CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

Department of Information Technology

BACHELOR THESIS

USE OF THE RUBY ON RAILS WEB APPLICATION ON THE CENTOS FOR THE GOVERNMENT

Author: Petr BARTONÍČEK

Supervisor: Ing. Miloš ULMAN, Ph.D.

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

Petr Bartoníček

Informatics

Thesis title

Use of the Ruby On Rails web application on the CentOS for the Government

Objectives of thesis

This thesis describes an application written in Ruby On Rails, development and deploy on CentOS7 within the environment of the e-government in Czech Republic. The main aim of thesis is to analyse implementation options with using open source technology.
The partial goals are such as:
- to make an overview of current web technologies,
- to describe a whole deployment solution and development environment for the application,
- to design fully functional solution of an application for selected government body and to make a comparative analysis.

Methodology

Methodology of the thesis is based on experience of web development and studying of information resources. The practical part is focused on deployment and development of an application with Ruby On Rails and CentOS7 as a secured open source platform with respect to the software engineering principles. The conclusion will provide a comparing of other technology.

The proposed extent of the thesis

30 – 40 pages

Keywords

Ruby On Rails, CentOS, Apache, Passenger, Bootstrap, Open Source

---

Recommended information sources

BLACK, David. Well-grounded rubyist. 2. S.l.: O'Reilly Media, 2014. ISBN 978-161-7291-692.
CentOS 7 Linux Server Cookbook. Second edition. US: Packt Publishing, 2016. ISBN 978-1785887284.
HARTL, Michael. Ruby on rails tutorial: learn web development with rails. Third edition. New York: Addison-Wesley, 2015. ISBN 01-340-7770-9.
Mastering CentOS 7 Linux Server. 1. US: Packt Publishing, 2016. ISBN 978-1785282393.
METZ, Sandi. Practical object-oriented design in Ruby: an agile primer. 1. Upper Saddle River, NJ: Addison-Wesley, 2013. ISBN 03-217-2133-0.
PUGLISI, Silvia. RESTful Rails Development: Building Open Applications and Services. 1. US: O'Reilly Media, 2015. ISBN 978-1-4919-1085-6.
RAHMAN, Syed. Bootstrap for Rails. 1. US: O'Reilly Media, 2015. ISBN 978-1-78398-727-6.

Expected date of thesis defence

2017/18 WS – FEM (February 2018)

The Bachelor Thesis Supervisor

Ing. Miloš Ulman, Ph.D.

Supervising department

Department of Information Technologies

Electronic approval: 18. 10. 2016	Electronic approval: 24. 10. 2016
Ing. Jiří Vaněk, Ph.D.	**Ing. Martin Pelikán, Ph.D.**
Head of department	Dean

Prague on 11. 07. 2017

Declaration

I declare that I have worked on my bachelor thesis titled "Use of the Ruby On Rails web application on the CentOS for the Government" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break copyrights of any their person.

In Prague on ………………… …………………

Petr Bartoníček

Acknowledgement

I would like to thank Ing. Miloš Ulman, Ph.D. for maintaining positive thinking and a supportive attitude to the subject of the thesis and for helping me to find the motivation to continue studying. I would also like to thank my family, especially my extraordinary mother and father. They have gone with me through the hard times and I will always be thankful to them. Also, my classmate and colleague Vlad Rakitov has been with me from the first day at CULS, and supports me in life as a best friend.

Use of the Ruby On Rails Web Application on the CentOS for the Government

Abstract

The bachelor thesis describes the development and deployment of a web application with databases. The application uses security features that can be used in a government environment. It is currently using an alternative programming language that provides an excellent, simple, and fast developmental process. The entire thesis deals with the methodology of development using agile methods and scrum technique. The main aim of the thesis is to create a theoretical alternative to the current form of development in the public sector as well as the government. The thesis shows an example of an alternative to the current state of the development of web applications using agile methods.

The introduction describes individual technologies and their axioms related to the development of secure applications. The practical part describes the use of individual technologies and demonstrates the implementation of the management system for public tenders. The basis of the entire system is CentOS, the mirror of Red Hat Linux Enterprise used by American government agencies such as the NSA or CIA. The web layer is Apache in the minimum configuration for functionality and the application layer is represented by Passenger. For server-side development, the Ruby with the Rails framework has been selected to offer a robust solution for the MVC design pattern. The client side serves the bootstrap and due to security reasons, only minimal JavaScript code is used.

Keywords: Ruby on Rails, CentOS, Apache, Passenger, bootstrap, open source

Použití webové aplikace v Ruby on Rails na CentOS pro vládu

Abstrakt

Tato bakalářská práce popisuje vývoj a nasazení do produkce webové aplikace s databází. Aplikace používá prvky bezpečnosti, díky nímž může být nasazena v prostředí vlády. Využívá v současné době již alternativní programovací jazyk, který umožňuje kvalitní, jednoduchý a rychlý proces vývoje. Celá práce bere v potaz metodologii vývoje za použití agilních metod. Cílem této práce je vytvořit teoretickou alternativu k současné podobě vývoje v prostředí veřejného sektoru za použití agilních metod.

V úvodu práce jsou popsány jednotlivé technologie a jejich axiomy vztahující se právě k vývoji bezpečné aplikace. V praktické části je popsáno použití jednotlivých technologií a ukázka implementace systému pro řízení veřejných zakázek. Jako základ celého systému byl zvolen server CentOS, což je zrcadlo Red Hat Linux Enterprise používaného například americkou vládní agenturou NSA nebo CIA. Webová vrstva je Apache v minimální konfiguraci pro funkčnost a aplikační vrstvu zastupuje Passenger. Pro vývoj na straně serveru byl zvolen jazyk Ruby s frameworkem Rails, který nabízí robustní řešení pro návrhový vzor MVC. Klientskou část obslouží bootstrap a z důvodu bezpečnosti pouze omezený kód v JavaScript.

Klíčová slova: Ruby on Rails, CentOS, Apache, Passenger, bootstrap, open source

Table of Contents

1 Introduction

The thesis covers topics in theoretical layers such as installation, management, running deployment, and development with Ruby on Rails on the CentOS server. Ruby on Rails is the framework based on the Ruby. Ruby is the objective-oriented language inspired by languages such as Perl, Smalltalk, and Lisp. Ruby, which is open source, was released in 1995 and has the stable version of 2.3.1. Rails framework, another open source, was released in 2004 and has the current version of 5.0.0.1. CentOS is a copy of Red Hat Linux Enterprise (RHLE) provided for free as open source, and the current version is 7. The main advantage of using CentOS is that the stable version is supported for 10 years. The database system that was chosen for this project is PostgreSQL, currently one of the most used database systems, released way back in 1986. PostgreSQL is a very well documented and maintained open source database system that supports even NoSQL databases. Managing the web server is done by Apache or Nginx with application layers such as Passenger or Puma. Apache is a Hyperrtext Transfer Protocol (HTTP) server, which is standard and used in "more than 35% servers online in 2015." [12] Apache was released in 1995 and was created to provide the free and friendly implementation of HTTP web servers. Nginx is the HTTP web server that runs on websites like Yandex and Mail.ru, and offers the same performance as Apache, or better. Apache servers uses Passenger for the application layer, while Nginx uses Puma.

The proposed solution provides a deep overview of using an open source tool and describes the whole in-house application lifecycle. The development is driven by stories. The thesis describes the security setup on the server for I/O with internal communication between daemons and services with SE Linux, as well as the user, development, deployment, and continuous integration management.

Rails has more to offer than just being part of framework language. With Ruby, it is possible to write scripts that are able to run on the server to help with process management. On the server, Git is used for managing application versions in an automated way, handled with Ruby scripts. Git, the version control system created by Linus Torvald, offers a quick solution for the versioning of application development; it has quickly become the standard only a few months since its release. All work is submitted to Git by committing with a message explaining what is being done. To ensure that the application is working correctly, the technique Test-Driven Development (TDD) should be employed. Test-Driven Development means that every code is supposed to have a test in order to prove that it is doing what it is supposed to be doing.

Agile development and scrum methodology fits into the solution of the developing application in the 21st century. Large Scale Scrum (LeSS) is a scrum framework that proposes two options: Huge LeSS for thousands of people with one product owner, and LeSS for multiple teams with up to eight people per team, as well as product owners for each team. The implementation of agile methodology is interesting from a theoretical point of view. Considering that within the Czech government, where Waterfall methodology still prevails, there are huge potential benefits from the increased flexible reactivity of the team on a 14-day scrum sprint basis.

Regarding facts about using open source in the Czech government, this could be a compensation for expensive licenses. The government currently forces municipalities, authorities, and ministries to place all data in digital form, while using mostly Microsoft and Oracle products; which is in contradiction with open source approaches. The solution proposed in the thesis would be secure, inexpensive, and capable of scaling and preparing for agile development. This is possible to achieve with open source. The thesis may also offer a guide for the faster implementation of using open source in the Czech government environment. All examples of code are provided in the Ruby language, represent a fully working code, and provide high-level overview of in-house solutions for managing project aims, tenders, and projects.

