank. While the latter offers a good hands on learning experience it lacks the possibility of adding new tasks and a grading system which could be integrated with TUT system. For the reason of having vast variety of learning environments author decides to compare the solutions most similar to proposed system.

Although a lot of solutions are available, there are no perfect matches which offer an environment with all the qualities mentioned earlier.W hen picking an existing solution, at least one requirement must be met. As an extra requirement it must be possible to adjust the rest of the system to the needs of this environment.

2.2.1. Virtual-C

Virtual-C is an IDE which helps teaching programming in C [1]. The system was developed with a few goals in mind - reduce the failure rates in *UniBW München* and enhance students capabilities in C programming.

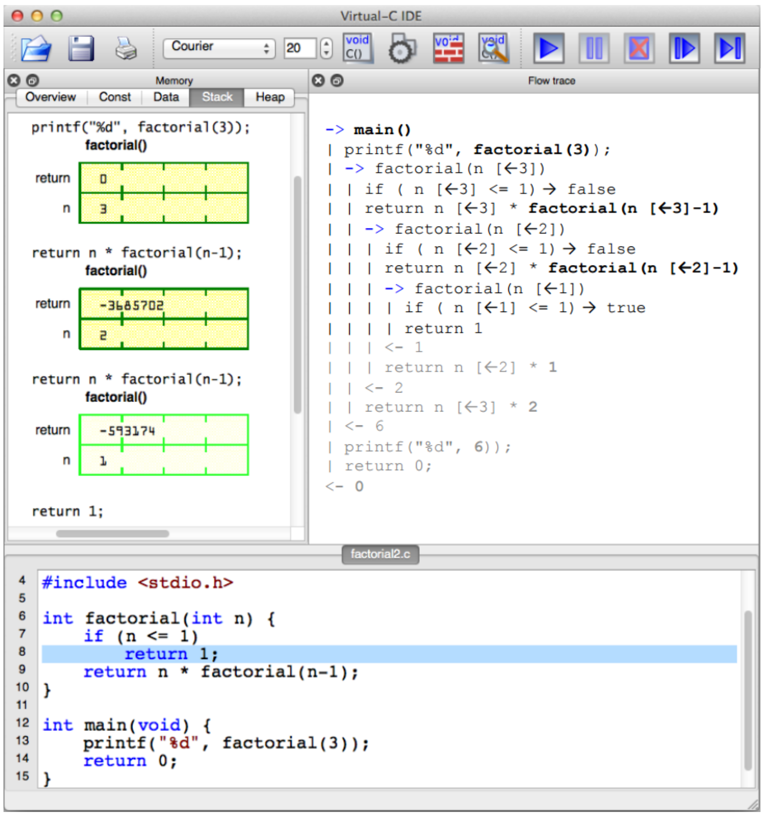
Mentioned IDE has bundled multiple useful tools within itself in order to achieve the goals set. Such tools include having integrated functional code testing, quiz and tasks within IDE (See Figure 1) and visual aids in form of memory layouts (See Figure 2). Having these functionalities greatly increases the speed of learning and understanding of programming concepts.



Tasks for Exercise Fibonacci (iterative)

	No.	Task	Comp.	Style	Link	Func.	Result	in %	Submission	Comment
☑	3	Add function call to `scanf()`	How to read the variable b? `scanf("%d", &b);`				passed	● 100%	Mo Sep 23 2013	--
☑	4	Calculating Fibonacci numbers	●	●	●	●	passed	● 100%	Mi Sep 25 2013	--
☑	6	$b_{max}(signed)=$	Maximum possible input of b? 46				passed	● 100%	Mi Sep 25 2013	--
☐	7	Give alternate types for n:	Which data types can extend the range (multiple answers allowed)? ☐ `unsigned char` ☐ `signed short` ☑ `unsigned int` ☐ `signed long` ☑ `long long`				--	--	Submit	--
☐	8	Calculation of the divine proportion					--	--	--	--

Figure 1. VIDE - Visualizing the flow trace of a recursive function [1]



```
printf("%d", factorial(3));
        factorial()
return  0
n       3

return n * factorial(n-1);
        factorial()
return  -3685702
n       2

return n * factorial(n-1);
        factorial()
return  -593174
n       1

return 1;
```

```
-> main()
| printf("%d", factorial(3));
| -> factorial(n [←3])
| | if ( n [←3] <= 1) → false
| | return n [←3] * factorial(n [←3]-1)
| | -> factorial(n [←2])
| | | if ( n [←2] <= 1) → false
| | | return n [←2] * factorial(n [←2]-1)
| | | -> factorial(n [←1])
| | | | if ( n [←1] <= 1) → true
| | | | return 1
| | | <- 1
| | | return n [←2] * 1
| | <- 2
| | return n [←3] * 2
| <- 6
| printf("%d", 6));
| return 0;
<- 0
```

```c
#include <stdio.h>

int factorial(int n) {
    if (n <= 1)
        return 1;
    return n * factorial(n-1);
}

int main(void) {
    printf("%d", factorial(3));
    return 0;
}
```

Figure 2. VIDE - Example of a quiz with integrated functional tests [1]

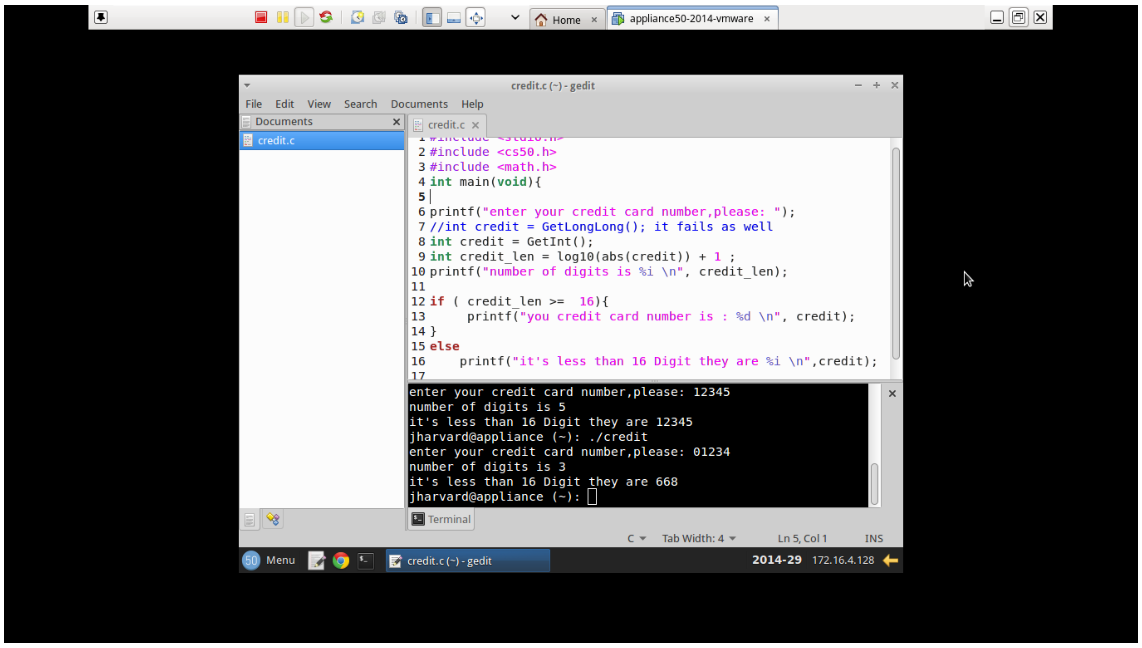
This IDE has solved its main goals by dramatically increasing students capabilities and decreasing dropping out rates. The strongest points of this system are the visual aids and its live coding and testing ecosystem. However since the project is with a closed source code making it impossible to integrate with our system.

2.2.2. CS50

Harvard University offers a course called *Introduction to Computer Science - CS50*. As a part of this course it aims to teach students how to think algorithmically and solve problems efficiently. Throughout the course following topics are covered: algorithms, data structures, encapsulation, resource management, security,m software engineering, and web development. Languages include C, Python, SQL and JavaScript plus CSS and HTML [5].

Historically students enrolled in CS50 had to be familiar with UNIX CLI basics. This was due to the course structure which was that students had to SSH to Harvard server cluster and then solve tasks there [6]. To get rid of that unnecessary constraint and offer the same uniform development environment for students and staff a virtual appliance was created.

The main resources available in CS50 Virtual Appliance (*appliance50*) for the students to use are check50, gedit, clang, Dropbox and cs50 header file. Screenshot of *appliance50* is in Figure 3.



```c
#include <cs50.h>
#include <math.h>
int main(void){

printf("enter your credit card number,please: ");
//int credit = GetLongLong(); it fails as well
int credit = GetInt();
int credit_len = log10(abs(credit)) + 1 ;
printf("number of digits is %i \n", credit_len);

if ( credit_len >=  16){
     printf("you credit card number is : %d \n", credit);
}
else
    printf("it's less than 16 Digit they are %i \n",credit);
```

```
enter your credit card number,please: 12345
number of digits is 5
it's less than 16 Digit they are 12345
jharvard@appliance (~): ./credit
enter your credit card number,please: 01234
number of digits is 3
it's less than 16 Digit they are 668
jharvard@appliance (~): 
```

Figure 3. CS50 - appliance50 [2]

The biggest pro with this solution is the development environment which is same for the staff members and students. However the task pool is limited and because the tasks seem to be defined in appliance each new task requires a new release of virtual appliance making this approach not viable.

2.2.3. ISCX

Teachers going through the topics at a certain pace is slightly aged way of teaching. Using the capabilities of today's technologies students should learn whenever, wherever at their own pace [7]. With this approach Vello Kukk has developed the system ISCX to allow this exact methodology.

ISCX is a learning environment in TUT, mainly integrating courses from Department of

Computer Systems. While this course has had at its peak over 10 courses, at the time of writing thesis there are six courses available of which four have been passed by the author.

The system uses competence system which is distributed between multiple skills and topics. Similar competences can be found in *My Field* in a nearby area (See Figure 4). Whenever students are learning a new topic they are clicking on a box on *My Field*, get and solve a task and if solved correctly, get points for it. The better students do the more 'blue' their field will look. By using complex algorithms the answers given in certain competence will affect adjacent topics as well thus making the studied topics stick by creating connections.

