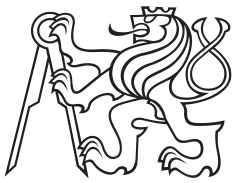
Master's thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering

Department of Computer Science

Animal Breeders Content Management System

Marek Polcar

May 2014

Supervisor: Tomáš Černý

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science and Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Marek Polcar**

Study programme: Electrical Engineering and Information Technology
Specialisation: Computer Science and Engineering

Title of Diploma Thesis: **Animal breeders content management system**

Guidelines:

Analyze breeders needs for publishing content on the Internet. Search for related work and similar existing systems. Based on the analysis propose a design and a structure that can capture animals' family relationships. Implement a content management system for breeders activity and perform all sorts of test tests including unit test, integration test, stress test, static code analysis and usability test.

Bibliography/Sources:

Christian Bauer and Gavin King. 2006. Java Persistence with Hibernate. Manning Publications Co., Greenwich, CT, USA.

Debu Panda, Reza Rahman, and Derek Lane. 2007. EJB 3 in Action. Manning Publications Co., Greenwich, CT, USA.

Diploma Thesis Supervisor: Ing. Tomáš Černý, MSc.

Valid until the end of the summer semester of academic year 2014/2015

L.S.

doc. Ing. Filip Železný, Ph.D.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, February 25, 2014

Acknowledgement

I would like to thank Tomáš Černý for being my supervisor.

I also need to thank my mother because she has been truly helpful throughout the entire work.

Last but not least, please allow me to thank the university for providing me with the priceless knowledge letting me create the system the thesis discusses.

Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 12. 5. 2014

.....

Abstrakt / Abstract

Tato diplomová práce pojednává o analýze, implementaci, nasazení a otestování služby, která umožní chovatelům zvířat publikovat svůj obsah na internetu.

Analýza ukázala, že chovatelé vynikají ve vytváření jedinečného a osobitého obsahu, avšak mají značné problémy s formou prezentace. Ta často postrádá logické členění a nesplňuje základní podmínky pro publikování obsahu na internetu. Chovatelé si tím, ač nevědomky, zbytečně zhoršují svou pozici při nabízení a prodeji odchovaných zvířat.

Vyvinutá služba ulehčuje chovatelům vkládání rozmanitého, chovatelsky specifického obsahu, přičemž se snaží zachovat jednoduché a přátelské uživatelské rozhraní.

Klíčová slova: systém pro správu obsahu, prototypování, responzivní web design, Maven, Java EE, PrimeFaces, Heroku, jednotkové testování, testováni použitelnosti

The goal of this thesis is to analyze, implement, deploy and test a service that will allow animal breeders publish content on the Internet.

The analysis shows that breeders excel at creating remarkable content but they miss a basic knowledge about the suitable form for it. The content they create is usually unlogically organized which makes it hard for a visitor to consume. In the end the breeders unknowingly harm their business by not being able to display their content in visually appealing and understandable way on the Internet.

The service that has been developed allows the breeders to publish various, breeding-specific content, but aims to keep the user interface simple and friendly at the same time.

Keywords: content management system, prototyping, mobile first, responsive design, Maven, Java EE, PrimeFaces, Heroku, unit testing, usability testing

Contents /

1 Introduction 1

1.1 Hypothesis 1

1.2 Executive Summary 1

2 Analysis 2

2.1 Main Use Cases 2

2.1.1 Puppies presentation 2

2.1.2 Choosing a Father for an Upcoming Litter 4

2.1.3 Keeping Animal's Diary 4

2.1.4 Having Complete Information about the Dog 5

2.1.5 Keeping Track of Dog's Descendants 5

2.2 Related Work And Existing Systems 5

2.2.1 Advertisement Portals 5

2.2.2 General Content Management Systems 6

2.2.3 Custom Made Website 7

2.2.4 Breeding Station Catalogues 8

2.2.5 Animal Pedigree Catalogues 8

2.3 Requirements 8

2.3.1 Priorities of Requirements 9

2.4 Use Cases 9

2.5 Domain Model 13

3 Software Design 15

3.1 Architecture 15

3.1.1 Programming Language 15

3.1.2 Web Server 15

3.1.3 Database Layer 15

3.1.4 Security 15

3.1.5 Hosting 16

3.1.6 Data Storage 16

3.1.7 Front-end Framework 16

3.2 Design Model 17

3.3 Graphic Design 18

3.3.1 Layout 18

3.3.2 Public Pages 20

3.3.3 Admin Pages 20

3.3.4 Responsive Design 21

4 Implementation 23

4.1 Encountered Problems and Solutions 23

4.1.1 Programming Language 23

4.1.2 Security 24

4.1.3 File Upload 24

4.1.4 Autosuggest PrimeFaces Component with Diacritics 25

4.1.5 JSF Composite Components 26

4.1.6 Markdown Support 26

4.1.7 Maven Spring Dependencies 27

4.1.8 Pedigree Recursive Construction 29

4.1.9 PrettyTime Integration 30

4.1.10 PrettyFaces Integration 30

4.1.11 Spring Beans View Scope Support 31

4.2 State Diagrams 32

4.3 Database Model 34

4.4 Screenshots 36

5 Deployment 39

6 Testing 41
6.1 Unit Testing 42
6.1.1 Test Driven Development 43
6.2 Integration Testing 44
6.3 Stress Testing 45
6.4 Usability Testing 46
6.4.1 Principles 46
6.4.2 Hypothesis 46
6.4.3 Testing 46
6.4.4 Results 47
7 Static Code Analysis 49
8 Future 51
8.1 Scope Extension 51
8.2 Additional features 51
8.3 Monetization 51
8.4 No Success Prediction 52
9 Conclusion 53
References 54
A Abbreviations 57
B Functional Requirements 58
B.1 Implemented 58
B.2 Not implemented 59
C Non-functional Requirements ... 60
D Use Case Scenarios 61
D.1 Station 61
D.2 News Section 62
D.3 Links Section 63
D.4 Guestbook 64
D.5 Animal 65
D.6 Litter 67
D.7 Custom Page 69
E Public Pages Content 71
E.1 About Us 71
E.2 News 71
E.3 Links 71
E.4 Guestbook 71
E.5 Contact 72
E.6 Animal Profile 72
E.7 Breed — (Fe)Males 72
E.8 Breed For Sale 73
E.9 Past Litters 73
E.10 Litter Profile 73
E.11 Retired 73
E.12 In Memory 74
E.13 Custom Page 74
F Admin Pages Content 75
F.1 Animals Admin 75
F.2 Litters Admin 75
F.3 Custom Pages Admin 75
G User Experience Test Scenario .. 76
G.1 Public Pages 76
G.2 Admin Pages 76
H CD Contents 78

Tables / Figures

6.1. Unit tests coverage 42

6.2. Usability problems 47

7.1. Source code statistics 49

7.2. Java code statistics 49

2.4. Advertisement Portal Item Example 6

2.5. General Content Management System Example 7

2.6. Custom Made Website Example 7

2.7. Breeding Station Catalogue Example 8

2.8. The project triangle 9

2.9. Actors in the system 9

2.10. Station use cases 10

2.11. News use cases 10

2.12. Links use cases 11

2.13. Guestbook use cases 11

2.14. Animal use cases 12

2.15. Litter use cases 12

2.16. Custom page use cases 13

2.17. Domain Model 14

3.1. Utilized technologies 16

3.2. Design Model 17

3.3. Page layout wireframe 18

3.4. Progressive Enhancement 19

3.5. Responsive Design Adaptation 21

3.6. Smartphone share of audience . 22

4.1. Cost analysis of good enough .. 23

4.2. Spring Data MongoDB 24

4.3. Autosuggest with diacritics 25

4.4. Autosuggest in frontend use ... 25

4.5. JSF composite component 26

4.6. Markdown support 27

4.7. Maven Spring dependencies ... 27

4.8. Spring core dependencies 28

4.9. Spring Data MongoDB dependencies 28
4.10. Spring Security dependencies .. 29
4.11. Pedigree recursive creation 30
4.12. PrettyTime JSF integration ... 30
4.13. PrettyTime frontend result 30
4.14. PrettyFaces and managed bean 31
4.15. PrettyFaces JSF link component 31
4.16. Spring beans view scope 32
4.17. Spring view scope integration . 32
4.18. Litter child state diagram 33
4.19. Animal state diagram 33
4.20. Litter state diagram..... 34
4.21. Database model 35
4.22. MongoDB animal document ... 36
4.23. About Us page screenshot 36
4.24. About Us page on mobile..... 37
4.25. Responsive menu screenshot ... 38
4.26. Sticky top bar screenshot 38
5.1. Deployment model 39
5.2. Deployment Schema 39
6.1. Cost to fix a bug 41
6.2. Testing Triangle 42
6.3. Parameterized unit test 43
6.4. Result of a unit test..... 43
6.5. Selenium test run 1st part 44
6.6. Selenium test run 2nd part 45
6.7. Stress Testing Chart 45

Chapter 1
Introduction

1.1 Hypothesis

There is no user-friendly and accessible solution on the Czech market that would allow breeding stations to advertise their services and products and facilitate business among various breeding stations on national and potentially international level.

1.2 Executive Summary

The proposed system targets two types of clients — breeding stations (professionals) and customers who want to buy a pet (non-professionals). Based on the research of current existing solutions and business needs of the breeding stations, proposed system will allow following:

Breeding stations:

- to create an easy-to-make, visually appealing Internet presentation
- to create a database of owned animals including their relationships to each other
- to create individual profiles for each animal
- to advertise the station/animals to potential buyers both from professional and non-professional community

Customers looking for a pet:

- to intuitively find all of the information about their potential pet
- to easily contact owners of the pets and facilitate the buying process

Considering the idea behind the proposed system is to be proven right, there is a potential for the system to become a standard for breeding purposes on an international level.

Chapter 2
Analysis

The proposed system is going to serve as a proof of concept of dogs breeding station content management system. Should the pilot be successful, business planning is to be done as well as probable scope extension of the system.

2.1 Main Use Cases

Proper understanding of real-live use cases is the essential thing when it comes to creating any kind of a new system. The main idea is that the proposed system must deal with the use cases in a smart unobtrusive way. Users must immediately fall in love with the new system and it's fresh functionalities because it is doing what they would expect it to do in a clear, friendly way.

2.1.1 Puppies presentation

The use case of presenting puppies is fairly simple. The potential buyer gets to know about certain breeding station by seeing the advertisement or some recommendation. Now it is time for the breeder to deliver more information about the puppies he has to offer to the buyer.

The best option for both sides is when the buyer is able to come over to breeder's house and see the whole litter along with the mother and also the environment that the puppies will be raised in for two months. In this scenario the breeder can explain everything the buyer needs to consider before buying one of the puppies. This case extremely well solves the use case of tranfering the much needed information from the breeder to a potential buyer. Unfortunately enough this option is not always feasible due to the distances and time cost of the whole process.

The Internet can certainly speed up the process. Since this paper describes the creation of a new web portal for now we will assume that the breeder doesn't have any website presentation on his or her own. [1]) But he or she needs to present how adorable the puppies are. A set of photos should solve the case. Most adopted way of sending pictures via the Internet in the Czech Republic is by using the email. So he or she sits down at the computer, selects the most recent photos, adds some comments about each of the puppies' characteristics. Estimated date of the offtake is also included and the `SEND` button is pressed. The breeder is happy because he or she satisfied the customer's thirst for the information and it took him or her only couple of minutes.

While this way of doing things is definitely feasible it is not very cost effective. Considering the breeding station is a popular one and has a lot of puppies for sale throughout

[1]) When creating a new system it is always the best to start from scratch. Core functionalities solving the use cases can arise more freely and be more accurate to the users' needs.

the whole year the activity of sending emails becomes boringly repetitive and needless to mention very time consuming. If only there was a way that would allow the breeder to compose the information along with attached photos only once. This forms to a very nice use case that the proposed system should handle.

