Sergey Levenets

JavaScript and Java

on the server-side

Bachelor’s Thesis

Information Technology

May 2016



DESCRIPTION

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | **Date of the bachelor's thesis**  06.05.2016 | | |
| **Author(s)**  Sergey Levenets | | **Degree programme and option**  Information Technology | | |
| **Name of the bachelor's thesis**  JavaScript and Java on the server-side | | | | |
| **Abstract**  The purpose of the study is to compare two different languages on the server-side. The thesis helps to get an idea how to build an application by using these popular technologies. This document may help a reader to select the technology for a development. The information from this study may be used for learning, building and improving web applications. It can be useful for IT students, teachers and web developers.  In the document, I made a research about Java and JavaScript, the client-server architecture and the back-end development with mentioned languages. After the planning and the workspace building, I started from the development of two web applications and ended up with results and overall picture of Java and JavaScript on the server-side.  The studies showed that both languages are truly competitive. Moreover, both technologies have huge developer community. Java looks like the best option for huge corporate servers. However, there are examples of JavaScript servers that successfully process enormous amounts of connections and data. The wisest decision is to investigate the problem deeply to choose the right technology. | | | | |
| **Subject headings, (keywords)**  JavaScript, Java, client-server, server-side, back-end, web development | | | | |
| **Pages** | **Language** | | | **URN** |
| 72 | English | | |  |
| **Remarks, notes on appendices** | | | | |
| **Tutor**  Timo Mynttinen | | | **Employer of the bachelor's thesis**  Mikkeli University of Applied Sciences | |

**CONTENTS**

1 introduction 1

2 Javascript and java 2

2.1 What is JavaScript? 2

2.2 What is Java? 4

3 the client-server concept 7

4 server-side possibility 11

4.1 A server with JavaScript 11

4.1.1 The engine 13

4.1.2 Node.JS 14

4.2 A server with Java 16

4.2.1 J2EE 16

4.2.2 Application servers 19

5 planning the work 24

5.1 Phases of the project 24

5.2 The Waterfall model 25

6 building the workspace 26

6.1 Installing the workspace 26

6.1.1 VMware and Ubuntu 27

6.1.2 Komodo Edit 31

6.1.3 Java Development Kit 32

6.1.4 Intellij IDEA 33

6.2 Test the workspace 35

7 building applications 37

7.1 Building the client 37

7.1.1 The HTML page 37

7.1.2 Hide elements with CSS 38

7.1.3 The client’s JavaScript 40

7.2 Building the JavaScript server 43

7.2.1 Installations 44

7.2.2 The server code 45

7.2.3 Launch and test 52

7.3 Building the Java server 54

7.3.1 Installations 55

7.3.2 The server code 59

7.3.3 Launch and test 64

8 results and analysis 65

Conclusion 66

BIBLIOGRAPHY 68

# 1 introduction

Today, almost everyone uses the internet. The popularity of online services, websites and web applications is growing with huge speed. Social networks, banking systems, online shops, entertainment, education, government services, science and health services almost every part of our lives are merged with this great invention called the Web. However, this is not a magic. The world needs people called developers and technicians to setup these services, so we can use them all.

Nowadays, the process of installation and development can be different. There is no single technology or solution for everything. Developers can achieve goals in many ways. Therefore, finding the best technology can be a huge question, because all technologies have certain problems: security, scalability, operating speed, latency, compatibility with other technologies, cost efficiency, implementation and development process.

The aim of my thesis is to investigate two competitive technologies on the server-side. These technologies are JavaScript and Java. The battlefield for the comparison will be development of the client-server application. Performance of the standalone server is important. However, interaction between a client and a server can be crucial sometimes. Therefore, I go through the server development and the interaction with the client to discover strengths and weaknesses.

Chapter 2 of the theoretical part considers information about Java and JavaScript as a language. Chapter 3 is devoted to the concept of client-server programing. Chapter 4 introduces basic information about server-side possibilities of these programming languages: libraries and frameworks. The next thing will be to plan my work and divide the whole process in smaller steps in Chapter 5.

The purpose of the practical part is to build a web application and to do the server-side with two different languages. Therefore, I go through the programming process and gain certain knowledge. To start the development, I will need to set up my workspace. To do so I need to install all the programs and libraries needed. Chapter 6 considers this process. Chapter 7 will go through the process of development. The output of the work accomplished is described in conclusion.

# 2 Javascript and java

## 2.1 What is JavaScript?

Let us see what is so special in JavaScript, why JS, and what other technologies exist, except JavaScript.

JavaScript was originally created in order to make the webpage “live”. Programs in this language are called scripts. In the browser they are connected directly to HTML, and once downloaded immediately performed. Program on JavaScript is an ordinary text. They do not require any special preparation. In this regard, JavaScript is quite different from the other language called Java. (Chapman 2014a)

When JavaScript was created, it originally had a different name: “LiveScript”. However, at that time, Java was a very popular language, and marketers decided that a similar name will make a new language more popular. It was planned that JavaScript is like a “little brother” of Java. However, the story had its own way. JavaScript has grown, and now it is a completely independent language with its specification, which is called the ECMAScript, and has nothing to do with Java. It has many features that complicate the development.

JavaScript can be performed not only in the browser, we just need a special program, which is called interpreter. The process of executing the script is called interpretation. To run the programs, no matter in what language, there are two ways: compilation and interpretation.

* Compilation: This is when the source code, using a special tool, which is called compiler, is converted into another language, to machine code. This machine code is then distributed and executed. In this case, the source code remains with the developer.

* Interpretation: This is where the source code gets another tool, which is called the interpreter, and executes it “as is”. At the same time, it applies the source code itself (the script). This approach is used in JavaScript for browsers. Modern interpreters convert JavaScript into machine code, optimize and then perform. Even during the execution, interpreters are trying to optimize the code. Therefore, JavaScript is very fast.

All major browsers have an integrated JavaScript interpreter, which is the reason why they can run scripts on the page. However, of course, JavaScript can be used not only in the browser. This full-fledged language program can be run on the server and even in a washing machine if it has an appropriate interpreter installed. Modern JavaScript is a “safe” general-purpose programming language.

In JavaScript the browser is able to perform everything that relates to the manipulation of the page, to interact with the visitor and, in some measure, with the server (Web Education Community 2011). Let me introduce basic page manipulations in the list below:

* Create new HTML tags, delete existing ones, change styles of elements, hide/show elements
* React to the visitor, process a mouse click, a cursor movement or a keyboard activity
* Sends a request to the server and download data without reloading the page (this technology is called "AJAX")
* Get/set the cookie, to query data, to display messages

Although, JavaScript is a fast and powerful language, the browser imposes on its execution some restrictions. This is done for the safety of users, so that an attacker can not use JavaScript to obtain personal data or to harm the user's computer. These restrictions are not there, if JavaScript is used outside the browser, such as a server. In addition, modern browsers provide their own mechanisms for installing plugins and extensions that have advanced features, but require special actions from the user to install. Following Chapman (2016b):

* JavaScript cannot read / write files on the hard drive or copy them. It does not have direct access to the operating system
* Modern browsers can work with the files, but this feature is limited to the designated directory, which is called sandbox
* JavaScript works in one tab and cannot communicate with other tabs, and windows
* From JavaScript, developers can easily send requests to the server, from which a page comes. Request a different domain is also possible, but less convenient, because there are also security restrictions

Why is JavaScript unique? There are at least three remarkable features of JavaScript:

* full integration with HTML / CSS
* simple things done simply
* supported by all widespread browsers, and is enabled by default

These three things are not present at the same time in any browser technology. Therefore, JavaScript is the most common way to create browser-based interfaces.

## 2.2 What is Java?

Java is both a programming language and platform. Java was developed by Sun Microsystems and later acquired by Oracle. (Beal, 2016)

First, Java is an object-oriented high level programming language. At compile time, Java code transformed into intermediate language code, which is called bytecode. In turn, the byte code is parsed and is executed by the Java (JVM) virtual machine that acts as a translator between the Java language and the hardware. All Java implementation must emulate the JVM, creating applications to run on any system that includes a virtual Java machine.

Second, Java is a software platform, versions of which are available for a variety of hardware systems. The platform includes the JVM and the application-programming interface in Java (API), which is an extensive set of ready-made software components (classes) to facilitate the development and deployment of applets and applications. API Java covers many aspects of development in Java, including the manipulation of basic facilities, network programming, security, XML generation and web-based services. API is organized as a set of libraries, called packets, which contain classes and interfaces for solutions related to each other problems. (Nourie and Pawlan 2007)

There are three editions of the Java platform. This allows application developers, service providers and hardware manufacturers to create solutions that meet the requirements of specific user groups. Following Will (2007), these editions are:

* Java SE (Java Platform, Standard Edition): Using the Java SE, developers can create and deploy Java-based applications to desktops and servers, as well as to develop embedded software for real-time systems software. Java SE includes classes needed to create a web-service, and the basic components of Java EE (Java Platform, Enterprise Edition);
* Java EE (Java Platform, Enterprise Edition): This corporate version of the platform helps developers build and deploy portable, robust, scalable, and secure server-side applications in Java. Developing the capabilities of Java SE, Java EE provides API web-services component models, remote communication and control for the implementation of enterprise software for the principles of SOA and Web 2.0;
* Java ME (Java Platform, Micro Edition): Java ME provides an environment to run applications developed for a wide range of mobile and embedded systems such as mobile phones, PDAs, set-top boxes and printers. This edition of the platform provides tools to create flexible user interfaces, robust security models, a full range of built-in network protocols as well as a strong support network and offline dynamically loaded applications. Applications based on Java ME specifications can be run on multiple devices at the same time and are able to make effective use of their system capabilities;

The main advantage of the Java language is expressed in the portability of the Java-applications’ ability to run on any hardware platforms and operating systems, because all of the JVM, regardless of the platform on which they work, are able to perform the same bytecode.

The language and the Java platform have excellent scalability. We can easily create applications for devices with limited resources, adapting the software, which was originally written for the desktop. In this case, the Java language is ideal for developing server-side web-based applications, with the help of which the user can access computing resources on the Web. The ability to safely execute code downloaded over the network was originally founded in the Java design so that this language provides a high level of security when using the Internet. Web-based applications run in a runtime environment called Web-containers which provide a variety of convenient services, including scheduling requests, security and parallel operation, lifecycle management as well as access to the API. A number of application servers was built for Java to perform as Web-containers of functions to other Java components, XML and web-services that interact with databases and dynamically shape the content of webpages. These servers also provide an environment for deploying enterprise applications and tools to manage transactions, clustering, security, connectivity and the required level of availability, performance and scalability. Java is a basis for a number of platforms, designed for software development and effective operation on different chip architecture systems. Java helps developers to solve the problems of the server, client and embedded systems.

One of the strengths of the Java virtual machine has always been its ability to easily juggle with multiple threads. The JVM is optimized for large multi-core machines, and it is not a problem to manage hundreds of threads. These magical capabilities are used by many websites with high traffic. Developers write code on their laptop, and then deploy the application on a powerful server where it uses multi-core power at full strength (Drake 1996). Ruby is one of the modern Java competitors. It has a clean and a live English syntax. However, when Ruby fans need high productivity, they turn to JRuby. This is version of Ruby that runs on top of the JVM, providing much better performance under high loads with multiple threads. Having invested a bunch of effort for reliable work with threads, engineers from Sun made the right decision.

Java has sustained success on the microchips market. Java has never been a popular tool for the development of desktop applications, but it has flourished in the mobile market segment, which has recently rushed up. The Android platform is built on Java and now Android devices are selling better than iPhone. This dominance is not something new. A cut-down version of the language and virtual machine, known as Java ME, is widely used worldwide on millions of phones.

**Key features of Java**:

* automatic memory management;
* enhanced exception handling;
* a rich set of filtering the input-output;
* a set of standard collections: array, list, stack;
* availability of simple tools for creating network applications
* classes that allow us to perform HTTP requests and process responses;
* unified access to databases:
* support generalizations;
* parallel execution of programs;

# 3 the client-server concept

Usually computers and programs that are part of the information system are not equal. Some of them have the resources (file system, a processor, a printer, database, etc.), and others are able to access these resources. The computer (or program) which can manage resources is called the server of the resource (file server, database server, and compute server). The client and the server can be located on the same computer or on different computers connected by a network. (Slideshare 2012)

A multi-level representation of computing systems can be divided into three groups of functions aimed at addressing various subtasks. The three groups are:

* Input functions and display data (providing user interaction)
* Application functions related to the subject area
* Resource management function (file system, database)

These functions are mainly provided by software, which can be represented in the form of interconnected components such as the following:

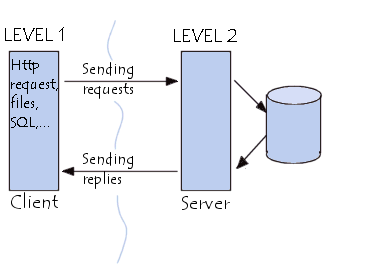
* View component is responsible for the user interface
* Application component implements an algorithm for solving a specific problem
* Resource management component provides access to the necessary resources

The client-server architecture defines the general principles of cooperation in the network. Servers are component suppliers of some specific functions (services). Clients are consumers of these functions.

The practical implementation of such an architecture is called client-server technologies. Each technology defines its own rules or uses existing rules of the interaction between a client and a server, called the exchange protocol.

**A two-tier architecture**

In any network, built on modern network technologies, there are elements of client-server interaction, usually based on a two-tier architecture. It is called Two-tier because of the need to allocate three basic components between two hosts (client and server).



**FIGURE 1. The two-tier architecture**

A two-tier architecture is used in client-server systems, where a server responds to client requests directly while using only its own resources. The server does not apply to third-party network applications and third-party resources to perform any part of the request (Figure 1). (CCM Encyclopedia 2008)

Components on the client-side or the server determines the following basic models of their interaction within the two-tier architecture:

* Terminal server provides distributed data representation
* File server provides access to a remote database and file resources
* Database server provides remote data presentation
* Application server provides the remote application

**Three-tier architecture**



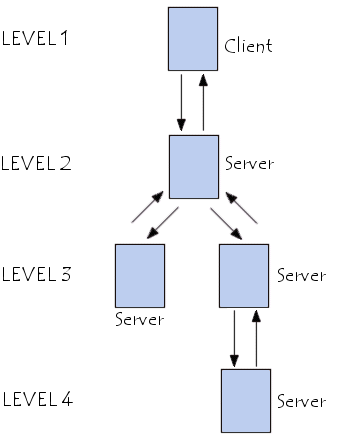
**FIGURE 2. The three-tier architecture**

Another trend in client-server technologies associated with the increasing use of distributed computing. They are implemented on the model of the application server, where the network application is divided into two or more parts, each of which can run on a separate computer. Dedicated parts of the application communicate with each other by exchanging messages in a pre-agreed format. In this case, a two-tier client-server architecture becomes a three-tier (Figure 2). (CCM Encyclopedia 2008)

Typically, the third link in the three-tier architecture becomes an application server, and then the components are as follows:

* Data presenting is on the client-side
* Application component is a dedicated application server (as an option, to perform middleware function)
* Resource Management is on the database server, provides requested data

Three-tier architecture can be upgraded to a multi-tier (N-tier) by allocating additional servers. Each of which will provide its own services and use the services of other servers at different levels. Abstract example model is shown in Figure 3.



**FIGURE 3. The N-tier architecture**

**Comparison of architectures**

A two-tier architecture is simpler, since the same server services all requests, however because of this it is less secure, and places high performance demands on the server.