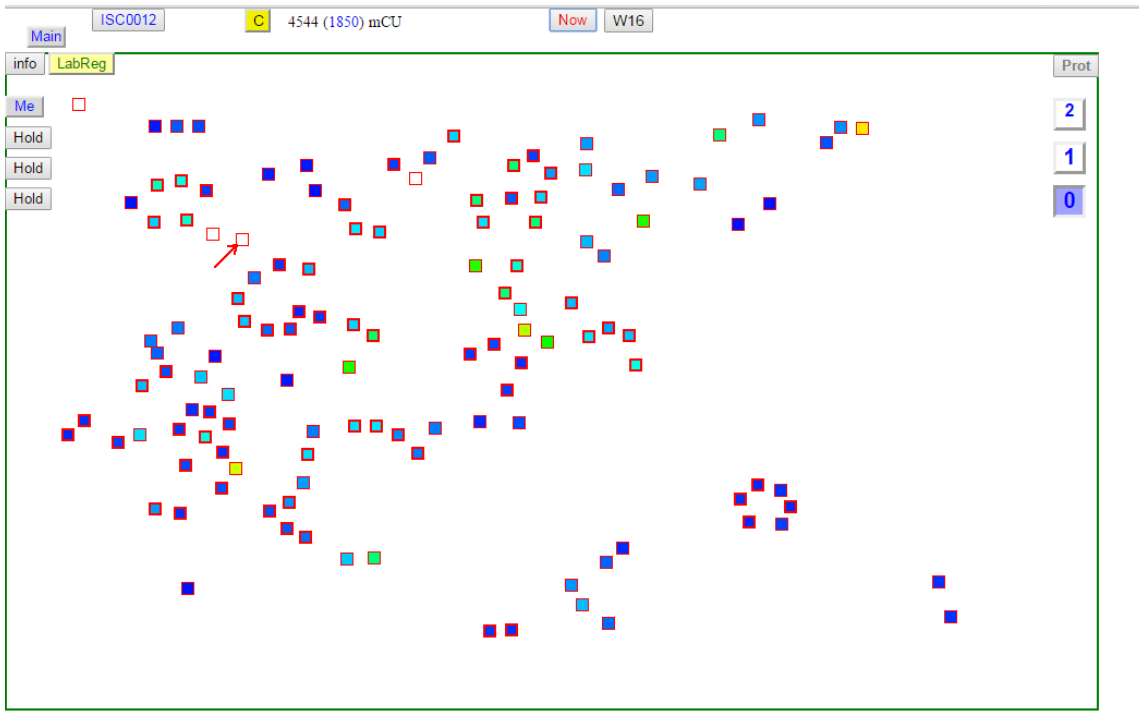
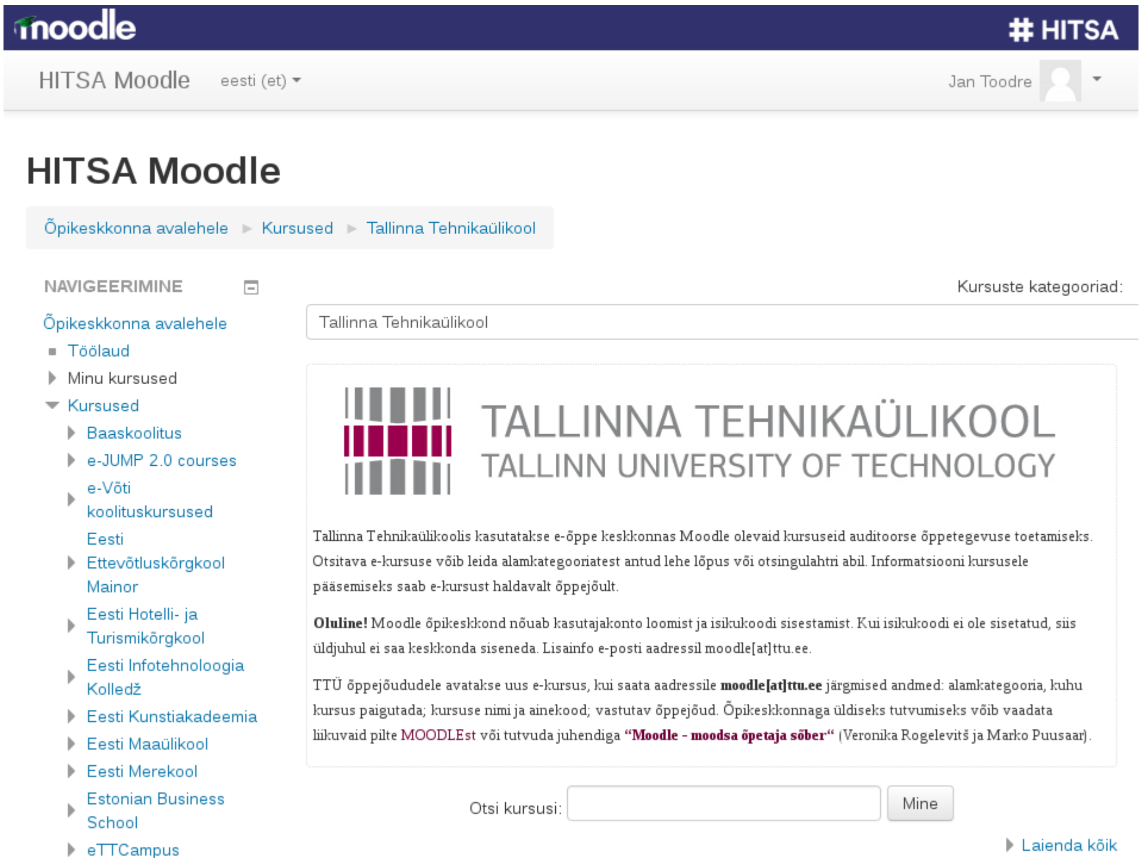


Figure 4. ISCX - My Field

One of the courses in the system which was passed by author is *Microprocessor Systems*. This course differs from other courses mainly because it expects solutions in a text field which will be manually checked by professors. This however, is a major drawback for the course consuming a lot of time from staff.

2.2.4. Moodle

Moodle is one of the most popular e-learning environment with more than 100,000,000 users [8]. The main motivation for the creation of this environment was to offer ways for distant learning [9], thus opening a global platform for different courses. Estonia's HITSA has set up Moodle environment for Estonian education system which is also used by TUT (See Figure 5).



Tallinna Tehnikaülikoolis kasutatakse e-õppe keskkonnas Moodle olevaid kursuseid auditoorse õppetegevuse toetamiseks. Otsitava e-kursuse võib leida alamkategooriatest antud lehe lõpus või otsingulahtri abil. Informatsiooni kursusele pääsemiseks saab e-kursust haldavalt õppejõult.

Oluline! Moodle õpikeskkond nõuab kasutajakonto loomist ja isikukoodi sisestamist. Kui isikukoodi ei ole sisetatud, siis üldjuhul ei saa keskkonda siseneda. Lisainfo e-posti aadressil moodle[at]ttu.ee.

TTÜ õppejõududele avatakse uus e-kursus, kui saata aadressile **moodle[at]ttu.ee** järgmised andmed: alamkategooria, kuhu kursus paigutada; kursuse nimi ja ainekood; vastutav õppejõud. Õpikeskkonnaga üldiseks tutvumiseks võib vaadata liikuvaid pilte MOODLEst või tutvuda juhendiga **"Moodle - moodsa õpetaja sõber"** (Veronika Rogelevitš ja Marko Puusaar).

Figure 5. Moodle - Courses, TUT

Moodle is aimed to provide a general purpose e-learning environment. This means that it will offer options which are suitable for most courses given in educational institutions. That explains why mostly in the basic software configuration professors can share materials, do simple quizzes with simple text input and multiple choice questions. This however is limiting to the purpose of environment proposed in thesis.

The strongest selling point of Moodle is that it is open source and widely adopted. This ensures a good support for generic problems and troubleshooting. However when customization is at hand, only plugins are available for Moodle. Mainly for programming Python and Java are well supported however lexing (scanning) and parsing plugin is available for C and C++ [10].

2.2.5. Comparison

The traits of each system are brought out in the following table:

Table 2. Comparison of Learning Environments

Environment	Virtual-C[11]	CS50[12]	ISCX	Moodle[13]
Source code	Closed Source	Closed Source	TTU internal	Open Source
Programming environment	IDE	Virtual Appliance with IDE	-	-
Test creation	Yes	Predefined	Yes	Yes
Student results summary	Maintainers server	Course site	Internal	Internal
Automated grading for code	Yes	Yes	No	No
Easy integration to existing system	No	No	No	No

Due to the fact that most of the existing solutions found are with closed source or not exactly fitting, it is reasonable to build a custom solution. The perk of building a completely new system is the possibility for full customization and flexibility. By Virtual-C description, it might seem just as a perfect fit for this thesis; however, it is just an IDE with closed source code. Even if it would cover some of the basic needs which this environment tries to solve, it is impossible to adjust the software according to our needs. Without having the ability to integrate our students' data with this system, it will not ease the grading and teaching process. Moodle, however, is open source and customizable, but the system itself is too complex to manually add modules and expect it to work flawlessly without putting tremendous amount of effort into the development process. Harvard's CS50 course has a working solution which is applicable in this environment, but the inner workings of the system are hidden from the general public. However, the concept of Virtual Appliance is used in this thesis and will be explained in further detail in section 2.5.1. The ISCX environment has a drawback with automated grading for the code, e.g. Microprocessor Systems course only accepts text and the code has to be manually evaluated by the professor. Since the system is developed in TUT, the collaboration is easier; however, the means of contribution are unexplored.

The final verdict is to build our own custom environment. The process and tools used for that will be covered in the next sections.

2.3. Virtualization

In simple terms, virtualization is hiding the physical characteristics of computing resources from the way in which other systems, applications or end users interact with these resources [14]. This allows end users to have available hardware to emulate special environments without the need to acquire the special hardware resources for it. Nowadays, computer hardware improves virtualization efficiency by having specific features such as VT-x and VT-d from intel and AMD-V from AMD.

As the technology advances, more solutions for virtualization emerge. There are multiple platforms for Virtualization software. The main types of virtualization are:

1. Hardware - also referred to as hypervisors;
2. OS-level;
3. Desktop;
4. Network;
5. Application.

In this thesis we focus on the hardware hypervisors and OS-level virtualization.

2.4. Application for this project

As stated earlier, one of the problems in programming classes is that students have different working environments. This leads to misunderstandings, unexpected results in grades and general discontent. It is essential to have a unified development environment for students to learn in and lecturers to teach in. In order to achieve the goals of this thesis, we need to provide a unified development environment with predefined tools and automated grading. There is most likely no better way to achieve it than using virtualization. The reason for using virtualized environment instead of having the actual environment setup as a requirement for students is because it is unreasonable to have people significantly modify their working environment for such a temporary time period. Virtualizing the environment allows the end users to set it up and later on clean up hassle-free.

If this infrastructure with provided virtual environments and automated grading is set up by the university, it guarantees the compatibility of developed software and less miscommunication between students and professors, thus eliminating the argument of *"It worked in my machine!"*.

2.5. Tools for this project

Prior to any actual environment setup, first the available options should be considered and the correct solution to fit the project's needs should be picked. It is useful to set the requirements in order to choose the right tools. For both of the environments it is required that software is free to use and can be freely installed by anyone. This rules out the operating systems of Macintosh and Windows - there is no way of legally acquiring copies of their software for free usage, thus leaving us with Linux. In addition to free use, the operating system should be widely used and supported. The operating system must have a support for essential development tools required for this course, thus ruling out some exotic distributions of Linux which are not designed for generic usage. As an extra, it is good to have a smaller footprint in terms of hardware resources. In short, the requirements of choosing the environments for this project are:

1. Cost - free;
2. License - free to use;

3. Support for basic development tools;
4. Support;
5. Community size - market share;
6. Hardware footprint.

For the testing and development environment it is very important to have the same software running on both ends - so that the code written can easily be compiled and run in the other environment. For programming in C, the essentials for developing software are the tools discussed in the next section.

2.5.1. Virtual Appliance

Virtual appliance is a preconfigured environment which can be set up easily in any major consumer level OS. In this project, the goal is to provide such environment in the form of a file which will be distributed to students and school computers, which can then be set up by using hypervisors. For hypervisors one of the best free and open source solutions is VirtualBox. The essential software bundle which has to be provided by the student is as follows:

- Linux Distribution;
- General purpose C Compiler;
- IDE for C coding;
- Fallback to IDE - text editor for coding in C;
- Utility to compile complex programs;
- Utility to check for memory leaks;
- Source code versioning software.

For some of those categories there are only a few options to choose from. This is due to the fact that either there is no market for this product or it is really expensive to produce and sell such software. Those categories are utilities for compiling complex software, utilities for memory leak detection and C compilers. In the following paragraphs, the author brings out the selected software for previously mentioned categories and the reasoning for why they were picked.