Figure 2.1. Litter state `planned`.

A litter becomes reality even before a single puppy is born. Breeding is a bussiness as any other hence some planning is required. The breeder knows mating times of all his or her bitches. Although it might not be definite who the father of the litter will be at the time of planning it does not really matter at this time fo the process. [1])

Requirement: Allow the user to make a planned litter with none, one or both parents and estimated date.

Figure 2.2. Litter state `current`.

The puppies are just born. There is not much to tell about them at this stage because they are simply too young. Perphaps except of their sex and color. The breeder still wants to share the happy event with the world so he or she needs to be able to say a few things about the litter along with the photo of the mother and her children sucked in at her milk-giving breast.

Requirement: Allow the user to change the state of a litter to `current`.

Requirement: Allow the user to compose text information talking about the litter as a whole as well as to attach pictures.

As the children get bigger and older the breeder needs to capture their weight, looks and slowly morfing personality. There is also need to tell the world whether the puppy is for sale, reserved or not for sale at all if the breeder decides to keep it.

Requirement Allow the breeder to set puppy's state within the selling process: `for sale`, `reserved`, `sold`, `not for~sale`. All of the possible states are pretty much self explanatory. Although special behaviour is observed when a child is sold. Until then the breeding station is the owner of the puppy. But selling the puppy of course means changing the ownership of the animal. Two situations can happen now. Either the buyer is an owner of another breeding staition or he is a person having the dog as a pet.

Requirement Allow the breeder to transfer the ownership of a puppy to either a breeding station or a person.

Figure 2.3. Litter state `archived`.

[1]) Looking for a proper father will be further discussed in next section.

Everything goes well and all of the children are either sold or not for sale at all. That means that the breeder can close the sellling phase and archive the litter.

Requirement: Allow the breeder to change the state of a litter to `archived` when all of the children are either `sold` or `not for~sale`.

The buyer might realize that purchasing a dog pet was not such a good idea and wants to return it back to the breeding station. Or he or she does not want to pay the repayments for it. As the breeder gets back the puppy in reality it also requires transfering the ownership back to the original breeding station. Assuming the buyer is not capable of doing it for whatever reason the breeder has to be able to do it.

Requirement: Allow the breeder to reclaim an ownership of already sold animal back to his or her breeding station without requiring permition of the buyer.

Or the buyer decides that he or she will rather sell the unwanted animal to some other person or breeding station.

Requirement: Allow the buyer to transfer an animal to another person or a breeding station.

2.1.2 Choosing a Father for an Upcoming Litter

It is very important for the breeder to have the best matching male possible for the bitch when it comes to mating. The genes play a huge role when deciding about the parenthood combinations. And of course the temporary spouses cannot be family related. This all means that the breeder often needs to look around a bit for a proper male.

Usually there is a list of stud males available for mating purposes at the website of the particular dog breed cynological club. There are two problems with this source of information. Firstly, although the animal listing could contain some information about the male dog it doesn't offer the family structure of ancestors that the person who looks for the proper dog would appriciate.

Secondly, to be enlisted in club's website the owner needs to contact the organization by himself and give them the necessary information along with a photo of the male dog. Which is obviously redundant because he could simply have all the information just in place at his or her breeding station website. Needless to say that he or she would need not to forget about updating the dog's information everytime the male gets a new reward or wins prestigious show.

2.1.3 Keeping Animal's Diary

There are two applicable use cases when keeping animal's diary comes in hand. The first case considers a buyer getting a puppy from a breeder. The happy buyer after some time surely wants to show how the dog had grown up and is enjoying the family vacation at the sea coast. So he sends an email describing the activity they had been doing with some pictures attached. The breeder is also happy because he or she can see that the animal is doing well. This positive feedback also happens to be the best content for breeding station's website. But now he needs to resize the photos so that they look good on the website and also upload and edit the whole thing. If only there was a way a buyer would be able to do the hard work of creating the content just once. And that is exactly the purpose of keeping dog's diary. Ideally the content then automatically appears on the breeder's website as a handy reference.

The second case is for breeder's need of presenting the updates about the breeding station. Usually the news section of a website is used for mentioning all the successes the dogs had received at the shows. But then the information gets lost with the time moving on. Needless to say it has no actual connection to the animal so anyone looking at the dog's profile is unnecessarily missing this information.

2.1.4 Having Complete Information about the Dog

Animal profile would not be complete without displaying the parents and a date of birth.

Requirement: Allow the breeder to set animal's parents and a birthdate.

There could be quite a lot of various information that a breeder would like to insert about the animal:

- animal's nature
- it's brief history
- list of awards won
- list of examinations passed

Requirement: Allow the breeder to create a general content describing the animal.

2.1.5 Keeping Track of Dog's Descendants

The more an owner of a male dog is presenting at his or her website the better. The saying "A picture is worth thousands words." also applies in this situation. The owner of a male is not able to keep the animal's profile page up to date with recent photos of it's children.

Requirement: List all the children and litters on the animal's profile page.

2.2 Related Work And Existing Systems

No complex solution has been found during the research that would solve all the uses cases that the breeder is dealing with. Hence this section talks about the options that each covers at least some of the use cases and is widely used amongst breeders nowadays.

The outcome of this section should be that the proposed system would take the best out of all the various options.

2.2.1 Advertisement Portals

There are two webportals [1]), [2]) in the Czech Republic that have the biggest market share when it comes to publishing ads of animals for sale. The biggest advantage of presenting the offer on such a site is that these sites tend to have great Search Engine Optimization (SEO). The result of that is when a potential buyer searches for a puppy using Google or any other search tool the advertisement webportal will typically be on the first page of the search results which gives the seller some certainity that people will find the puppy.

[1]) `http://www.bazos.cz/`
[2]) `http://www.ifauna.cz/psi/inzerce/r/`

Figure 2.4. Advertisement Portal Item Example

On the other hand they both suffer from more or less the same bad approach for selling children of animals. To put it simply a puppy is not like a chair. It grows with age. It has got parents and siblings. All of that information could be easily lost within one single advertisement. Additionally, with every site a breeder decides to publish an advertisement on the amount of time the breeder needs to spend if keeping the photos and information up to date increases.

The proposed system should make a big effort to accomplish the same great results for it's content to be displayed within the first pages in search engines. It should also embrace the ease when creating the offer that comes with understandable interface and clear form of filling the information.

2.2.2 General Content Management Systems

Since there is nothing like the proposed system the breeders tend to make use of general content management systems. They usually come for free. They are great for creating general content that does not change very often and does not have some sort of logical structure. The problem is that these systems are not sufficient for breeders needs.

Breeders are forced to come up with their own consistent form of structuring the content. Futhermore they have to stick with the created form and put a lot of time and energy to keep the content nice and organized. It comes as no surprise that most of the breeders are simply not capable of doing so.

Figure 2.5. General Content Management System Example

The lesson learnt here is that the proposed system should be free for all in some basic version. It should also allow breeders to easily create content without having them worry about the form that the content will be displayed in. The system will make sure that all of the information a breeder had put in is consistent and well organized letting the breeder stay focused on what is most important for him or her — the content itself.

2.2.3 Custom Made Website

Some breeders realize that having a professional web presentation is one of many things that undoubtedly help the business. They have the website created with some sort of custom content management system. But no breeder would go that far that he or she would order and pay for creating a system with the functionalities as the proposed system will have just for his or her single breeding station. In fact the breeder might not even realize that he or she could benefit from such a system.

Figure 2.6. Custom Made Website Example

Breeders usually buy an internet domain along with a custom made web presentation. The domain name reflects the name of the breeding station and acts as a form of professionalism. The proposed system should also allow a breeder use own custom domain or at least have some sort of Uniform Resource Locator (URL) branding in place.

2.2.4 Breeding Station Catalogues

There are couple of websites [1]) [2]) that try to collect all the breeding stations. They usually provide a filter to sort the stations by breed or location. The breeding station gets a chance to describe it's activites as well as insert it's contact information and some pictures.

Figure 2.7. Breeding Station Catalogue Example

It is not really sufficient to use such a service as a full presentation of a breeding station but could serve well when improving SEO by building backlink structure. It also might bring some visitors to the owner's website from such a service.

2.2.5 Animal Pedigree Catalogues

A pedigree database[3]) is only one use case solving service. It is missing an essential context information about the animals. Furthermore a breeder needs to create the entry manually and make the animal information redundant since it is already present on the breeding station's website.

2.3 Requirements

Functional requirements describe what the system must do. They are derived from the real-world user stories. Some were mentioned earlier in the Analysis chapter 2. There is a complete list in the appendix B.

Non-functional requirements are often called qualities of a system. Those for the proposed system are enclosed in the appendix C

[1]) `http://www.celysvet.cz/seznam-chovatelske-stanice-psu.php`
[2]) `http://www.hafici.cz/psi/chovatelske-stanice-psu/`
[3]) `http://www.pedigreedatabase.com/`

2.3.1 Priorities of Requirements

The proof of concept nature of the proposed system tells us that the requirements need to have priorities assigned because there is only limited time and resources available for the project. A Venn-diagram 2.8 style chart of the `project triangle` shows visually the potential overlaps between speed, quality and low cost, along with the inability to accomplish all three indicates the bottomline that the quality of the project is the variable that will be shifted in time. As for the proposed system matters that boils down to fewer requirements being implemented.

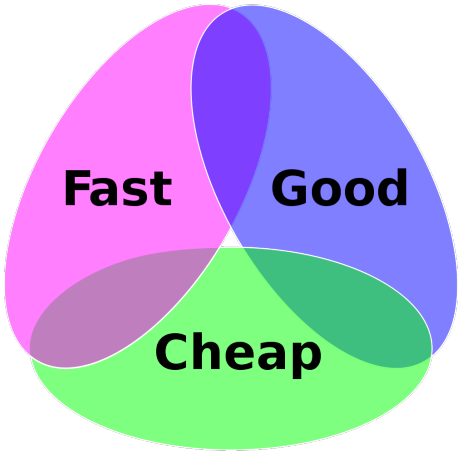


Figure 2.8. The project triangle as a "pick any two" Venn-diagram.

`Fast` refers to the time required to deliver the product, `Good` is the quality of the final product, and `Cheap` refers to the total cost of designing and building the product. This triangle reflects the fact that the three properties of a project are interrelated, and it is not possible to optimize all three – one will always suffer.

2.4 Use Cases

As Craig Larman says in his book [1], a use case diagram is an excellent picture of the system context; showing the boundary of a system, what lies outside of it, and how it gets used. It serves as a communication tool that summarizes the behaviour of a system and its actors.

The figure 2.9 shows the actors in the system. The `Guest` is just a random visitor. The `Admin` is usually a breeder.

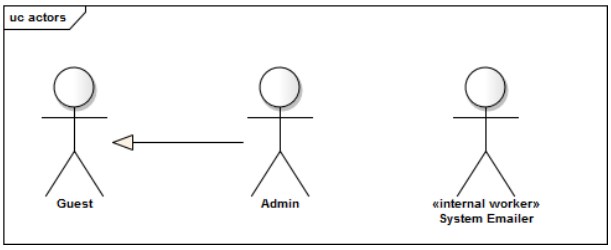


Figure 2.9. Actors in the system

In the following figures primary actors are located on the left side whereas supporting actors are shown on the right side. The figures 2.10, 2.11, 2.12, 2.13, 2.14, 2.15, 2.16 show the use cases for the breeding content management system. Models put together related use cases. Each model consists of system boundaries and actors who are associated with use cases. The use cases cover all the functional requirements and define the relationships between the users and the system itself.