Three-tier architecture is difficult, but because the functions are distributed between the second and third-level servers, this provides (CCM Encyclopedia 2008):

* A high level of flexibility and scalability
* High security (protection can be defined for each service or level)
* High performance (tasks can be distributed between servers)

# 4 server-side possibility

## 4.1 A server with JavaScript

Let us look at an extremely progressive approach, when we build server-side code using JavaScript.

Many developers have always believed JavaScript is just a “gadget” for the browser, some kind of semi-language, which is good only for the management of forms and manipulating the DOM-tree webpage. Some people still think that the java word in the name means something. Indeed, the language is very simple. However, real programmers long ago learned to do wonderful things with it, giving us amazing, convenient online services that we use every day. Many of these pros have gone further and, soberly looking at the language itself and its capabilities, especially in terms of working with events, decided to build the server with JavaScript. Developers get the opportunity to write all of the website parts using the same language. In addition, JS is great, just perfect for different web manipulations. It is very simple and at the same time flexible, allowing us to write code in different paradigms: from the usual procedural to object-oriented programming in a mixture with a functional style.

Most importantly, JS is completely asynchronous. This means that any code will not be executed sequentially, as in the case of PHP/Perl scripts, but at a time when all the data for it is ready. After all, the Web does not require a lot of computing power, because most of the time the server waits for the events such as acquisition data from forms, a sample of the database or a response to the request to another server. Common PHP script at these moments is in idle state. Therefore, the entire thread is in idle state, not allowing the server to use it for other users. In this case, using JavaScript we simply indicate which function to perform, when a certain event happens. That is all. At this time, other code can easily be performed. Such a function is called a callback or an event handler. Although writing really complex code in this manner will be a bit uncomfortable, especially if our function depends on several events at once, but there are solutions using different frameworks. These solutions are more powerful and elegant than all of these PHP / Ruby / Python (Wayner 2011).

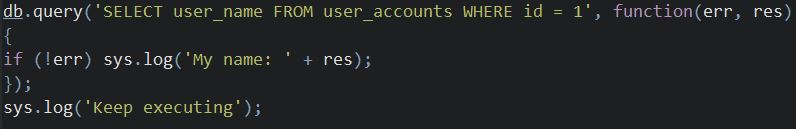
To show why this asynchronous is so important I will introduce two typical examples by using the PHP and JavaScript code that perform in the same way. Examples will show the sequential execution restrictions. Let me start from PHP (Figure 4):



**FIGURE 4. We send a simple SQL-query to the database with PHP**

On the first line we send a simple SQL query to the database to get the user name with the id = 1. Note that at this point the script is stopped. The next line will not be executed until the moment when the request will not be processed by the database. The result will not be returned to the variable $result. Yes, in this example, it will take thousandths of a second, but in real services, queries are much more complex, databases size up to gigabytes, and such requests can simultaneously be a couple of thousand.

Next, we try to carry out the code on JS by using asynchronous style (Figure 5):



**FIGURE 5. We send a simple SQL-query to the database with JavaScript**

Again, we create a database query, but in addition to the SQL expression, also a handler function (callback) is transmitted in the request. This function will be called when the response from the database will be received. But to this point, the script will not stop in any case. For example, on the next line we simply display a string to the console to indicate that the script continues immediately after the request, without waiting for its completion. In fact, at the heart of any version of the server-side JavaScript, the concept of events and callbacks (event handlers) incorporated. We can describe own events. Then the progress of the application will depend on the events that occur as a result of user activity on the page or are generated within the server itself (as in the case with reference to the database). These actions, which must be performed in the case of events, are described inside the event handler functions.

### 4.1.1 The engine

In short, the use of JavaScript not only on client-side, but also on the server-side is good and even pleasant. Another question, how to implement it. Today there are four main engines used on servers, and they can be listed and described as follows.

**Rhino** is the engine from the Mozilla Company, written in Java and supports the latest 1.7 version of the JS standard, which also adds its own language extensions and objects. The main advantage of the engine is to work on top of the standard JVM. Therefore, it can be used in any environment where Java is running. In other words, it is possible to use modern web servers such as the Jetty, but still code with JS. By the way, the Rhino is used in the cloud hosting from Google. However, performance is a problem. It depends, on the one hand, from the engine and the technologies used there, such as JIT-compilation, on the other hand, from the work of JVM. In fact, many testers who say that Rhino is slow, forget that the engine has two modes of operation:

* the interpretation, when the script is converted into Java byte code every time;
* the compilation, when this transformation takes place only once, and then repeatedly executed

The first mode is beneficial when we are debugging code that changes every minute, the second is more suitable for a working version of the project, working under load.

**SpiderMonkey** is an another engine from the Mozilla. This is world's first JS engine, written in the Netscape. Today, it is open and is used in popular products like Firefox, Adobe Acrobat, and even in one of the server emulator for online game Ultima online. Later on, developers greatly modified it by adding JS compilation directly in assembly code, and renamed into TraceMonkey - this is the engine used in Firefox 3.6. SpiderMonkey is mainly used in the software, which is written in C / C ++ and requires a scripting language.

**Futhark** is the engine from Opera, which in addition to the browser is used in their service Unite and on their servers serving the Opera Mini (mobile browser). The engine is closed and it is not used outside Opera itself.

**V8** is the engine from Google which is used in Chrome and is the basis for a future Chrome OS. Today, V8 is the fastest and most powerful engine in which JS code is directly converted to the target processor assembler. This allows V8 to be the fastest, relative to other engines. Additionally, many contrivances are used for optimization, such as follows:

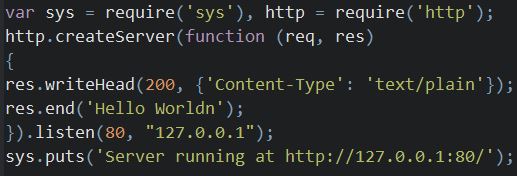
* A compiled code can be stored in the memory.
* The code can be optimized on the fly (e.g., compiler can decide to remove blocks of code, when they generally might not be employed).

The most popular and rapidly growing server platform, Node.JS, is built on the basis of this engine.

### 4.1.2 Node.JS

Probably after the release of Chrome, developers have realized that such a fast engine can be used successfully on the server. The first experience was the V8cgi project that simply allows users to write server-side scripts that work with any web server on the standard CGI protocol. Further experiments led to the birth of the Node.JS project. Node is a fully independent platform that includes, in addition to the engine, the integrated server (HTTP and TCP / UDP / Unix-socket) and a basic set of libraries as well as provides a fully asynchronous work with files and network devices.

The project is developing so rapidly and extensively that now it is ready for industrial use. This, in particular, is proved by experience of people from Plurk (an Asian Twitter). Their developers completely moved their Comet server, originally written in Java and solid JBoss Netty, on Node.JS. According to their responses, they reduced memory consumption literally for gigabytes. They still have more than hundreds of thousands of simultaneous connections. It will take few lines of code to start a HTTP server that can handle thousands of connections asynchronously (Figure 6):



**FIGURE 6. Starting the HTTP server**

The server itself is written with C ++ and Assemble, however most of the libraries of the distribution is developed with JavaScript. The base server includes a set of basic functions. The rest is left for developers who have already written hundreds of different libraries and frameworks. However, the youth of project does not give a confidence in its stability. Some trivial tasks may not be "ready to download" solutions, but it happens and vice versa. There are many implementations which are radically different. While most libraries are written with JavaScript, there are those that require the compilation of the module to the server, which promises much greater speed. They simply extend the standard server API.

The main feature of the Node, in addition to the full asynchronous, is the single-threaded model. In other words, all operations are performed in one running thread, even if the server have thousands of simultaneous users. It is recommended just to load multiple copies of applications for the implementation of multi-threading and the use of all cores of modern processors. However, we can take WebWorker of the HTML5 standard and distribute applications to work on several child processes. Since there is no multi-threading it does not mean that system will be slow. Remember that the web application does useful work very quickly, and most of the time just waiting for something.

The second feature of the architecture is the Node eventfulness. Almost every operation has callbacks, generates an event, and the user is given an object EventEmiter, through which developers can literally with one line generate own events (this is easy, because the event is just a string with the name and a list of parameters that are passed to the handler).

Node itself is built around EventLoop, a global event handler loop, which at every tick checks whether the data is ready for any of the specific callbacks. If there is a data, the code execution begins. When there is no code, then forward to the next call. The cycle is executed inside the engine, written in C, whereby it happens very quickly. In addition, the server has a very efficient integrated garbage collector (GC), so even thousands of connections do not cause memory overflows and server falls. Node.JS has built-in native system to work with events.

## 4.2 A server with Java

### 4.2.1 J2EE

According to Brown (2002), J2EE includes many technologies to reach any goal. Let me describe them below.

**Enterprise JavaBeans (EJB)** is a component model that simplifies the development of middleware software, providing services such as the transaction management, security, and database connectivity.

**Portlet Specification** defines a set of API to create portals on Java, covering aspects such as aggregation and presentation of information, personalization, and security

**JavaMail** is the API presenting a set of abstract classes that manage a mail system.

**Java Message Service (JMS)** is an API, which supports the creation of portable Java-based applications based on the messaging engine. It defines a common set of basic concepts and programming strategies for all messaging systems compatible with JMS.

**JavaServer Faces (JSF)** provides a programming model that helps to create web applications through the page layout of reusable UI components and bind these components to data sources, and the events generated by the client.

**JavaServer Pages (JSP)** provides to web developers the ability of a quick creation and easy support for dynamic, cross-platform webpages that share the user interface and the generation of content, so designers can change the layout without touching the dynamically generated content.

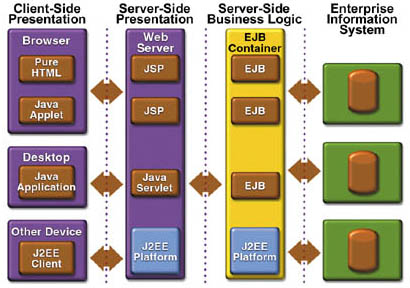
**The standard tag library for JavaServer Pages (JSTL)** is a set of special tags that provide a standard format for the implementation of the actions necessary to many web applications.

**Java servlets** extend the functionality of web servers, providing a cross-platform, component-based approach for creating web applications, free of performance limitations of CGI.

**J2EE Connector Architecture (JCA)** provides a standard architecture for connecting J2EE applications to heterogeneous enterprise information systems (EIS). This architecture defines a set of scalable and secure mechanisms on the basis of transactions, by which the EIS vendors can provide a standard resource adapters to be included in the application server.

**Specification J2EE Management (JMX)** defines a management information model for the J2EE platform. This model has been specifically designed to interact with many systems and management protocols. It provides a standard abilities for comparison with a common information model (Common Information Model - CIM), information base SNMP management (SNMP Management Information Base - MIB) and the Java object model using the server EJB component (J2EE Management EJB Component - MEJB).

**Java Transaction API (JTA)** is a high-level protocol-independent API that provides access to transactions for programs and media server applications. Java Transaction Service (JTS) specifies the implementation of the transaction manager that supports JTA and implements, on the underlying level, comparison with the object transaction services. A distribution of JTS transaction is realized with the help of Inter-ORB Protocol.



**FIGURE 7. The model of the J2EE application**

The model of J2EE applications developed based on JavaTM 2 Platform, Enterprise Edition, simplifies the design principles of scalable and widely available internet and intranet based applications (Figure 7).

The question is what features are missing in the applications made with JavaTM 2 Platform, Enterprise Edition. The solution of various complex tasks, such as the business operations management, lifecycle management, merging of resources, should be performed by corporate applications. All of this are built into the platform, and is performed automatically for all supported components. Therefore, developers of components and applications can focus on other tasks, such as a business logic and a user interface.

Another advantage of J2EE is that the application model puts functional levels into certain types of components. The business logic is contained in the components of Enterprise JavaBeansTM (EJB). The interaction with the client may be carried out by usual HTML webpages, through the webpages using applets based on the Java technology, Java servlets API and JavaServer PagesTM technology or by standalone Java applications. The components communicate freely, using different standards: HTML, XML, HTTP, SSL, RMI, IIOP, and others. Reusability of J2EE components provides developers a competitive choice. J2EE allows them to assemble applications from a combination of standard widely available components and their own custom components. It is expected that a number of standardized functional features of Java 2 Enterprise Edition will already be able to solve a wide range of tasks such as the development of components of applications for the business sector and solving problems for vertical markets. This means that the website for the purposes of electronic commerce can be built by using a combination of EJB ready-to-go components or the modified EJB components to create specialized services for customers.

This approach to the construction of web services means that less time is required to develop. Moreover, a high performance, portability and ease of use within a wide range of platforms can be achieved. The main advantages of this approach are: the higher code productivity, the best strategy to manage computer resources and a greater return from investment in the technology.

**Java has a flexible user interaction.** J2EE provides a choice of the graphical user interface within the company intranet or the World Wide Web. Customers can use desktop computers, laptops, PDAs or cell phones. On the client-side, the simple user interfaces can use HTML and Java applets. The support of HTML means that prototypes will run faster, and it will cover a wide range of users. As an additional service, J2EE has Java PlugIn, which connects applets. J2EE also works with standalone Java applications.

In order to support the server capability for dynamical update of its contents J2EE has both Java servlets API and JavaServer Pages (JSP) technologies. The Java servlet API allows developers easily implement the server by using all the advantages of Java API features. JavaServer Pages Technology combines the prevalence of HTML with possibility to place Java programming scripts on the server.

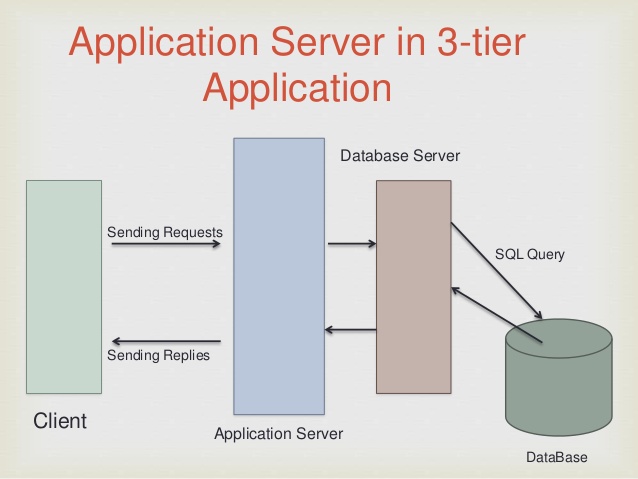
**Enterprise JavaBeans is a flexible design model for business components.** TheEnterprise JavaBeans technology had wide growth after micro software filled the market. The reason is that this technology simplifies the approach of the development of multi-tier applications, concealing their complexity, and allows developers to focus on business logic. The J2EE platform is the result of the natural development of Enterprise JavaBeans technology.

The EJB technology let developers to simulate a wide range of objects that are used across the enterprise, defining two types of EJB components: Session Beans and Entity Beans. Session Beans represents the client's session. For example, the implementation of user transactions on the site. Entity Beans display an entity, such as a data in the relational database, and contains operations with the targeted data. Entity Beans are designed to provide stability and security of data.

### 4.2.2 Application servers

The submission of application logic into a separate layer gives developers more flexibility in a distributed information systems creating. Accommodation and program execution on the server-side reduce the hardware requirements of customers and the problem of interoperability in a heterogeneous network environment.

Application server is a service program that provides client access to applications running on the server. The application server is usually stands out as the middle tier (Figure 8).



**FIGURE 8. The application server as the middle tier**

* A first level: The interface is usually graphical (GUI)
* Middle level: There is executable code, usually hosted on a dedicated server
* The third level, the background, the database: This includes data access and transaction management.

In a network environment, the application server is the intermediate layer between the client’s front-end and database servers.