Linux Distribution The most important choice is the operating system on which all the other applications will run. Due to the restrictions given in Section 2.5, the operating system will be one of Linux distributions. The key aspects of deciding which Linux distribution to pick are hardware requirements, support, community and support for the essential tools mentioned in this section. When people are new to Linux and are not sure which distribution to choose, it is a wise idea to either look at the most popular distributions or search by criteria from distrowatch [15]. The author finds that this approach is acceptable for use in this environment due to the fact that students who are unfamiliar with Linux will possibly use a similar approach to find a suitable distribution.

Picking three of the most popular distributions of the past year result with Mint, Debian and Ubuntu. The author finds it reasonable to make the decision based on the most popular distributions because if a distribution is popular, it means that the community of it is growing and the support for it is also better both from the community and official distributors. It should be mentioned that SUSE Linux Enterprise Edition has a free edition called openSUSE, however, it is not added to the comparison because Debian based systems have larger communities.

Even though the End of Life for Mint and Ubuntu is later, the installation media is more than twice the size of Debian (table 3). This renders Debian the winner, as it does take the least disk space. In addition to that, all the other OSes are based on Debian, thus it is reasonable to assume that the software packages which work on Mint and Ubuntu most likely will also work in Debian.

Table 3. Comparison of Linux distributions

Distribution	Mint	Debian	Ubuntu
Version	18.1 Serena	8.0 Jessie	16.04
End of Life	2021-04	2020-05	2021-04
Based on	Debian, Ubuntu LTS	-	Debian
Default desktop managers	Cinnamon, GNOME, KDE, MATE, Xfce	Cinnamon, GNOME, KDE, MATE, Xfce, LXDE, etc.	Unity
32-bit ISO size	Xfce: 1.6 GB	Xfce: 647 MB	desktop: 1.5 GB

C Compilers In courses Programming I and Programming II it is important to cover the main aspects such as staff's familiarity with the software, support for different C standards, support for C++, possibility of doing cross compiling and supporting different targets. Selection process does not take into account the fact that Clang 4 is superior to GCC 7 in terms of performance [16]. In this case, the selected compiler is GCC mainly because of staff's familiarity with it, as in other compared aspects they are quite similar (Table 4).

Table 4. Comparison of GCC and Clang compilers

Compiler	GCC 7.1	Clang 4.0
required C standard support	C89, C90, C99, C11	C89, C90, C99, C11
Usage in IAG courses	Widely used	Mentioned
Target support	x86_64, arm, avr, msp430, etc.	x86_64, arm, avr, msp430, etc.
Supported languages	C, C++, Go	C, C++
Supported OS	Linux, Mac OS X, Windows	Linux, Mac OS X, Windows

Integrated Development Environment Integrated Development Environment is software which makes coding easier and more comfortable for the developer. This is achieved by having a variety of tools bundled together with language specific syntax highlights increasing readability. Usually C IDEs have buttons and keyboard shortcuts for compiling and running the software, which in default setup makes the compilation easier - students might find it dreadful to compile and run programs from command line at first.

For this task, although there are a lot of IDEs available in the market, the professors mostly use Geany. The main reason for this could be that it is a simplistic piece of software which has the basic features necessary for building C programs and has less problems than its alternative Code::Blocks.

Text editor with Command Line Interface As there are three most popular command line text editors out there, all three shall be added to comparison - vim, Emacs and nano [17]. Due to the fact that two of them are complex and highly configurable text editors, one might even replace their fully functional IDE for one of these editors (Table 5). Due to this reason, the author believes that both vim and nano should be available in the virtual appliance - for those who want something simplistic and those who want to fully configure their own IDE.

Table 5. Comparison of Text Editors

Text editor	vim	Emacs	Nano
Learning curve	steep	steep	flat
Configurable	yes	yes	no
GUI	yes	yes	no
Plugins / extensions	yes	yes	no
Authors familiarity	familiar	-	familiar

Utilities make and valgrind Make is an utility for helping with the compilation of complex C programs. Instead of having to type in long build commands, the stages of compilation can be grouped. This makes compiling such software easier and more maintainable. Having a makefile for the project, one only has to write 'make' for the project to build.

Valgrind is an utility for discovering memory leaks. This piece of software will mainly be used in conjunction with programs that utilize dynamic memory allocation. As an extra, it also supports memory error detection, thread error detection and multiple profilers [18].

Source code versioning software Version Control Software is very important to learn as every developer will eventually have to get involved with it. There is a reason why this sort of software is widely used - it is proven to be unreasonable to send different software revisions via e-mails and then trying to keep a track of different versions of the software.

When comparing two most widely used VCS Git and Subversion (also known as *SVN*) (Table 6)[19] , Git seems to emerge as the winner. Git has a distributed repository model, meaning that coding can be done without having an internet connection and modifications can be made without the fear of undoing somebody's work. As for choosing the VCS, since the author has prior experience with Git, it is the preferred choice. In addition to the forementioned properties, Git has a web interface GitLab which will be used in this project in conjunction with Testing environment (See 2.5.2).

Table 6. Comparison of VCS

CVS	git	subversion
Development status	active	active
Repository model	Distributed	Client-Server
Revision IDs	SHA1 Hashes	Numbers
Web interface	Bitbucket,GitLab, GitHub	Trac, ViewVC, WebSVN
GUI	GitKraken, gitk, etc.	Nautilus, TortoiseSVN, etc.
Authors familiarity	familiar	-

Chosen software In conclusion, the software chosen for the development environment is brought out in the previous chapters. The chosen software for Virtual appliance is as follows:

1. Debian 8.0 Jessie - Operating System;
2. GCC - C compiler;

3. Geany - IDE for C coding;
4. vim - Fallback IDE, text editor for coding;
5. make - Utility to compile complex programs;
6. Valgrind - Utility to check for memory leaks;
7. Git - Source code versioning software.

Exporting Virtual Appliance After picking out the software bundle, it is necessary to download the OS installation media and run the installer in Virtual Machine. This can be done by using VirtualBox. Once the operating system is installed, the manager of Virtual Machine should install the bundle of software and then export it as an appliance. This can also be done with VirtualBox - export it as an *ova* file. This ova file can then be distributed to students.

Once the students receive the file, they have to install virtualization software such as Virtualbox or VMware Workstation. Essentially, this file can be imported and set up with a few simple steps - by specifying the resources that shall be allocated to this appliance. After that, the system is ready for use.

2.5.2. Testing environment

Testing environment utilizes the OS level virtualization. For this occasion, there is a tool called Docker. Docker is a software containerization platform [20] which allows users to set up environments for their applications that are independent of the host operating system. This is the preferred method over having Virtual Machine for it because it has less overhead in terms of default system utilities, third party software and drivers. For the testing environment, it is necessary to install the same tools required for Virtual Appliance with a few exceptions, as testing the development is not important. That means no IDEs are required for testing.

Testing environment is for Automated grading of the system, which means it will be explained in greater detail in section 3.2.

3. Core System

One of the key parts of this system is the application. It will be explained in greater detail throughout this section - the architecture, its key components and technologies to be used.

Learning Environment consists of three main parts - one being the application and user interface for students and professors, second an automated grading system and third a virtual appliance. The last one is the smallest part of this system in terms of complexity and setup.

The core system is visualized in Figure 6. Node application (Figure 7) is a web application running on Linux server, which is the center of Learning Environment. Application is responsible for spawning docker containers, receiving requests from automatic grader, GitLab and end users.

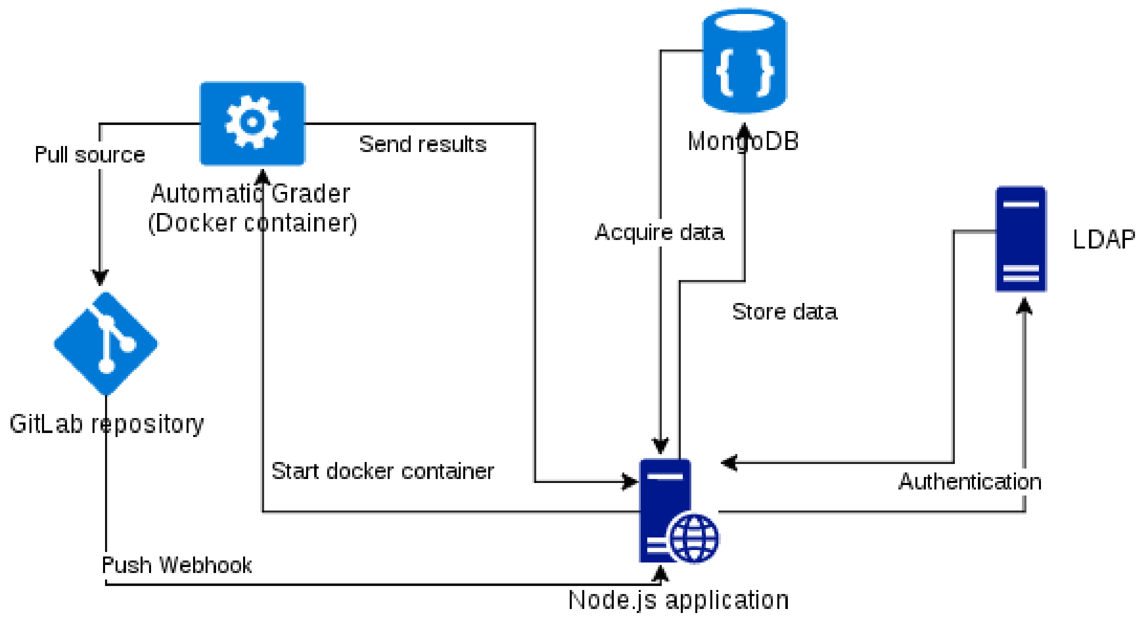


Figure 6. System Architecture

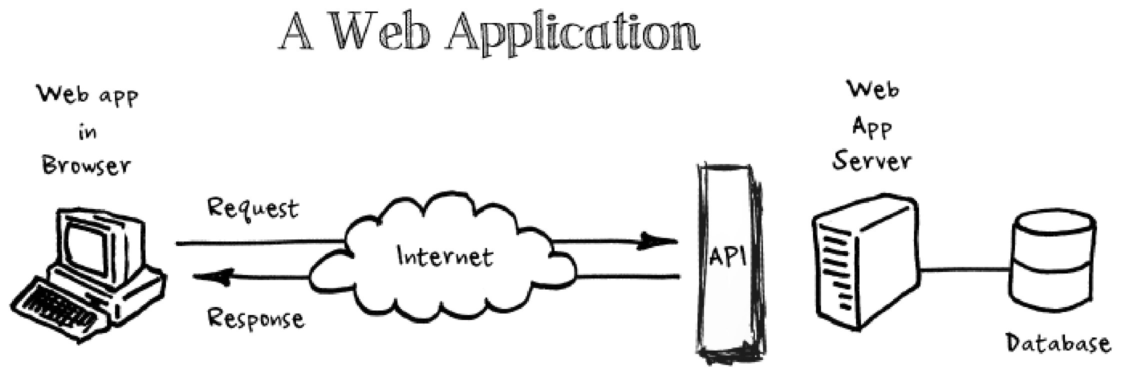


Figure 7. A Web Application [3]

The Students workflow is as follows (See Figure 8) - this assumes that the initial setup is done - UNI-ID, git config, repository setup in gitlab:

1. Access web page;
2. Look for a task that is not yet completed;
3. Create directories according to the structure;
4. Write the code according to the task description;
5. Test solution and make git add, commit, push;
6. GitLab receives pushed code, sends a POST request to node app (webhook);
7. Application processes request;
8. Application starts Automatic grader and provides student data;
9. Automatic Grader (AG from now on) will pull source from GitLab;
10. AG requests for Grading instructions from app;
11. AG looks for completed tasks (specific directories);
12. AG executes tests for tasks that have been completed;
13. AG sends all the results to application;
14. App updates results and leaderboard;