There is a complete scenario list in the appendix D.

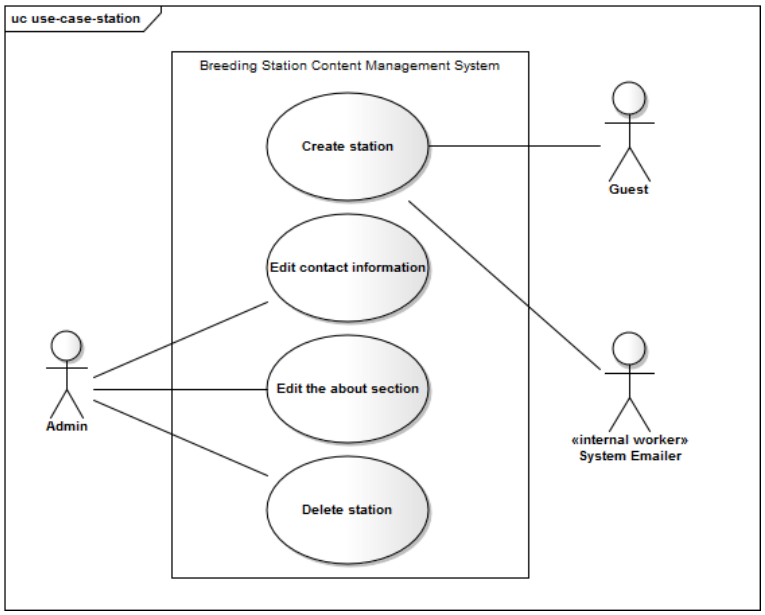


Figure 2.10. Station use cases. Scenarios are appended in D.1

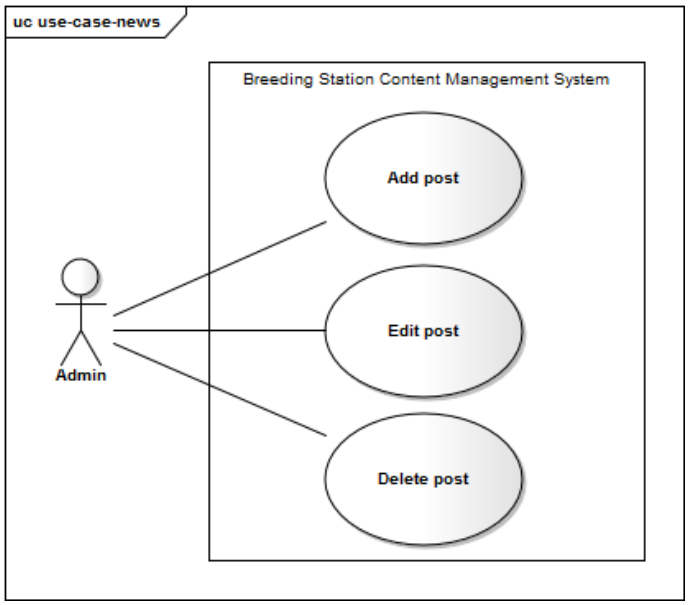


Figure 2.11. News use cases. Scenarios are appended in D.2

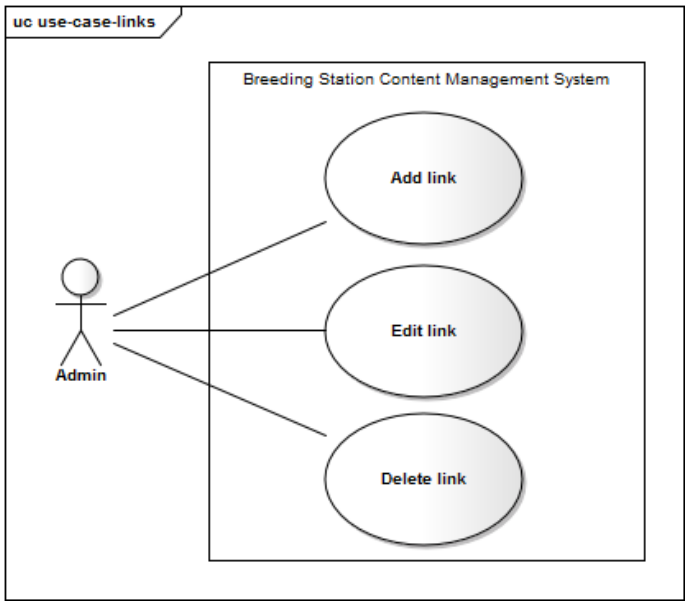


Figure 2.12. Links use cases. Scenarios are appended in D.3

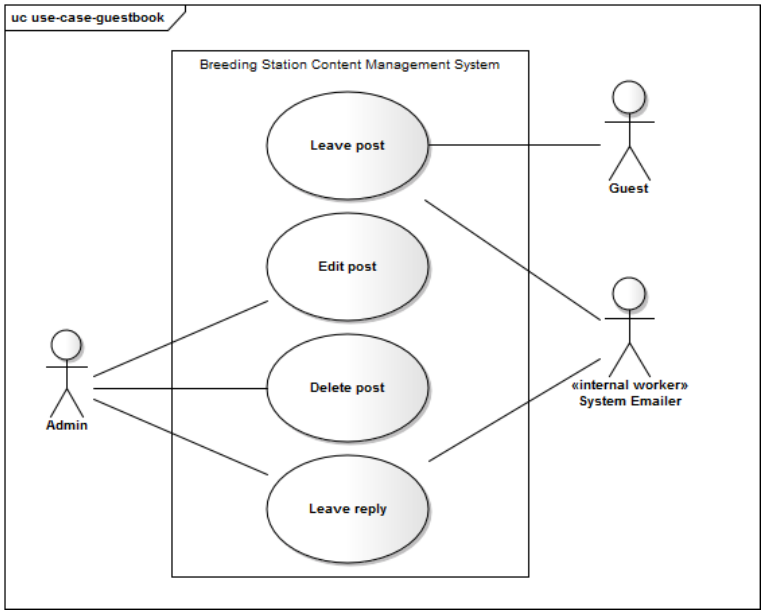


Figure 2.13. Guestbook use cases. Scenarios are appended in D.4

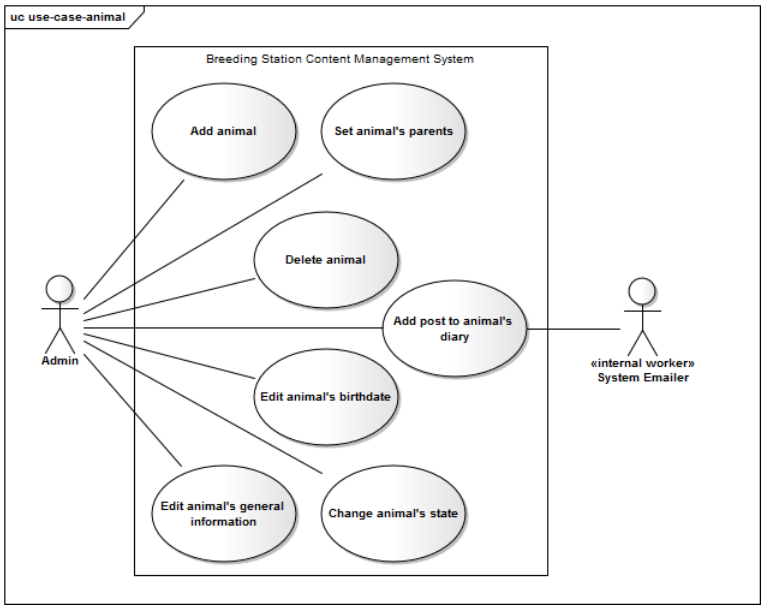


Figure 2.14. Animal use cases. Scenarios are appended in D.5

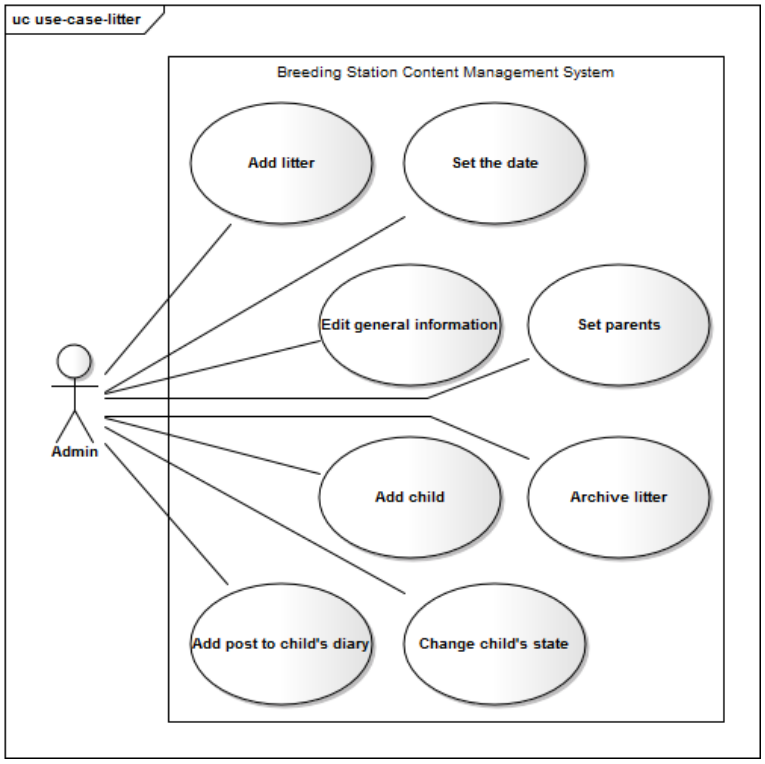


Figure 2.15. Litter use cases. Scenarios are appended in D.6

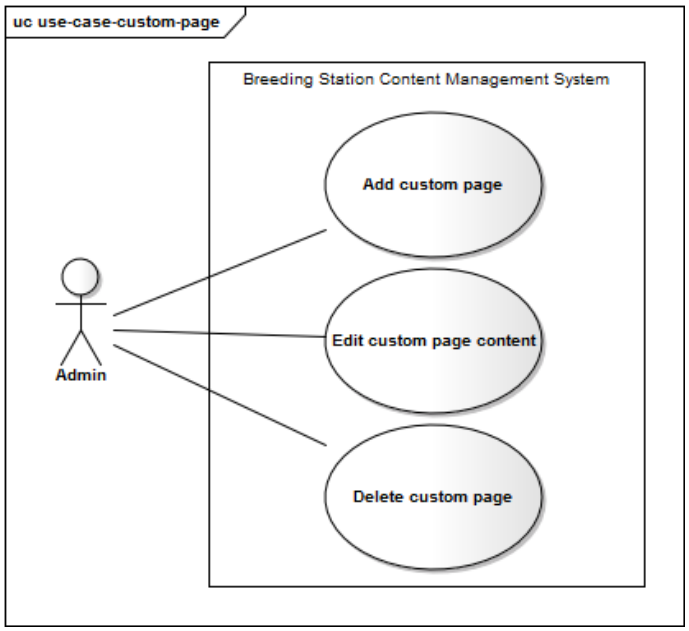


Figure 2.16. Custom page use cases. Scenarios are appended in D.7

2.5 Domain Model

Domain model consists of classes and their attributes. Those are based on the important verbs and nouns identified in requirements and use case work.

The figure 2.17 shows that the class `Station` is the essential one. Everything else is tied to it. It also has got the 1:1 relationship with the `User` which represents a breeder in the system.