The business logic may be implemented on the server-side as a remote code and a distributed code. In the first case, the server can access terminals and clients. This interaction corresponds to the model of "terminal server". In the other case, clients can receive the components of the server application and carry them on their side (examples are JavaScript, applets, Flash).

Customers can interact with applications through the server API (Java client-servlet or container-servlet). The interaction through third-party services, in the first place via the web server, provides more flexibility and versatility. The concept of an application server is traditionally associated with the Java platform, pointing out that the server of Java applications is the implementation of the servlet specification, possibly in the form of the JSP, and some other maintenance services such as connection to the database. The application server provides an environment in which applications can be operated, regardless of what and how they are doing. Therefore, to answer the question of whether some service software is an application server, it is necessary to compare its functions with the following list of features:

* It provides a container model for applications.
* It provides maintenance services for the programs.
* It provides management of applications and / or tools for development.
* It complies with industrial standards and specifications.
* It provides the website maintenance, because of the real demand for technologies based on the WWW.

**The question is how to implement an application server.** According to the list above, the category of the application servers includes, for example, traditional terminal-server systems, CGI technology, the Java servlet containers and others.

**Terminal Server** provides an environment for a remote execution of programs, which acts as the operating system itself. An access to them is carried out by remote control protocols (Telnet, SSH, RDP and VNC) from the client’s software (a terminal emulator controls the remote desktop, etc.). The management of a running program performed by the emulated client user interface (text or graphic). On the server-side, the interaction with the operating system software is realized through system calls. It is also managed by the operating system. The development can be conducted with any language available in the operating system.

**Common Gateway Interface (CGI)** is the technology to access applications through a web server. Differences from the terminal server here is that the user interface is provided in the form of webpages. Requests of web clients, to access the application hosted in a dedicated directory (usually cgi or cgi-bin), are redirected to their input through standard input stream (stdin). The application returns results, in the form of hypertext, to a server via stdout.

**Java has own application servers.** The Java platform is the industry standard. It allows programmers to create components of the unified interoperable software solutions for a variety of systems in which the Java virtual machine can be started (JVM). The servlet container is one of the architectural components of J2EE, which is the environment for servlets execution. A servlet is a Java-based application that runs on the server-side (as opposed to an applet). The servlet container can work as a full standalone server. However, it is more commonly used together with other server software. The servlet container provides communication between clients and servlets and assumes functions such as creating a software environment for the running servlet, the identification and authentication of customers and the organization of the session for each of them. The servlet container concept allows us to create both universal and specialized application servers. Examples of the implementation of the servlet container are Apache TomCat, Apache Geronimo, JBoss, GlassFish and IBM WebSphere Application Server (WAS).

Following Pravin (2009), I will put basic application servers’ **advantages** on the list below:

* The integrity of the code and data

Placing business logic on a dedicated server or a limited number of server computers guarantees access to a updated and upgraded software for all customers. This eliminates the risk of data access and management of legacy and possibly incompatible applications.

* Centralized control

Modifications in the configuration of applications, such as, for example, the change of the database server, are performed centrally.

* Security

Centralized resources, through which the service provider can control access to data and application components, allow us to perform an authentication of potentially unreliable clients in the middle layer and does not affect the database level.

* Performance

The application server can meet the challenges of balancing network traffic and load balancing between other physical servers in the system.

* Total cost of ownership

The set of the above-mentioned advantages, in addition to the redistribution os equipment costs for the client-to-server communication, may lead to cost savings for the organization. Moreover, a software rental policy can affect total cost of an ownership. In fairness, it should be noted that the cost of the server software, as well as its implementation and maintenance costs could be quite high.

However, application servers have **disadvantages** such as following:

* Centralization

Systems that are based on the application server have one major flaw, which can be found in all centralized solutions: The server fall will lead to the unavailability of programs for all customers. In addition, the network connection failure will lead to the same effect.

* Data protection

This problem is relevant to any network solutions using public network infrastructure for data transmission.

# 5 planning the work

## 5.1 Phases of the project

I have learnt at the university and own projects that any development requires a planning process. Otherwise, the work will fail or will be completed not as it was mentioned from the begining.

First, I have never had working experience in web development. Therefore, I lack knowledge in this topic, despite the fact that we had related courses at the university and the fact that I invested my free time in this. I investigated theory needed for the practical part, but before starting the development I have to carry out more research on the exact development.

I plan to gain further knowledge from web sources such as forums, articles and blogs. The reason for this is actually developers’ community. In this world, currently, developers publish a lot of information and problem solutions by pieces before it can be summed up into a whole book. Moreover, people can freely ask for solutions from the live community. One example is StackOverFlow.com. Defining a task and finding solutions from the web sources would be working and learning processes for me.

Second, I will build my workspace. This process includes the installation of all the required programs and their configuration. I have to decide which technologies, frameworks, OS, a compiler and a text editor I need for my development. The compatibility of versions might be a problem, and therefor this step requires a lot of attention and information.

The third step will be the building of the applications. During this process I will spend a lot of time reading articles, forums and blogs to make everything work properly. There is a possibility that I will have to change an architecture or technologies used for my applications, due to lack of experience or a better solution.

The fourth step is to build the overall picture to analyze every process of development, provide easily accessible thesis results and to make a proper conclusion.

## 5.2 The Waterfall model

I have chosen waterfall model as my develoment methodology, and it can be introduced as follow:

Modern systems are so large and complex that their creation require an entire team of specialists from different fields: programmers, analysts, system administrators, testers and end users. They all work together to develop programs that contain millions of lines of code.

The oldest known model for multi-level development process is the Cascade or simply the Waterfall model. Every step of the development continues from the previous one. In order to move on, to a new stage, we must completely end the current.

The Cascade model is simple and clear, but not so practical as before. In a fast and dynamically changing requirements, a strictly structured process can become a hindrance to the successful completion of the system. Today, therefore, the waterfall model is mostly used by large companies for large and complex projects that require comprehensive risk control.

**The pros and cons of the Cascade model can be listed as follows:**

Advantages:

* Full documentation of each stage
* Accurate planning of time and costs
* Transparency of processes for the customer

Disadvantages:

* The need for the approval of the full scope of requirements for the system even at the first stage
* The need to change the requirements later will lead to a return to the first stage and re-conversion of all of the work done
* Increase in the cost and time needed in case necessary requirement changes have to be done

# 6 building the workspace

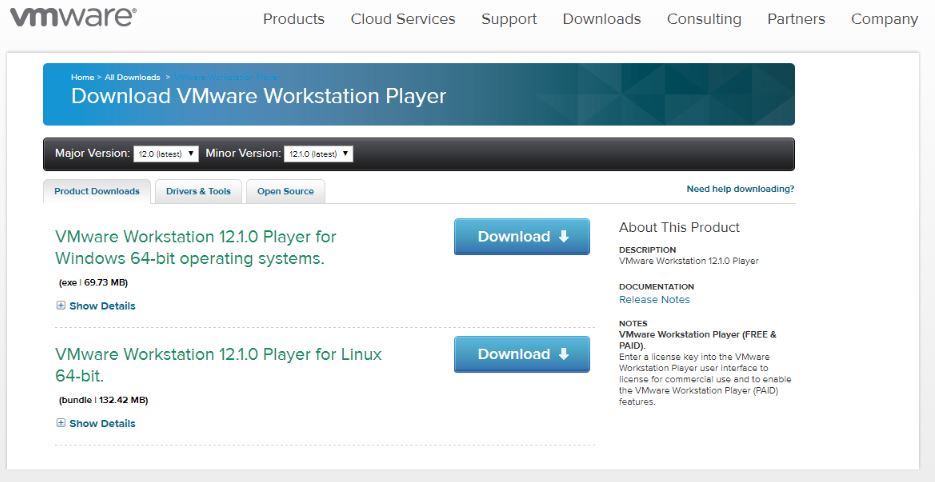
## 6.1 Installing the workspace

The combination of libraries and programs that is necessary to setup for the successful development of applications is what we call workspace. For my practical part I have to prepare a workspace for the client/server development. As I explained earlier, I am going to work with Java and JavaScript. However, I have decided that the operating system for my servers will be one of the Linux distributions (Ubuntu). I have good working experience related to Ubuntu and command line. It would be more comfortable to configure and launch everything by using Linux Terminal. Since my main OS is Windows, I am going to use the VMware Workstation software to make a virtual machine, where Linux will be installed. After Linux will be ready, I will install the Java development kit (later JDK), Integrated Development Environment (IDE) and the text editor. Moreover, the rest of the workspace will be ready during the next chapter, because I do not have sufficient knowledge about the requirements. This means that in this current chapter I will install and setup the basic components. Bear in mind, we must care about capability of products we install. If a person has the x32 system, then install x32 (or x86) products. I have Windows 10 x64, therefore I use the x64 products. Let me put this all on a list:

* VMware Workstation. ”VMware Workstation Pro transforms the way technical professionals develop, test, demonstrate and deploy software by running multiple x86-based operating systems simultaneously on the same PC” (VMware Workstation, 2016)
* Ubuntu, Linux distribution – popular operating system, best for running a server
* Komodo Edit, smaller version of Komodo IDE, it is text editor with limited functions of IDE
* Java Development Kit, package necessary for development and launching of Java applications
* Intellij IDEA, the Java IDE, compiler and tools

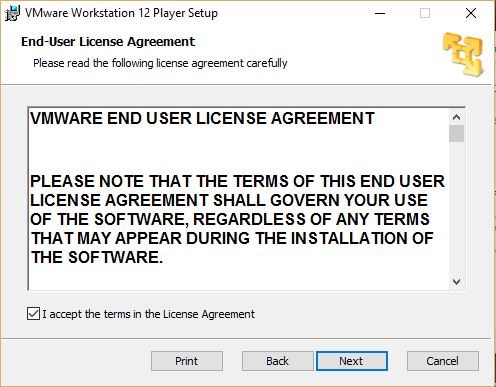
### 6.1.1 VMware and Ubuntu

Next step, we start from downloading the VMware software, as we can see in Figure 9. The Workstation version is free for non-commercial use.



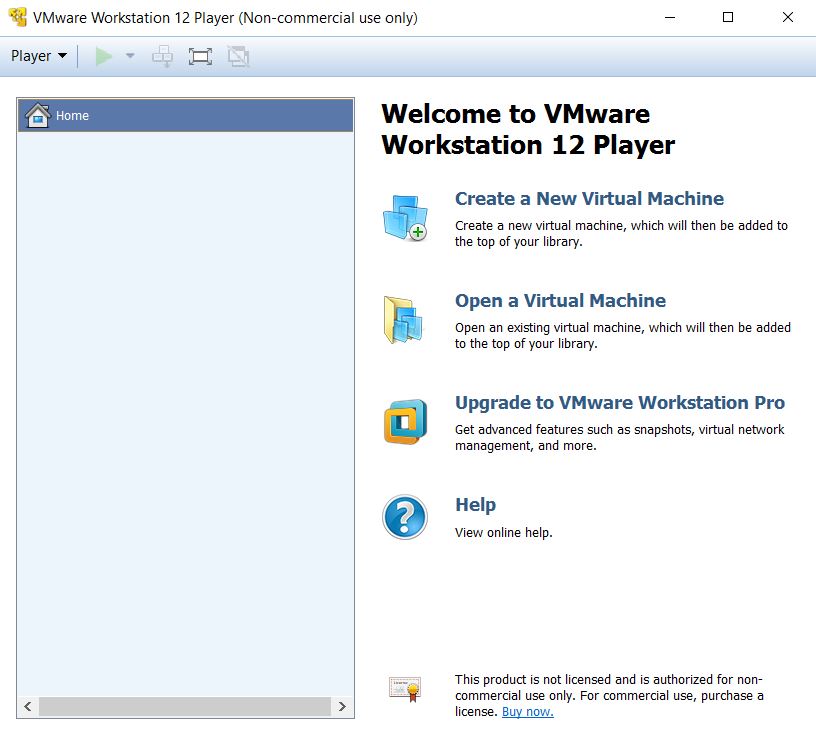
**FIGURE 9. The VMware download page**

After downloading, during the installation we need to accept the agreement shown in Figure 10.



**FIGURE 10. The VMware agreement**

We have our software installed and now we are ready to create a virtual machine with Ubuntu on it (Figure 11).

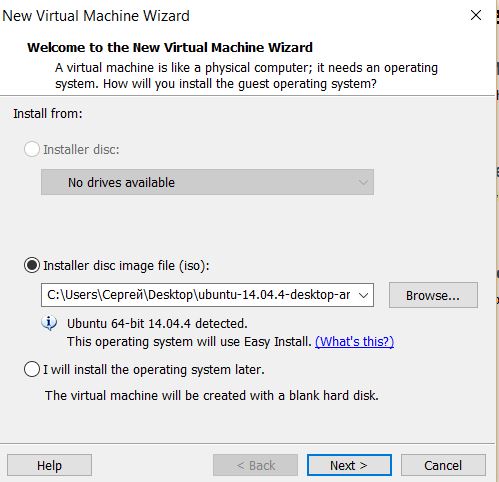


**FIGURE 11. Create a New Virtual Machine**



**FIGURE 12. The Ubuntu download page**

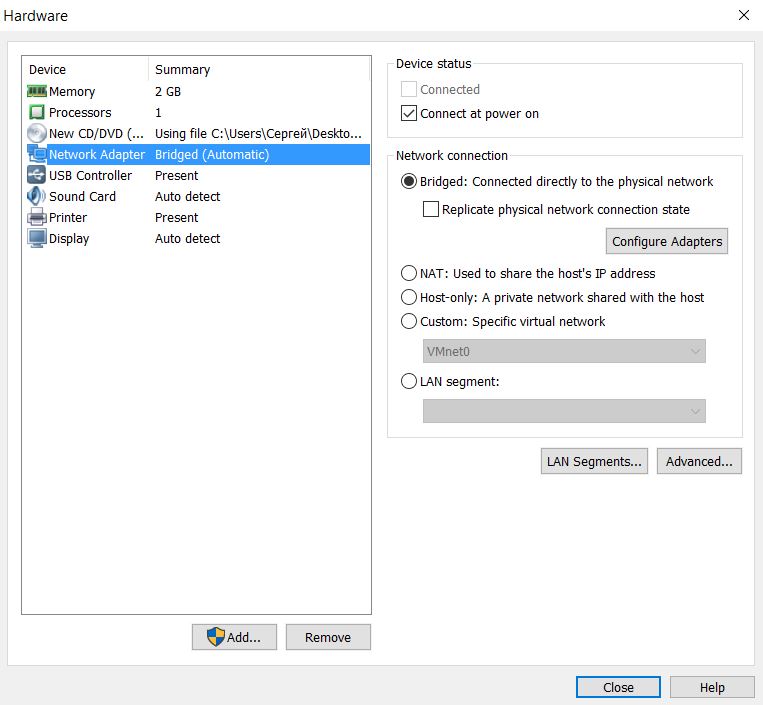
The process of installation begins, and there are a couple of settings we need to change. First, VM will ask to choose the ISO file with the Ubuntu version. We can download it for free from the Ubuntu official website (Figure 12). In Figure 13, we choose our ISO file with Ubuntu.