15. App sends direct url to detailed report to email / slack;

16. Student can now view updated results table.

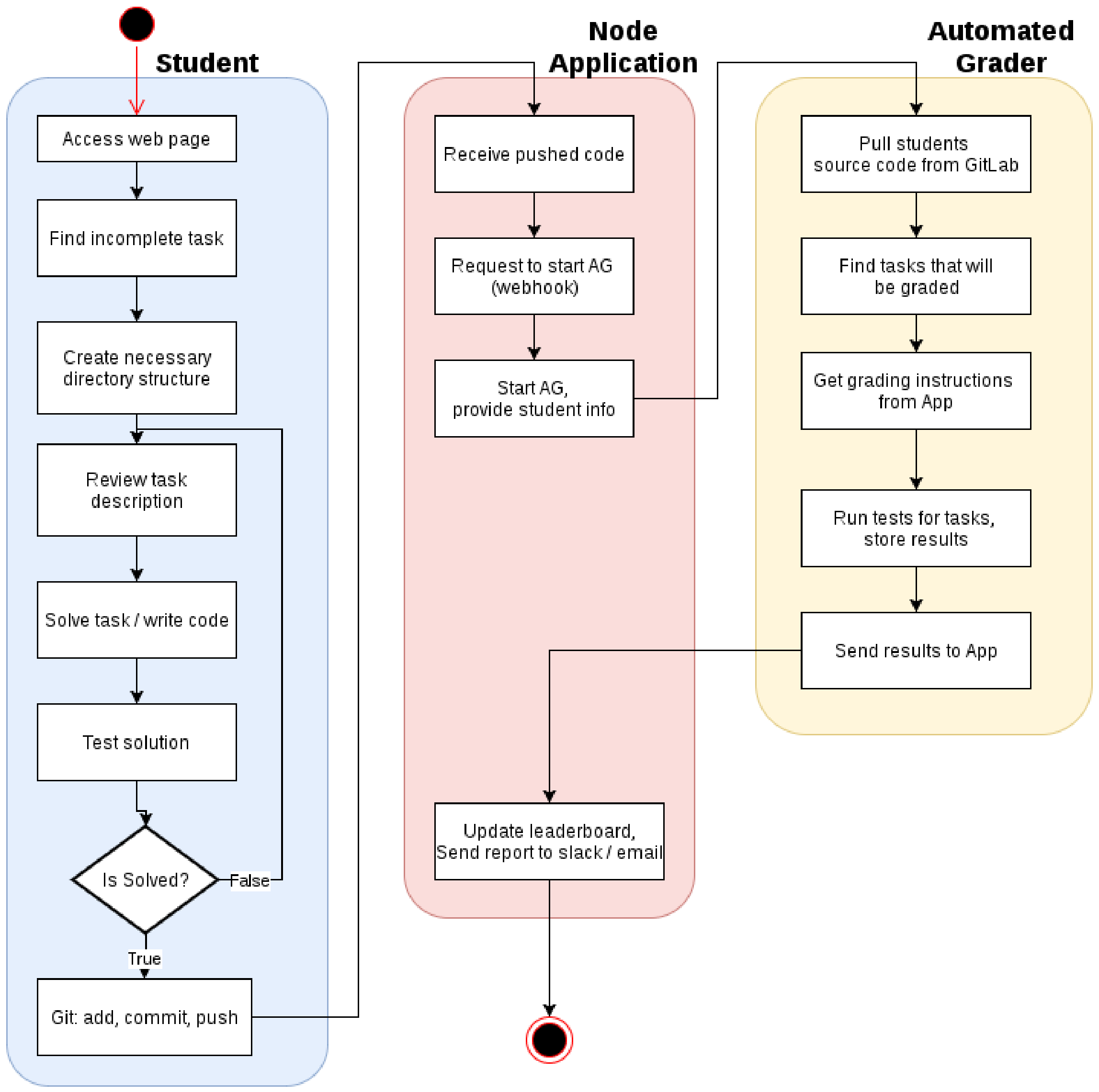

Student
Access web page
Find incomplete task
Create necessary directory structure
Review task description
Solve task / write code
Test solution
Is Solved?
False
True
Git: add, commit, push
Node Application
Receive pushed code
Request to start AG (webhook)
Start AG, provide student info
Update leaderboard, Send report to slack / email
Automated Grader
Pull students source code from GitLab
Find tasks that will be graded
Get grading instructions from App
Run tests for tasks, store results
Send results to App

Figure 8. Students workflow

From the professors view, it is possible to do multiple things, such as:

- Manually modify and enter results;
- Update testing environment;
- Add new tasks (see 3.2.1, 3.2.2 and 3.2.3);

- Verify Plagiarized submissions;
- See an overview of the whole system.

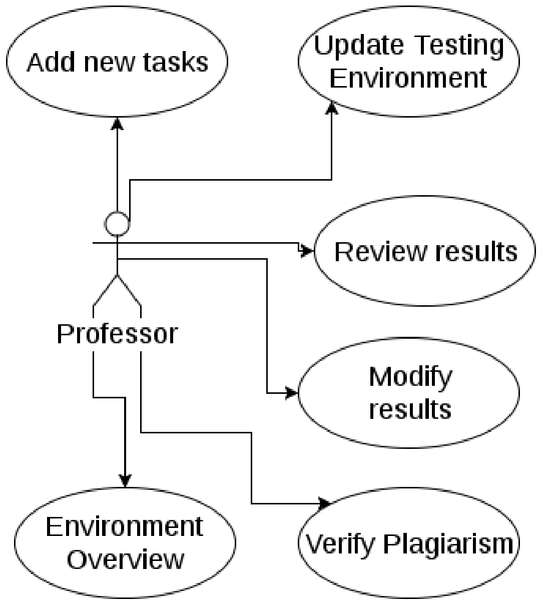


Figure 9. Professors actions

Since students are sometimes given tasks which are more creative or cannot entirely be graded by AG, the system must have an option for the professor to manually enter the results. For the students it means that on the results page they are informed that AG gave x points and the rest must be given by the professor - this will prevent misunderstandings when the results change.

3.1. Application

This application will be created with Node.js - JavaScript runtime built on Chrome's V8 JavaScript engine. Node on its own has the essential web server features, but there is also the world's biggest package management system called npm. This package management system has a vast variety of packages available to be installed from CLI. One of the major pros of npm is that it takes care of all the dependencies for the developer. All one has to do is specify the necessary packages, their versions and how those packages should be updated. At the same time, this dependency handling is its pitfall - it creates a lot of overhead where some packages might redundantly get added to the project. Versioning in packages follows the guidelines of semantic versioning [21] , which in its essence means that the versions are with the pattern of MAJOR.MINOR.PATCH, where:

- MAJOR versions are with incompatible API changes;
- MINOR versions are with backwards compatible functionality updates;
- PATCH versions with backwards compatible bug fixes.

For this application, most of the code is written in JavaScript. Should there be a necessity for functionality which is not given by default, the developers can modify V8 engine and add extra functionality by writing it in C++ and exporting it as a JavaScript function, which can then be called.

3.1.1. Experience

In addition to the time usage benefits that come from using this application, professors will be able to use this system to modify grades more easily and get instant results without swapping environments. Current practice for TA's is to create intermediate results table for lab, attendance and homework, then merge the final results table with their intermediary one. This is time-consuming and inefficient.

This system, however, is not only aimed to improve the professors' lives, but it should also give a good experience for students. By centralizing information and resources like results, notifications, tasks, general information about course, it is possible to make everybody's life easier. As an extra feature, the system intends to gamify the point system - a leader board and a hall of fame shall be created for this purpose. This should give the students some incentive to study more and better in order to get to the top of the leader board. This will not only make them competitive towards each other, but it will also make students strive for better results. It should be noted that on the hall of fame only student code and number of gathered points is shown, in order not to break the Personal Data Protection Act, which, if broken, can be fined with up to 32,000 euros [22].

3.2. Automated Grading

Automated grading is the second key part of this thesis - it is the system which will improve the professors' life quality by giving them more time. The main concept of this system is to keep it updated continuously and have its task pool grow, allowing students to choose between tasks which they wish to complete. However, for such a system to be maintainable and working, there have to be some requirements and constraints set.

As a by-product of this system, multiple tools will be developed to ease the work of professors who wish to create more tasks in this environment. Utilities developed for this subsystem contain searching for the tasks that have been finished, saving intermediary results and later posting them, finding correct build commands, self testing and many more.

3.2.1. Requirements and Constraints

For this system to be maintainable, a strict structure and logic must be applied. Since this subsystem can be isolated, it shall have its own repository and maintainers separate from the main application. This allows the continuous integration to work both for the application and automated grader.

The main constraint to this system is having the testing software run on Linux - same OS as in Virtual Appliance. This sets some limitations, yet gives much freedom, which will be discussed in the next section.

Second constraint is the directory structure. To ease the naming convention and navigating trough the directory tree, the author suggests the depth 3, levels starting from the root of the project directory- for example, */category/topic/task*. To give an illustrative example of this directory structure for a task which requires a student to make a *Hello world!* program:

- category - io (input/output);
- topic - printf;
- task - hello.

This constraint applies to both the template for the automated grading system and the students' repository. Otherwise, finding the correct task to grade would be significantly harder and would require some configuration files which might be more challenging to use, depending on the students' setup. However, it should be fairly easy to create a directory in Git repository with the following command: *mkdir -p io/printf/hello*.

Regarding naming convention and directory structure, there are two more constraints. One of the constraints requires students to name their source file (which contains the main function) with the same name as the parent directory it is in. This constraint eases the usage of default build utility. Using the example given above, the source file name would be *hello.c*.

The second constraint requires maintainers and professors who wish to add a new task to the system create a directory called *test*. This directory contains self tests for the task. Self test for the task is a model program written by the professor which will result in maximum score if the task instructions are followed correctly. Similarly to the previous requirement, test directory must have a file with a name of *test.c*.

Another requirement for automated grading is the grading software - this also has to be written either by the maintainer or the professor. This piece of software has to verify that the model solution provided in the test directory (e.g. *io/printf/hello/test*) does actually pass all tests. This is a security mechanism to prevent complaints from students when they have correct and working solutions and get less points than advertised.

The last requirement, which has to be met before any task can be considered active in the whole environment, is that all the self tests have to pass per task. When a new task is added, the automated grader docker container shall be rebuilt to support checking new tasks.

In short, the following requirements must met by (Student - S and Maintainer - M):

- directory structure with following notation: *category/topic/task* - SM;
- testing software inside previously mentioned directory - M;
- task with self test directory *test* - M;
- solution with the parent directory name - SM;

- testing software which runs on Linux - M.