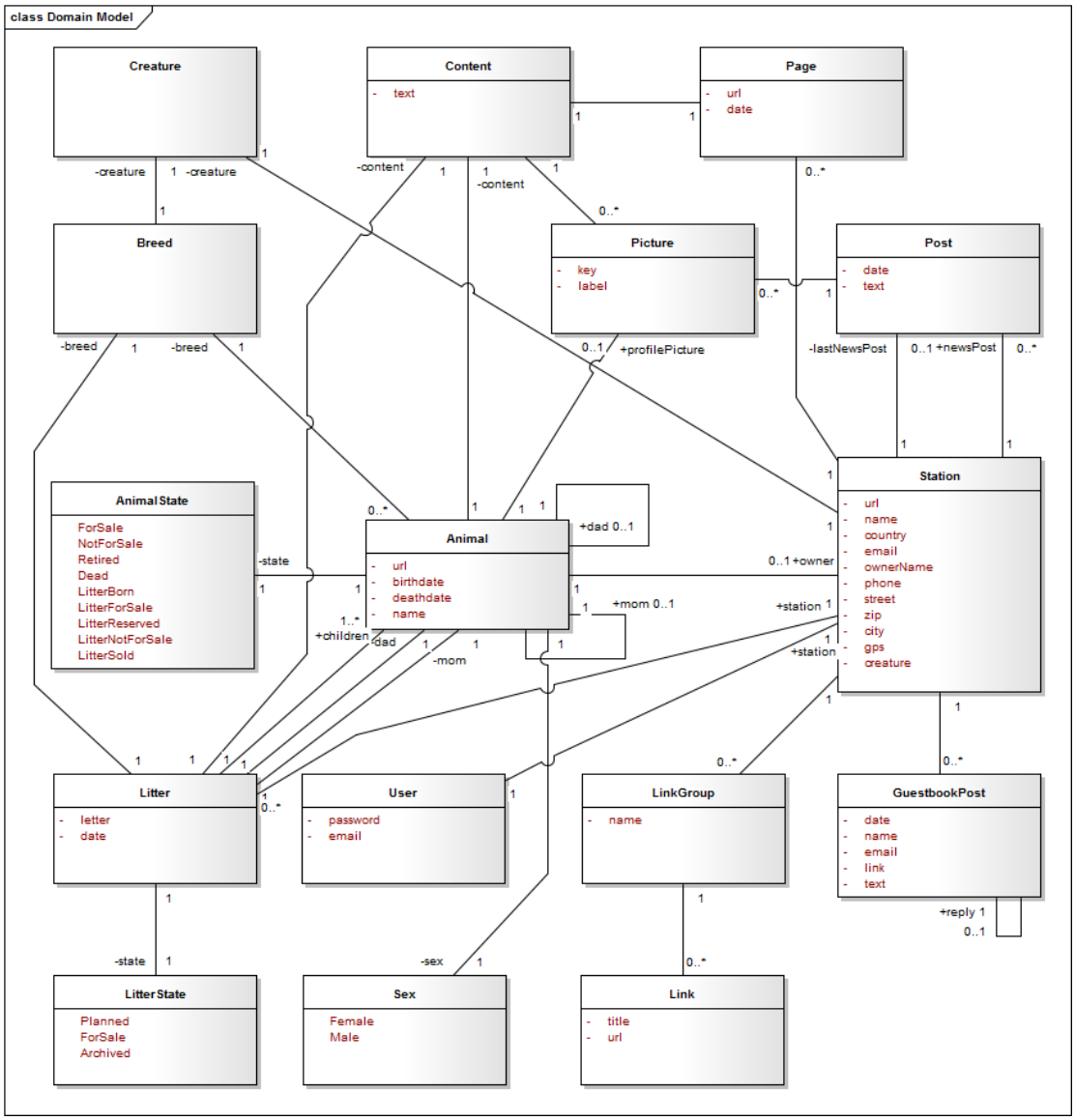


Figure 2.17. Domain Model

Chapter 3
Software Design

3.1 Architecture

Each and every architecture facet solution has been examined with respect to recent best practises and verified techniques.

3.1.1 Programming Language

Java[1]) has been chosen as a programming language because it is proven by the years yet still claimed to be the fastest option [2] when it comes to server-side web applications.

As for the web framework, choice was Java Server Faces (JSF) framework with PrimeFaces[2]) component suite. JSF provided the component-based approach and PrimeFaces itself features 100+ rich set of JSF components. PrimeFaces also offers Asynchronous JavaScript and XML (AJAX) partial rendering which is in combination with component-based backing beans a very powerful weapon.

PrettyFaces library[3]) provided friendly URLs which is one of the non-functional requirements.

3.1.2 Web Server

Apache Tomcat[4]) was chosen to act as a web server. It is open-source, simple, transparent and low-resource demanding servlet container.

3.1.3 Database Layer

The requirements for the database system were speed, reliability and the possibility of fast prototyping. NoSQL architecture satisfies all the needs. The leading document database is MongoDB[5]) so it was chosen for the project. It supports indexes, replication, auto-sharding, fast in-place updates, map/reduce and much more.

3.1.4 Security

Spring Security framework[6]) takes care of user authentication and authorization.

[1]) http://www.java.com/en/
[2]) http://www.primefaces.org/
[3]) http://ocpsoft.org/prettyfaces/
[4]) http://tomcat.apache.org/
[5]) http://www.mongodb.com/leading-nosql-database
[6]) http://projects.spring.io/spring-security/

3.1.5 Hosting

Most of the resources alocated for the development of the proposed system must be invested in implementing the features because a proof of concept is to be created. Thus cloud hosting is very appropriate for the matter since the basic configuration is offered for free and everything works out of the box.

Platform as a service (PaaS) is a category of cloud computing services that provides a computing platform and a solution stack as a service. PaaS offerings facilitate the deployment of applications without the cost and complexity of buying and managing the underlying hardware and software and provisioning hosting capabilities.

Figure 3.1. Technologies used in implementation.

Heroku platform[1]) offers a huge variety of addons, a stack located in Europe, 512 MB free memory plan and super easy git-push like deployment. All of that makes it perfect for such prototyping.

3.1.6 Data Storage

Heroku only offers ephemeral filesystem[2]). For that reason a permanent filesystem solution is required in order for the system to be able to store the pictures that users will upload. Amazon S3[3]) comes in hand with it's 5 GB free storage and easy to use Application Programming Interface (API). Amazon has also got servers in Ireland which makes it latency wise convenient for users browsing in the Czech Republic.

3.1.7 Front-end Framework

Twitter Bootstrap[4]) has been utilized. It has got a very complete set of Cascading Style Sheets (CSS) components. Unlike it's competitors, it does not expect the developer to understand underlying CSS perfectly, but tries to offer everything there is to easily bootstrap a project.

[1]) `https://www.heroku.com/`
[2]) `https://devcenter.heroku.com/articles/dynos#ephemeral-filesystem`
[3]) `http://aws.amazon.com/s3/`
[4]) `http://getbootstrap.com/`

3.2 Design Model

Design model is created by applying the General Responsibility Assignment Software Patterns (GRASP) to a domain model.

The getters and setters methods were left out in the figure 3.2. According to [3] the model should try to achieve the utter clarity.

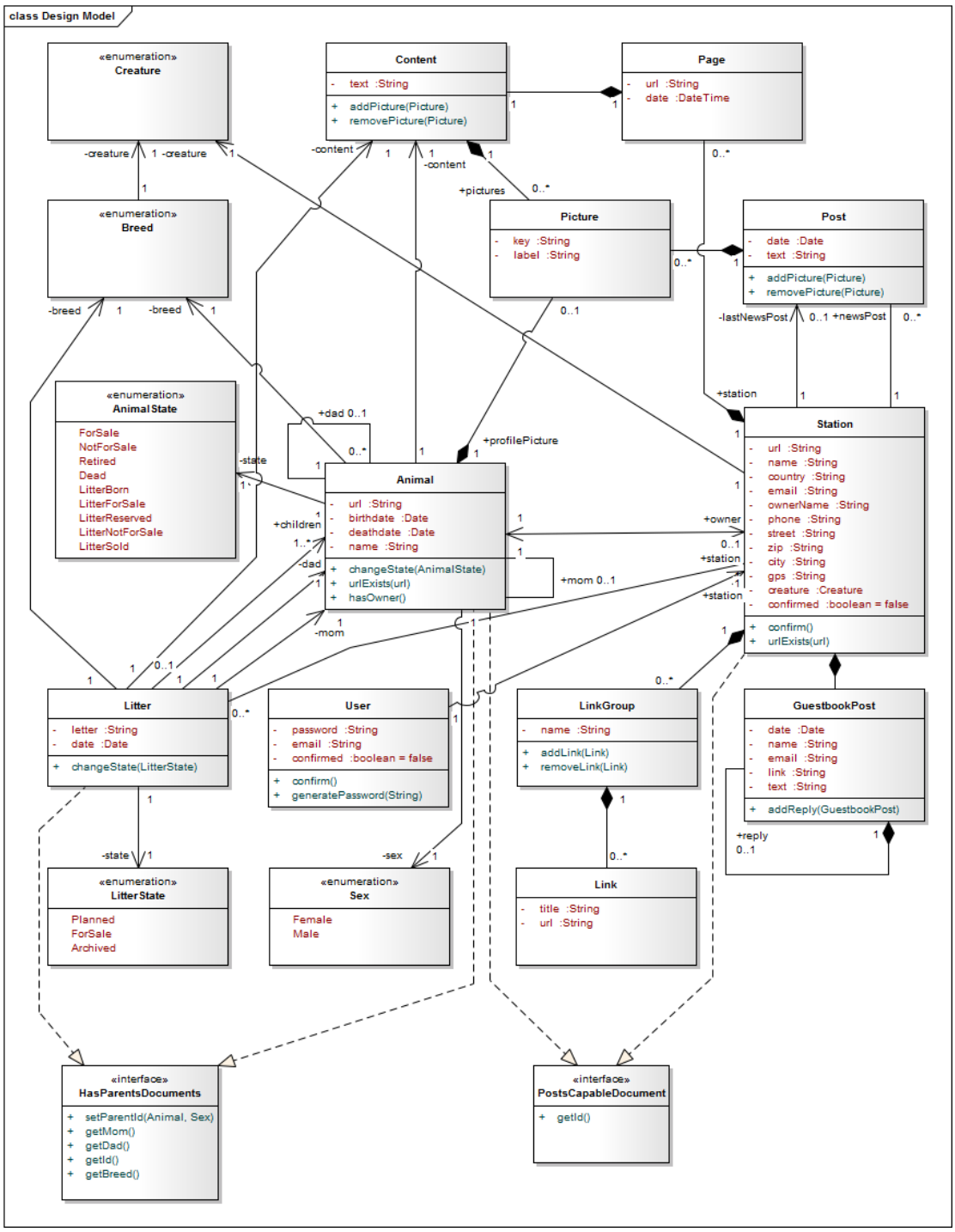


Figure 3.2. Design Model

3.3 Graphic Design

It is important to make sure the design is cleverly fabricated, as a flaw in design can lead to a flaw in coding, which in turn can lead to a flaw in the system.

3.3.1 Layout

The analysis of current breeders websites revealed couple of important repeating elements and patterns. Those form a basic page structure.