2 Objectives and Methodology

The following section provides high-level overview of the thesis itself.

2.1 Objectives

The thesis describes the production environment from the server to the application and its development with agile approach. An application written in Ruby On Rails, development and deployment on a CentOS server within the environment of the government in the Czech Republic. The main aim of thesis is to show and investigate possible implementation options, using open source technology and agile methodology, to describe the basics regarded to literature, and to compare the methodology and critical overview of the chosen technology.

The partial goals are as follows:

- Make an overview of current web technologies such as Ruby on Rails and JavaScript
- Describe the entire deployment solution for the client JS and server side application as well as the development environment for the application itself
- Design a functional solution of an application for a selected government body and create a comparative analysis

2.2 Methodology

The methodology of the thesis is based on experience of web development and the study of information resources. The practical part focuses on deployment and development of an application with Ruby on Rails and CentOS7 as a secured open source platform with respect to the software engineering principles. The conclusion will provide a comparison of other technology and development approaches. Comparative analysis shows different methodology of development and SWOT analysis shows the possible strengths and weaknesses for each chosen technology in the context of public sector.

3 Literature Review

This section provides the description of the axiom and paradigm languages and the leveraging potential for further implementation. Every section has a small conclusion regarding the experience and obtained knowledge from the reviewed literature.

3.1 Server

The first web server has been running the very first web page in CERN, Geneva. [27] NeXT, the computer that has been running the first web server, runs on the NeXT STEP operating system, which is a Unix-based operating system, still the core of all Mac OS in use today. The difference in the core of CentOS; Linux versus Unix based systems, is not huge. The two cores were developed separately — while Unix was meant to be a money generating solution, Linux was free of charge and it quickly became the open source standard for everyone. The initial intention was to have a powerful machine work with data, and naturally, the web server was created. The first "distributed network" was established by IBM with their servers in order to share data across the organization, 10 years before the existence of the World Wide Web.

The server hosts the database, the web server, and the application server.

3.1.1 Installation and Maintenance

The server has to be prepared as an installation Compact Disc or on a Flash Disk. The ISO file can be downloaded on the official website https://centos.org/. There is also a possibility to implement an automated installation through the TCP/IP and reach the maximal potential of deployment of the whole solution; however in this case, Ruby may be used to provide a solution for mass deployment.

First and foremost is the update and upgrade of the server OS. Linux usually uses a package manager, and in the case of CentOS, it is Yellow-dog Updater Modified (YUM). The YUM package manager regards to running a system update. Applying patches and updates is a regular task for every server administrator, and an up-to-date system can help increase or ensure the security of ones server, as software bugs and vulnerabilities are found often and must be fixed promptly. [2]

IP addresses will be defined within the range of local area network. "While a dynamically assigned IP address or DHCP reservation may be fine for most desktop and laptop users, if you are setting up a server, it is often the case that you will require a static IP address. From web pages to e-mail, databases to file sharing, a static IP address will become a permanent location from which your server will deliver a range of applications and services." [2]

The purpose of opening Port 80 is to show the web page for development and production and it is done by using firewall-d, a service that allows easy configuration of iptables. [4] Iptables offer an editable file that covers complete network setup in the environment of the Linux operating system. Port 80 is in commonly used for websites all over the world, so it is means that it is most vulnerable in this case.

SE Linux is used on the server to ensure security, especially for internal purposes. SE Linux is “a mature technology for hardening your Linux system using additional security features added to the basic security system. It has been around for many years in the CentOS.” [2] However, it is not broadly used due to the difficult level of manageability and setup. SE Linux is heavily used in the government in the United States, especially in the branches of information services. The NSA is one the biggest donors and developers of the SE Linux secure layer.

The server uses SSH, “SSH is the Secure Shell connection that is used to connect remotely to a Linux machine. It is the main tool used by system administrators for remote management of their infrastructure. It is one of the essential tools that could be find in a basic installation of CentOS 7 and almost all Linux distributions by default.” [4] SSH will be here to provide a quick solution to push the development version to the production, directly from the development server to production with the Git version control system. “Secure Shell (SSH) is the basic toolkit that provides remote access to your server. The actual distance to the remote machine is negligible, but the shell environment enables you to perform maintenance, upgrades, the installation of packages and file transfers; you can also facilitate whatever action you need to carry out as the administrator in a secure environment. It is an important tool; as the gateway to your system.” [2] The authentication is performed by keys. A door works well as an example. The door needs a key and the key has to be in the possession of the entity that is trying to reach the door and open it. The key may be created in advance for SSH, but everything is digital. The user has a private key that is possessed only by the user. The unique identifier should be the only one in the universe thanks to algorithms employing immense prime numbers to create the keys and almost zero probability to of identical private keys being generated. Afterwards, the private key is concatenated and checked with the public key on the door, which unlocks the door for the user.

3.1.2 Web Server Layer Apache/Nginx

The web server is the underlying layer below the app server and above the Linux core. It provides access to the physical files on the server for the user, without letting him or her really see what is on the server and how it is implemented. The web server handles the errors and server time. If the resource cannot find standards in place, then the answer to the request to the server will be 404. In case the server will be not able to respond to the error in the form of error 500, it is thanks to the protocol HTTP, which is fully standardized. To mine the maximum from this technology, HTTPS should be standard and the server should include a certificate from the authority, which will provide a crypto connection between the client and the server.

Apache and Nginx are HTTP web servers that understand requests from the client and answer to them. Web servers are written in C, and Gcc is used for compiling, which is already included upon installation of any Linux.

“The Apache Web server is a remarkable piece of software. The basic package distributed by the Apache Software Foundation is quite complete and very powerful, and of effort has gone into keeping it from suffering software bloat. One facet of the package makes it especially remarkable: it includes extensibility by design. In short, if the Apache package right out of the box does not do what you want, you can generally extend it so that it does.“ [11]

“Nginx is more efficient than its competitors. First, and foremost, it's faster. By making use of asynchronous sockets, Nginx does not spawn processes as many times as it receives requests. One process per core suffices to handle thousands of connections, leading to a much lighter CPU load and memory consumption. Secondly, its simplicity of use is remarkable. Configuration files are much easier to read and tweak with Nginx than with other web server solutions, such as Apache; a couple of lines are enough to set up a complete virtual host configuration.” [12]

Both servers have the possibility to be used as mail servers and they are extensible by modules. This is crucial to in order to achieve desired functionality and have opportunity to scale the solution in the future.

3.1.3 App Layer Co-Working with Web Server Passenger/Puma

Running the Ruby on Rails application on Passenger as a “real time application,” provides debugging/monitoring logs and both app layers are able to manage massive amounts of connections; nowadays largely used across the online world. Both of them work primarily with Ruby on Rails. They behave as a module on the web server, Apache cooperates with Passenger, and Nginx cooperates with Puma. What is interesting is that Passenger can work as a standalone server, and therefore should be used as a quick development environment server. Passenger and Puma should be used in basic installations with only minimal modifications in order to concentrate focus on the main parts of the full web stack development environment.

The application server may be optimized for a multi-thread app. Unfortunately, it is rather difficult to write a Ruby code that is thread safe. The violation of threads is a huge problem and the solution needs to be scaled for more than 1,000,000 users. It is in this case that it is a more suitable solution to stay with a single thread application. For every new connection, Passenger and Puma create a scope environment and each work with one thread. Special care should be taken during injection of the C code into Ruby. Writing part of a code in C should be a huge advantage; however, with respect to the difficulty of maintenance, it is always better practice to stay with Ruby for full-stack.

3.2 Database

PostgreSQL is a strong relation database working with NoSQL. The “database management system (DBMS). Data can be structured as tabular data, semi-structured as XML documents, or unstructured data that does not fit a predefined data model.” [15]

Relation databases work with atomized data models and provide standard solutions with private, foreign key relation. NoSQL database is driven by CAP theorem. “The CAP theorem states that it is impossible for a distributed computing system to simultaneously provide all three of the following guarantees:

- Consistency: All clients see (immediately) the latest data even in the case of updates
- Availability: All clients can find a replica of some data even in the case of a node failure. That means even if some part of the system goes down, the clients can still access the data
- Partition tolerance: The system continues to work regardless of arbitrary message loss or failure of part of the system” [15]

3.2.1 PostgreSQL

The standard port for PostgreSQL is 5432. Good practice is to keep all the communication with the DB interface in the server and not allow anyone to call the DB directly from outside, and this may be achieved by the client JS call. This implementation will not keep a single API call from client. The passwords for the DB should be stored in ENV variables on the server and shall not be accessible from the outside world. The DB should be constantly backed up. Back-ups should be made within the organization on a different server behind the firewall and NAT of an internal network. All processes should be checked by SE Linux and any violations should be strictly forbidden.