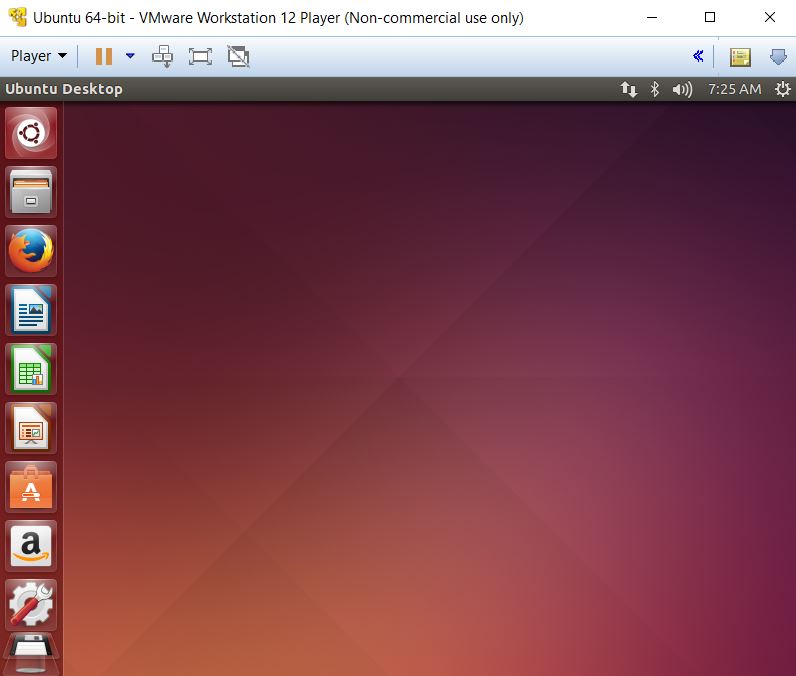
**FIGURE 13. The disc image file**

The next important thing to remember is to change networking parameters to have the internet on our virtual machine. To do this we need to go to the Network Adapter parameters. Then in the network connection section, we need change the radio button selection to the Bridged mode as we can see in Figure 14. The bridged mode means that our Virtual Machine will have straight access to the physical network from the main OS.

After all, the Ubuntu installation begins. It takes a couple of minutes to install. However, during installations we setup the username, the name of the PC, passwords and personal configurations. Finally, we have a virtual machine with Ubuntu installed on it (Figure 15).



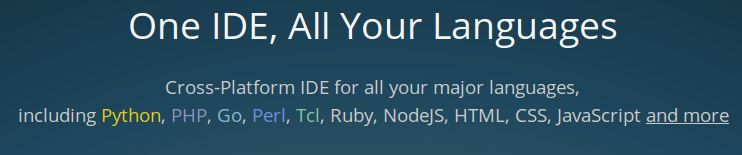
**FIGURE 14. The bridged mode option**



**FIGURE 15. Ready Vmware and Ubuntu**

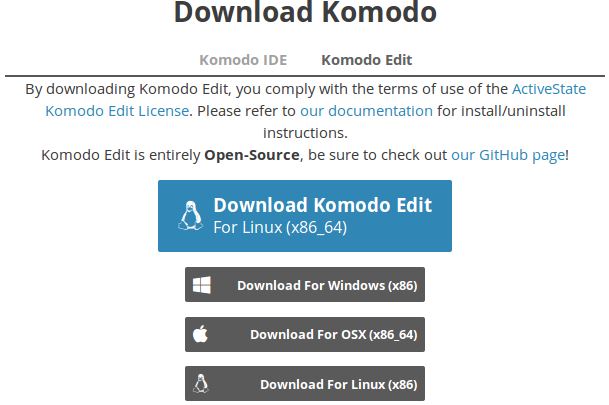
### 6.1.2 Komodo Edit

As it was mentioned before, JavaScript does not require a compiler or an external interpreter. Actually, everything we need for the web development with such technologies as HTML, PHP, CSS and JavaScript is just a text editor. Of course, that is not practical and there are many professional editors, which were designed for the web development. My preference is Komodo Edit. This version is free for community but has limited functions compared to Komodo IDE (Figure 16), which requires a license. However, even the lite edition has everything I need for the further development.



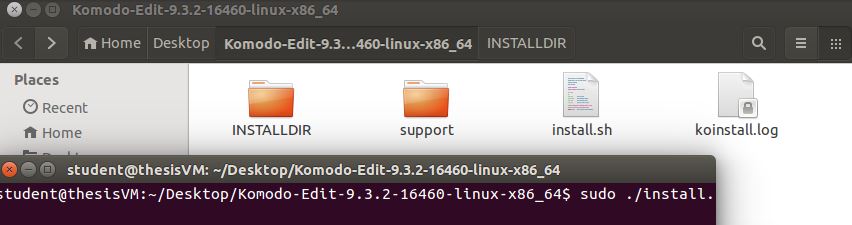
**FIGURE 16. The Komodo IDE intro**

We can download Komodo Edit from the official webpage. We can choose the suitable vesrion and again we need to know what kind of a system we have. In my case, I will download the x64 version for Linux (Figure 17).



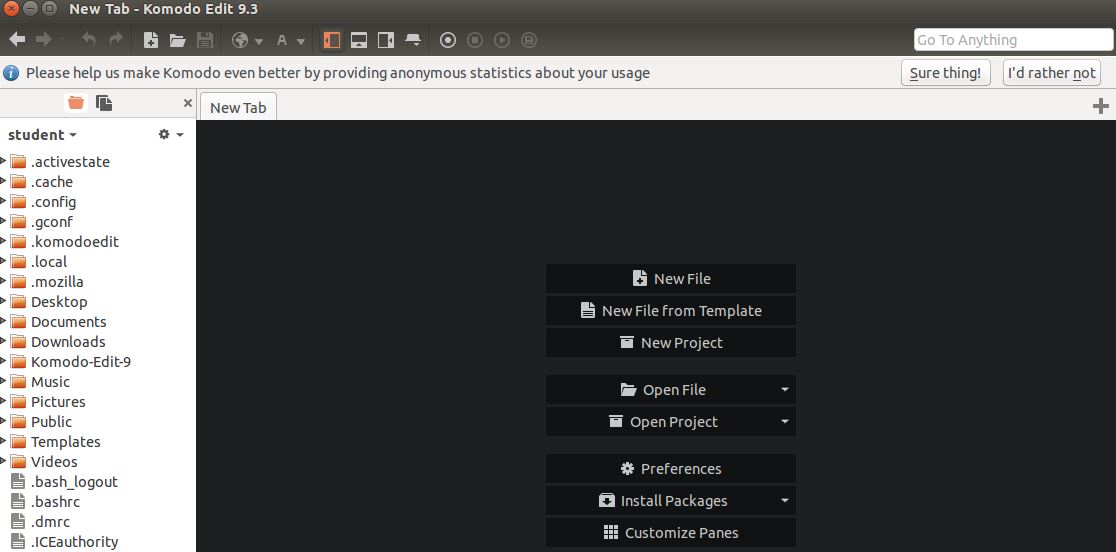
**FIGURE 17. The Komodo Edit download page**

After downloading, we have a tar.gz file extension, a compressed file. To install Komodo we need to unzip-decompress the file into an empty folder/directory. Next step, will be done from the command line. We go to the created directory with command: cd (change directory) and execute a script *install.sh*. We can see this process in Figure 18.



**FIGURE 18. Install Komodo from the command line**

Now, we can launch our new software (Figure 19).



**FIGURE 19. Komodo is installed**

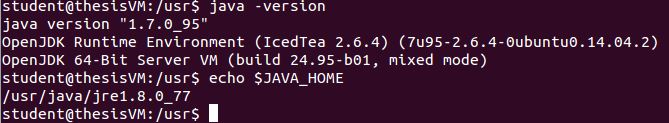
### 6.1.3 Java Development Kit

For Linux, there are two platform versions: Open JDK and Oracle JDK. The difference is in the way how both are supported. Open JDK is supported by community since it was released. The Orcle version is supported by the Oracle company. The difference in technologies is not that noticeable, especially for small groups of developers. However, there is a disadvantage for the Oracle version. The problem is that Oracle distributes Java Environment by themselves and do not rely on other companies. In this case, it is easier and more comfortable to install Open JDK. To do so we just need only one command typed in the terminal as shown in Figure 20.



**FIGURE 20. The command to install OpenJDK**

In Figure 21, I check the version of Java to understand that my operating system can see JDK and that it was installed successfully.



**FIGURE 21. Check the Java version**

The last thing here would be the variable path configuration. We need this step, because sometimes a software needs to know where to find the Java machine. I setup up PATH in Figure 22, also with a command line.

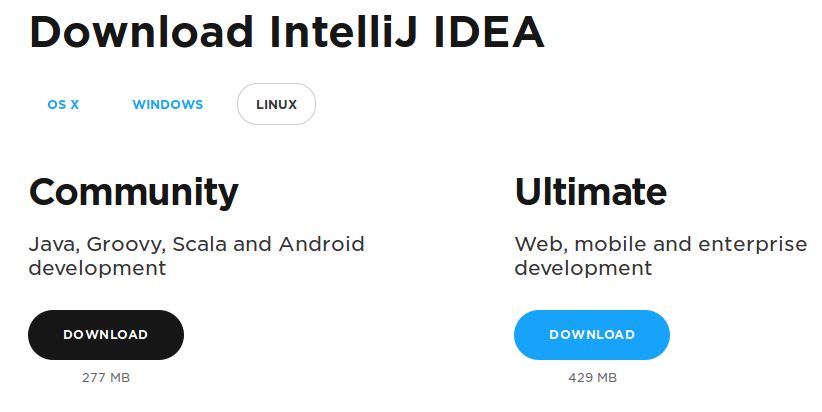


**FIGURE 22. Setting the JAVA\_HOME path**

### 6.1.4 Intellij IDEA

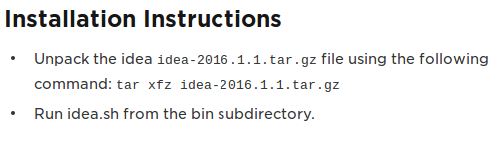
Intellij IDEA is my favorite Java IDE. I have spent a lot of time working with it. This is the obvious choice for me. Eclipse and NetBeans are also very popular and competitive.

It is possible to download IDEA from the official webpage (Figure 23). There are two versions of it what is basic scenario. One version is limited but free, another require paid license. I personally use the community version, which is free. For my thesis, the community version is enough.



**FIGURE 23. The IDEA download page**

On the download page there is an installation instruction for Linux (Figure 24). The process is similar to the Komodo installation (see 6.1.2 Figure 18)



**FIGURE 24. Installation instructions for Linux**

The installation window will appear with basic simple configurations. Moreover, we need to accept the agreement. Finally, IDEA is ready (Figure 25).



**FIGURE 25. Idea is ready**

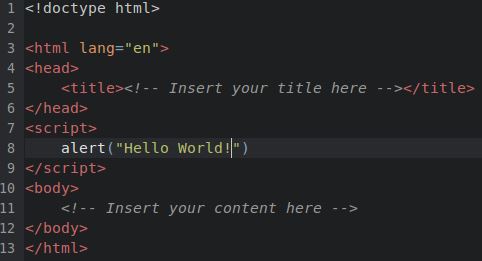
## 6.2 Test the workspace

Now, after all installations I need to test Komodo Edit, JDK and IDEA.

**Komodo Test**

I will start with Komodo Edit. I will build a simple webpage with the basic JavaScript alert function:

* Start the Komodo Edit
* It will ask how we would like to begin our project. Choose *New File from template*
* As a template, we choose *HTML* and type the name for this file. After that, Komodo will create the HTML file
* Basic code structure will be placed in the new file
* Before the *<body>* tag, place JavaScript with function *alert(“HelloWorld!)* (Figure 26)
* Final step, just simply open the HTML file (Figure 27)



**FIGURE 26. The alert function**

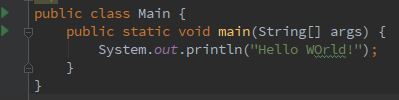


**FIGURE 27. The “Hello World” alert**

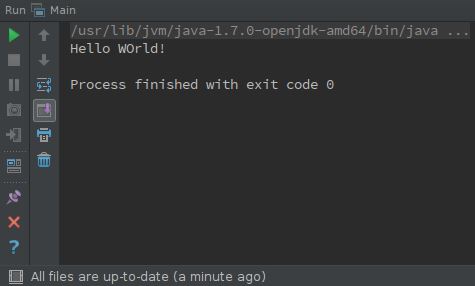
**IDEA and JDK**

To test JDK and IDEA we need to build a project with IDE:

* Go to *File – New - Project*
* As a project SDK we need to choose installed JDK, press *Next - Next*
* Type a project name and choose place for it
* Project files generated, now go to *File – New – Java Class*
* Type a name for the class
* Now, we need to code a little bit to print “Hello World” to the console
* Make *Public static void main* method and in this method call a function *System.out.println(“HelloWorld”*) (Figure 28)
* Then, go to *Run – Run “Class name”* and get a console message “Hello World!” (Figure 29)



**FIGURE 28. The println function**



**FIGURE 29. “Hello world” is in the console**

In this chapter I downloaded all the needed software, installed and configuried it. Before we set up the development environment, we need to collect the information about our operation system. All these steps are necessary for the actual process of application development.

# 7 building applications

In this chapter, I will build two web applications based on the client-server technology. The idea for web applications will be a web chat, because the target of my thesis is to compare two popular languages in the server-side environment. Therefore, it would be a good practice to learn the weaknesses and strengths of languages. The web chat includes building the client, building the server, and setting up communication between the client and server. The communication is the most interesting part, because we make two completely different technologies work together.

For both, Java and JavaScript application, I will build a client based on the web technologies such as HTML, CSS and JavaScript. The reason for this is the world trend. Currently, the market is separated as follows: Java is good for the server-side and JavaScript for the client-side. We are still able to find web clients based on Java, using the applets technology. However, this approach is too old, and all progressive developers prefer JavaScript for this. Therefore, there are many talks about JavaScript on the server-side, because it would be a beautiful approach to build the whole application with one language. During my practical part, I will gain experience from development with both technologies following the current world trend. The web chat can have many features. I will introduce the basic features first. For users, the basic functionality would be account creation, messaging, chat logs and nicknames.

I will explain the code briefly. However, do not forget to read comments on the pictures. Comments start with the symbols “//”.

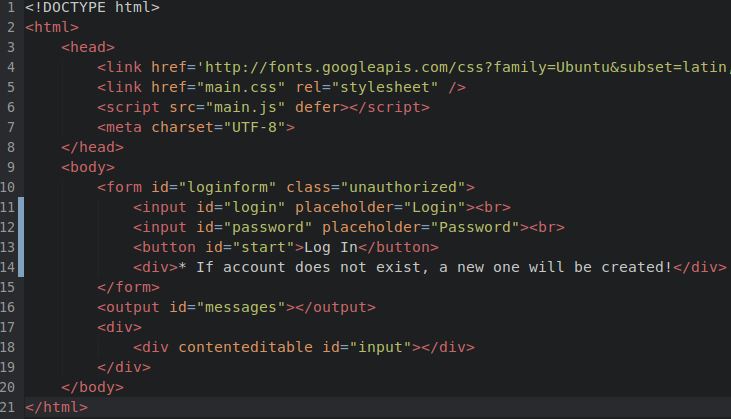
## 7.1 Building the client

The client for both applications is a webpage, based on HTML, CSS and JavaScript.

### 7.1.1 The HTML page

We create a simple html file using Komodo function – “Create file from a template”. To the code that was created by the template, we need to add following:

* Link a css file to the page using html tag <link href=”main.css”></link>
* Link our client logic script file to the page using the HTML tag <script src="main.js" defer></script>. We also put the “defer” attribute here to make sure that the script in will load after the page
* Create a basic login form as in Figure 30. Here we add input boxes for the login, password and a button. Notice that our *FORM* has *class=”unauthorized”,* it will be used later as logic to hide the *input* *DIV* element
* Add the *<output id=”messages”></output>* tag, which will be a container for all chat messages
* Last thing is the *input* DIV element. Later, with CSS I will hide it before the moment when user will be authorized



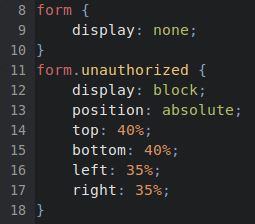
**FIGURE 30. The HTML page of the application**

### 7.1.2 Hide elements with CSS

The basic login logic for the application is to show necessary elements to users. During, the authorization we need to see login and password inputs with the button. We do not need chat elements before the authorization, because it will be confusing. After the authorization, we need to hide the authorization form and show chat elements.