3.2.2. Possibilities

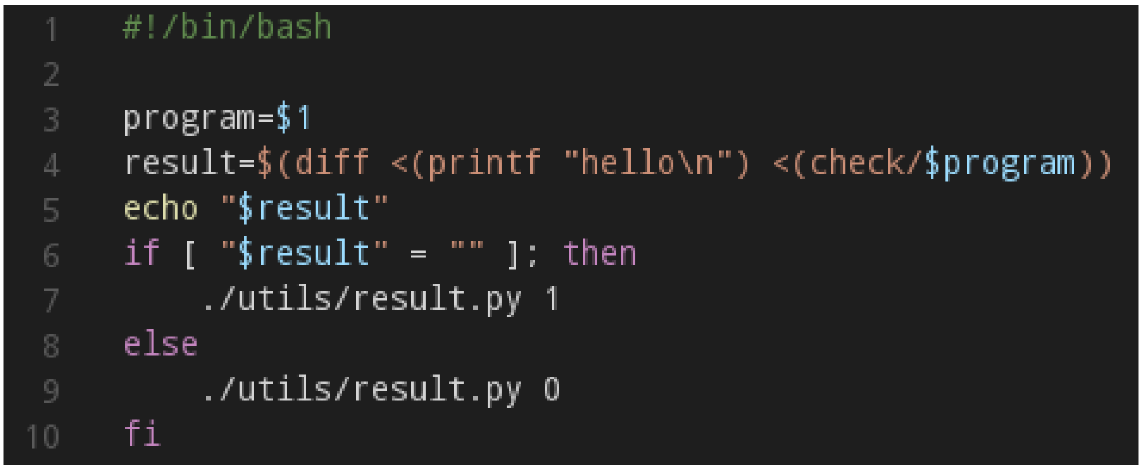
When taking into account all the previously mentioned requirements and constraints, one will wonder what can be done in such a system with so many constraints. As mentioned before, having grading software run on Linux is both a constraint and a possibility.

This option gives the maintainer the opportunity to create testing software by using any combination of command line tools which have already been created for Linux, such as grep, sed, cut, awk, etc. Furthermore, not only does it give the freedom to use other binaries, it also allows the testing software to be written in any language which is supported in Linux. A few scripting and programming languages in which the tests could be written in are ruby, python, php, bash, C++, etc.

Having these opportunities in this environment means that it could be used not only for teaching and grading C, but also other languages that can be used in Linux. This means that as a result of this thesis not only professors and TAs in courses IAG0581 and IAG0582, but also other courses which might teach Java, C++, Python or some other language, can reap the benefits of this Learning Environment.

The author created two tasks which were tested with bash and php scripts in conjunction with grading script and default build script, where the default building script will either look for a task specific build command, makefile or just the .c file with the parent directory name. However, the grading script will make use of the system environment variables; thus, when this script gets called from the grading software, it will only take the result in points as a parameter.

As a proof of concept, a task was created, which should be the first *Hello world* program. The requirements are for the program to compile and print *hello* with a trailing newline (Figure 12). It should be noted that the default build utility will always give a point for the task if there are no errors. *Hello world* program is checked with a script *check.sh* (Figure 10).



```
#!/bin/bash

program=$1
result=$(diff <(printf "hello\n") <(check/$program))
echo "$result"
if [ "$result" = "" ]; then
    ./utils/result.py 1
else
    ./utils/result.py 0
fi
```

Figure 10. POC - Grading software written in bash

3.2.3. Adding new tasks

The process of adding new tasks to the environment has a simple concept which can be summarised with a few following steps to be followed to create a new task. It is required to pull the auto grader repository before one can proceed with the creation of a new task.

1. create a task with clear instructions using web interface - *this adds entry to database*;
2. create a model solution based on task instructions;
3. create an automatic grader based on task instructions;
4. verify that model solution gets maximum points;
5. push the code to repository.

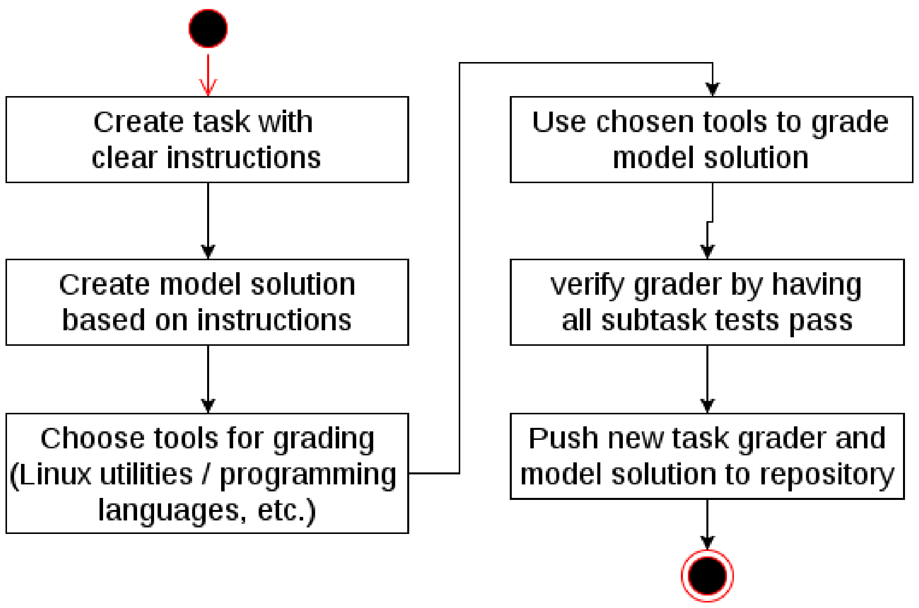
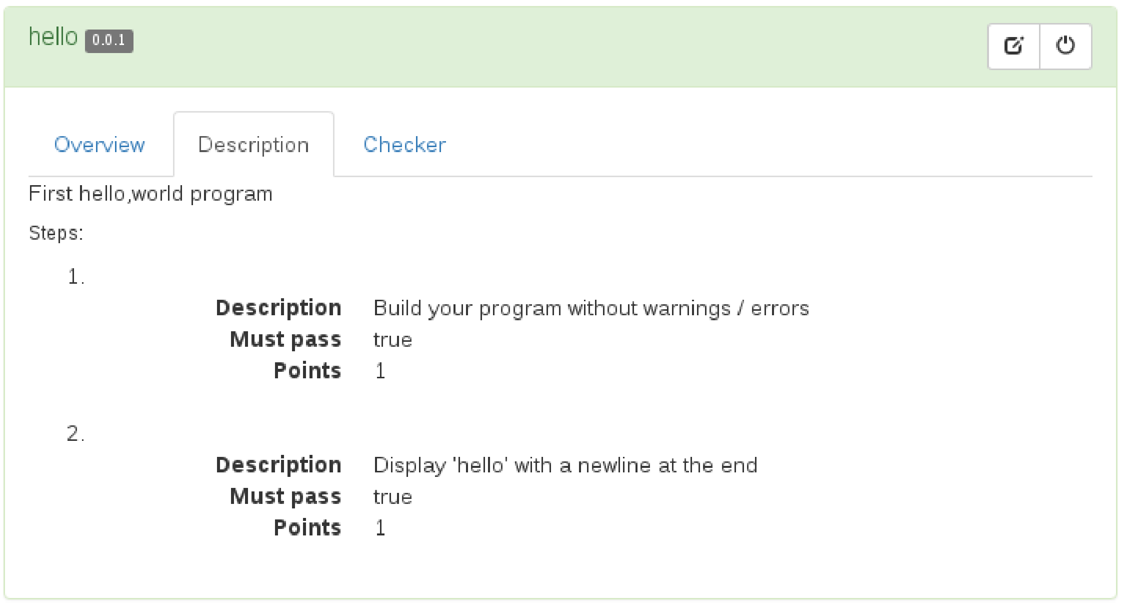

Create task with clear instructions
Create model solution based on instructions
Choose tools for grading (Linux utilities / programming languages, etc.)
Use chosen tools to grade model solution
verify grader by having all subtask tests pass
Push new task grader and model solution to repository

Figure 11. Adding new tasks

After these steps are done, the CI (See 4.3.3) and application will take care of the rest. The model solution will then be verified by using the developed grading script. Should the results be as expected, the grading docker container will be rebuilt and put into action. This allows the maintainer to add new features without having to learn and manage docker. Should everything succeed, the professor will see the added task in the admin panel (Figure 12).



First hello,world program

Steps:

1.

Description	Build your program without warnings / errors
Must pass	true
Points	1

2.

Description	Display 'hello' with a newline at the end
Must pass	true
Points	1

Figure 12. POC - task preview

3.3. Server Side Solution

This is the part of the application which does the most work in the environment. Everything will be going trough this Node application (Figure 6). Proof of Concept Node application listing can be seen in Appendix A. All the requests from Automated Grader, client, GitLab will be processed here. It is very important to make the Server Side Solution very performant in order to serve all the students, which in year 2017 was 286 students from Programming I and 198 students from Programming II course. It would be reasonable to assume that the active users would be somewhere between the people who declare Programming II and the number of students that successfully finish Programming I. Should this project be a success and approved by other professors as well, it would serve even more students.

Server side solution is the part of the application that will not be seen by the end user. In this section, the author of this thesis will provide an overview of the technologies used in developing POC application for this course.

3.3.1. Technologies

As the reader already knows, the main application is a Node.js application running on V8 JavaScript engine. However, having the vanilla set up of Node.js is not sufficient to simply create a fully fledged system. In order to avoid reinventing the wheel and possibly creating less secure and error prone sub modules, it is wise to use the components made available in the npm.

When running a Node application, the current state of the code will be run. For as long as the process runs, the source code updates will not affect the way that the application behaves. In order to always have the latest version of the code run on the server, nodemon is installed. Nodemon runs the application and monitors for any changes in source files. However, template files and HTML, CSS files will not trigger the application restart.

Web server For creating the web server a fast, unopinionated, minimalist web framework is used - express.js [23]. This framework allows the web server to easily be set up without binding one to some specific tools.

Database As with many systems, this system also requires a way to store and request data. Since the application is written in JavaScript, it is reasonable to use MongoDB for this purpose. MongoDB is different from usual relational databases, as it is a document store. Documents in this case are JSON objects, which could be thought of as rows in relational databases. Similarly to tables, they are called collections in the document store. Further comparison of document store and relational database is out of the scope of this thesis. MongoDB can easily be interfaced and accessed by using JavaScript. If relational databases require one to learn SQL syntax, in this case the knowledge of pure JavaScript is required for making queries. To make development and standard database query usage easier, a wrapper library for MongoDB is used - mongoose. Mongoose makes creating new collections easier - it is used to describe the schema of the collection and later on storing and requesting data.

Code Application code is written in ECMAScript 6 JavaScript standard. Using the latest ES standard allows the usage of new features and some functionality which is essentially syntactic sugar. The drawback to using the latest standard is that not all web browsers support it. However, it is still possible to utilize the latest features offered by the standard and have it work for everybody. This can be achieved by using a tool called babel. Babel is a JavaScript compiler which will compile the new ES standards into a widely supported ES5 standard. For the usage of a specific preset, it is reasonable to create a *.babelrc* file, which describes which ES standard and preset are best to use for the project.

Templating Templating is essentially creating a blueprint and sub-modules for the pages. When used, it will at first build the page based on the template and insert the requested data during the build, depending on the data which is desired to be shown. For templating, engine express-vue was used - Server Side Rendering for Vue. Vue is a small, fast, reactive framework which can be used for templating. In the POC solution, the rendering was on the server side - before the page is sent to the client, the entire page is built on the server. This method is useful when doing SEO, but it reduces the amount of clients it can serve at the same time, meaning there is an increase in time between user requesting the web page and receiving it. For this environment, the same solution is not viable, hence only the template should be sent to the client, resulting in the page rendering on the end user's machine.

3.4. Client Side Solution

In order to create the user interface, HTML is used in conjunction with the most popular framework for responsive web - Bootstrap. This framework allows the developers to concentrate on building the site, instead of creating CSS code for the site to look decent. When using this framework, it is necessary to only write HTML and add required classes to DOM elements in order for it to look presentable. To add extra functionality, the use of JavaScript libraries and frameworks is required.

Because this environment has data which changes over time, it is required to communicate with the server. Therefore, queries should be made to request the most up to date data. For this purpose, there is a widely used library which makes requests to RESTful API easier - jQuery. This framework does not only allow us to make requests more easily, but it is also required by Bootstrap framework in order to manipulate DOM.