3.3 Application RUBY/HTML/CSS/JS

In order to deliver a quality code, every HTML and CSS code should be controlled by the validator with respect to standards for HTML and CSS. Standards are covered by World Wide Web Consortium (W3C). "HTML (the Hypertext Markup Language) and CSS (Cascading Style Sheets) are two of the core technologies for building Web pages. HTML provides the *structure* of the page, CSS the (visual and aural) *layout,* for a variety of devices. Along with graphics and scripting HTML and CSS are the basis of building Web pages and Web Applications." [16]

Ruby code will be developed and tested with TDD. "TDD takes an inside-out approach, usually starting with tests of domain objects and then reusing these newly created domain objects in the tests of adjacent layers of code." [5] BDD should be used alternatively. "BDD takes an outside-in approach, creating objects at the boundary of an application and working its way inward, mocking as necessary to supply as-yet-unwritten objects." [5] Using TDD, there is a necessity to test everything before it is deployed. A crucial factor is code coverage, *especially* in the case of development of an application for the government. TDD does not mean that mocking is not necessary; it is important to make the decision if the integration tests are enough, or the mocking and interfaces implementation are necessary to be placed to have 100% code coverage. In the case of a unit test, the desired behaviour is to see how many method/function parameters exist and what will happen if one will be added or removed. Proper implementing of the unit test will not cover the whole code so that mocks are in place. Mocks may provide alike response for an interface without any call to API, which might be helpful for huge applications with thousands of tests.

JS code should respect the ECMAScript standard. ECMAScript is covered by the Mozilla foundation. "Ecma Standard defines the ECMAScript 2016 Language. It is the seventh edition of the ECMAScript Language Specification. Since publication of the first edition in 1997, ECMAScript has grown to be one of the world's most widely used general purpose programming languages. It is best known as the language embedded in web browsers but has also been widely adopted for server and embedded applications." [17] Babel should be implemented as a pre-processor for using the latest version. It offers a possibility to write JS code in a new ECMAScript standard and it transforms to the form of supported older versions for the browser or server environment. For the government, there should be an emphasis on compatibility. Internet Explorer will not work with this solution, not even with clear JS due to different implementation from Microsoft, without respect to the standards. Much should be solved with frameworks such as jQuery or Vue.js, which provide full, native support for IE, Mozilla, Opera, and Safari.

3.3.1 MVC Architecture

MVC stands for “Model View Controller.” It is the most basic web structure and it has been in use since the late 1990s. The Model communicates only with the DB and the light Controller will provide authentication/authorization for the Model and View. The View should hold every UI element which is implemented on the site: JS and HTML/CSS. To secure the site from XSS, CSO, SQL-injection; there should be no usage of parameters directly from client, but rather manage them in the Controller on the back end and serve to the Model which will then through the authentication providing the data/information to the View. It is good practice to restrict both View and Model-Controller so the client will be not able to access sensitive data. Any API call from the client is strictly forbidden and means that API calls may be implemented only on the server side, rather than with custom Controller logic.

To get up to speed with development, Ruby takes care of all standards and any basic applications will have a MVC structure as shown below in Fig. 1.

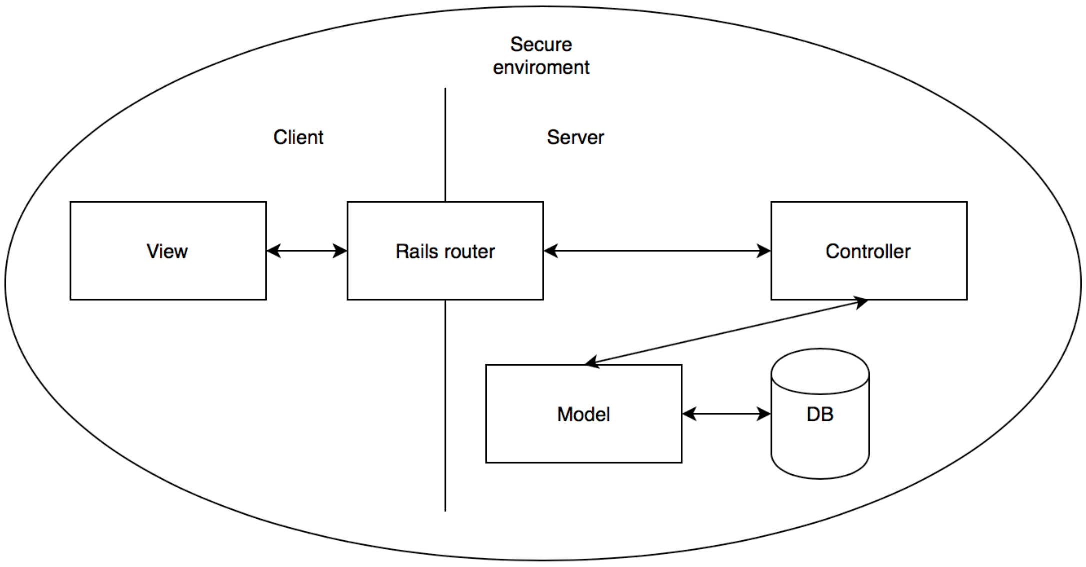


Fig. 1 - MVC in Ruby on Rails [27]

3.3.2 Application in Ruby on Rails

DRY stands for “don’t repeat yourself.” Every function used more than once should use a method that will solve the problem. The following basic rules are adopted for the thesis:

- “UTF-8 as the source file encoding
- Two spaces per indentation level (aka soft tabs); no hard tabs
- Unix-style line endings
- One expression per line
- Single-line format for class definitions with no body
- Avoid single-line methods
- Use spaces around operators; after commas, colons, and semicolons, around {and before}
- Never use unless with else
- Keep the controllers skinny
- Logic resides in the models” [6]

The application will be an all-in-one, using the capabilities and native function of Ruby and Rails Framework. Every part of the application prefers to have already-written solutions, so that the gems are used. Gems are packages of reusable code, distributed for Ruby. The front-end will be written as embedded Ruby and JavaScript, it is native HTML pre-processor which includes the Ruby code. Ruby on Rails has it own ORM layer, object relation model called "active record." Active record will handle the communication with the PostgreSQL database.

3.3.3 REST Application in Ruby on Rails With Separate Front-End

REST stands for "Representational State Transfer." "REST is the architectural style of the Web, consisting of a set of constraints that, applied to components, connectors, and data elements, constitute the wider distributed hypermedia system that is known today: the World Wide Web."[6] The application in Ruby on Rails might be written as API and connected to the JS server that will handle the JavaScript front-end separately. This way, the logic of the front-end and back-end will be separated. Here is the opportunity to use framework such as REACTJS, Ember, AngularJS, or Vue.js.

"JSON (pronounced "Jason"), which stands for JavaScript Object Notation. It is widely used as a data storage and communication format on the Web. JSON is similar to JavaScript's way of writing arrays and objects, with a few restrictions. All property names have to be surrounded by double quotes, and only simple data expressions are allowed—no function calls, variables, or anything that involves actual computation. Comments are not allowed in JSON."[9] JSON is used for the communication between the front-end and back-end. Ruby has a built-in function to work with JSON and JavaScript is the basic data structure. Here there is a possibility to use the NoSQL database.

This will implement possible connection to any custom application. There is increasing potential of this solution for using mobile applications or other websites in the future. The same data, credentials, and users know the flow, which is especially valuable in Government environments.

3.3.4 Versioning of Application with Git

The use of Git manages the version of the application on the server and in similar ways for production. "There are many options for version control, but the Rails community has largely standardized on git, a distributed version control system originally developed by Linus Torvalds to host the Linux kernel." [3]

"Git is strongly recommended, not only because it's nearly a universal practice in the Rails world, but also because it will allow you to back up and share your code more easily." [3]

3.4 Security

The server should only be utilized for internal usage because there is no motivation or intention to keep the server accessible from the outside of the organization like a government body. Every request on the server should reload the page and create a new token for authorization in order to avoid spoofing the traffic on the network implementation, as can be done with oAuth 2.0. [21] In practice, this is done by the Ruby on Rails framework itself, so the connection client to the server and vice versa, is secured by convention over the configuration. Additionally, the Object-Relation Mapping, part of the Ruby on Rails framework layer ActiveRecords, is secured by design. This provides a robust solution for avoiding attacks such as middlemen where the attacker is spoofing the local network with tools such as Wireshark. This is enough evidence that the solution is clear and hardly hackable when it is correctly implemented.

Basic security must also be fulfilled. This means that the server should be updated on a daily basis. Code should go through code review and refactoring might be done after every sprint when a final product is shipped. Common practice is to order a penetration testing company that will provide an overview of possible vulnerabilities within the system.