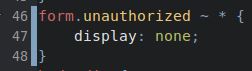
To do this we need to specify some attributes in the CSS file:

* First, in Figure 31, there are two “form” selectors. One of them with the “unauthorized” class selector. Another has the display property with the “none” value. The logic here, all forms on the page with the “unauthorized” class will be shown; all forms without this class will be hidden by the display property.



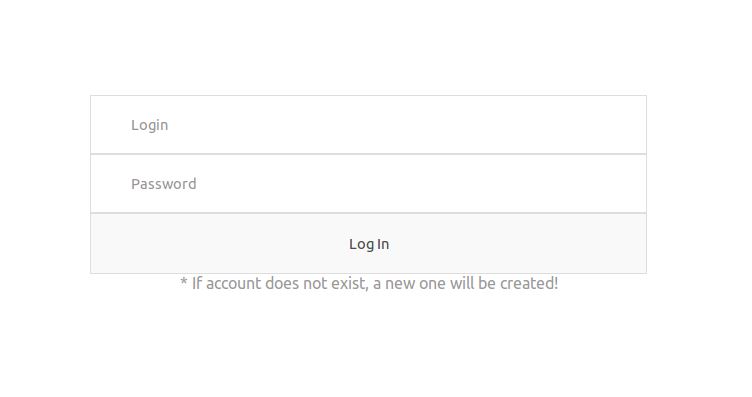
**FIGURE 31. CSS form settings**

* Second, in Figure 32, I hide all siblings of the form with the “unauthorized” class. The “Siblings” word means all elements under one parent element. In Figure 36, we can see that “output” and “input” DIVs are siblings of the main form.



**FIGURE 32. Hide all form’s siblings**

As a result, when the basic form has the class specified as “unauthorized”, our “output” and “input” DIVs are not displayed. When I will delete this class value with our script, elements will be swapped. The rest of the CSS file is just view settings. In Figure 33, we can see the login form in action.



**FIGURE 33. The login form in action**

### 7.1.3 The client’s JavaScript

To make the code simple I will substitute the code *document.getElementById* with the *$* symbol (Figure 34).



**FIGURE 34. Reducing the code**

Next, we open the connection to the server and wait for incoming messages as shown in Figure 35.



**FIGURE 35. Create the connection**

We create the connection, in Figure 35, using WebSocket with parameters: address “ws://localhost” and port “9000”. Then, we start to wait for a message from the server with the *ws.onmessage* callback. After we get the message (we parse it into usable object), the function with the *message* parameter will be executed. I define the function immediately. The function will check the event type and choose what to do.

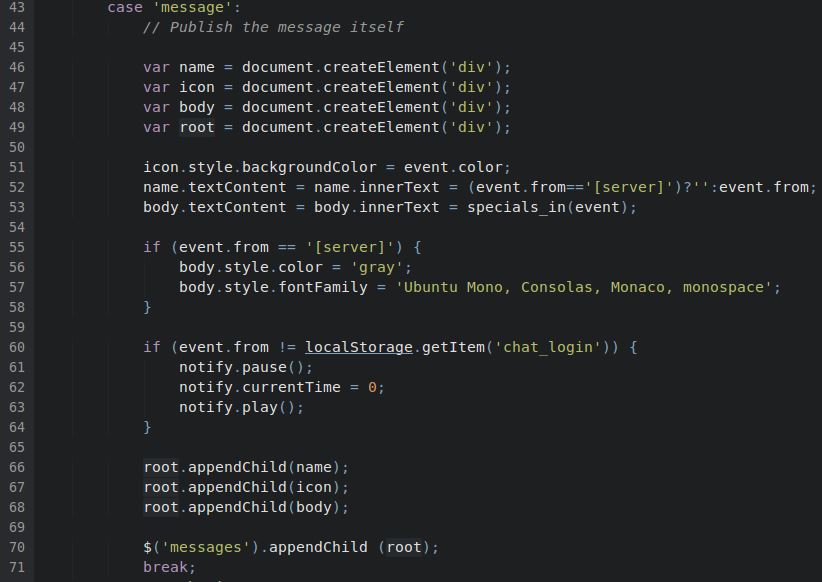
When the client gets the “message” event, it publishes the message that we get from the server. To show the message on the screen, the script creates four *div* elements, as shown in Figure 36:

* *name div* will contain login/nickname of the user
* *icon* will contain color information
* *body* will contain the message itself
* *root* will be a container for above-mentioned DIVs

Then, the script changes:

* icon’s background color for a color from the message object
* *name.textContent* for the message’s *from* variable, if the message is from the server nickname div will be empty
* *body.textContent* for the message’s *message* variable

For the server’s system messages the script changes font color and font family for the BODY element. To publish elements I use the *appendChild* function to put three div elements into one *root* element. *Root* is the main div element that contains the whole message. Final step here is to use the same function to append *root* straight to the <output> element with id=”messages” on the HTML page.

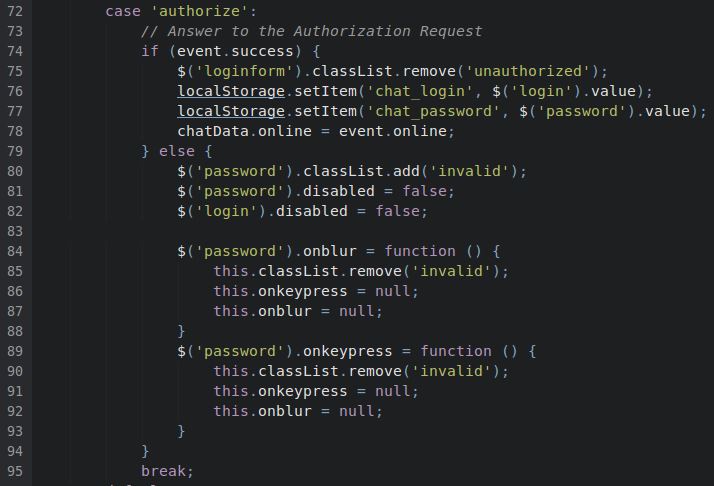


**FIGURE 36. Get the “message” event and publish the message**

When the client gets the “authorize” event, the script checks if the authorization was successful, as shown in Figure 37. If the authorization was successful:

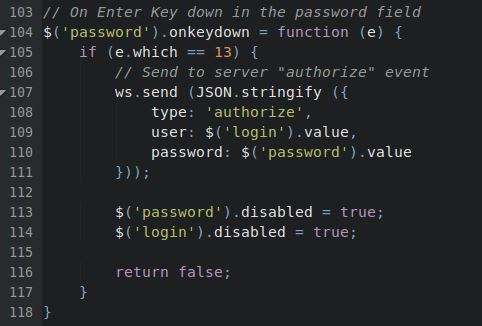
* the script removes the “unauthorized” class from the main form on the HTML page, this will remove the form from the screen and let <output> and input elements appear
* the script puts information about the login and password into the HTML5 local storage, using variables: *chat\_login* and *chat\_password*

If the authorization was not successful, the script will let us know and ask to try again.

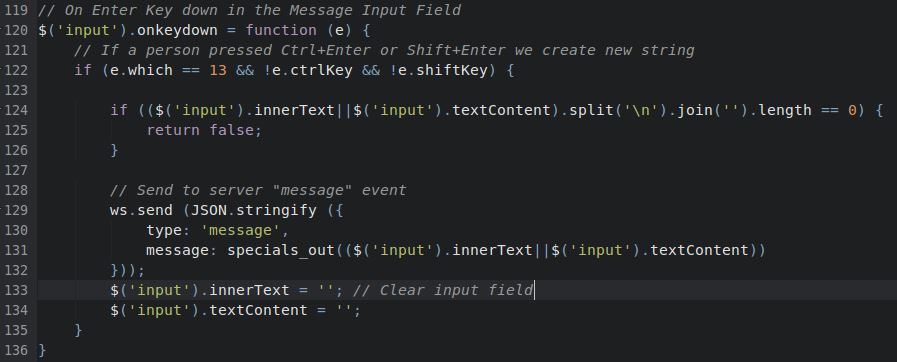


**FIGURE 37. The authorization event**

In Figure 38 and 39, there are functions, which send the data to the server. The data is presented as the JSON object. For the authorization request, the object contains variables: *type*, *user* and *password*. For the message, the object contains variables: *type* and *message*. Pressing “Enter” will send objects to the server, where based on the “type” variable the server will react. Currently, the chat has two message types: “authorize” and “message”.



**FIGURE 38. Send the “authorize” event to the server**



**FIGURE 39. Send the “message” event to the server**

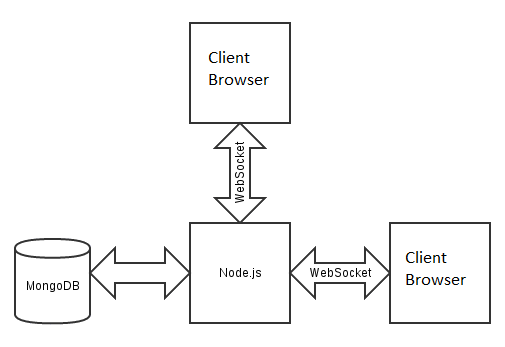
## 7.2 Building the JavaScript server

The very first application will be the JavaScript server with key features listed below:

* Saving messages in the database
* An authorization
* The connection to the client via WebSocket
* Messaging
* The online list

In Figure 40, we can see the architecture of the application. Let me put all on a list:

* **MongoDB** is a database for the server
* **Node.js** is ”an open-source, cross-platform runtime environment for developing server-side web applications” (Wikipedia 2016b)
* **WebSocket** is full-duplex communication protocol over the TCP-connection for exchanging messages between the browser and web server in real time.
* **JSON** is ”short for JavaScript Object Notation, and is a way to store information in an organized, easy-to-access manner. In a nutshell, it gives us a human-readable collection of data that we can access in a really logical manner” (Lengstorf 2009).
* **Browsers** (Opera, Firefox and Chrome)



**FIGURE 40. The JavaScript server architecture**

The list of files and the brief explanation:

* main.js, this file contains the client-side logic and connected in the main html file
* server.js is the Node.js script for the server, we launch it from the Linux terminal
* main.html is the main page, start point for the user
* main.css is the style of the webpage

### 7.2.1 Installations

Before we start development, we have to install NodeJS, MongoDB and dependencies. Dependencies make it possible for technologies to work together and use each other. We have installed Ubuntu before. Therefore it will not be a difficult task. Everything would be done from the command line.

* Installing NodeJS with the command shown in Figure 41



**FIGURE 41. The command to install NodeJS**

* Installing MongoDB with the command shown in Figure 42



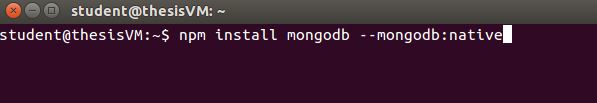
**FIGURE 42. The command to install MongoDB**

* Installing NPM, the package manager from NodeJS. Helps to setup dependencies (Figure 43)



**FIGURE 43. The command to install NPM**

* Installing MongoDB drivers for NodeJS using NPM with the command shown in Figure 44



**FIGURE 44. The command to install the MongoDB dependence**

* Installing WebSocket drivers for NodeJS using NPM with the command shown in Figure 45



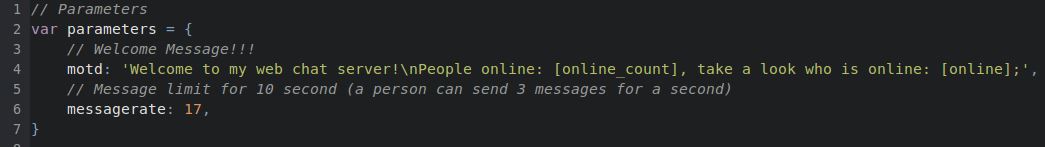
**FIGURE 45. The command to install the WS dependence**

Last installations have been done and now we are ready to proceed to the development process.

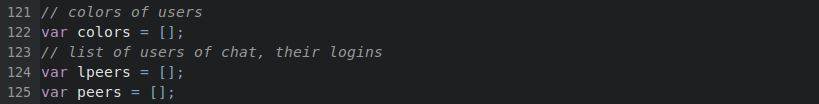
### 7.2.2 The server code

* **Create variables and a welcome message**

First, I will create basic variables for arrays (Figure 47) and the welcome message (Figure 46). Arrays will contain the list of logins, names and colors of users.



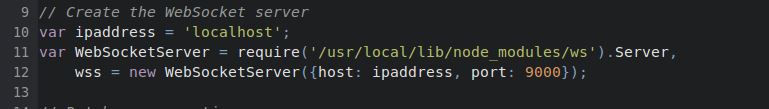
**FIGURE 46. The welcome message creation**



**FIGURE 47. The creation of arrays**

* **Create the WebSocket server on port 9000 (Figure 48)**

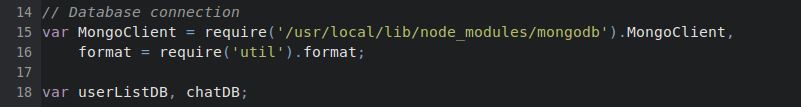
The address is localhost because I use the local machine. We need also specify the path to WebSocket dependencies, which were installed before.



**FIGURE 48. The WS server creation**

* **Create the database connection (Figure 49)**

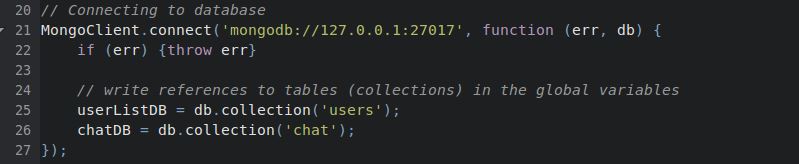
Specify the path to MongoDB dependencies, which were installed before. Also, create local global variables *userListDB* and *chatBD.*



**FIGURE 49. The Database connection creation**

* **Connecting to the database (Figure 50)**

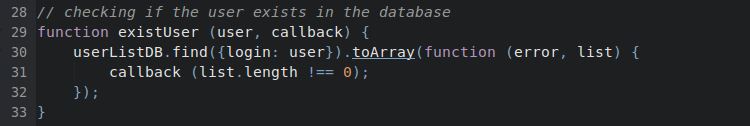
Connect to the database using created connection from Figure 49. Place references to database collections into global variables *userListDB* and *chatBD.*



**FIGURE 50. Connecting to the database**

* **Function: existUser (Figure 51)**

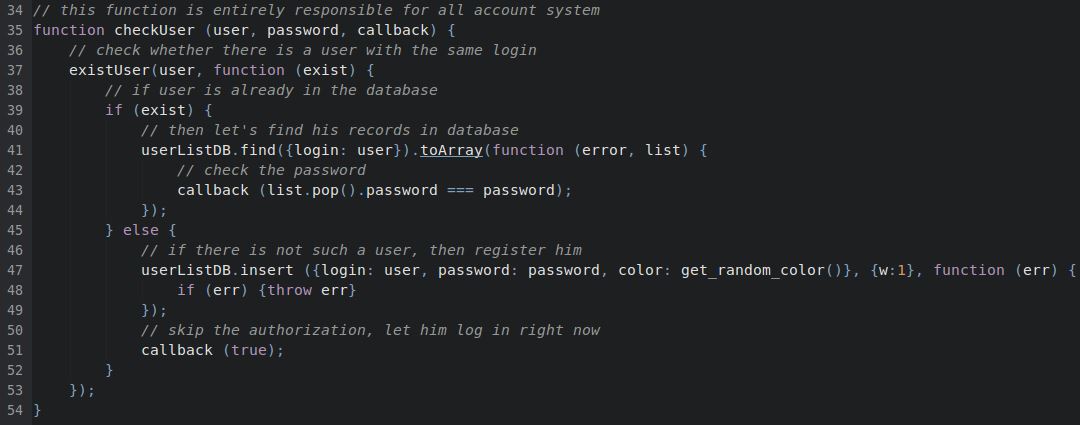
Function checks if the given user is in the database.