As explained in the previous subsection, the templating system should be on client side. This is good for numerous reasons. For one, it will save server resources as there is no need to render the page for every client. A second pro for this solution is the possibility to cache commonly used files in the client's computer, meaning that there is no need to send them over the network if the file version is the same as the file available locally.

To further exhaust the possibilities of caching and optimizing the resources used by the server that hosts our application, it is useful to investigate the opportunities which are offered by CDN. CDN is Content Distribution Network, which, as the acronym indicates, distributes content. Most importantly, this network is separated from the application and will therefore reduce the work load of the environment hosting server. CDNs can be used for most of the frameworks and libraries that are included with the HTML script and style tags. In this particular case, jQuery, Bootstrap and Vue would be distributed to the end users by CDNs.

4. System Development

The main overview of the Learning Environment was given in the previous sections. However, just giving an overview of how it should be done does not necessarily make the development process easy and quick. In order for this project to excel, it is necessary to set some rules to what and how to develop.

4.1. Project

This project will be an open source project. This allows those people who are not necessarily connected to the university to contribute as well. One of the reasons for this is having external pairs of eyes to verify the code, thus making the system less buggy in the long term. By gathering people with different sets of expertise, the system will become more secure, the code will be of better quality and the system can be supported by various people.

The whole environment will consist of multiple repositories to ensure CI/CD (See 4.3.3). The separation of repositories is due to the isolation - different subsystems are deployed differently. At the time of writing this thesis, the necessity for one registry and three repositories is evident:

- Docker containers registry;
- Application repository;
- Dockerfiles for AG repository;
- Task repository.

Docker registry Docker registry is essentially the same as Git repository with one difference - it holds different container versions and layers. This registry is necessary, because using cloud solutions for a project without a budget is unreasonable and expensive for private repositories. Most cloud registries that offer free private repositories only offer one.

Repositories All of the repositories will be in GitLab Community Edition, which is hosted on ATI server. The task repository will contain the task model solutions and grading software. For the grading software, some Linux tools which are not installed by default should be added to the Docker image. Only the source code and scripts should be stored in this repository.

4.2. Tools

For the project work to flow, correct tools should be chosen. As an ancient proverb goes: if you are handed a hammer, everything will start looking like a nail - this is one of the reasons why the whole project is open source and some tools to ease the development are taken into use.

4.2.1. Version Control System

As already mentioned in the previous sections, the VCS used for this project and environment is Git with a web interface called GitLab.

GitLab GitLab is a web interface for managing Git repositories. Because in developers' everyday life there are multiple tools that are used, GitLab has also integrated them into their platform. It does not only add extra features to barebone Git repository, but also makes its usage easier- for example, managing groups, members and permissions.

In addition to core VCS features, GitLab offers CI/CD options, issue tracking system and scrum board. These extra features allow users to use one unified system for development without the need to create extra accounts and switch between different environments, thus cutting down productive time. Whenever a new issue is created, the assignee will get a notification in the opening page of GitLab. To further increase productivity, it is possible to integrate Slack with webhooks, so that for each desired action a notification will be sent in Slack to the users.

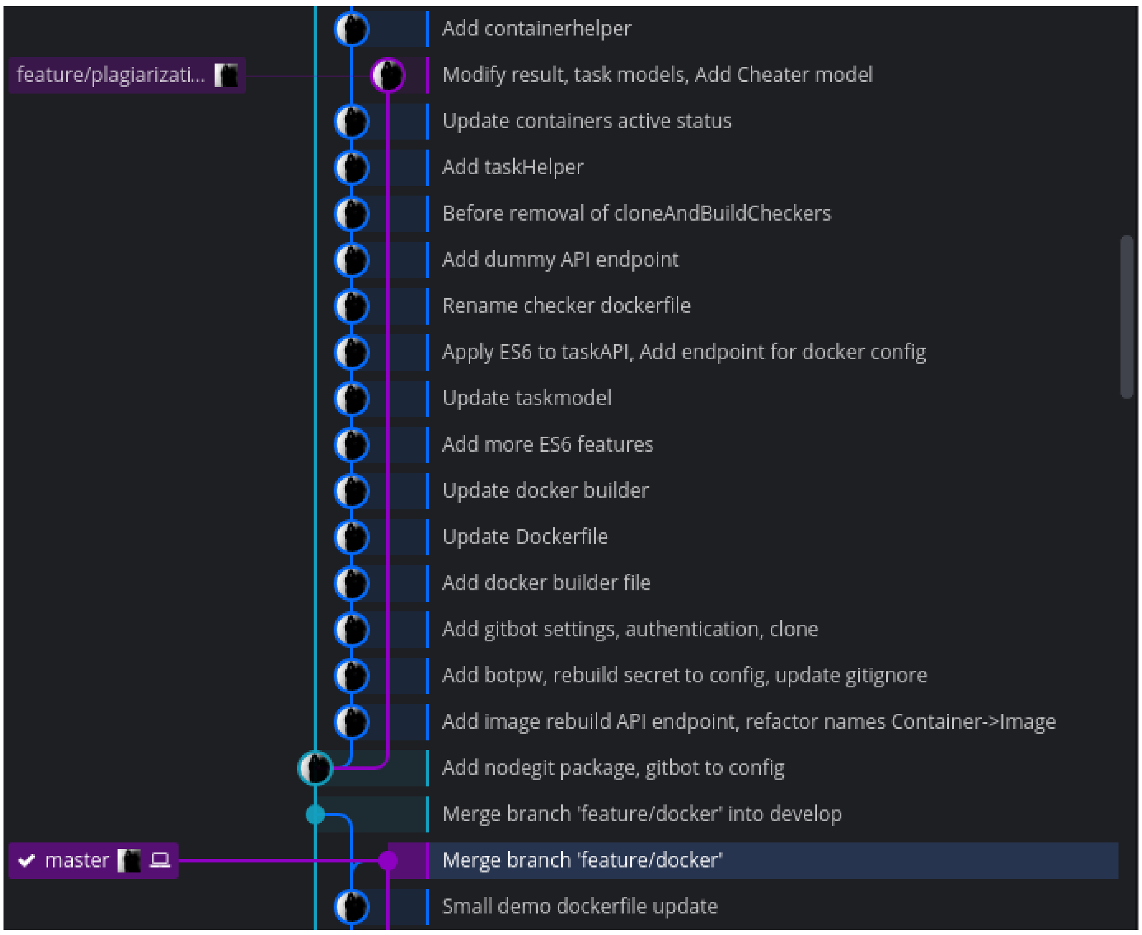
Slack Slack itself is not a VCS. However, it can be integrated with GitLab. Slack is an instant messaging multiplatform application. It is widely adopted in silicon valley by many startups. Because of its many integrations, it allows to improve everyday workflow. For example, if a team uses Dropbox for documenting the system, it can easily be integrated in slack without having to open the dropbox application to share files.

Slack allows to create private and public channels with a specific purpose. In this project, it is reasonable to create channels based on the subsystem. Slack would also be used as the main means of communication instead of e-mails.

4.2.2. Development tools

For a smooth development process, picking compatible tools is a must. For this purpose, it is suggested that everyone who wants to contribute to the project will have the same software bundle installed.

Usually, VCS Git is used. However, by default Git only offers a command line interface and a subpar graphical application. For some people this is very dreadful, and they would rather not get familiar with it. For this purpose, GitKraken from axosoft is used. It offers a visually pleasing Git experience with basic features - log, graph, committing, branching, merging, rebasing, etc. Visualisation of commit history can be seen from Figure 13.



Add containerhelper
Modify result, task models, Add Cheater model
Update containers active status
Add taskHelper
Before removal of cloneAndBuildCheckers
Add dummy API endpoint
Rename checker dockerfile
Apply ES6 to taskAPI, Add endpoint for docker config
Update taskmodel
Add more ES6 features
Update docker builder
Update Dockerfile
Add docker builder file
Add gitbot settings, authentication, clone
Add botpw, rebuild secret to config, update gitignore
Add image rebuild API endpoint, refactor names Container->Image
Add nodegit package, gitbot to config
Merge branch 'feature/docker' into develop
Merge branch 'feature/docker'
Small demo dockerfile update

Figure 13. Git commit history in GitKraken

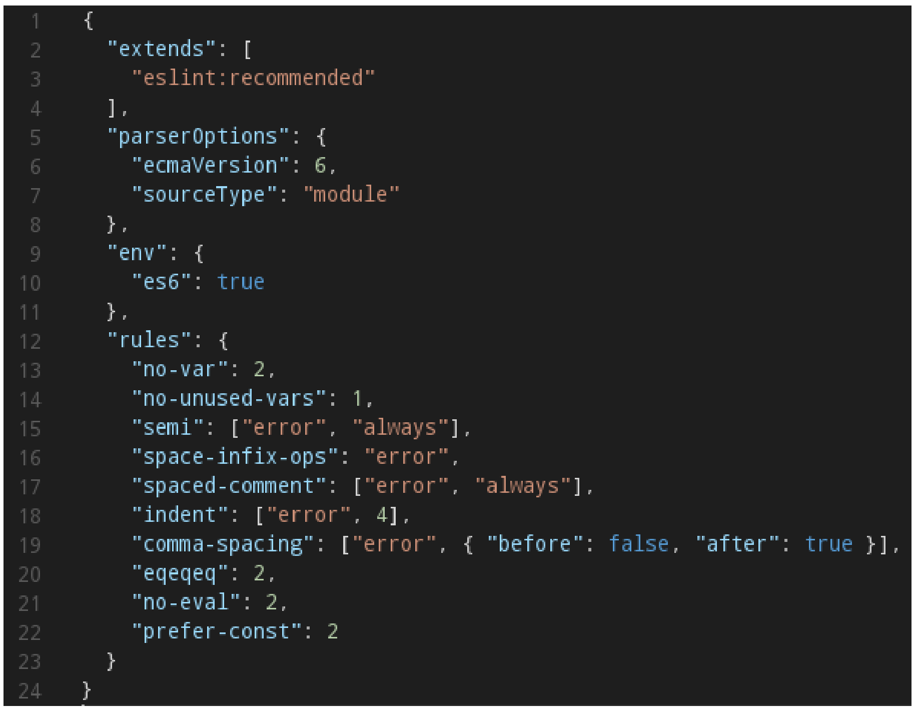
Because this project uses newer technologies, knowledge of Docker is preferred. It is reasonable to expect a developer to know how to create new images, run containers and modify *Dockerfiles* and *docker-compose.yml* files.

Managing application processes can be cumbersome. To ease the process, it is suggested to use pm2 - *Process Manager 2*, which is a watcher for node applications. It is possible to launch a single process, but also a cluster of processes, which allows almost no downtime when upgrading. To make the developers' life easier, it also offers a clean CLI.