3.5 Agile Development

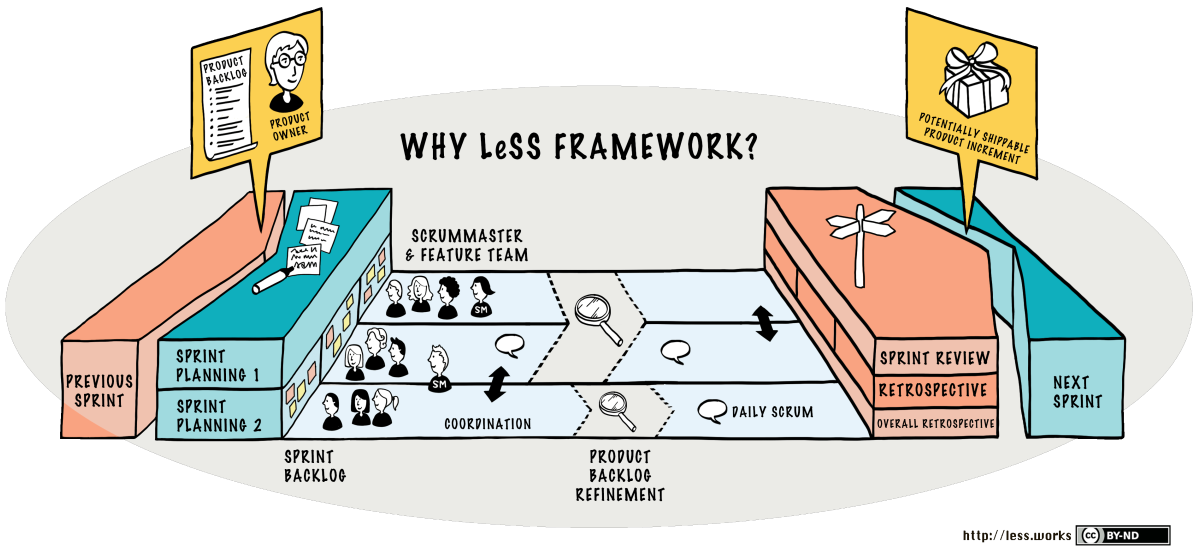


Fig. 2 – Less framework [18]

Agile, Scrum development may be taken from many sides but the basics are clear. The Product Owner (PO) has a team that works on a 14 days basis. Rather than finishing the product itself, it is better to go through the small iteration cycle in order to avoid the difficulty of the complex solution. The team gains knowledge and creates value for itself. Every two weeks there should be a refinement where the scrum teams and POs sit together and score the task. Normal time is no longer used and story points are employed. Every sprint of the standard length is 14 days has 80 story points. The points are based on previous experience and are spread across the tasks. The PO should be aware of every difficulty in business logic and prepare the stories/use cases/tasks for each sprint. Stories should be as small as possible, keeping in mind: something that is impossible to achieve for the team now, might be easy in half a year. The idea is not to bring in new people, but rather to let the older members grow and educate. Teams are supposed to communicate together and share the knowledge and experience with the stories. Since the scrum is only for 14 days, the period is very well supportive of the idea of working with new technology. The government is not a living place with all of the regulations, deregulations, laws, restrictions, and licenses. Scrum may help to open the government to the world of free licenses and technology. [18]

A huge advantage in this approach is the PO, who may call PM from waterfall methodology, does not need to know anything about the technology. The implementation is up to the team members and if it is not possible, there is still a way for it to be postponed. Trends are used for guiding the project team leader who is first in everything and is always willing to help. The problem is that there is lack of such skilful people. The agile scrum with inexperienced developers will produce a great team without a PM or team leader and speeds up development. [18]

This approach was chosen for the thesis as a one of the most common implementations of Scrum.

3.6 E-Government in the Czech Republic

The practical part offers free licenses for use with own servers, which are necessary for the implementation. The Czech government paid “4,600,000,000 CZK” [13] for licenses and services, although this money also covered the support in addition to the licenses and services. An official has a salary defined by Czech Law that groups of developers will have ¼ the salary of the consulting company IBM. The developer is considered as a technical worker for the government body within the Czech Republic. However, the thesis offers a different approach, the law about using electronic form of data in most parts of government was written in “2009”. [14]

IBM uses mostly Microsoft technology for their servers, webservers, and licences. This is extremely expensive compared to open source, which costs nothing to use. For instance, a public tender management system should be used at the Czech Ministry for Regional Development. The current number of the employees regarded to the organization structure is “127,” [19] not including the receptionist.

4 Practical Part

The practical part describes the proposed implementation in order to improve the processes around development and deployment. The development process employs the Agile methodology with stress on the internal team growth in terms of knowledge and effectivity. Implementation with the whole installation phase and basic security setup must be done by the team. Continuous integration includes UAT and various environments. This should cover the whole process and provide an overview of possible alternatives to the legacy systems implementation that are currently in use. Showing the basic user stories is the case of the environment of the Ministry of Regional Development as a public government body example.

4.1 Considerations for the Chosen Technology

Programming language distribution is an important indicator, to have a secure application with as little work as possible. The Ruby, respectively the Rails framework, sets the standard for MVC which is followed by almost all other MVC frameworks, such Laravel for PHP or .NET itself. No government body will adopt PHP since it is not the enterprise standard and most of the consultants, especially those in the Czech Republic, adopt Microsoft technology instead. In the graph bellow, Ruby has the lowest penetration on the market, which is better for use in such environments in an effort to avoid everyday attacks inside the network.

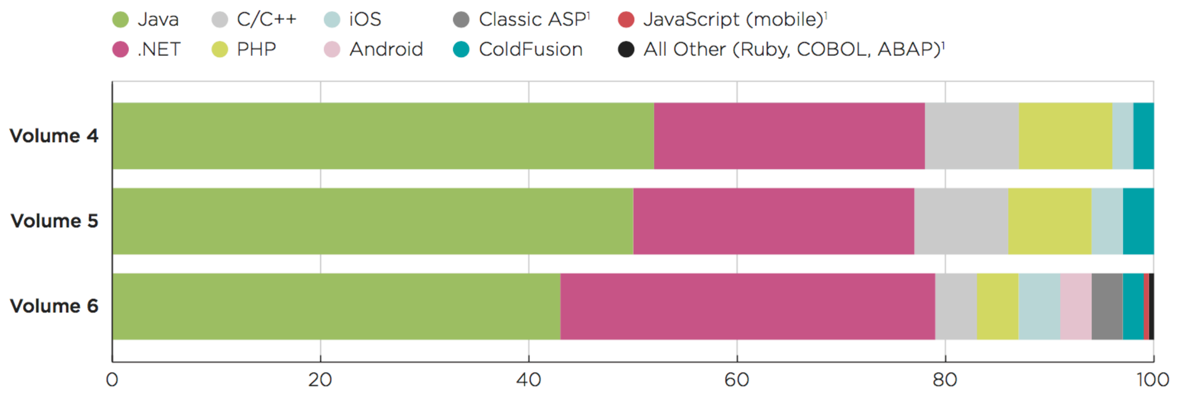
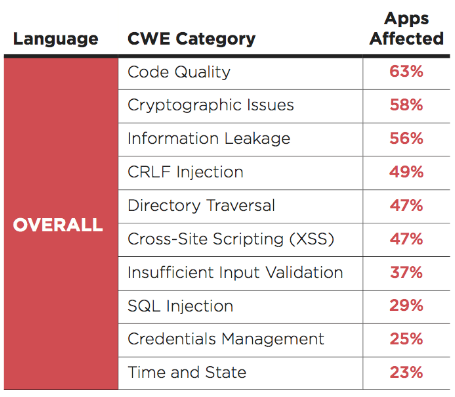


Fig. 3 – Language distribution [20]

From the graph above, is clear that Ruby is not the most used language; however, it defines the standards and it has enterprise-like architecture without possibilities of everyday attacks. The word “attack” refers to the table on the following page.



Language	CWE Category	Apps Affected
OVERALL	Code Quality	63%
	Cryptographic Issues	58%
	Information Leakage	56%
	CRLF Injection	49%
	Directory Traversal	47%
	Cross-Site Scripting (XSS)	47%
	Insufficient Input Validation	37%
	SQL Injection	29%
	Credentials Management	25%
	Time and State	23%

Fig. 4 – Top 10 vulnerabilities by Open Web Application Security Project (OWASP) [20]

These were the top vulnerabilities regarded to the Web Application. Regarding CRLF Injection, the Rails did not allow alteration programmatically code from its own convention, and was a possible problem for the legacy systems. Directory traversal was not possible since there was no vulnerable JavaScript code possible in a legacy system. SQL injection may have been a problem for both systems, while the Rails framework SQL checked if there was insufficient authorization then it was not possible to inject query, coming from convention in the MVC. If someone had access to the data, the query could not be injected. Credential management was the last point — in this case, a legacy system on a Windows server with active directory was more secure; however, the credentials were travelling on a local network. Since the implementation of the application was behind the proxy, internal attack was possible, but not in the case of Rails. Once again, convention took care of the encryption and decryption of the credential for the team, as well as double-checked the query. The rest of the vulnerabilities were up to the people where the stress might be on the training within the organization. This was done as a story by the scrum team thanks to Agile approach; there were not 10 tasks that needed to be done before training, but rather only three tasks were executed and the next sprint would take care of the re-education of the users.

The Ruby could also be used as a scripting language on any platform, which made it the preferable choice. PHP could do the same, but not with the same performance as Ruby. RHEL used Ruby for deployment of huge amounts of installation across the Internet. Therefore, automation was most likely done by Ruby as well.