**FIGURE 51. Function: existUser**

* **Function: checkUser (Figure 52)**

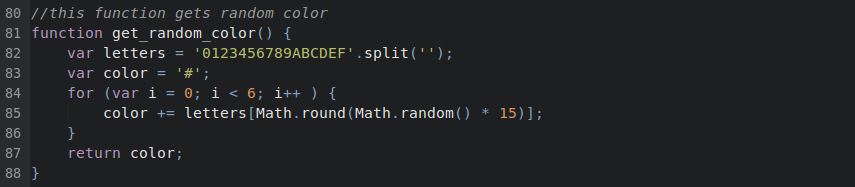
The logic in this function is responsible for the entire account system. It checks if the user in the database, finds his data, checks the password, registers the user and lets him log in.



**FIGURE 52. Function: checkUser**

* **Function: get\_random\_color (Figure 53)**

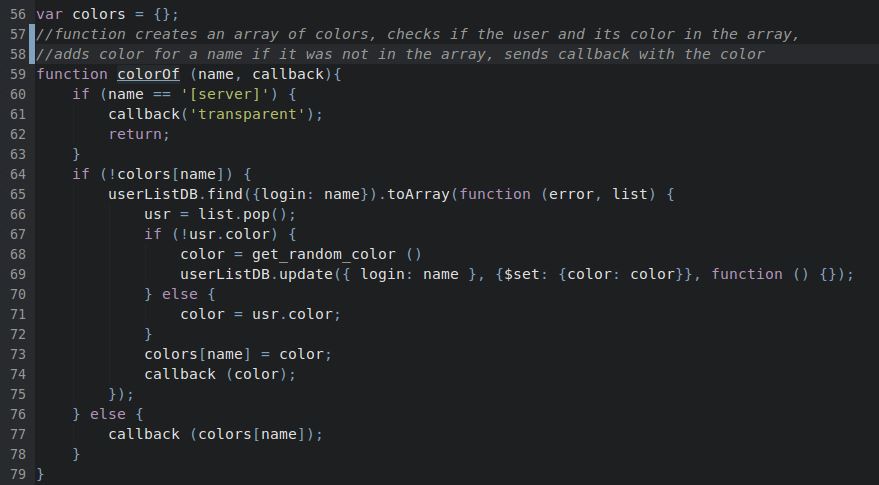
The name of this function speaks for itself. It generates a random color.



**FIGURE 53. Function: get\_random\_color**

* **Function: colorOf (Figure 54)**

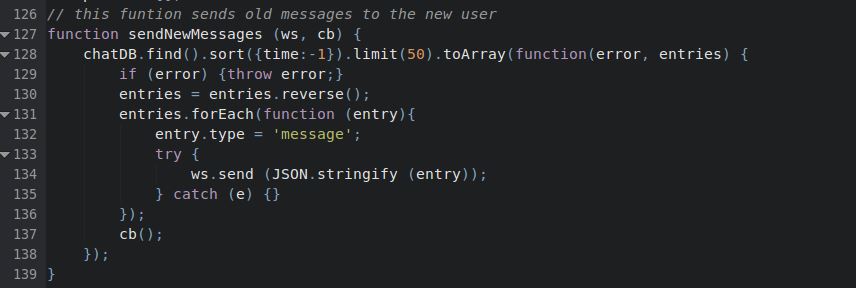
The function creates an array of colors, checks if the user and its color are in the array, adds the color in the array, and sends a callback with the color.



**FIGURE 54. Function: colorOf**

* **Function: sendNewMessages (Figure 55)**

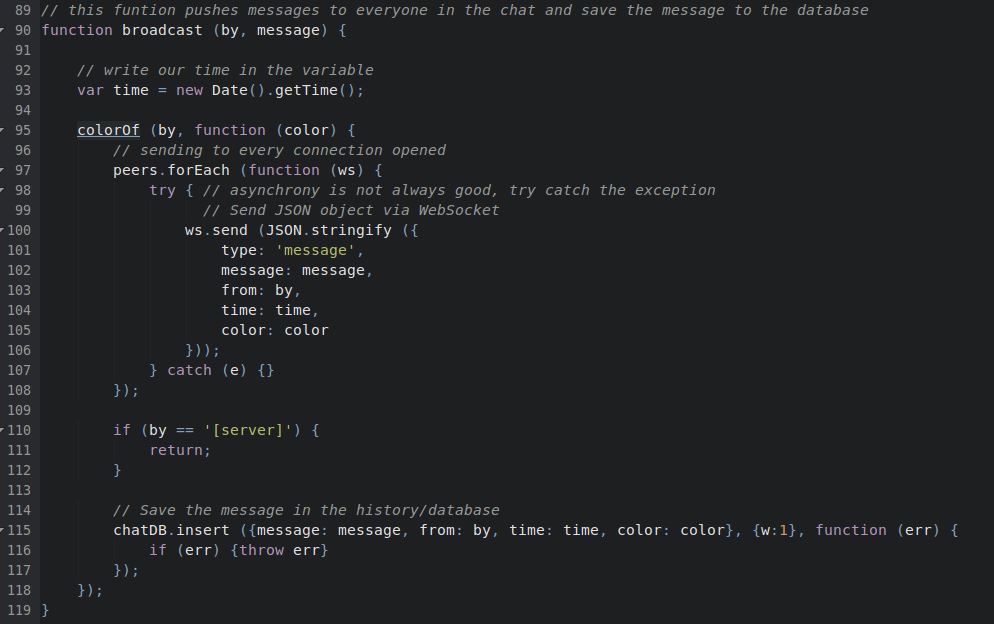
The function sends old messages to new chat participants.



**FIGURE 55. Function: sendNewMessages**

* **Function: broadcast (Figure 56)**

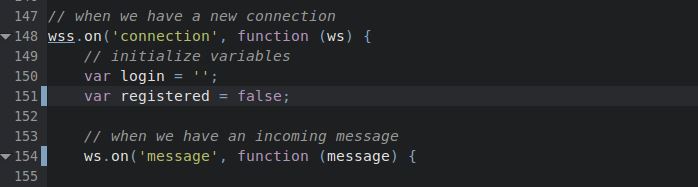
The function sends messages as a JSON object via WebSocket to all chat participants and save it to the database/history.



**FIGURE 56. Function: broadcast**

* **Logic: When we have a new connection**

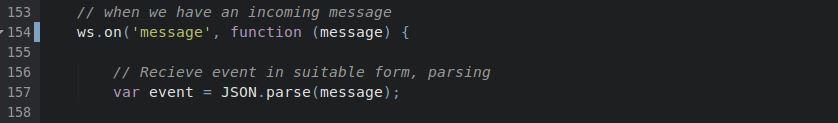
When the server gets a new connection, it starts an anonymous function. This function initializes new local variables and proceed to the next step (Figure 57).



**FIGURE 57. The server gets a new connection**

* **Logic: When we have an incoming message**

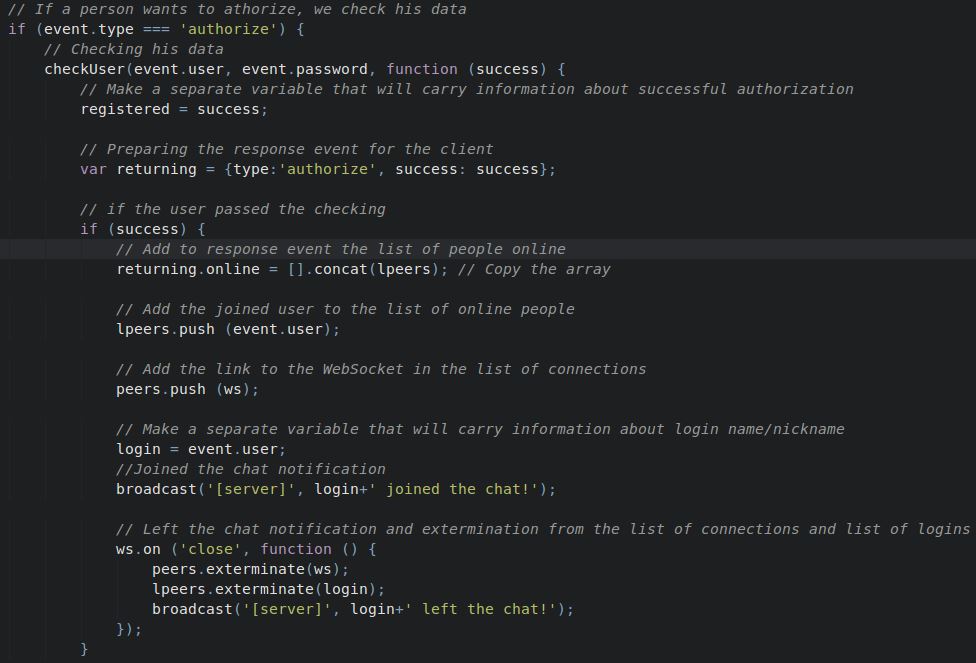
When the server gets a new message, it starts an anonymous function. This function prepares message, transforming it into a suitable form by the *JSON.parse* method (Figure 58). Then, the function proceeds to the next step, where our server reacts to a message.



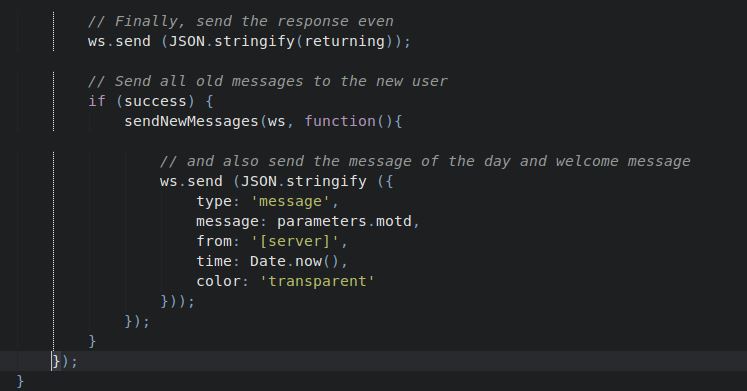
**FIGURE 58. The server gets a new message**

* **Logic: message type – “authorize” (Figure 59)**

If the message type is “authorize”, the server uses *checkUser* function to check user’s data and prepares a response message for the client (*returning* variable). If the user passes, the server adds the list of people online to the response message, the joined user to the list of people online, and the link to the WebSocket to the list of connections. The server broadcasts “Joined the chat” or “Left the chat” notifications. In the end, the server finally sends the “authorize” response event with the authorization success, the list of people online, old messages and the message of the day (Figure 60).



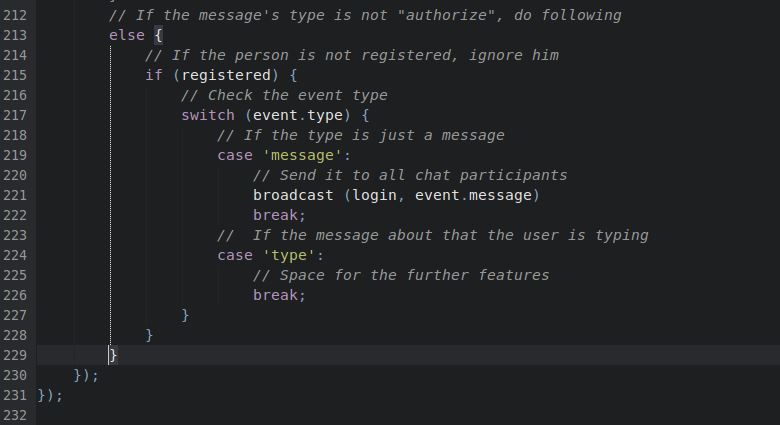
**FIGURE 59. The server reacts to “authorize” message type**



**FIGURE 60. Sending the authorization response**

* **Logic: message type is not “authorize” (Figure 61)**

Here, the server checks if the user, who sent the message, is registered. Then, based on message’s type the server performs actions. This support of many message types provides multiple features. However, currently my web application has only one message type. After the server gets this message, it sends information to all chat participants using the *broadcast* function.

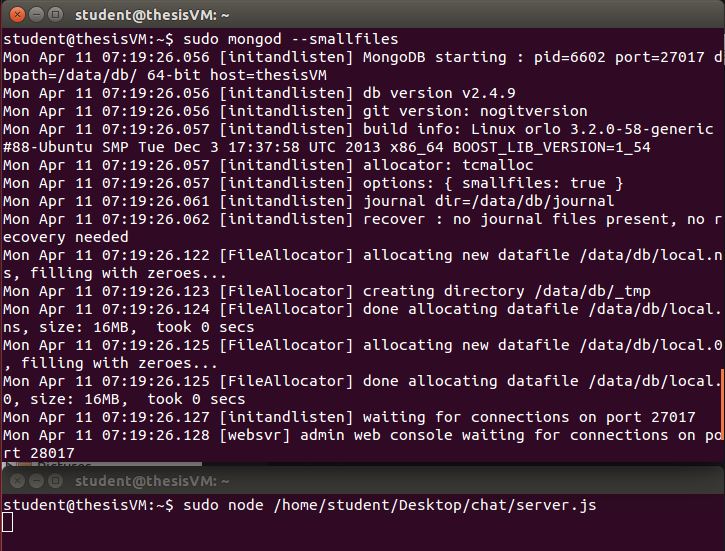


**FIGURE 61. The server reacts to the non-authorize message**

### 7.2.3 Launch and test

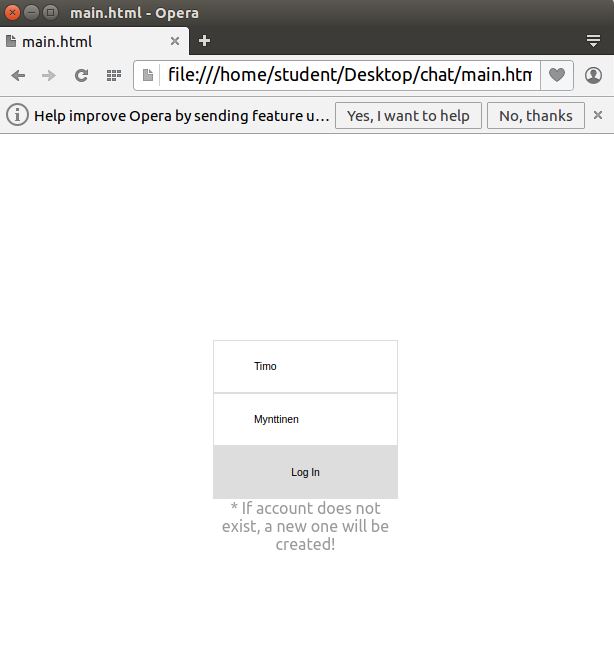
At this point, we have everything we need to test the web application. As clients, I am using three browsers: Opera, Chrome and Firefox. To start the application we need to do following:

* Use the command line *sudo mongod –smallfiles* from the terminal to launch the MongoDB database (Figure 62)
* Use the command line *sudo node server.js (or specify the path the o server.js file)* from terminal to launch the Node.js server (Figure 62)
* Open *the main.html* file in browsers



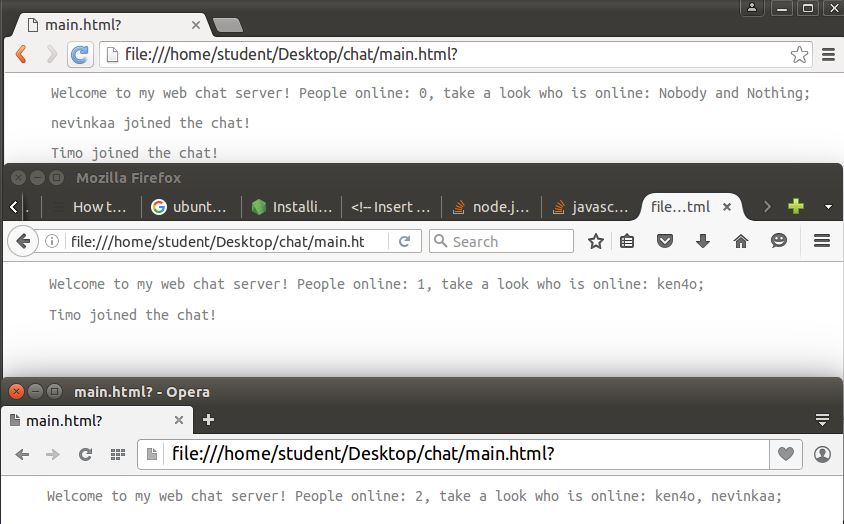
**FIGURE 62. Launching MongoDB and Node.js**

Figure 63, shows the authorization page.