4.2.3. Style Guide

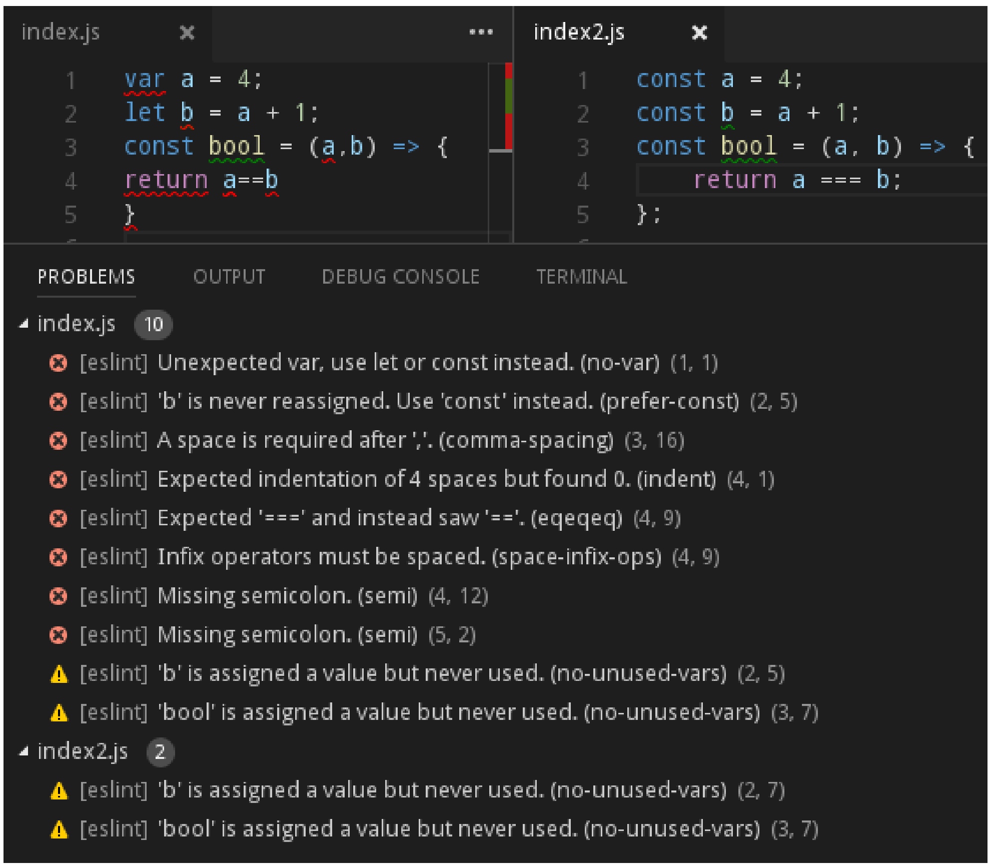
For any project to be maintainable, the structure and workflows should be described. For the codebase to be uniform, a style guide will be enforced. Instead of having someone read the code line by line to check if it conforms to standards, a tool is used - ESLint. This allows the codebase to be readable by anyone and offers style tips to everyone, regardless of their coding habits.

ESLint is a piece of linter software, which will analyse the code style and throw warnings or errors in IDE according to the style definition. Style is defined in a file *.eslintrc* which has rules[24] on how to write code. For any new users of this tool, a widely used style guide is made by AirBNB [25]. A few example rules can be seen in the Figure 14. An example of incorrectly styled code next to correctly styled code with an ESLint output can be seen in Figure 15. This tool can be used with various IDEs, for example Visual Studio Code and Atom.



```
{
  "extends": [
    "eslint:recommended"
  ],
  "parserOptions": {
    "ecmaVersion": 6,
    "sourceType": "module"
  },
  "env": {
    "es6": true
  },
  "rules": {
    "no-var": 2,
    "no-unused-vars": 1,
    "semi": ["error", "always"],
    "space-infix-ops": "error",
    "spaced-comment": ["error", "always"],
    "indent": ["error", 4],
    "comma-spacing": ["error", { "before": false, "after": true }],
    "eqeqeq": 2,
    "no-eval": 2,
    "prefer-const": 2
  }
}
```

Figure 14. ESLint - Example .eslintrc configuration file



```
var a = 4;
let b = a + 1;
const bool = (a,b) => {
return a==b
}
```

```
const a = 4;
const b = a + 1;
const bool = (a, b) => {
    return a === b;
};
```

```
index.js  10
  [eslint] Unexpected var, use let or const instead. (no-var)  (1, 1)
  [eslint] 'b' is never reassigned. Use 'const' instead. (prefer-const)  (2, 5)
  [eslint] A space is required after ','. (comma-spacing)  (3, 16)
  [eslint] Expected indentation of 4 spaces but found 0. (indent)  (4, 1)
  [eslint] Expected '===' and instead saw '=='. (eqeqeq)  (4, 9)
  [eslint] Infix operators must be spaced. (space-infix-ops)  (4, 9)
  [eslint] Missing semicolon. (semi)  (4, 12)
  [eslint] Missing semicolon. (semi)  (5, 2)
  [eslint] 'b' is assigned a value but never used. (no-unused-vars)  (2, 5)
  [eslint] 'bool' is assigned a value but never used. (no-unused-vars)  (3, 7)
index2.js  2
  [eslint] 'b' is assigned a value but never used. (no-unused-vars)  (2, 7)
  [eslint] 'bool' is assigned a value but never used. (no-unused-vars)  (3, 7)
```

Figure 15. ESLint - Visual Studio Code Example output

4.3. Workflow

This section defines some rules for the development and work process.

VCS Git Flow will be used. This approach allows easy tracking of git commit history. As a general rule of thumb, there are two main branches - an immutable branch *master* and *develop*. Most of the development process will be happening on the *develop* branch. Whenever a new version of software is released, one of the commits from *develop* will be merged into *master* branch.

Each new functionality, hotfix that is added to the environment shall get a new feature branch. After it is fully developed, a merge request is issued, upon which one of the project maintainers or lead developers will do code review for the committed portion of the code. Once the code is approved, it will be merged back to develop branch.

4.3.1. Design pattern

Using different design patterns in software projects is very common. In this project, the development pattern will be to develop microservices and API first. This allows the developers to be sure that the application logic works and only then can they proceed with the user interface. Furthermore, if this is the approach used then the front-end solution can be chosen later down the development process - for example, which front-end framework to use - react or vue.

Microservices Microservice architecture is a design pattern used in programming to simplify understanding and developing of a system. This allows communication with and between services by API. Using this method has the benefits of having smaller components, ease of use for CD and also eliminating the long-term commitment to a certain technology stack [26]. The mentioned approach allows the application to scale wide, meaning more instances can be set up as a cluster, which in turn results in servicing more clients.

API API is *Application Programming Interface* and in the context of this thesis referred to as RESTful API. RESTful API allows the microservices to communicate with one another mainly to execute some action, store or request some data.

4.3.2. Contributing

As for every open source project, there are means of contributing. For this project, at the end of each semester the contribution possibilities to this environment are announced. Those who wish to contribute should make a request. After making a request to contribute, one will be given access to the according repository (see 4.1). Then the person shall be assigned a task and a new branch should be created (See 4.3). After that, the process goes as described in section 4.3. One way to motivate people to contribute is to offer incentives. Whenever a task is assigned and later on completed, the tags which are in the task will be added to the person's profile, in a sense making it a competence cloud based on keywords similar to Stack Exchange.

As the environment is enormous, there are multiple areas to contribute to. To name a few areas which require more attention:

- Virtual Appliance;
- Main application;
- Tasks and checking software;
- Task creation utilities;
- Documentation;
- Dockerization;
- Bug fixes;
- Integrations with Slack;
- Continuous Integration.

4.3.3. Continuous Integration and Continuous Delivery

Continuous Integration is the means of merging developer's working copies into mainline several times a day. Continuous Delivery is the process of having software released in short cycles. These two approaches used in software development allow the users to have the most up to date services available to them. In conjunction with the work flow whenever feature and hotfix branches are merged back to develop a CI/CD, a pipeline will be started with automatic tests. Should they pass, the system which was updated shall be deployed and made available to the end users. The same process applies for merging *develop* branch to *master* with an addition of creating a new software version - *tag* in terms of Git (See [21] for versioning conventions).

CI/CD tools are integrated in GitLab CE, therefore making the usage of such an approach easier. To set this pipeline up, there are few things that have to be done:

- Setting up runners;
- Assigning runners to projects;
- Writing configuration file;

- Configuring CI/CD pipeline.

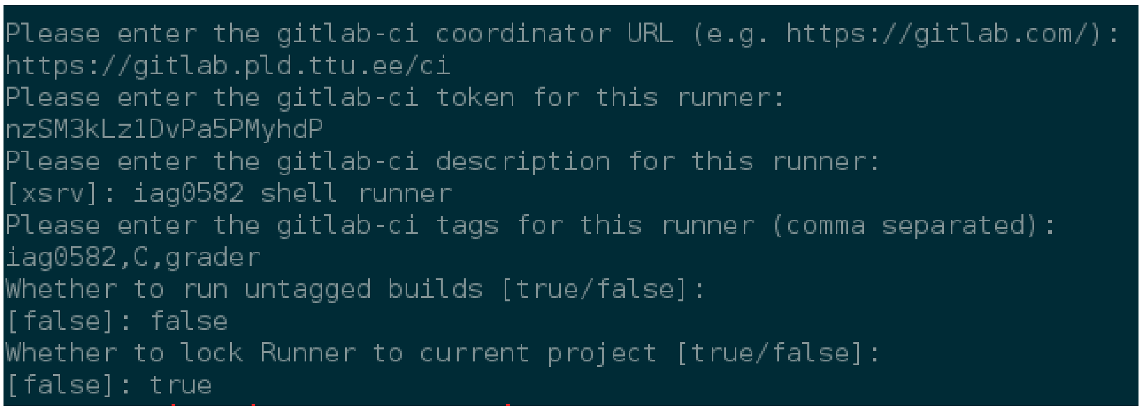
Runner Runners are pieces of software in GitLab which will execute the instructions described in *gitlab-ci.yml* file (See 4.3.3). These runners can be hosted in the local machine as well as remote machines. They are mostly used for environment setup, automated testing, deployment and clean up.

GitLab supports multiple types of runners [27]:

- Shell - Builds will be run in hosted OS;
- Docker - Builds will be run in Docker container;
- Docker-SSH - Builds will be run in a Docker container by opening an SSH connection;
- VirtualBox - Builds will be executed in Virtual Machine;
- Parallels - Builds will be executed in Virtual Machine;
- SSH - Builds will be run in remote server by opening an SSH connection;
- Kubernetes - Executor uses Kubernetes Cluster for software builds.

From the above list, most commonly used executors are Docker and Shell, both of which are used in this project. Shell executor is mainly used for deployment and Docker mainly for new tests and running them.

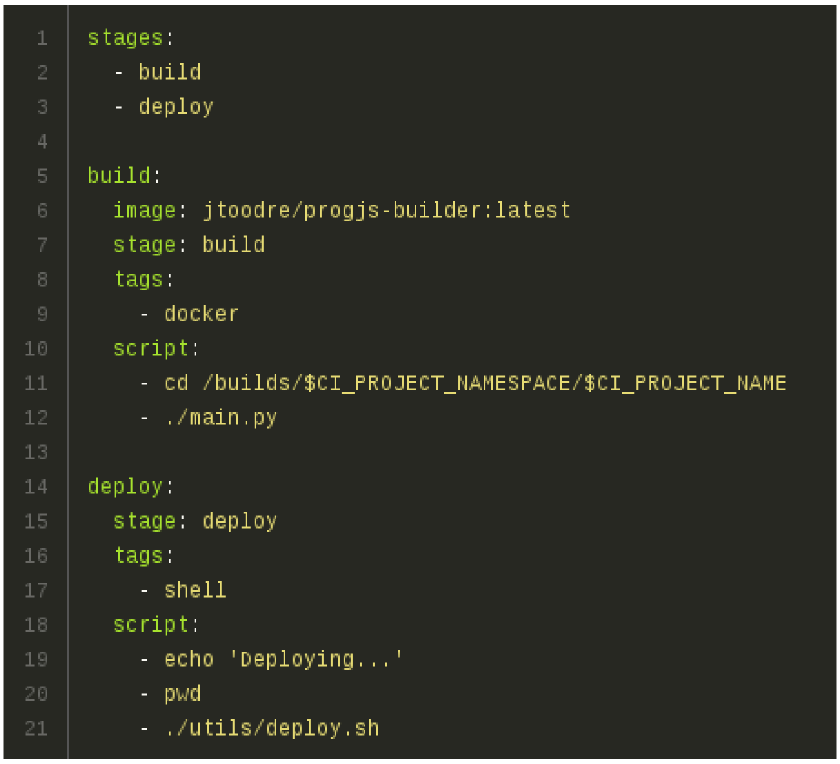
Setting up a runner is usually done in the server which hosts GitLab CE by running *Gitlab CI Multi Runner*. This program will ask for information about the runner, e.g. what will be its name, tags, the type of the runner and type dependant information. Default configuration of a runner can be seen in Figure 16.



```
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
https://gitlab.pld.ttu.ee/ci
Please enter the gitlab-ci token for this runner:
nzSM3kLz1DvPa5PMyhdP
Please enter the gitlab-ci description for this runner:
[xsrv]: iag0582 shell runner
Please enter the gitlab-ci tags for this runner (comma separated):
iag0582,C,grader
Whether to run untagged builds [true/false]:
[false]: false
Whether to lock Runner to current project [true/false]:
[false]: true
```

Figure 16. Runner registration

Configuration file Configuration for CI/CD will be in *gitlab-ci.yml* file. In the file, maintainers should describe the processes- which of them should be run and how they should be run. When writing configuration files and using sensitive data, such as credentials for login, it is wise to use GitLabs *Secret Variables*. They allow the usage of variables, which will be replaced when the build is run, so that the contributors cannot see any sensitive data from the builds. Such variables and predefined variables start with *$* and are usually in upper case. An example of this configuration file can be seen in Figure 17.