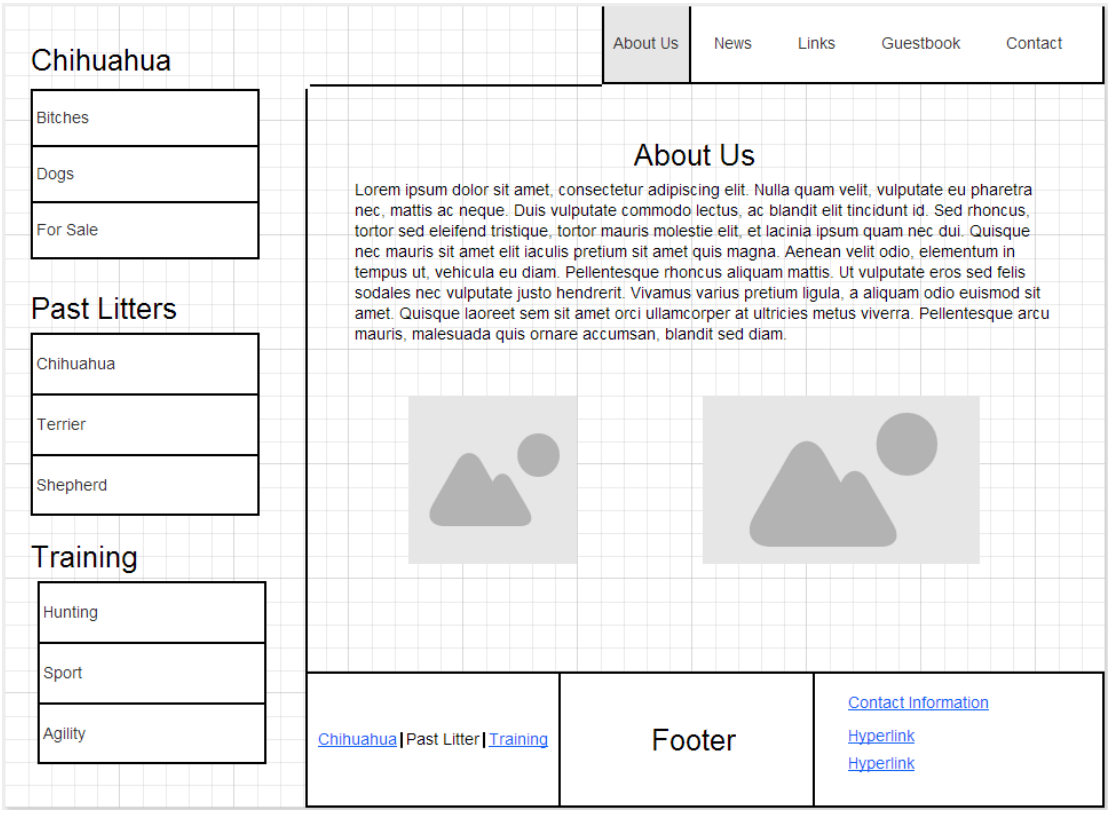


Figure 3.3. Page layout wireframe

Top Menu

Visitors perceive top right corner of a webpage as a less important area that holds links to static information such as guestbook or contact page.

The items in the menu are sorted in particular order which should reflect the timely sequential steps that a visitor should take. Firstly he or she starts at `About Us` page then continues through `News` and `Guestbook` finally reaching `Contact Page`.

Left Menu

The upper part of the left menu is where the important links usually are. It is also a great place for dynamically created content since there is vertical space for any number of items in the menu. This way the system will be able to properly display a menu with all the pages about the animals in a consistent manner.

Picture 3.3 shows that the most important content `breeding animals` is located in the upper part of the left menu whereas sections `past litters` and `custom pages` are located in the lower part.

Content Section

The entire layout is content centric based on the mobile first approach [4]. It says the designer should take the advantage of progressive enhancement rather than applying graceful degradation as seen in the figure 3.4. This way the content is focused on the most important elements. Since mobile phones have only small resolution display which is a portrait mode (width smaller than height) the content is vertically linear. The visitor is given the direction in which he or she should consume the content — simply from the top to the bottom. He or she does not get distracted by less important side elements.

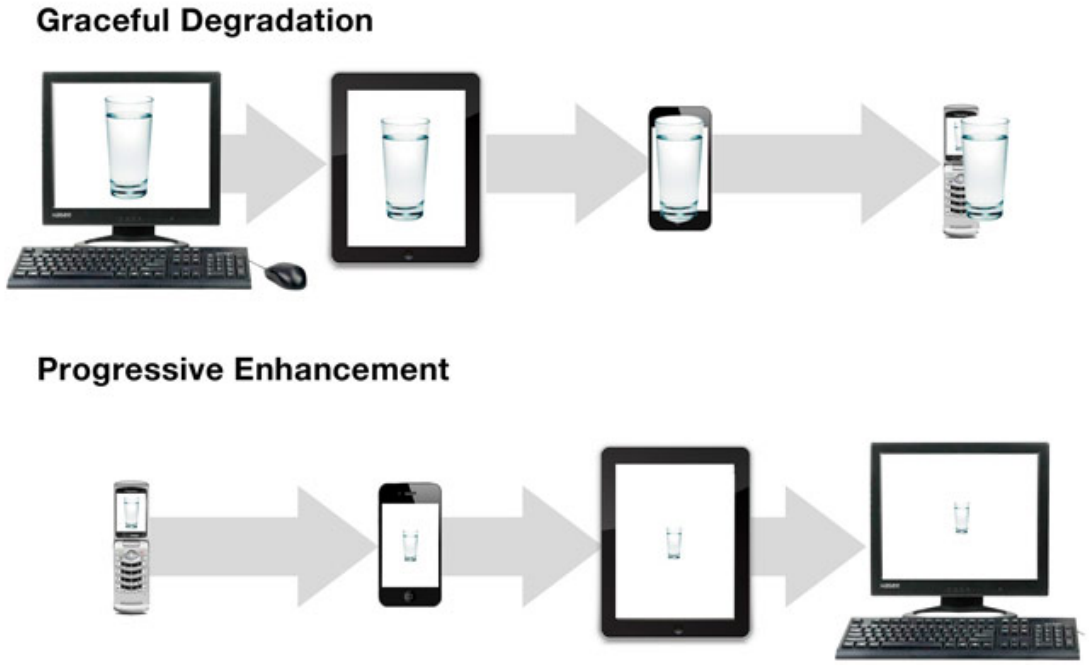


Figure 3.4. Progressive Enhancement [5]

Second phase of the design is moving from mobile layout to desktop layout. The designer now feels free because he or she has much more space to use. It does not work the other way because if the designer starts with desktop layout and then proceeds to mobile layout he or she feels sad simply because not everything from desktop layout can fit to smaller size mobile screen.

But still the stress is put to keep the layout as clean and simple as possible so that the visitor is not disctracted even when using desktop computer. In this manner only left sidebar navigation menu is added in desktop mode. The vertically linear approach is still persisted.

Footer Area

Footer section is composed of two parts. Firstly, upper part contains complete list of breeding station pages as well as the contact information. It serves the visitor

as an overview and lets him decide where would he or she like to continue browsing. Also having the contact information in the footer is essential since it is the fundamental mean of communication between a potential buyer and a breeder and so it has to be displayed on every page.

Secondly, the lower part of the footer contains information about the user session. A breeder gets the option of signing in or out in this section. There are also links that let visitor navigate throughout the entire system. One can create new breeding station following the link in this section. That is very desirable behaviour since the proposed system has to be self-promoting.

3.3.2 Public Pages

There is a list of all public pages the proposed system will serve in the appendix E.

Public pages are accessible without signing in. The speculation is that these will make more than 95% of the traffic. The proposed system has to use as few resources as possible. That will ensure that the content is delivered to the visitor's browser in the fastest way. Such practises include:

- No useless stylesheet and JavaScript files loaded.
- Only neccessary database requests.
- Properly designed database indexes.
- The smallest picture sizes transfered — no scaling down in the browser.
- Pictures stored in a cloud — ensures best possible latencies due to closest server location.

The content consists of reusable elements. That solves two purposes. Firstly, a developer can take advantage of repeating elements and create reusable components which means following the Don't Repeat Yourself (DRY) principle thus producing less code. Secondly, it also serves well the user because he or she can get acquainted with the elements hence feel more comfortable browsing the website.

Each page has to be represented with meaningful URL. That way the visitor knows where he or she is within the website or where he or she would be taken by clicking on the link.

3.3.3 Admin Pages

There is a list of all admin pages the proposed system will have in the appendix F. There is not that many of them as one would have expected thanks to the utilization of inline editing features. Embracing this approach keeps the complexity of the website at minimum although the system is complex and there is a lot of input required from the breeder. He or she simply edits the information right where the information is displayed which is practical and intuitive.

By putting the admin tools right along with the information itself the system makes use of AJAX partial rendering. Let's say the breeder wants to assign a father to an animal. He or she accomplishes that by navigating to the animal's profile page and then clicking on the `add father` button. The website sends an AJAX request to the server and only the area showing who the father is gets changed with the content just received from the server that allows the breeder to assign the father. That way the interface feels

like a native application due to it's low latency. It is also payload efficient since only the smallest portion of HTML code is transfered over the network.

Even stronger emphasis is put to reuse components in admin pages due to a bigger complexity of the components. There are buttons and textareas and pop ups and more buttons. The sooner gets the breeder familiar with the interface of the system the better. Consistent reusable components definitely help.

3.3.4 Responsive Design

Responsive design is one that adapts to various screen sizes. For successful adaptation the layout needs to change as well as font sizes, flow of elements and picture sizes. The purpose is that the visitor must be able to comfortably consume and browse the content no matter what device is at use.

Figure 3.5. Responsive Design Adaptation. [6]

There used to be a special mobile version of a website when the smartphones hit the market not that long time ago but due to the wide diversity of screen sizes and resolutions present in today's technology marketplace, mobile versions are often no longer sufficient in delivering optimal user experiences. Yet some services(e.g. Facebook) still use a mobile version instead of fully responsive one.

The numbers supporting the need for mobile/tablet website version speak for themselves. Czech Statistical Office counted the internet use of mobile and tablet devices in the Czech Republic in Q2 2013 to be 22,3% [1]).The Monetate Q4 2013 Ecommerce Quarterly [2]) published February 2014 gives insight 3.6 on smartphone vs tablet vs desktop share of audience for large Ecommerce brands.

[1]) http://www.czso.cz/csu/2013edicniplan.nsf/t/31002945B2/$File/97011338.pdf
[2]) http://monetate.com/research/

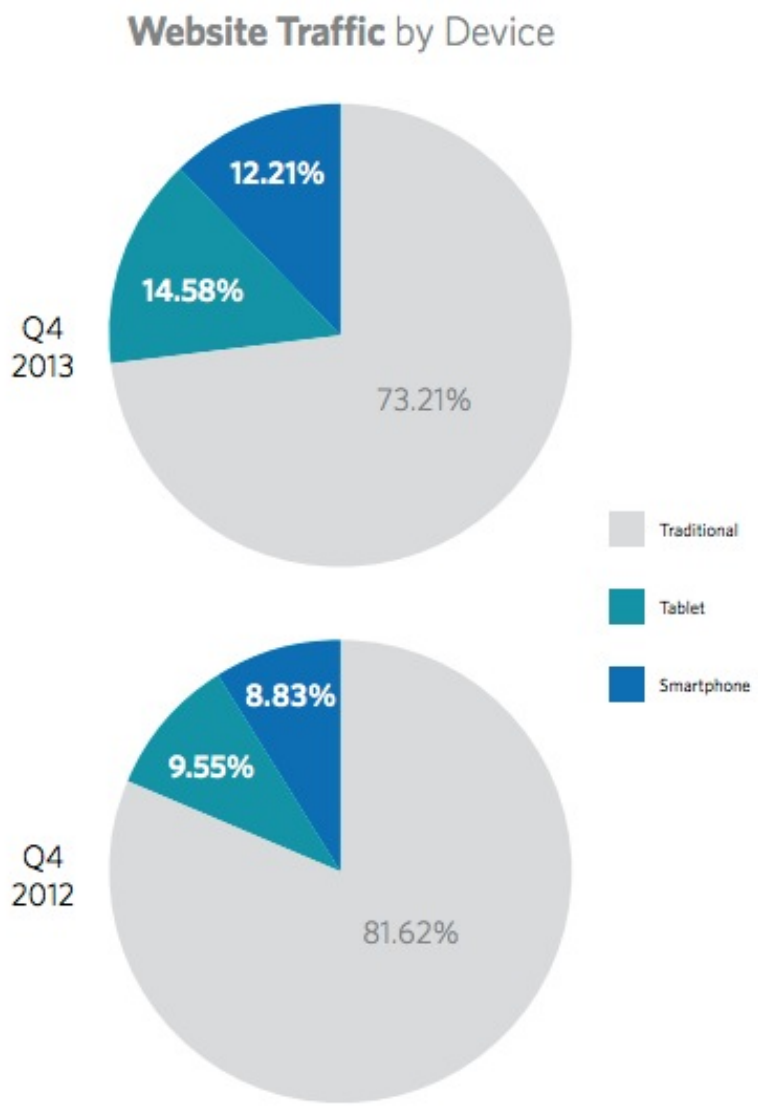


Figure 3.6. Smartphone vs tablet vs desktop share of audience for large Ecommerce brands. [7]

Chapter 4
Implementation

A diligent execution of the implementation phase is crucial for the proof of concept nature of the system. The focus is set on the most important requirements with keeping limited resources (manpower and time) in mind.