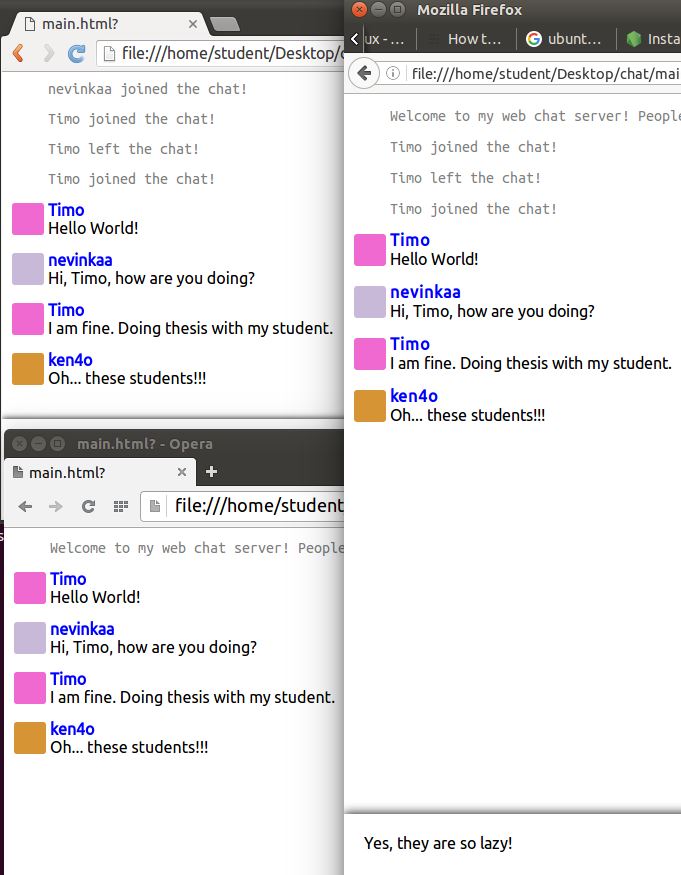


**FIGURE 63. Authorization page**

In Figure 64, there are three browsers right after the authorization. The last browser shows that two people are online: “ken4o” and “nevinkaa”. The first browser shows information when the users *Timo* and *nevinkaa* joined the chat.



**FIGURE 64. Browsers right after the authorization**



**FIGURE 65. The chat in action**

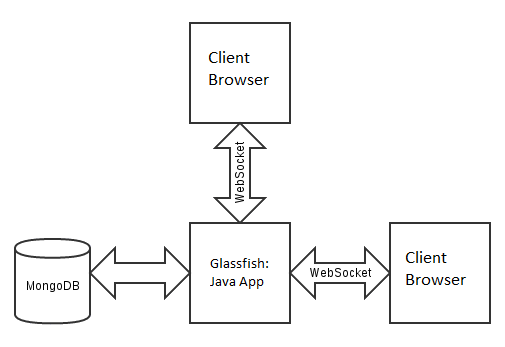
Now, we can send messages between browsers via the WebSocket connection. The data is processed on the Node.js server by the server.js script. The example of the message exchange is shown in Figure 66.

## 7.3 Building the Java server

The second application will be the Java server. The key features are the same as it was for the JavaScript server (see 7.2 section).

In Figure 66, we can see the architecture of my application. Let me put all on a list:

* **MongoDB** is a database for the server
* **Glassfish** is an open-source application server
* **WebSocket** is a full-duplex communication protocol over the TCP-connection for exchanging messages between a browser and a web server in real time
* **JSON** is ”short for JavaScript Object Notation, and is a way to store information in an organized, easy-to-access manner. In a nutshell, it gives us a human-readable collection of data that we can access in a really logical manner” (Lengstorf 2009)
* **Browsers** (Opera, Firefox and Chrome)
* **Apache Maven 3** is a tool which simplifies and manages a Java project build process: compilation, creating the jar, creating a software distribution



**FIGURE 66. The Java server Architecture**

The list of files and brief explanation:

* main.js file contains the client-side logic
* index.html is the main page
* main.css is the style of the webpage
* pom.xml is “an XML file that contains information about the project and configuration details used by Maven to build the project” (Apache 2016)
* ChatEndpoint.class is the basic class with the server logic

### 7.3.1 Installations

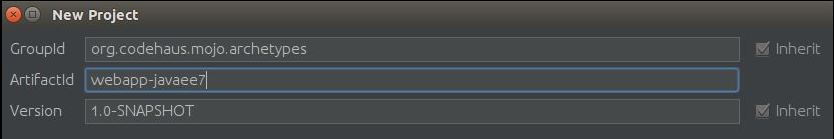
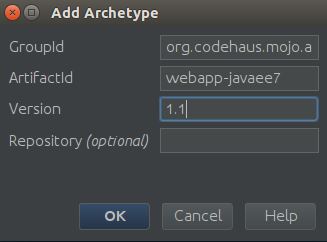
Previously, I have installed IDE, OpenJDK and MongoDB. However, for this project, I am still missing Maven 3. With Maven, I can easily install the MongoDB driver, BSON (Binary version of JSON), Glassfish and other necessary libraries. In Figure 67, we can see a command line, which installs the latest Maven version.



**FIGURE 67. The command to install maven**

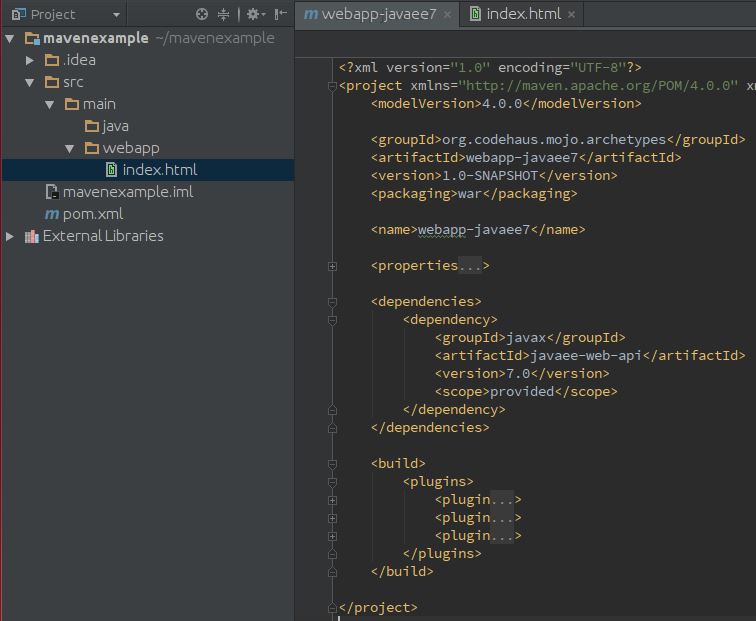
To continue the installation process, I need to generate the pom.xml file with Maven.

* Open Intellij IDEA
* Go to *File* – *New* – *Project*
* At left side bar, select Maven
* Press *Add Archetype*
* Type GroupID and ArtifactID for “webapp-javaee7” archetype, as shown in Figure 68, press *OK*
* As a project SDK choose installed JDK, press *Next*
* Type GroupID and ArtifactID for “webapp-javaee7” archetype, as shown in Figure 68, press *Next - Next*
* Type project name and specify project location, press *Finish*
* Project files are generated



**FIGURE 68. GroupID and ArtifactID**

Now we have the *pom.xml* file, and directories for Java classes and the webpage. Generated files are shown in Figure 69. Moreover, xml file is opened in the figure.

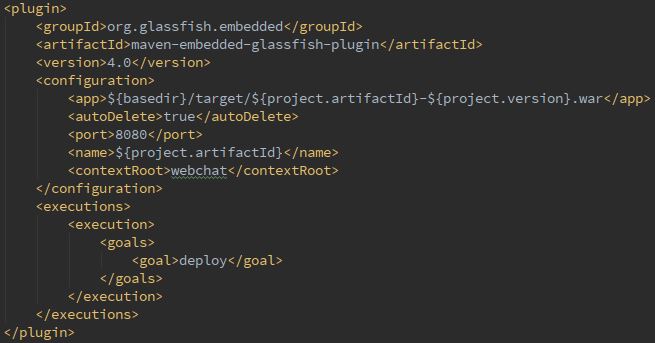


**FIGURE 69. Generated files**



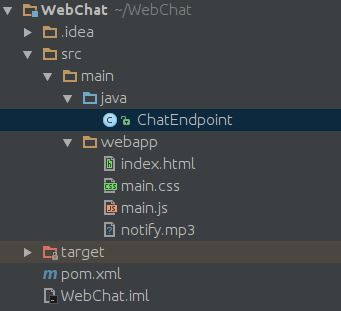
**FIGURE 70. Additional dependencies**

Notice, that *pom* contains tags: *<dependencies>* and *<plugins>*. By using these tags, we can install new dependencies and plugins. This will be my next step. In addition to the *javaee-web-api* dependency, I have added BSON, the MongoDB driver and JSON dependencies, as shown in Figure 70. MongoDB require BSON to work properly. The JSON dependency simplifies processing of JSON messages. I also use the *maven-embedded-glassfish* plugin (Figure 71), which allows us to use Glassfish without downloading/installing.



**FIGURE 71. The Glassfish plugin**

To finish this section we need to place all client files, which were created earlier, in the *webapp* directory. Notice that we have to rename the *main.html* file into *index.html*. Create new Java class in *java* directory. I called it *ChatEndpoint.* The class will contain the entire server logic. In Figure 72, we can see directories and files of my project.



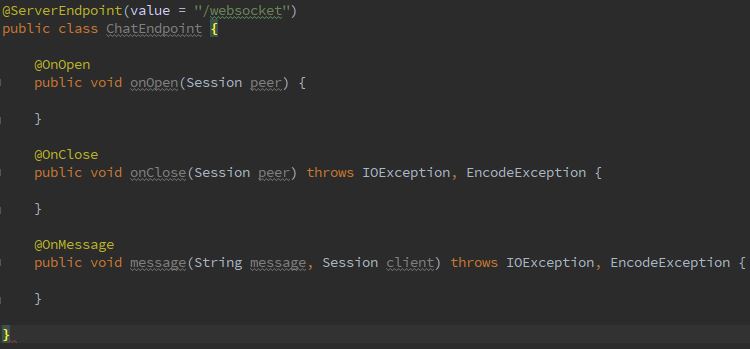
**FIGURE 72. Directories and files of the project**

### 7.3.2 The server code

The basic logic is the same as it was for the first server. Therefore, the *ChatEndpoint* class contains same functions and the structure. However, Java is not JavaScript. The syntax of languages is different. In addition to this, both languages have own features. For example, Java has better collections and annotations when JavaScript has callback functions. To avoid repetition, I will point out key differences of functions and the logic.

* **A backbone of the class**

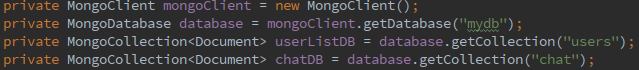
First, create the backbone of the class, as shown in Figure 73. This contains three methods: *onOpen*, *onClose* and *message*. Every method has the WebSocket annotation (e.g., *@onOpen*), which allows calling these methods according to the WebSocket activity. The class also has the *@ServerEndpoint* annotation. This makes the WebSocket technology to use the this class. As a parameter, we are able to point out the URL.



**FIGURE 73. The backbone**

* **Connecting to the database (Figure 74)**

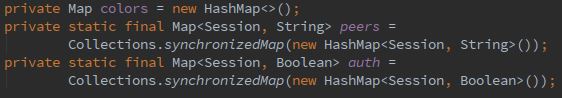
Connect to the database and place references to collections into global variables *userListDB* and *chatBD.* If a database or a collection does not exist, it will be created.



**FIGURE 74. Connecting to the database**

* **Create collections (Figure 75)**

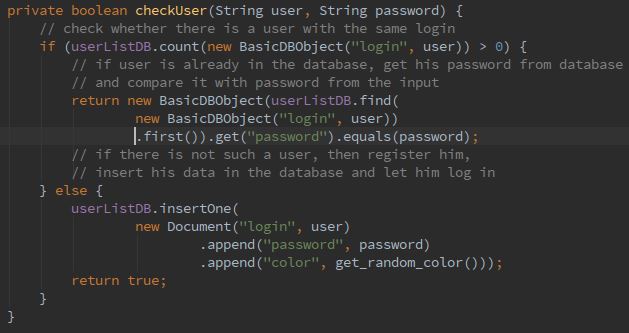
Collections will contain the list of logins, connections and colors of users.



**FIGURE 75. Collections**

* **Function: checkUser (Figure 76)**

Function checks if the user exists in the database, finds his data, checks the password, registers user and lets him log in. The function does not have a callback and returns Boolean value.



**FIGURE 76. The *checkUser* function**

* **Function: get\_random\_color (Figure 77)**

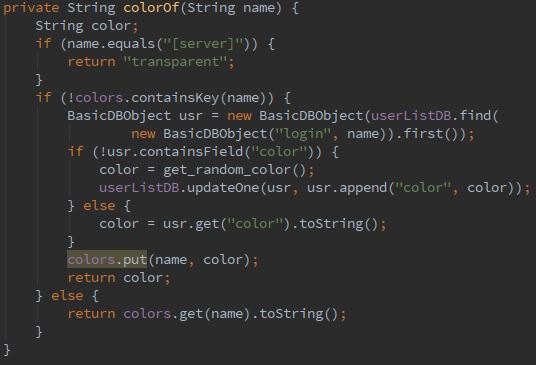
The function generates random color.