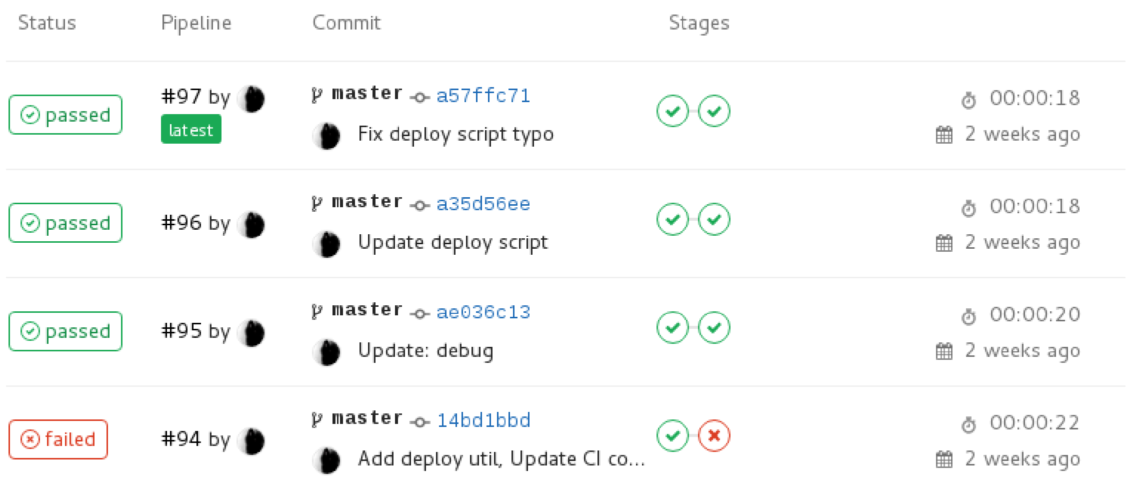
```
stages:
  - build
  - deploy

build:
  image: jtoodre/progjs-builder:latest
  stage: build
  tags:
    - docker
  script:
    - cd /builds/$CI_PROJECT_NAMESPACE/$CI_PROJECT_NAME
    - ./main.py

deploy:
  stage: deploy
  tags:
    - shell
  script:
    - echo 'Deploying...'
    - pwd
    - ./utils/deploy.sh
```

Figure 17. Example CI/CD configuration file

After having the configuration file and runners set up, one can reap the benefits of CI/CD in their project. By default, each time a contributor pushes new code to master or develop, branch pipelines shall be run. Taking the example configuration file (See 17), the pipeline output can be seen in Figure 18.



Status	Pipeline	Commit	Stages	
passed	#97 by latest	master a57ffc71 Fix deploy script typo	✓ ✓	00:00:18 2 weeks ago
passed	#96 by	master a35d56ee Update deploy script	✓ ✓	00:00:18 2 weeks ago
passed	#95 by	master ae036c13 Update: debug	✓ ✓	00:00:20 2 weeks ago
failed	#94 by	master 14bd1bbd Add deploy util, Update CI co...	✓ ✗	00:00:22 2 weeks ago

Figure 18. Example CI/CD pipeline output

Conclusion

In this thesis, a learning environment for programming in C was proposed. Having compared multiple solutions which are already available, it was decided that a custom solution should be built to solve the problems postulated. For the programming environment, a similar solution to CS50 Appliance is used. The main purpose of the appliance is to offer uniform environment with essential tools for programming in C. The same tools must also be available in the automated grading system. In addition to the previously mentioned tools, some utilities are provided for the professors to create grading software.

This work offers suggestions for the tools and frameworks to be used when building this environment. There are also generic guidelines available with given tools on how to develop the system and how to contribute to it. In order to achieve the best environment possible, the project will be open source, thus allowing enthusiasts to contribute to this project. Contributing guidelines include workflow and style guides which can be used with existing tools.

As a result of developing this environment, students will have a vast variety of tasks at their hand. This is achieved by the help of contributors and the concept of automatic grading. Since the tools used for automatic grading software are limited to Linux only, it means that if some other programming languages can be checked by using Linux tools, they are automatically usable within this system. In other words, the developed system is scalable and its usage is very desirable when teaching programming.

References

[1] A. B. Dieter Pawelczak, “Virtual-c a programming environment for teaching c,” April 2014.

[2] A.ezzat, “Getint() can not read more than 10 digits and proplems with zero!” September 2015. [Online]. Available: https://cs50.stackexchange.com/questions/12324/getint-can-not-read-more-than-10-digits-and-proplems-with-zero

[3] R. Drummond, “A node.js application on the amazon cloud. part 1: Installing node on an ec2 instance,” April 2013. [Online]. Available: http://www.robert-drummond.com/2013/04/25/a-node-js-application-on-the-amazon-cloud-part-1-installing-node-on-an-ec2-instance/

[4] D. McFarland, “The real reason why everyone should learn to code,” http://blog.teamtreehouse.com/havent-started-programming-yet, November 2014.

[5] “Cs50 syllabus,” August 2016. [Online]. Available: http://docs.cs50.net/2016/fall/syllabus/cs50.html

[6] T. MacWilliam, “How did the idea of cs50 appliance originate?” April 2015. [Online]. Available: https://www.quora.com/How-did-the-idea-of-CS50-appliance-originate/answer/Tommy-MacWilliam?srid=uO406

[7] V. Kukk, “Vello kukk: õppimine 21. sajandil,” June 2012. [Online]. Available: http://arvamus.postimees.ee/878860/vello-kukk-oppimine-21-sajandil

[8] Moodle, “Moodle stats,” May 2017. [Online]. Available: https://moodle.net/stats/

[9] “Moodle history,” July 2015. [Online]. Available: https://docs.moodle.org/33/en/History

[10] O. Sychev, “Blocks: Formal languages block,” December 2016. [Online]. Available: https://moodle.org/plugins/block_formal_langs

[11] “Virtual-c ide,” January 2017. [Online]. Available: https://sites.google.com/site/virtualcide/home

[12] “Cs50 appliance 19,” January 2017. [Online]. Available: https://manual.cs50.net/appliance/19/

[13] “Moodle repository,” January 2017. [Online]. Available: https://github.com/moodle/moodle

[14] “Virtualization in education,” IBM Global Education, Tech. Rep., October 2007.

[15] distrowatch, “Search distributions,” 2017. [Online]. Available: https://distrowatch.com/search.php

[16] M. Larabel, “Gcc 7.0 vs. llvm clang 4.0 performance with both compiler,” https://www.phoronix.com/scan.php?page=article&item=gcc7-clang4-jan&num=1, January 2017.

[17] M. Tanjim, “Best text editors for linux command line,” July 2016. [Online]. Available: https://itsfoss.com/command-line-text-editors-linux/

[18] V. Developers, “Valgrind,” March 2016. [Online]. Available: http://valgrind.org/

[19] “Comparison of version control software,” April 2017. [Online]. Available: https://en.wikipedia.org/wiki/Comparison_of_version_control_software

[20] “What is docker,” November 2016. [Online]. Available: https://www.docker.com/what-docker

[21] T. Preston-Werner, “Semantic versioning 2.0.0,” December 2016. [Online]. Available: http://semver.org/

[22] Riigikogu, “Personal data protection act,” December 2007. [Online]. Available: https://www.riigiteataja.ee/en/eli/ee/529012015008/consolide/current

[23] “Express - fast, unopinionated, minimalist web framework for node.js,” March 2016. [Online]. Available: https://expressjs.com/

[24] JS Foundation, “Eslint - rules,” April 2017. [Online]. Available: http://eslint.org/docs/rules/

[25] Airbnb, “Airbnb javascript style guide,” March 2017. [Online]. Available: https://github.com/airbnb/javascript

[26] C. Richardson, “Pattern: Microservice architecture,” 2017. [Online]. Available: http://microservices.io/patterns/microservices.html

[27] GitLab, “Executors,” April 2017. [Online]. Available: https://docs.gitlab.com/runner/executors/#selecting-the-executor

A. POC - Node Application app.js

The main JavaScript file which is necessary to run the Node app is shown in Listing 1. This application imports the necessary modules in order to run the application. These modules include router, template engine, database interface, configuration and API endpoints. When this application is run, it will start a web server on port 3000 which can be accessed either by localhost or local IP. Once the web server is run, on the line 45 the program checks whether it was run for testing purposes or it was deployed. When testing, it is important to close the process in order to make the GitLab CI pipeline succeed, otherwise the application would run for the time period set, and then close with the test, marked as failed.

```
1  // Using util to log info with timestamps
2  import util from 'util';
3  import express from 'express';
4  import expressVue from 'express-vue';
5  import mongoose from 'mongoose';
6  // Require user modules
7  import config from './config';
8  // Require controllers
9  import seedAPI from './controllers/seedAPI';
10 import taskAPI from './controllers/taskAPI';
11 import feedbackAPI from './controllers/feedbackAPI';
12 import containerAPI from './controllers/containerAPI';
13 import mainRouter from './controllers/mainRouter';
14
15 const app = express();
16
17 const port = process.env.PORT || 3000;
18
19 app.use('/', express.static(__dirname + '/assets'));
20
21 app.set('views', __dirname + '/views');
22
23 app.set('vue', {
24     componentsDir: __dirname + '/views/components',
25     defaultLayout: 'layout'
26 });
27 app.engine('vue', expressVue);
28 app.set('view engine', 'vue');
29
30
31 // Establish connection to database (currently no authentication)
32 mongoose.Promise = global.Promise;
33 mongoose.connect(config.getDbConnectionString());
34
35 // Routes
36 app.use('/', mainRouter);
37 app.use('/api/task', taskAPI);
38 app.use('/api/seed', seedAPI);
39 app.use('/api/feedback', feedbackAPI);
40 app.use('/api/container', containerAPI);
41
42 app.listen(port);
43
44 util.log(`Server started on port ${ port }`);
45 if (process.env.LOCALDEPLOY === 'true') {
46     setTimeout(() => {
47         console.log('Run successfully.');
48         process.exit(0);
49     }, 5000);
50 }
```

Listing 1. Node application main JavaScript file