Every aspect needs to be though through cautiously in regards to the prototyping style of the development. The goal is to get the quality "good enough" in terms of the number of core requirements implemented as well as sufficient degree of implementation quality.

The article by James Bach [8] says: "The utilitarian view of quality is framed in terms of positive and negative consequences. The quality of something should be considered good enough when the potential positive consequences of creating or employing it acceptably outweigh the potential negatives in the judgment of key stakeholders."

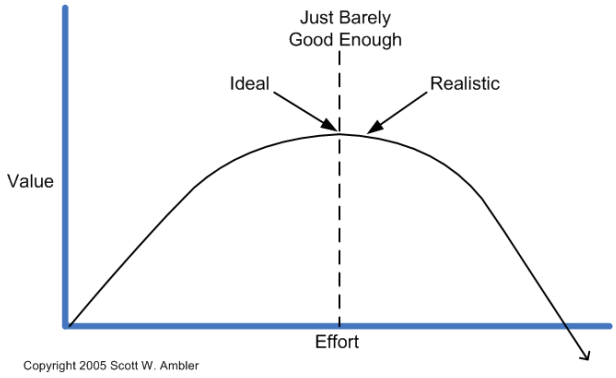


Figure 4.1. Cost analysis of just barely good enough.[9]

4.1 Encountered Problems and Solutions

4.1.1 Programming Language

At the beginning the developer tried the combination of SpringMVC framework with Thymeleaf templating library[1]) but there were two things that he found really annoying. Firstly, Spring MVC is a request-based framework which in the end meant creating endless amount of controllers. Secondly even worse, Thymeleaf has got no User Interface (UI) components in it's repertoire. Every UI element would have been implemented from scratch which is not desired when the focus should be placed on fulfilling the requirements in the shortest time.

Spring Data MongoDB[2]) provides an integration with the MongoDB document database. Easy integration with Spring Context using annotations is possible. There

[1]) `http://www.thymeleaf.org/`
[2]) `http://projects.spring.io/spring-data-mongodb/`

is for example feature “Automatic implementation of Repository interfaces including support for custom finder methods.” which allows only defining the method such as `findByUrl` in the code example in the figure 4.2 `a)` and injecting the automatically created interface implementation into a service class using annotation 4.2 `b)`.

```java
package cz.cilf.chs.service;

import cz.cilf.chs.d.Station;
import org.bson.types.ObjectId;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;

@Repository
interface StationRepository extends MongoRepository<Station, ObjectId> {

    Station findByUrl(String url);
}
```

a)

```java
import cz.cilf.chs.d.Station;
import javax.inject.Inject;
import javax.inject.Named;

@Named
public class StationService {

    @Inject
    private StationRepository stationRepository;

    public Station findByUrl(String url) throws StationNotFoundException {
        Station station = stationRepository.findByUrl(url);

        if (station == null) {
            throw new StationNotFoundException(url);
        }

        return station;
    }
}
```

b)

Figure 4.2. Spring Data MongoDB

4.1.2 Security

A password is stored in the database as a hash which is an encrypted sequence of characters obtained after applying certain algorithms and manipulations on user provided password. On the other hand saving the passwords into the database in it’s plain form is considered unsecure because anyone who gets access to the database would be able to see the passwords.

The problem today is that hardwares have become so fast that any brute force attack using dictionary and rainbow tables can crack any password that has been hashed by simple hash alghoritm. To solve this problem, general idea is to make the hash function slow enough to impede attacks, but still fast enough not to cause a noticeable delay for the user. This feature is essentially implemented using some CPU intensive algorithms such as `PBKDF2`, `Bcrypt` or `Scrypt`. These algorithms take a work factor (also known as security factor) or iteration count as an argument. This value determines how slow the hash function will be. When computers become faster in future the work factor can be increased to balance it out. The proposed system uses for hashing passwords `PBKDF2WithHmacSHA1` which is implementation of `PBKDF2` algorithm in Java.

Authorization is more complicated because the standard approach when a user is authorized upon successful login could not be employed because the user obtains `admin role` only if he or she is the owner of the currently displayed breeding station.

4.1.3 File Upload

Pictures is the only file content that users are allowed to upload. The maximum size for the file is 10 MB and only ten pictures can be uploaded at once. Pictures get uploaded to the system running on Heroku. Each picture is then downsized to 640 pixels and 145 pixels thumbnail version. Both pictures are transfered to Amazon S3 Storage placed into a breeding station’s folder having an unique name.

4.1.4 Autosuggest PrimeFaces Component with Diacritics

It is essential for Czech users to be able to type the words without the diacritics but match corresponding words containing diacritics.

At the time of the implementation the possibility to provide own JavaScript function for matching the suggested words in the PrimeFaces' SelectOneMenu component was broken. The easiest workaround was to replace PrimeFaces' method that had not been utilized with custom method as you can see on the first line in the figure 4.3.

```
PrimeFaces.widget.SelectOneMenu.prototype.endsWithFilter = function (value, filter) {
    return value.removeDiacritics().indexOf(filter.removeDiacritics()) !== -1;
};

(function () {
    "use strict";

    var dia = "áäčďéěíĺľňóôõöŕšťúůűüýřžÁÄČĎÉĚÍĹĽŇÓÔÕÖŔŠŤÚŮŰÜÝŘŽ";
    var noDia = "aacdeeillnoooorstuuuuyrzAACDEEILLNOOOORSTUUUUYRZ";

    function removeDiacritics(str) {
        var tmp = "";

        for (var p = 0; p < str.length; p++) {
            if (dia.indexOf(str.charAt(p)) !== -1) {
                tmp += noDia.charAt(dia.indexOf(str.charAt(p)));
            } else {
                tmp += str.charAt(p);
            }
        }

        return tmp;
    }

    if (!String.prototype.removeDiacritics) {
        String.prototype.removeDiacritics = function () {
            return removeDiacritics("" + this);
        };
    }
}());
```

Figure 4.3. Autosuggest JavaScript matching with diacritics

There is the result of the implemented function in the figure 4.4. It matches the words typed without diacritics to the ones with diacritics as well as the other way around.

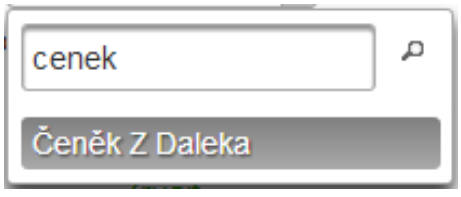


Figure 4.4. Autosuggest frontend component

4.1.5 JSF Composite Components

JSF composite components have been utilized in order to follow the Don't Repeat Yourself (DRY) principle . There is a litter component in the figure 4.5. It takes the `cz.cilf.chs.d.Litter` object as a required parameter and displays a litter header, a list of children and a litter detail link.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:composite="http://java.sun.com/jsf/composite"
      xmlns:chs="http://java.sun.com/jsf/composite/chs"
      xmlns:pretty="http://ocpsoft.com/prettyfaces" xmlns:f="http://java.sun.com/jsf/core"
        >

<composite:interface>
    <composite:attribute name="litter" type="cz.cilf.chs.d.Litter" required="true"/>
</composite:interface>

<composite:implementation>
    <chs:litterHeader litter="#{cc.attrs.litter}"/>
    <chs:animalListSmall animals="#{cc.attrs.litter.children}"/>

    <div class="row text-right">
        <pretty:link mappingId="litterProfile" styleClass="btn btn-primary">
            <f:param value="#{cc.attrs.litter.stationUrl}"/>
            <f:param value="#{cc.attrs.litter.breed.url}"/>
            <f:param value="#{cc.attrs.litter.letter.toLowerCase()}"/>
            <f:param value="#{cc.attrs.litter.id}"/>
            Zobrazit detail vrhu
        </pretty:link>
        <br/>
        <br/>
    </div>
</composite:implementation>
</html>
```

Figure 4.5. JSF composite component

4.1.6 Markdown Support

It is always a good practise to offer an advanced functionality for more experienced users. That is the reason why Markdown support has been implemented. Users can take advantage of it while creating the post or page content. Upon saving the inserted text content the HTML is stripped for security reasons. The `PegDownProcessor`[1]) includes various plugins such as support for tables or automatic links creation.

[1]) `https://github.com/sirthias/pegdown`

```
@Named("markdown")
@Scope("view")
class MarkdownConverter implements Serializable {

    @Inject
    MarkdownProvider provider;
    private Map<String, String> cache = new HashMap<~>();

    public String convert(String rawText) {
        if (StringUtils.isEmpty(rawText)) {
            return null;
        }

        if (!cache.containsKey(rawText)) {
            cache.put(rawText, provider.processMarkdownText(rawText));
        }
        return cache.get(rawText);
    }
}

@Named
@Scope("session")
class MarkdownProvider implements Serializable {

    private PegDownProcessor processor;

    @PostConstruct
    public void init() {
        processor = new PegDownProcessor(Extensions.ALL + Extensions.SUPPRESS_ALL_HTML);
    }

    public String processMarkdownText(String markdownText) {
        return processor.markdownToHtml(markdownText);
    }
}
```

Figure 4.6. Markdown support beans

4.1.7 Maven Spring Dependencies

There is an issue with integrating Spring Security using Maven with the latest release of Spring Framework. Spring Security has transitive dependency on Spring version 3.2.6 but in the system there is already defined Spring version 4.0.0. Version mismatch can and will cause a lot of trouble in compile time as well as runtime.

To solve this issue Spring BOM (Bill Of Materials) is utilized. A BOM dependency keeps track of version numbers and ensures that all dependencies (both direct and transitive) are at the same version. An added benefit of using the BOM is that you no longer need to specify the `version` attribute when depending on Spring Framework artifacts.

There is a part of `pom.xml` regarding the spring dependencies in the figure 4.7.