4.1.1 Legacy Systems and Their Part

The legacy systems for the management and maintenance of the data within the government body was implemented in C# and .NET. This was the case for the Czech Republic in 2015 [13], using connectors to the DB on local network to the Oracle DB. [13] SSH tunnels were used from server to server on a local network and the certificates were kept updated. DB connectors faced possible vulnerabilities, but there was not such an update policy as existed for SSH. For instance, DB connectors such as ODBC could not be updated as easily as SSH with zero-day exploits. In order to update open-SSH, it was sufficient to write a single command in Linux. However, in order to update DB connectors, the package had to be downloaded and manually added to the Window server, where it then had to be attached to the DB.

4.1.2 Specific Type of Web Application in Terms of Request Response

Basically, there were two types of applications, single page and multipage. Single page was written in JavaScript and was not secure using open API call from the client without the encryption of anything since it was on the client side. There were attempts to do partial rendering more than 50% on the server with Node.js. The multipage app used a template engine and served the static pages with prefilled variables or data on the server.

4.1.3 Potential of Further Implementation

After the revision of vulnerabilities and the spread of the language itself on the market, it was clear that Ruby on Rails had at least the same security as legacy systems, or better. Accessibility was everywhere within the government on a local network. Performance was not what should have been checked, since Ruby on Rails and C# or .NET were still in development, the performance may have varied and the sprint for refactoring could have helped solve almost any issue.

Both systems were compiled through the compiler or through the VM — the VM and compiler may have been implemented for both of them. Ruby was written in language C and natively supported code in C, which may have helped with the optimization of performance.

4.1.4 Out of Scope TLS, SSL, Certification Authority

A certification authority was in effect. Ruby on Rails with Passenger ran on HTTPS seamlessly, so it was necessary to take care of the certification by organization. The government in the Czech Republic used its own certificates for each ministry, so this was not the issue. The certificate had to be assigned to the web page and used any tunnelling technology such as TLS or SSL for calls to the server. For small implementation, DB servers could have been a part of the production server. SE Linux handled the security internally for the server and its daemons in the core of Linux.

4.2 User Stories and Technical Solutions

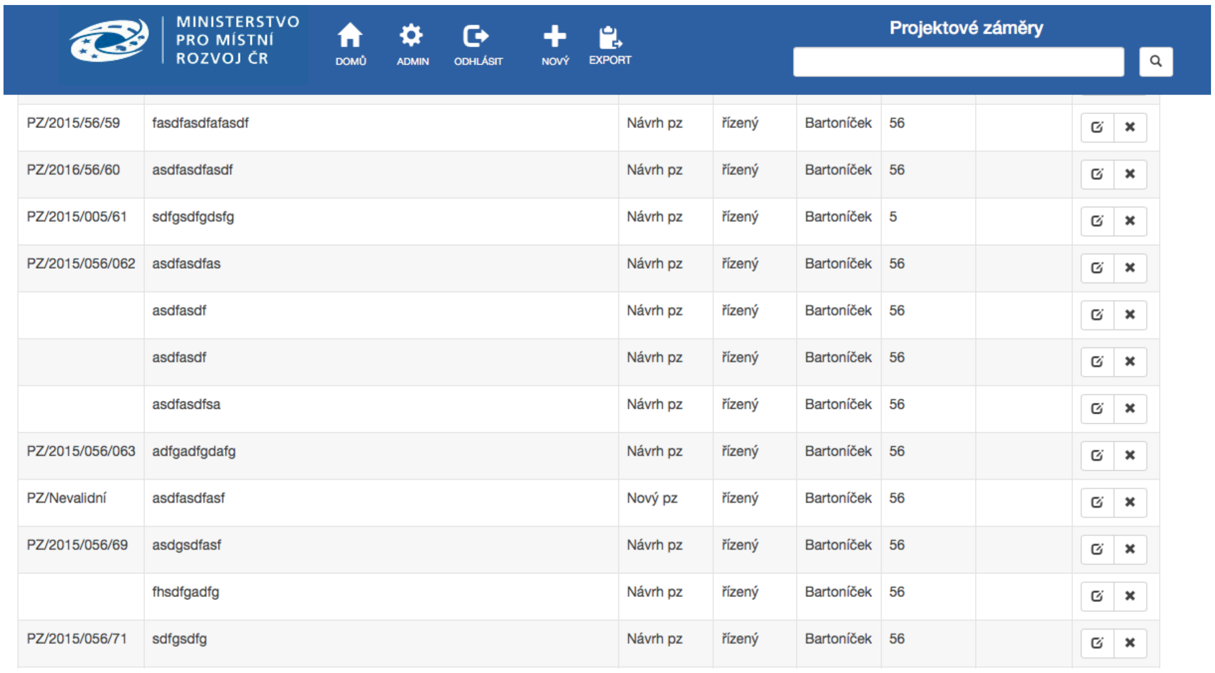
The size of the user stories was to fill one sprint for the scrum team. The time frame for the sprint was 14 days.

A technical solution was something that was provided by the team as an outcome from a conversation between the team members. Additionally, the basic wireframes were a part of the technical documentation.

4.2.1 Public Tenders

User story: an employee of the Ministry of Regional Development, wanted a basic view of public tenders.

Technical documentation: created was a view of basic information from public tenders that were already in the DB. This information was shown to the user and a static header was created. Lastly, the scaffold and bootstrap were used.



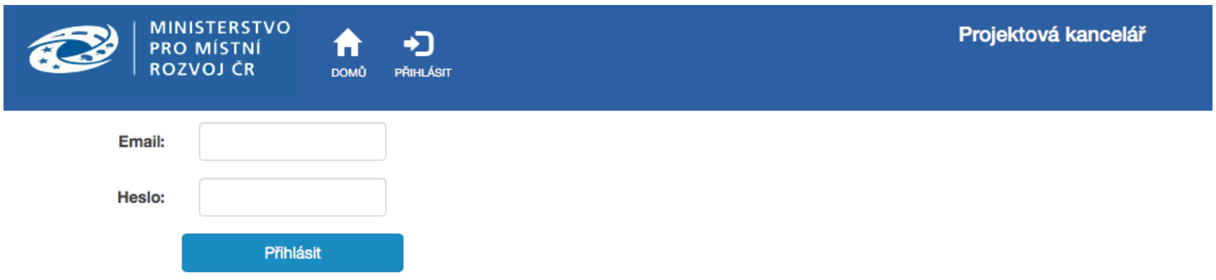
PZ/2015/56/59	fasdfasdfafasdf	Návrh pz	řízený	Bartoníček	56
PZ/2016/56/60	asdfasdfasdf	Návrh pz	řízený	Bartoníček	56
PZ/2015/005/61	sdfgsdfgdsfg	Návrh pz	řízený	Bartoníček	5
PZ/2015/056/062	asdfasdfas	Návrh pz	řízený	Bartoníček	56
	asdfasdf	Návrh pz	řízený	Bartoníček	56
	asdfasdf	Návrh pz	řízený	Bartoníček	56
	asdfasdfsa	Návrh pz	řízený	Bartoníček	56
PZ/2015/056/063	adfgadfgdafg	Návrh pz	řízený	Bartoníček	56
PZ/Nevalidní	asdfasdfasf	Nový pz	řízený	Bartoníček	56
PZ/2015/056/69	asdgsdfasf	Návrh pz	řízený	Bartoníček	56
	fhsdfgadfg	Návrh pz	řízený	Bartoníček	56
PZ/2015/056/71	sdfgsdfg	Návrh pz	řízený	Bartoníček	56

Fig. 5 – Wireframe: User Story 5.2.1 list of public tenders

4.2.2 Administration

User story: user log-in and log-out.

Technical documentation: the basic menu was viewed. The top bar was still handling the main section, while the menu of each section was a part of the main view. The list of the user was added within the scope of the sprint (when possible).

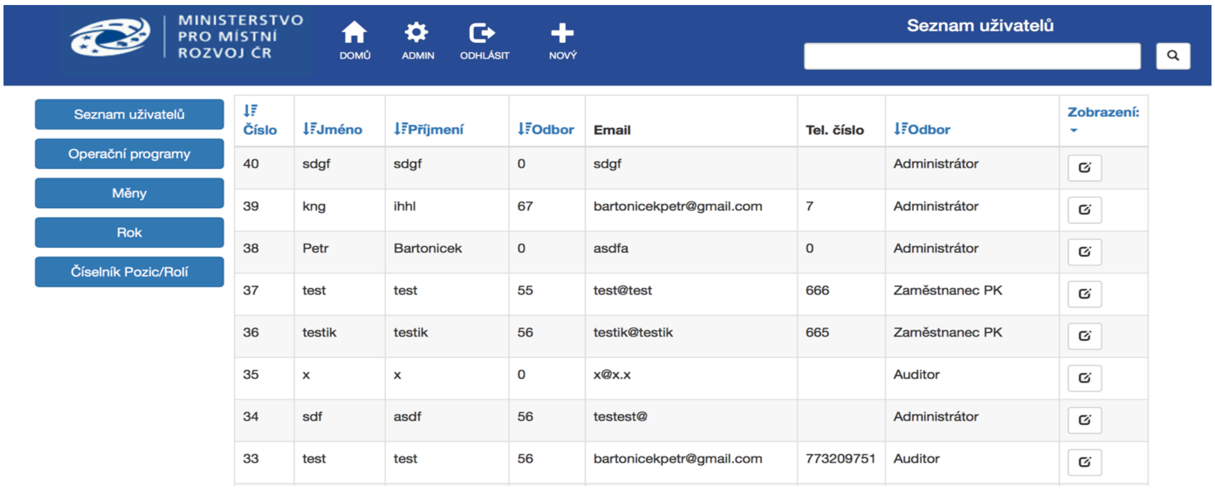


Email:

Heslo:

Přihlásit

Fig. 6 – Wireframe: User Story 5.2.2 login window



Číslo	Jméno	Příjmení	Odbor	Email	Tel. číslo	Odbor	Zobrazení:
40	sdgf	sdgf	0	sdgf		Administrátor	
39	kng	ihhl	67	bartonicekpetr@gmail.com	7	Administrátor	
38	Petr	Bartonicek	0	asdfa	0	Administrátor	
37	test	test	55	test@test	666	Zaměstnanec PK	
36	testik	testik	56	testik@testik	665	Zaměstnanec PK	
35	x	x	0	x@x.x		Auditor	
34	sdf	asdf	56	testest@		Administrátor	
33	test	test	56	bartonicekpetr@gmail.com	773209751	Auditor	

Fig. 7 – Wireframe: User Story 5.2.2 optional for sprint list of the users

4.3 Implementation

4.3.1 Server Installation

First and foremost, the server was downloaded. For the purposes of the thesis, CentOS 7 was used as the server. The code below was the code that ran the update and installation of Ruby 2.2.2. This was the version that was used for the development of the thesis.

4.3.1.1 Linux Update and Upgrade

```
$ yum update
$ yum upgrade
$ reboot
```

4.3.1.2 Ruby Installation

```
$ yum install libffi-devel openssl-devel libedit-devel gdbm-devel readline-devel
$ cd install
$ wget http://cache.ruby-lang.org/pub/ruby/2.2/ruby-2.2.2.tar.gz
$ tar xzvf ruby-2.2.2.tar.gz
$ cd ruby-2.2.2/
$ ./configure
$ make
$ make install
$ exit
```

4.3.1.3 Optional Setup Proxy in Nano Editor

```
$ nano /etc/profile.d/proxy.sh
export http_proxy=http://[IP:Port]
export https_proxy=http:// [IP:Port]
export ftp_proxy=http:// [IP:Port]
# Ctrl + O, name proxy.sh
# Press Enter
# Ctrl + O, name proxy.csh “save under different name / press Y”
# Ctrl + X
# Press Enter
$ exit
$ reboot
```

4.3.1.4 DB PostgreSQL 9.4 Installation

The DB was installed and the DB service started on Port 5432 if the port was free, otherwise the next free and iterating port was chosen. Creation of the user in the DB and installation was done in the Ruby application.

```
$ su
$ yum install http://yum.postgresql.org/9.4/redhat/rhel-7-x86_64/pgdg-centos94-
9.4-1.noarch.rpm
$ yum install postgresql94-server postgresql94-contrib postgresql94-devel
$ /usr/pgsql-9.4/bin/posgresql94-setup initDB
$ nano /var/lib/pgsql/9.4/data/pg_hba.conf
Change all “ident” with “md5”
$ systemctl start postgresql-9.4.service
$ systemctl enable postgresql-9.4.service
$ su – postgres
$ createuser –s [user name]
$ psql
postgres=# \password [user password]
postgres=# \q
$ exit
$gem install pg --pre -- --with-pg-dir=/usr/pgsql-9.4/
$ exit
```

4.3.1.5 Web Server

Apache web server was already installed in CentOS, even in the basic installation. The application server was installed and the Ruby on Rails development/production environment had full setup.

4.3.1.6 Application Server

The passenger was installed to Apache as an application server and module, but this was already in place thanks to the installation of the CentOS server.

```
$  su
$ gem install Passenger --pre
$ yum install libcurl-devel httpd-devel apr-devel apr-util-devel
$ passenger-install-apache2-module
$ nano /etc/httpd/conf/httpd.conf
```

4.3.1.7 Node.js

Additionally, it was necessary to also have Node.js installed on the server, for some of the rail’s functionality as a partial DOM rendering. Thanks to the specific DOM rendering, the application felt like an SPA.

```
$ su
$ yum install nodejs
$ exit
```

4.3.1.8 Versioning Control System Git

The installation of Git to the server was automated during development. Git was initialized in the folder with the application and the current version was obtained from the production. The development server was possibly duplicated and served as a pre-production and production server. Git push moved everything.

```
$ yum install git
$ git --version
$ sudo adduser git
$ su git
$ cd
$ mkdir .ssh && chmod 700 .ssh
$ touch .ssh/authorized_keys && chmod 600 .ssh/authorized_keys
```

4.3.1.9 SE Linux – Basic Setup

The Apache server was in the /var/www folder, and it started SE Linux and added Postgres as the Postgres daemon. It was the only one allowed to communicate out of the core of CentOS Linux.

```
$ getenforce
$ setenforce 1
$ getsebool httpd_can_network_connect_db
$ setsebool –P httpd_can_network_connect_db on
```

4.3.2 Environments and Continuous Integration

4.3.2.1 Three Environments (Dev, PreProd [Pre-Production], and Prod [Production])

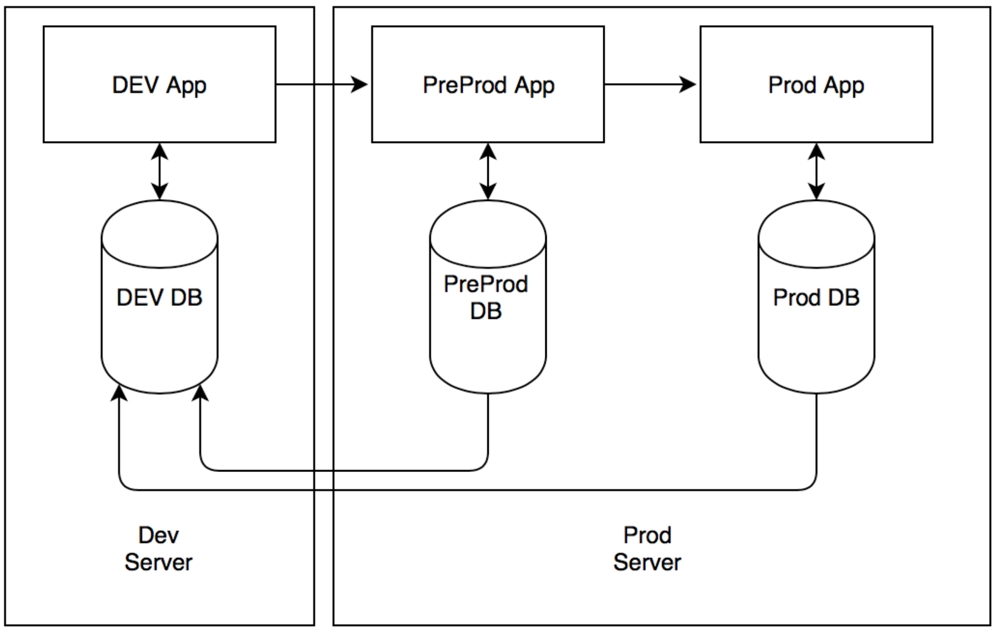


Fig. 8 – Three environments for the application

The application was composed from the developer server, which was any Mac/PC with Git. There was a setup for all three environments in the application. The case described a scenario where only Git was used to deploy everything to the server. The server only needed a manual reboot in case of an error. Since the environment for production and pre-production were the same, there were not be any issues with migration. The databases were migrated manually for development purposes. The application was developed incrementally and there was no intention to show real data to the testing team. There was however, motivation by the Scrum methodology that the team would create enough testing data in order to test what was delivered on their own behalf.

4.3.2.2 Continuous Integration

Once the code was tested in the pre-production environment, it was deployed to the production process. The rails framework offered $ db:migration and moved the metadata related to the DB of the application. The continuous integration only used the Git and Rails command line interface, which deployed everything without stopping the application.

4.4 Development

The development of the app from the moment of creation showed the basic bootstrapped scaffold generated view. The generated scaffold were user-finalized stories "Public Tenders" and "Administration."

4.4.1 Creating a New App

```
$ rails new PublicTenders --database=postgresqlU
```

New app generation dealt with the PostgreSQL database. To be aligned with the CI, a file /config/database.yml from the root app structure needed to be updated. The database YAML file held the configuration of all environment databases. The file names and credentials were obtained from the environment of the system. This helped secure the app, even if someone compromised the server and obtained the code, he or she would not be able to get the credentials without root access or access to a specific user.

```
production:
  secret_key_base:
  database:
    :host: localhost
    :name: ENV[‘PROJECT_AIMS_USER_DB_NAME]
    :username: ENV[‘PROJECT_AIMS_USER_NAME’]
    :password: ENV[‘PROJECT_AIMS_PASSWORD’]
```

The Git repository was initialized after setup was completed, which prevented passwords from being exposed.

```
$ git init
$ git commit –am “Initial commit”
```

4.4.2 DB Modelling with ORM Active Records

Modelling the DB was done with Ruby on Rails. Active records provided a layer that worked securely with the data in the DB and provided a robust platform that applied the changes to the DB.