**FIGURE 77. Get random color**

* **Function: colorOf (Figure 78)**

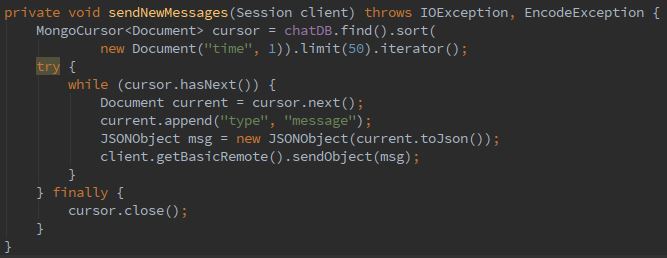
The logic is the same as for the JavaScript server. The function checks if the user and its color are in the array, adds color for the name (if it was not in the array), and updates the database. The function returns the color.



**FIGURE 78. The *colorOf* function**

* **Function: sendNewMessages (Figure 79)**

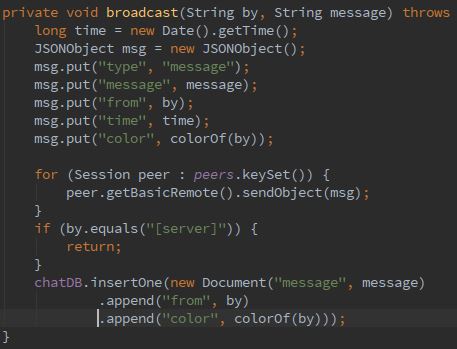
The function sends old messages to new chat participants.



**FIGURE 79. The *sendNewMessages* function**

* **Function: broadcast (Figure 80)**

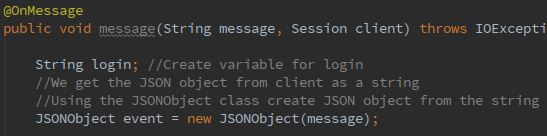
The function sends messages to all chat participants and save it to the database/history.



**FIGURE 80. The *broadcast* function**

* **Logic: When we have an incoming message (Figure 81)**

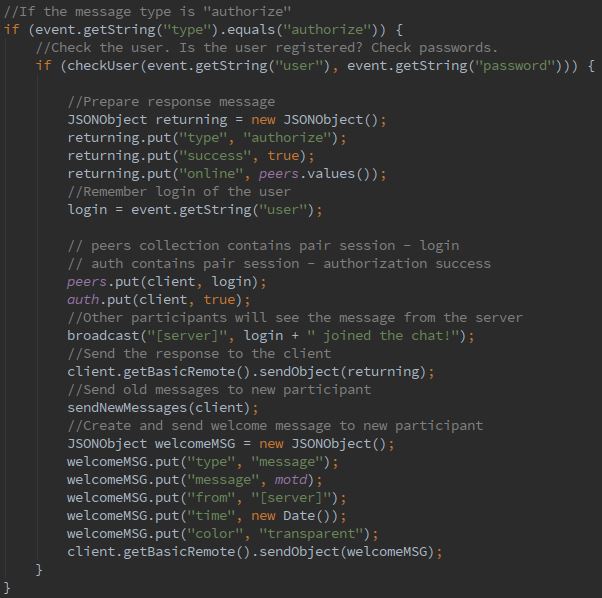
Create a local variable for login. The client sends the JSON object. However, the server gets the String message. Using the JSONObject class create JSON object from the string. Then, the function proceeds to the next step, where our server reacts to messages.



**FIGURE 81. *OnMessage***

* **Logic: message type – “authorize” (Figure 82)**

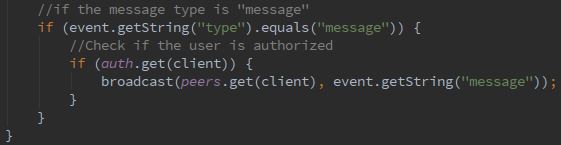
If the message type is “authorize”, the server uses *checkUser* function to check the user data. After the authorization, prepares a response message. Collections remember the login, the WebSocket session and the authorization success. The server broadcasts “Joined the chat” notification. In the end, server finally sends the “authorize” response event with information about the authorization success, the list of people online, old messages and the message of the day (Figure 82).



**FIGURE 82. The message type is “authorize”**

* **Logic: message type is “message” (Figure 83)**

After the server gets the message of this type, it checks if the user is authorized. Then, the server sends the information to all participants of the chat using the *broadcast* function.



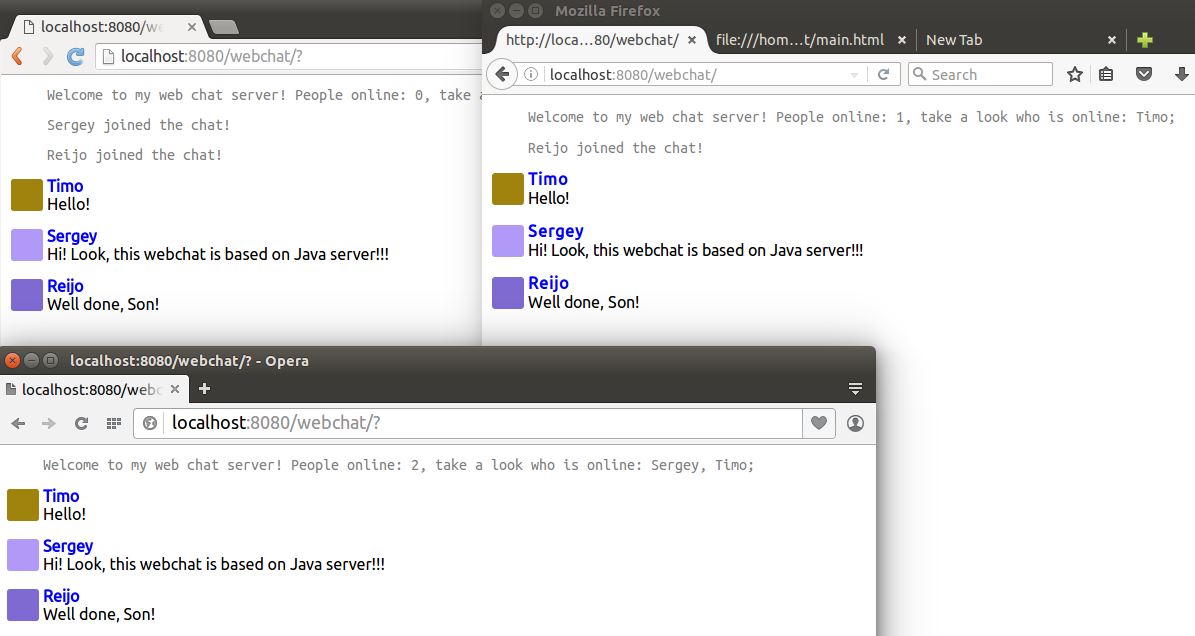
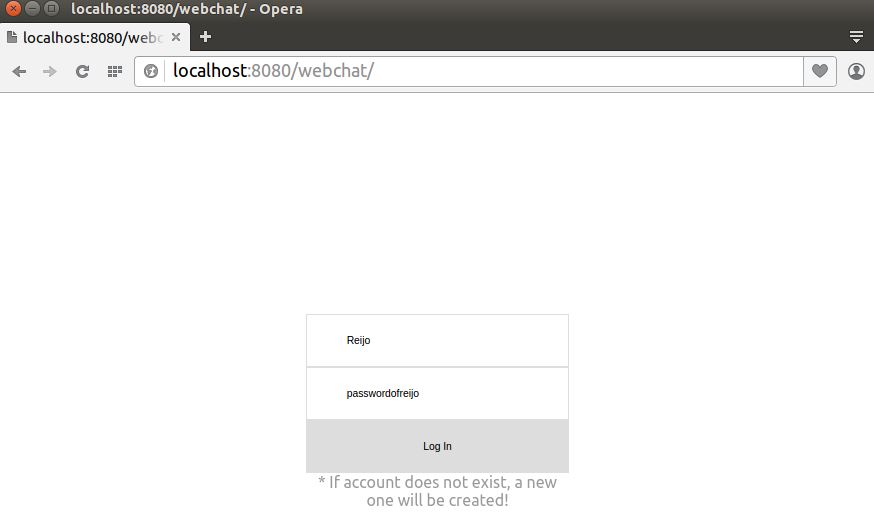
**FIGURE 83. The message type is “message”**

### 7.3.3 Launch and test

Now, we can launch and test the web application. As clients, I am using three browsers: Opera, Chrome and Firefox. To start the application we need to do following:

* Use the *sudo mongod –smallfiles* command from the terminal to launch MongoDB database
* Use the *sudo mvn package embedded-glassfish:run* command to deploy the server
* Opena browser and type *localhost:8080/websocket* in the address field

In Figure 84, we can see the login page and the chat in action.



**FIGURE 84. The chat in action**

# 8 results and analysis

In the end, I have got two web applications. Both clients have exactly the same code and even files. For the client I have used HTML, CSS and pure JavaScript. This is the most popular development stack for the web client. Web applications have different server sides. Main differences are programming languages, a build process and the Java application server (Glassfish). Shared technologies are the MongoDB database, the JSON object and the WebSocket connection.

The installation process was a lot easier for NodeJS. It took 5 terminal commands to install everything we need for the development. Moreover, you can skip the installation of Komodo Edit because you can start programming in any txt editor. However, by using Maven, developers have more control over all dependencies and plugins. Maven provides XML file. The file has all information and even can show errors related to unavailability of dependencies. Dependencies do not need to be stored locally because they will be downloaded during the server deployment according to the XML file. This simplifies the team development and sharing between outsource developers. The analogy for NodeJS is *package.json* file. However, developers need to work with the file carefully to avoide problems and have to use the *npm install* command to install all dependencies listed in the *package.json* file. It was easier to work with JSON objects and MongoDB during the NodeJS development. This is not surprising because JSON objects are based on JavaScript, and MongoDB is more adapted for Node. Moreover, Java does not has the callback function feature. Callback funtions allow developers to build more interesting logic. Nevertheless, the Java development is more strict and structed. There is no any part of code that cannot be reproduced with Java. The Java code logic is clear. The Java object-oriented possibility provides more potential for massive servers.

On my virtual machine it takes approximately 32 seconds for Java to deploy the Glassfish server. Meanwhile, NodeJS needs around 3 seconds. As a result we have two web chats. Overall experience is good for both chats. The speed and functionality are the same for both chats. However, NodeJS passes through authorization slightly faster. In addition, Java WebSocket has problems with connetions on page refreshing. This cause a bug I am not able to fix for the moment.

# Conclusion

The very first thing I have noticed during the development is that Java and JavaScript mainly differ in the number of features and complexity level. JavaScript is more free in comparison with the Java implementation. For example, in JavaScript we can skip the declaration of variables, and the data type conversion is easier. The JavaScript language requirements are also much less strict in matters of syntax and type checking. In addition, the source code in JavaScript do not need to be compiled, unlike of Java programs, JavaScript is interpreted language. JavaScript interpreter reads the program line by line, and reports errors after each line, rather than after processing the entire code of the program, as the Java compiler does. Therefore, during my development process I noticed that JavaScript program can be developed and debugged much faster. As Java, JavaScript is an object-oriented language, although it has no classes, no built-in mechanisms of inheritance, which are standard for Java. In the Java language programs use the object classes, and these programs are fully object-oriented. Moreover Java and JavaScript are among the safest languages ​​in the sense that each of them supports tools that do not allow the data recording to a hard disk.

The Node.js platform is the most popular realization of JavaScript on the server-side. Although Node.js has got a variety of modules, it is still significantly lose to the capabilities of the powerful set of Java libraries. Java is a solid and popular technology with huge positions on the market. There are millions of Java servers working around the world. However, Node.js is developing with huge speed and surprises with its swiftness. I have noticed that Node.js documentation is more updated and fresh. Community questions about Node.js are also more up-to-date. It looks like developers of the entire Web have got a new toy to play and forgot about Java. During the development, I actually started to understand why people like the idea of JavaScript on the server-side. The possibility to build a client and a server by using only one programming language excites developers around the world. As a result, we can observe the great growth of the Node.js platform and its community.

The studies showed that both languages are truly competitive. Moreover, both technologies have huge developer community. Java looks like the best option for huge corporate servers. However, there are examples of JavaScript servers that successfully process enormous amounts of connections and data. . As a result, the wisest decision is to investigate the problem deeply to choose the right technology, because certain tasks work best with certain solutions. The thesis can be useful for IT students, teachers and web developers. For further developments, the best option would be to build another Java application by using only J2EE or the Java full stack. It can be a good idea to build the Java web client too. Moreover, the size of applications is truly important. Therefore, a development of massive servers would show more advantages and disadvantages.

# BIBLIOGRAPHY

The Apache Software Foundation, 2016. Introduction to the POM. The company’s WWW page. https://maven.apache.org/guides/introduction/introduction-to-the-pom.html. Referred 31.03.2016.

Beal, Vangie 2016. Java, basic information. WWW page. http://www.webopedia.com/TERM/J/Java.html. No updating information. Referred 06.03.2016.

Brown, Simon 2002. Introduction to the Java 2 Platform, Enterprise Edition (J2EE). WWW page. http://www.javaranch.com/journal/2002/10/J2EE.html. No updating information. Referred 15.03.2016.

CCM Encyclopedia 2008. Networking - 3-Tier Client/Server Architecture. WWW page. http://ccm.net/contents/151-networking-3-tier-client-server-architecture. Updated 2016. Referred 07.03.2016.

Chapman, Stephen 2016a. What Exactly Is JavaScript? WWW page. http://javascript.about.com/od/reference/p/javascript.htm. Updated 2016. Referred 03.03.2016.

Chapman, Stephen 2016b. What JavaScript Can Not Do? WWW page. http://javascript.about.com/od/reference/a/cannot.htm. No updating information. Referred 03.03.2016.

Drake, Donald G. 1996. Introduction to Java threads. WWW page. http://www.javaworld.com/article/2077138/java-concurrency/introduction-to-java-threads.html. No updating information. Referred 10.03.2016.

Lengstorf, Jason 2009. JSON: What It Is, How It Works, & How to Use It. WWW page. http://www.copterlabs.com/json-what-it-is-how-it-works-how-to-use-it/. No updating information. Referred 20.03.2016.

Nourie, Dana and Pawlan, Monica 2007. Introducing the Java Platform. WWW page. http://www.oracle.com/technetwork/topics/newtojava/intro-139083.html. No updating information. Referred 09.03.2016.

Pravin, 2009. Advantages of Application Server. WWW page. http://data-centers.in/advantages-of-application-server. No updating information. Referred 16.03.2016.

Slideshare, 2012. Client-server concepts. WWW slides. http://www.slideshare.net/avinash3189/client-server-concepts. No updating information. Referred 13.03.2016.

VMware Inc, 2016. Introduction of VMware Workstation Product. The company’s WWW page. https://www.vmware.com/products/workstation. No updating information. Referred 21.03.2016.

Wayner, Peter 2011. JavaScript conquers the server. WWW page. http://www.infoworld.com/article/2621690/application-development/javascript-conquers-the-server.html. No updating information. Referred 17.03.2016.

Web Education Community, 2011. What can you do with JavaScript? WWW page. https://www.w3.org/community/webed/wiki/What\_can\_you\_do\_with\_JavaScript. Updated on 18.11.2011. No updating information. Referred 10.03.2016.

Wikipedia, 2016a. Application server. WWW page. https://en.wikipedia.org/wiki/Application\_server. Updated on 15.04.2016. Referred 20.03.2016.

Wikipedia, 2016b. NodeJS. WWW page. https://en.wikipedia.org/wiki/Node.js. Updated on 24.04.2016. No updating information. Referred 22.03.2016.

Will, Brian 2007. A brief explanation of Java versions. WWW page. http://brianwill.net/blog/2007/03/03/a-brief-explanation-of-java-versions/. No updating information. Referred 14.03.2016.