```
<properties>
    <spring.version>4.0.3.RELEASE</spring.version>
</properties>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-framework-bom</artifactId>
            <version>${spring.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <exclusions...>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
    </dependency>
```

```
    <dependency>
        <groupId>org.springframework.data</groupId>
        <artifactId>spring-data-mongodb</artifactId>
        <version>1.4.1.RELEASE</version>
        <exclusions...>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-web</artifactId>
        <version>3.2.3.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-config</artifactId>
        <version>3.2.3.RELEASE</version>
    </dependency>
</dependencies>
```

Figure 4.7. Maven Spring dependencies

You can see the transitive dependencies of Spring Framework 4.8, Spring Data MongoDB 4.9 and Spring Security 4.10 in the diagrams below.

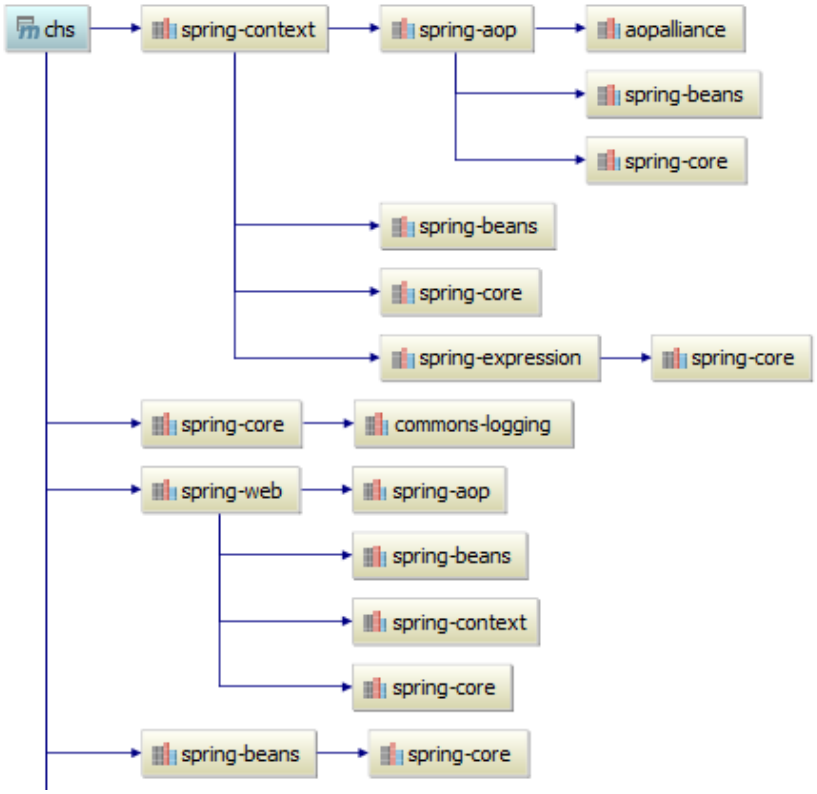


Figure 4.8. Maven Spring core transitive dependencies diagram

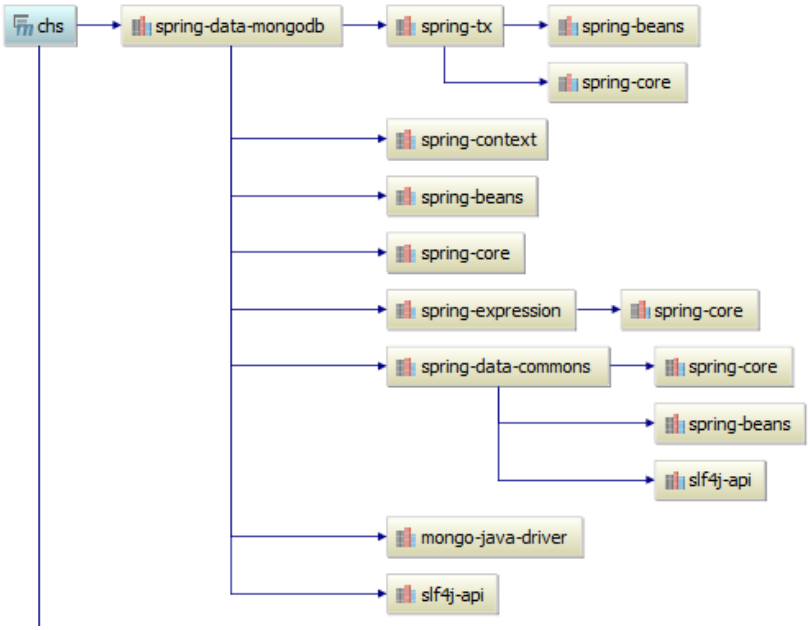


Figure 4.9. Maven Spring Data MongoDB transitive dependencies diagram

Figure 4.10. Maven Spring Security transitive dependencies diagram

4.1.8 Pedigree Recursive Construction

The animal profile page offers a possibility to display an animal's pedigree. There is a bean responsible for constructing the pedigree in the figure 4.11. It recursively traverses the ancestors tree until the depth reaches the `NUMBER_OF_ANCESTORS` constant.

```java
@Named
@Scope("request")
@URLMapping(id = "pedigree", pattern = "/#{breedUrl}/#{animalUrl}/pedigree", viewId = "/faces/pedigree.xhtml")
public class AnimalProfilePedigree {

    private static final int NUMBER_OF_ANCESTORS = 4;
    @Inject
    private AnimalService animalService;
    private PedigreeNode pedigreeNode;

    @PostConstruct
    public void init() {
        Animal animal = animalService.findByAnimalUrlAndBreed("animalUrl", Breed.get("breedUrl"));
        pedigreeNode = buildPedigreeTree(animal, 0);
    }

    private PedigreeNode buildPedigreeTree(Animal animal, int depth) {
        PedigreeNode node = new PedigreeNode(animal);

        if (depth == NUMBER_OF_ANCESTORS) {
            return node;
        }

        if (animal.getDadId() != null) {
            node.setDad(buildPedigreeTree(animalService.findOne(animal.getDadId()), depth + 1));
        }

        if (animal.getMomId() != null) {
            node.setMom(buildPedigreeTree(animalService.findOne(animal.getMomId()), depth + 1));
        }

        return node;
    }

    public PedigreeNode getPedigreeNode() { return pedigreeNode; }
}
```

Figure 4.11. Pedigree recursive construction

4.1.9 PrettyTime Integration

Showing the relative date instead of the absolute one is very popular these days. The PrettyTime[1]) library is used to provide such functionality. In the figure 4.12 is an example of JSF usage. The figure 4.13 shows the frontend result of a guestbook post saying "před chvílí" which means "moments ago" in English.

```
<h:outputText value="#{post.date}" converter="org.ocpsoft.PrettyTimeConverter"/>
```

Figure 4.12. PrettyTime JSF integration

Figure 4.13. PrettyTime frontend result

4.1.10 PrettyFaces Integration

One of the non-functional requirements states that the system has to have nice URLs. There is no native support for this functionality in JSF framework. That is why PrettyFaces library has been utilized. PrettyFaces offers configuration in XML or using annotations.

There is a guestbook bean in the example 4.14. You can see the annotation `@URLMapping` having `id` key which equals to `mappingId` used for creating links

[1]) `http://ocpsoft.org/prettytime/`

as in the figure 4.15. The `pattern` key is the url that it would match and the `viewId` points to the JSF file that will be displayed.

```
@Named
@Scope("view")
@URLMapping(id = "guestbook", pattern = "/#{stationUrl}/guest-book", viewId = "/faces/guestbook-page.xhtml")
public class Guestbook implements Serializable {

    @Inject
    private Controller controller;
    @Inject
    private GuestbookService guestbookService;

    private List<GuestbookPost> posts;

    @PostConstruct
    public void init() {
        posts = guestbookService.findAll(controller.getStation());
    }

    public List<GuestbookPost> getPosts() { return posts; }
}
```

Figure 4.14. PrettyFaces and managed bean

A `link` component from `pretty` namespace is used in order to create a link. It takes the `mappingId` and necessary parameters to build the URL. This way of constructing URLs gets convenient because the URL pattern itself is only defined in the bean annotation.

```
<pretty:link mappingId="guestbook">
    <f:param value="#{controller.stationUrl}"/>
    Guestbook
</pretty:link>
```

Figure 4.15. PrettyFaces JSF link component

4.1.11 Spring Beans View Scope Support

JSF framework offers very convenient `view` lifetime bean scope. The bean with such a scope gets created when a user enters a certain URL and gets destroyed when he or she leaves to another URL. Most of the PrimeFaces components are AJAX based. When a component makes an AJAX `Postback` (performs a POST request to the very same URL) the bean does not get destroyed thus it is able to keep it's state for these future AJAX requests.

Since Spring Data MongoDB has been utilized it was necessary to autowire the Spring annotated services into the beans which in the end meant using the Spring autowiring mechanism and Spring beans altogether. But Spring does not have a `view` bean scope. The figure 4.16 shows a custom implementation of the `view` scope for Spring beans and it's integration 4.17 in the spring context XML descriptor.

```java
package cz.cilf.chs.spring.customscope;

import org.springframework.beans.factory.ObjectFactory;
import org.springframework.beans.factory.config.Scope;

import javax.faces.context.FacesContext;
import java.util.Map;

/**
 * Implements the JSF View Scope for use by Spring.
 * This class is registered as a Spring bean with the CustomScopeConfigurer.
*/
public class ViewScope implements Scope {

    public Object get(String name, ObjectFactory<?> objectFactory) {
        if (FacesContext.getCurrentInstance().getViewRoot() != null) {
            Map<String, Object> viewMap = FacesContext.getCurrentInstance().getViewRoot().getViewMap();
            if (viewMap.containsKey(name)) {
                return viewMap.get(name);
            } else {
                Object object = objectFactory.getObject();
                viewMap.put(name, object);
                return object;
            }
        } else {
            return null;
        }
    }

    public Object remove(String name) {
        if (FacesContext.getCurrentInstance().getViewRoot() != null) {
            return FacesContext.getCurrentInstance().getViewRoot().getViewMap().remove(name);
        } else {
            return null;
        }
    }