4.4.2.1 DB Creation and Migration

Rails provided an out-of-the-box CLI tool that administrated the DB. The Rake tool created the DB and migrated everything. At that point, the first story had a basic application that was built on. This finished the application implementation and provided the first valuable product.

```
$ rake db:create
$ rake db:migrate
```

The application was secure with basic DB objects. The Devise gem was used for user password encryption, decryption, and authentication. Devise also implemented authorization, which was used in the model, controller, and view. The entire implementation was secure. Devise was also added to the gem file /Gemfile, where the names and versions of all the gems were stored. After that, it was installed with the $ bundle.

```
$ bundle install
$ rails generate devise:install
```

For the authentication, Devise generated the model. The file called /DB/migrate/[timestamp_action_modelName].rb was updated so it had all of the fields that were needed for the User model in the story.

```
class Users < ActiveRecord::Migration
  def change
    create_table :users do |t|
      t.string :first_name
      t.string :last_name
      t.string :email
      t.integer :phone_number
      t.integer :department
      t.string :password_digest
      t.timestamps null: false
    end
  end
end
```

```
$ rails generate devise User
$ rake DB:migrate
```

The solution offered a simple environment that set up a wise security authentication. Devise automatically crypted all of the passwords and handling sessions; and provided complete log-in, log-out, lost password, and notification functionality.

4.4.3 Scaffolds

In terms of MVC, scaffolding allowed the frame of any application to be created. The scaffold generated models /app/models/, view /app/views/ and controller /app/controllers/ automatically. In Rails, this was extremely powerful because it also created routes, and everything was prepared automatically. Scaffold could be changed and automatically generated the template with bootstrap, which sped up development.

```
$ rails generate scaffold ProjectTender name:string subject:string
price_expected:string authority:integer investment_source:string
password_digest:string real_start_date:string active:Boolean
```

Rails used scaffolds and custom scaffolds were a set of tools in the Devise framework. The installations of devise:view and devise:controllers completed the Administration story and provided the UI and back-end for the full authentication of the user.

```
$ rails generate devise:view User
$ rails generate devise:controllers Users
```

The Devise controller generator also created namespace users, where all authenticated routes went. In the generated scaffold ProjectTender; model, view, and controller were changed and used user namespace, which was created with Devise. Routes were also changed manually. Each was evaluated and the value of user_sign_in? was checked. If there was a false user; he or she was redirected back and had to log in.

4.4.4 Routes

The file /config/routes.rb contained basic set-up for the routes. The Rails framework route was defined based on the name of the controller and view. Routes thus provided setup for request/response-allowed actions, as well as namespace for login users.

```
Rails.application.routes.draw do

  get    'login'     => 'sessions#new'
  post   'login'     => 'sessions#create'
  get    'logout'    => 'sessions#destroy'
  delete 'logout'    => 'sessions#destroy'
  get    'user'      => 'static_pages#user

  resources :users
  namespace :user do
        resources :projectTenders, only: [:new, :edit, :create, :update]
  end
end
```

4.4.5 ERB and Bootstrap

ERB provided robust mark-up that extended the HTML file and rewrote variables in the HTML template stored in the view. The following code shows how a variable name was created, which was sortable and had the label “Name.”

```
<%= sortable "name", "Name" %>
```

4.4.5.1 ERB

Log-out linked the ERB and created the links automatically, which routed the application log-in. With the small helper function, whole logic was moved out and reused on each page as a dynamic header icon.

```
def login_logout
    if current_user
```

```
      link_to "<span class='glyphicon glyphicon glyphicon-log-out'></span>< >Log-
out </p>".html_safe, logout_path, title: "Log-out"
    else
      link_to "<span class='glyphicon glyphicon glyphicon-log-in'></span><p
style='font-size:40%; margin-left: -17%'> Log-in</p>".html_safe, login_path,
title: "Log-in"
    end
end
```

With helper, the function was accessible with only the function name. All of the helpers followed the naming of the view in the application. Helpers were stored in /app/helpers/.

4.4.5.2 Bootstrap

Bootsrap was added with a gem called Gem 'bootstrap-sass', '~> 3.3.6'; this gem was written in SaSS so it was allowed to write SaSS instead of CSS. This was actually good for development; however, it had to be compiled before every deployment. The production environment ran this as a CSS-compiled library. This was done automatically with CLI tool running command:

```
$RAILS_ENV=production bin/rails assets:precompile
```

All of the assets were stored in /app/assets/. The bootstrap was stored in /app/assets/stylesheet/ and it was called ERB in the header of all HTML templates.

```
<%= stylesheet_link_tag    'application', media: 'all', 'data-turbolinks-track' =>
true %>
```

For the bootstrap to be included, SaSS variables were added to the file application.css in /app/assets/stylesheet/.

```
@import "bootstrap-sprockets";
@import "bootstrap";
```

This provided standard bootstrap components accessible from anywhere in the views. For the easiest and fastest deployment/development for the team, standard bootstrap was used.

4.4.6 JavaScript

JavaScript was added in the same way and the application used Turbolinks that served the assets to the exact elements that needed them. The link was included in the header as it was for CSS. The path where ".js" files were stored was /app/assets/stylesheet/.

```
<%= javascript_include_tag 'application', 'data-turbolinks-track' => true %>
```

The thesis included much literature about JavaScript; however, there was no reason for it to be used. JavaScript code was an inevitable option for UI specific behaviour, which is why it was used in the application on the client browser.

5 Results and Discussion

The results were provided in the form of a SWOT analysis and the discussion was regarded as a comparative analysis on the theoretical basis for the methodology as Waterfall and Scrum with Agile environment in mind. SWOT compared the technology and feasibility within the environment of the Czech government. Both analyses provided a broad overview and understanding of the strengths and weaknesses for each chosen software.

5.1 SWOT Analysis of a Chosen Technology in the Public-Sector Body

The SWOT analysis described the overall solution and possibility of the implementation regarded to the theoretical part of the thesis. For the thesis goals to be fulfilled, the SWOT analysis provided an overview of the thesis technology and methodology. The SWOT analysis included the top ten vulnerabilities with respect to OWASP. [22] Since the White Hat security statistic report changed in 2014, it has compared the spread of security issues across web applications, instead of comparing languages. White Hat security reports offered a reliable source of data to be used as a reference in the SWOT analysis. [23] With respect to both of the reports for SWOT, Ruby on Rails was considered as a secure platform. The thesis observation and obtained knowledge showed how simple it was to be developed with Ruby on Rails.

	HELPFUL	HARMFUL
INTERNAL	STRENGTHS • Secure by design (Rails and Devise) • Easy to develop and extend • Great developer basement • Anybody can learn • All software is open source	WEAKNESSES • Low amount of developers available in the Czech Republic • No support from big consultancy companies such as IBM, Accace, KPMG
EXTERNAL	OPPORTUNITIES • Speed up development • Slowly re-educate the public sector • Automation through easiest language	THREATS • Public body has no experience with Ruby on Rails • Educating public employees is expensive • Support from community might disappear

Tab. 1 – SWOT analysis of the technology used in the thesis

5.1.1 Strengths

The development with Ruby on Rails was easy and secure due to the out-of-the-box design. It was rather simple to develop and re-use developed code — one of the Ruby on Rails mottos is: “don’t repeat yourself.” With the amount of developers in 2011 and 2012, Ruby had a similar number of gem modules as CPAN, pearl modules repository. Rails used

Ruby brilliant syntax and created a domain specific language, which was easy to learn and use. This may have helped educate the public sector. Software used with Ruby on Rails has always been always free.

5.1.2 Weaknesses

There has been a lack of developers for Ruby on Rails in the Czech Republic. Basically, none of the big consultancy and software companies have supported open source software. As part of the implementation, there was a consideration for the creation of the team, especially for Ruby on Rails.

5.1.3 Opportunities

There has been opportunity to drastically speed up the development of custom applications in the public sector and to help grow new paths for the government, with an attitude to slowly re-educate people in the public sector. The effectivity of processes may have increased, and support may have been transferred to an internal service as part of an implementation of free software. Automation has been the future, and in order to save time and lower risks, people must begin with the most readable language integration.

5.1.4 Threats

The public body has had no experience with open source in the public sector. The last large project was supposed to be EET; however, this was handed to IBM. It might sound like an impossible task, but in order to change the technology and re-educate people, it must be agreed upon across entire the government body. Regarding the spread of Ruby on Rails framework, there has been a low likelihood of Ruby implementation as the main open source language in the public sector. No one may surely say what the future of the framework is, or when the core team will stop working on it.

5.2 Comparative Analysis of the Methodology

Comparative analysis was used as a broad term across scientific fields. The definition of the comparative analysis for the thesis was: "Comparing item by item with the alternates." The comparative analysis used the Agile manifesto as a reference for Agile practice. [26] The reference for the Waterfall methodology [25] was from the wiki.c2.com community, which has operated since 1995. The page focuses on people, projects, and patterns among software development.

5.2.1 Agile vs. Waterfall Methodology

Agile manifesto principles were methodologically compared to the Waterfall from a theoretical point of view. The comparison provided possible answers from the Waterfall methodology to the Agile manifesto. The path of Agile obviously showed a different approach to letting people decide. The mindset in both methodologies was greatly different. Agile was customer-oriented while Waterfall was more product-oriented. The pattern also showed that Agile was not fit for projects that were expected as an outcome fixed deadline and finalized documented product without feedback.

Agile Manifesto Principles	Waterfall Principles
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software. [26]	Highest priority is on finishing the task within the scope, without feedback. This might extremely slow down the project.
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. [26]	It is almost impossible to do the changes late in the project due to high value of these changes it is better to implement a workaround.
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. [26]	Development is going step-by-step so there is no possibility for the shorter timescale.
Business people and developers must work together daily throughout the project. [26]	Everything is prepared and nothing blocking the developers, testers and analytics from work in defined timeframe.
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. [26]	Every person is replaceable and with well documented project you do not necessarily need to use top people. People can be changed and the project will be not affected. Maintenance of the project is done by management rather than team members.
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. [26]	There is clearly defined work to be done so no interaction such is needed and seniors might full their roles of high-effective developers, testers, analysts.
Working software is the primary measure of progress. [26]	Slow increment is helping to build whole difficult solution and be delivered until the deadline.
Agile processes promote sustainable development. The sponsors, developers, and users should beable to maintain a constant pace indefinitely. [26]	Whole development is done for fix time frame and price is well known to everyone. No changes are trade-offs of well documented project.
Continuous attention to technical excellence and good design enhances agility. [26]	Everything is defined by standards from the beginning to the end of project and nothing might be change. This provide the sustainability and maintainability of the project.
Simplicity--the art of maximizing the amount of work not done--is essential. [26]	Every detail has to be considered in order to deliver whole solution as defined.
The best architectures, requirements, and designs emerge from self-organizing teams. [26]	Architecture and all requirements are defined before the work starts.
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. [26]	Manager is controlling the performance and adjusting it accordingly.

Tab. 2 – Comparative analysis of the Agile manifesto vs. Waterfall approach

The analysis on the previous page provided the overview of differences in Agile and Waterfall. This analysis clearly shows the fact that Agile and Waterfall have different management attitudes, form teams, and management. The Agile team has been driven by the people within the team, which has been helping the whole organization. The Waterfall team only takes care of tasks, from start to finish. Agile stakeholders only pass their wishes on to the team, and the team creates tasks with responsibility within the team. Waterfall stakeholders give the budgets to the managers, which are supposed to drive the team and be responsible for the tasks. Only managers are responsible in Waterfall — not the team itself. The mindset of the team and management in both approaches has to be changed. Waterfall has a defined deadline and deterministic path. Agile offers an open-minded path to the goal.

6 Conclusion

The main aim of the thesis was to investigate and describe application of the CentOS server with Ruby on Rails. The literature review provided broad knowledge on the topic and went through the development, implementation, and technology for the thesis. The practical part solved the implementation and development of the Use Cases from the Scrum methodology. The result and discussion part covered the implementation, deployment, and development with two analyses. SWOT analysis covered the chosen technology and comparative analysis.

The first partial objective was to create an overview of current web technologies such as Ruby on Rails and JavaScript. The overview was done in the literature review and provided the basic building blocks for the practical part.

The second partial objective was to describe a whole deployment solution for the client JS and server side application, as well as a development environment for the application itself. The description of such application was mentioned in the literature review. However, the application in the practical part was written as MVC, without separated servers for the front-end and back-end. It was decided that care of security without JavaScript was much easier.

The third partial objective was to design a functional solution of an application for a selected government body, and to create a comparative analysis. The design of the application was done in the practical part. Comparative analysis was part of the result and discussion.

With respect to the main and partial goals, the conclusion of the thesis is as follows — "The challenge was not just to train people and let them develop, but rather to change the mindset of people. When using Ruby on Rails as an open source framework on the CentOS server with agile methodology, creating the path for future development in a public sector is rather idealistic and unfeasible."

The result was especially useful for the common public sector. A public body that implements any software, must have an overview of the challenges, in order to implement any web applications using open source by their own employees.

7 References

[1] BLACK, David. Well-grounded rubyist. 2. S.l.: O'Reilly Media, 2014. ISBN 978-161-7291-692.

[2] CentOS 7 Linux Server Cookbook. Second edition. US: Packt Publishing, 2016. ISBN 978-1785887284.

[3] HARTL, Michael. Ruby on rails tutorial: learn web development with rails. Third edition [online]. New York: Addison-Wesley, 2015 [cit. 2016-09-12]. ISBN 01-340-7770-9. Available at: http://3rd-edition.railstutorial.org/book.

[4] Mastering CentOS 7 Linux Server. 1. US: Packt Publishing, 2016. ISBN 978-1785282393.

[5] METZ, Sandi. Practical object-oriented design in Ruby: an agile primer. 1. Upper Saddle River, NJ: Addison-Wesley, 2013. ISBN 03-217-2133-0.

[6] PUGLISI, Silvia. RESTful Rails Development: Building Open Applications and Services. 1. US: O'Reilly Media, 2015. ISBN 978-1-4919-1085-6.

[7] RAHMAN, Syed. Bootstrap for Rails. 1. US: O'Reilly Media, 2015. ISBN 978-1-78398-727-6.

[8] *APIS ON RAILS* [online]. 2014. USA: Softcover, 2014 [cit. 2016-09-12]. Available at: http://apionrails.icalialabs.com/book/.

[9] HAVERBEKE, Marijn. *Eloquent JavaScript: A Modern Introduction to Programming*. 2nd Edition. USA: No Starch Press, 2014. ISBN 978-1593275846.

[10] OSMANI, Addy. *Learning JavaScript Design Patterns*. 1. USA: O'Reilly, 2012. ISBN 978-1-449-33181-8.

[11] COAR, Ken a Rich BOWEN. *Apache Cookbook*. 2nd edition. USA: O'Reilly, 2007. ISBN 978-0-596-52994-9.

[12] *Nginx HTTP Server*. Third Edition. UK: Packt Publishing Ltd., 2015. ISBN 978-1-78528-033-7.

[13] Centrální nákup softwarových produktů Microsoft & Oracle MVCR: Prezentace. *Ministerstvo Vnitra České Republiky* [online]. 2015, **2015**(6), 75 [cit. 2016-09-13]. Available at: http://www.mvcr.cz/soubor/prezentace-workshop-mv-se-zadavateli-produkty-ms-oracle-03-09-2015-v11-ppsx.aspx.

[14] *Zákon č. 111/2009 Sb., o základních registrech*. In: . Praha, CZ: Vláda ČR, 2009, ročník 2016, poslední změna – zákon 312/2013 Sb. Also available at: http://www.szrcr.cz/file/167.

[15] JUBA, Salahaldin, Achim VANNAHME a Andrey VOLKOV. *Learning PostgreSQL*. 2015. UK: Packt Publishing Ltd., 2015. ISBN 978-1-78398-918-8.

[16] HTML & CSS Standard. *World Wide Web Consortium (W3C)* [online]. USA: W3C, 2016 [cit. 2016-09-13]. Available at: https://www.w3.org/standards/webdesign/htmlcss.

[17] *JS standard ECMA-262*. 7th. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf: Ecma International, 2016.

[18] LeSS. *Large-Scale Scrum* [online]. US, 2005 [cit. 2017-07-12]. Available at: https://less.works.

[19] *Organizační struktura MMR* [online]. ČR: MMR, 2017 [cit. 2017-07-12]. Available at: https://www.mmr.cz/cs/Ministerstvo/Ministerstvo/Organizacni-struktura.

[20] *STATE OF SOFTWARE SECURITY: Focus on Application Development* [online]. 2015, **2015**(6), 20 [cit. 2017-01-13]. Available at: https://info.veracode.com/state-of-software-security-report-volume6-pt2.html.

[21] *OAuth 2.0* [online]. US, 2017 [cit. 2017-07-20]. Available at: https://oauth.net/2/.

[22] *OWASP Top 10 Most Critical Web Application Security Risks* [online] 2017, [cit. 2017-11-13]. Available at: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

[23] *WEB APPLICATIONS SECURITY STATISTICS REPORT 2016* [online] 2017, [cit. 2017-11-13]. Available at: https://info.whitehatsec.com/rs/675-YBI-674/images/WH-2016-Stats-Report-FINAL.pdf.

[24] *The Scrum Guide^{TM}* [online] 2017, [cit. 2017-11-13]. Available at: http://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100.

[25] *Wiki.c2 Water Fall* [online] 2017, [cit. 2017-11-13]. Available at: http://wiki.c2.com/?WaterFall.

[26] *Agile manifesto principles* [online] 2017, [cit. 2017-11-13]. Available at: http://agilemanifesto.org/principles.html.
e
[27] *First website online* [online] 2017, [cit. 2017-11-13]. Available at: http://info.cern.ch/hypertext/WWW/TheProject.html

8 List of Figures

9 List of Tables