    public void registerDestructionCallback(String name, Runnable callback) {...}

    public Object resolveContextualObject(String key) { return null; }

    public String getConversationId() { return null; }

}
```

Figure 4.16. Spring beans view scope implementation

```xml
<bean class="org.springframework.beans.factory.config.CustomScopeConfigurer">
    <property name="scopes">
        <map>
            <entry key="view">
                <bean class="cz.cilf.chs.spring.customscope.ViewScope"/>
            </entry>
        </map>
    </property>
</bean>
```

Figure 4.17. Spring view scope integration (spring-application-context.xml)

4.2 State Diagrams

Litter child states diagram in the picture 4.18 shows the states the animal can get within a litter which state is `for sale`.

The child is added as `born` which gives a breeder the time to decide if he or she would like to keep the animal or not.

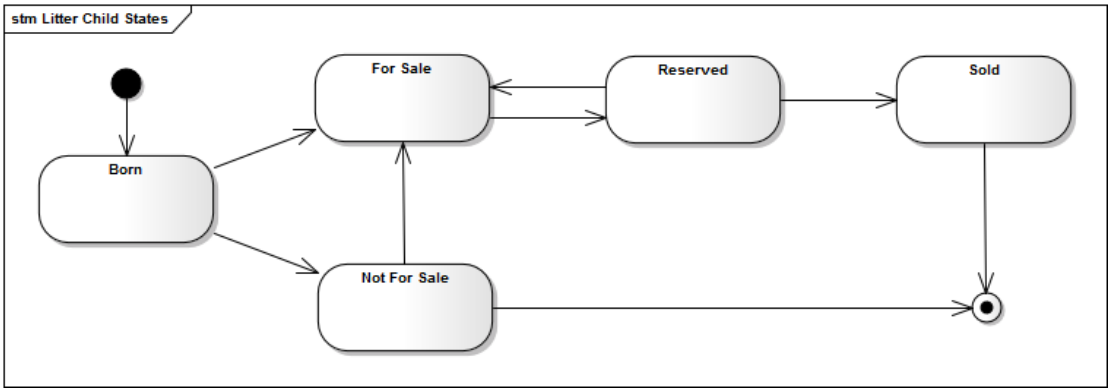


Figure 4.18. Litter child state diagram

An animal becomes `active` either upon creating, or by archiving the litter in which the animal has been born.

An animal is moved to `Retired` section of the website once it gets the state `retired`. Similarly the animal transfers to the section `In Memory` upon dying.

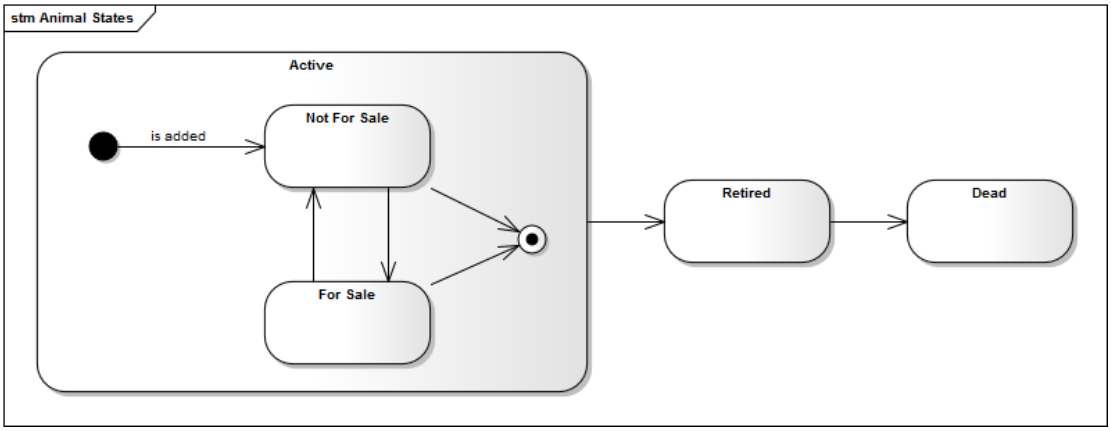


Figure 4.19. Animal state diagram

The section `For Sale` of the website shows adult animals with `for sale` state and litters with `planned` and `for sale` states. Otherwise the section becomes empty. That should guide the breeders to create litter ahead with the state `planned` so that the most important part of the website does not get empty.

A `planned` litter can have none, one or both parents. A first child can only be added if both parents are assigned. The litter becomes `for sale` upon adding the first child.

The breeder is allowed to archive a litter once all of the children get `not for~sale` or `sold` state. Upon archiving the litter is moved to `Past Litters` section of the website.

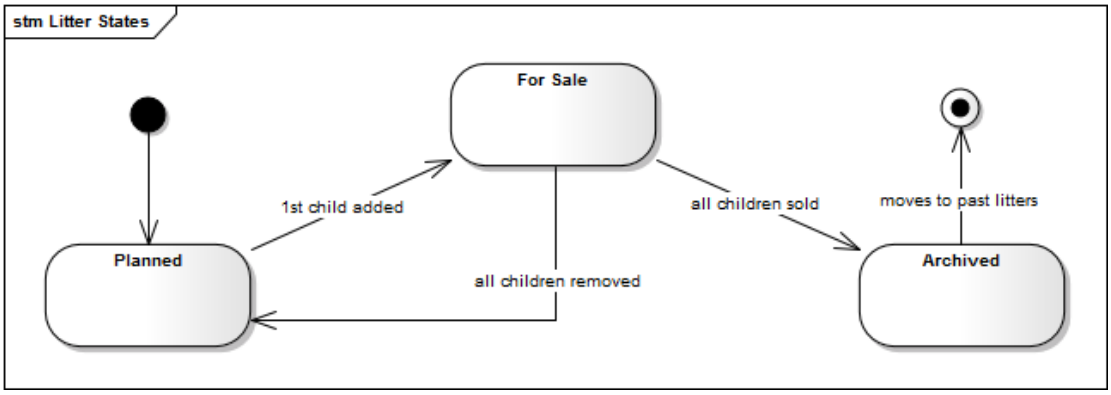


Figure 4.20. Litter state diagram

All of the constraints introduced in the figures above should help the breeder follow a logical path of managing litters.

4.3 Database Model

There are no M:N relationship tables in the model 4.21. MondoDB is NoSQL database also known as schemaless, and therefore the one-to-many or many-to-many relation is stored the same way as in Java objects. The object (document) would contain the array of either foreign keys or documents as for example in the figure 4.22 there is a `Picture` object acting as a `profilePicture` attribute. The biggest advantage is that there is only need to access one collection (= table in SQL world) to retreive one document with all the information. The only limitation is the 16 MB limit per document.

Every time there is an aggregation used in the model 4.21 it actually means that the target object lays within the source object. Again the `profilePicture` in the figure 4.22 is a perfect example.

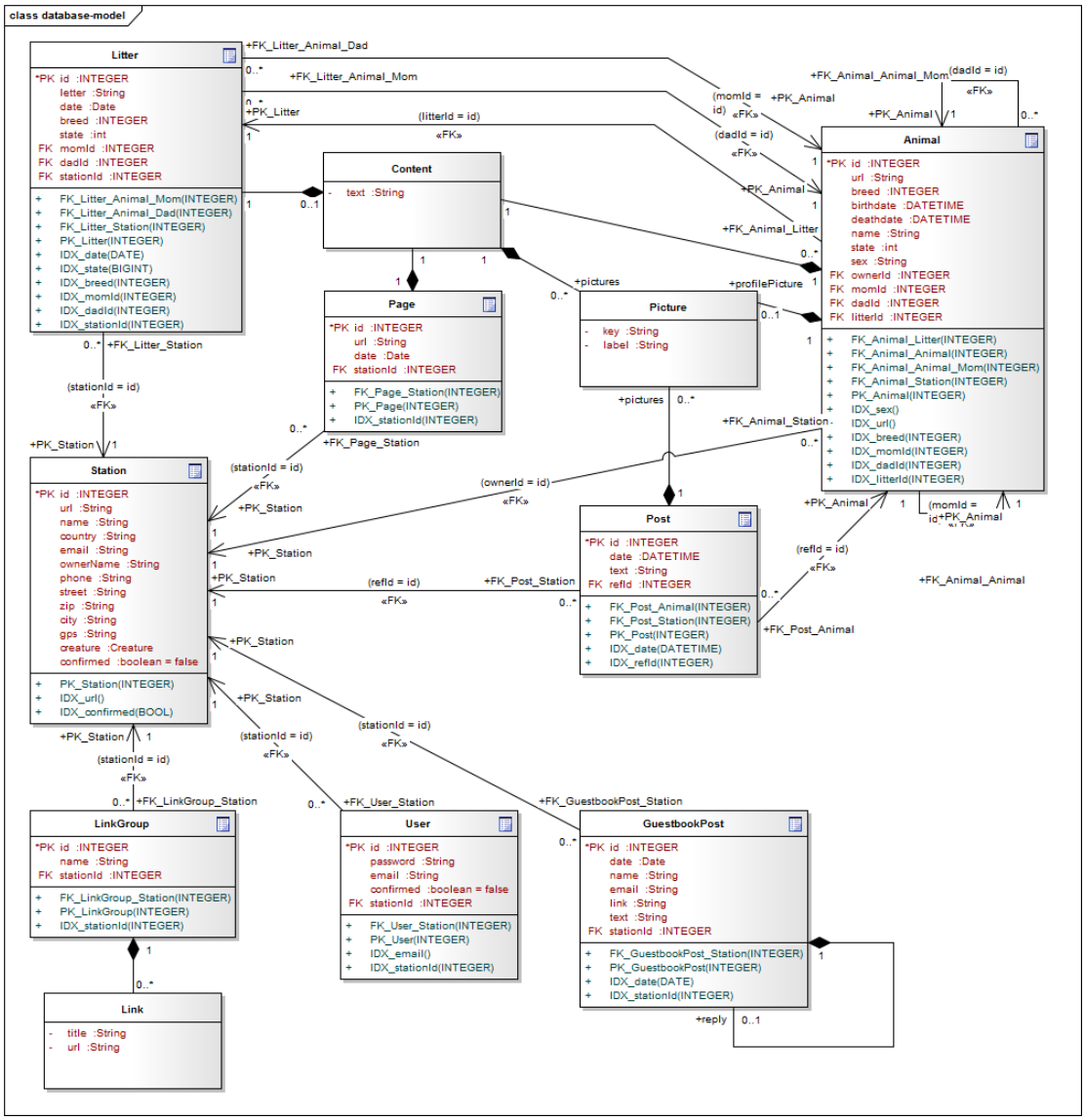
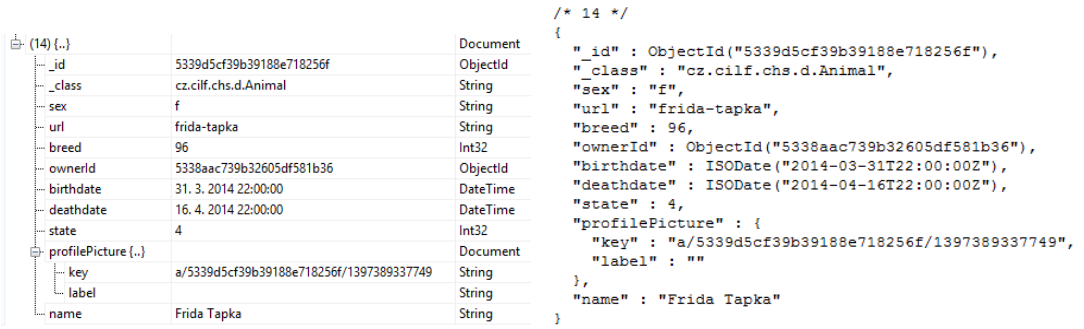


Figure 4.21. Database model

There is an example of a MongoDB animal document in the figure 4.22. On the left side `a)` there is a representation of the document in MongoVUE[1]) explorer whereas on the picture `b)` there is an actual JavaScript Object Notation (JSON) object as it is saved in the database.

The `_id` key is primary key automatically generated by MongoDB database server and it is unique within the database. The `_class` key is a reference to the type of serialized Java class generated by Spring Data MongoDB[2]). The `sex`, `breed` and `state` keys are ids of Java enumerations.

[1]) `http://www.mongovue.com/`
[2]) `http://projects.spring.io/spring-data-mongodb/`



(14) {..}		Document
_id	5339d5cf39b39188e718256f	ObjectId
_class	cz.cilf.chs.d.Animal	String
sex	f	String
url	frida-tapka	String
breed	96	Int32
ownerId	5338aac739b32605df581b36	ObjectId
birthdate	31. 3. 2014 22:00:00	DateTime
deathdate	16. 4. 2014 22:00:00	DateTime
state	4	Int32
profilePicture {..}		Document
key	a/5339d5cf39b39188e718256f/1397389337749	String
label		String
name	Frida Tapka	String

a)

```
/* 14 */
{
  "_id" : ObjectId("5339d5cf39b39188e718256f"),
  "_class" : "cz.cilf.chs.d.Animal",
  "sex" : "f",
  "url" : "frida-tapka",
  "breed" : 96,
  "ownerId" : ObjectId("5338aac739b32605df581b36"),
  "birthdate" : ISODate("2014-03-31T22:00:00Z"),
  "deathdate" : ISODate("2014-04-16T22:00:00Z"),
  "state" : 4,
  "profilePicture" : {
    "key" : "a/5339d5cf39b39188e718256f/1397389337749",
    "label" : ""
  },
  "name" : "Frida Tapka"
}
```

b)

Figure 4.22. MongoDB animal document

4.4 Screenshots

The left bar menu in the figure 4.23 is generated by the system depending what content the breeder had put in. The background image has been chosen to suit properly the dog breeders' business. On the other hand, it had to be general enough to go well with all the possible breeds and look elegant yet professional.

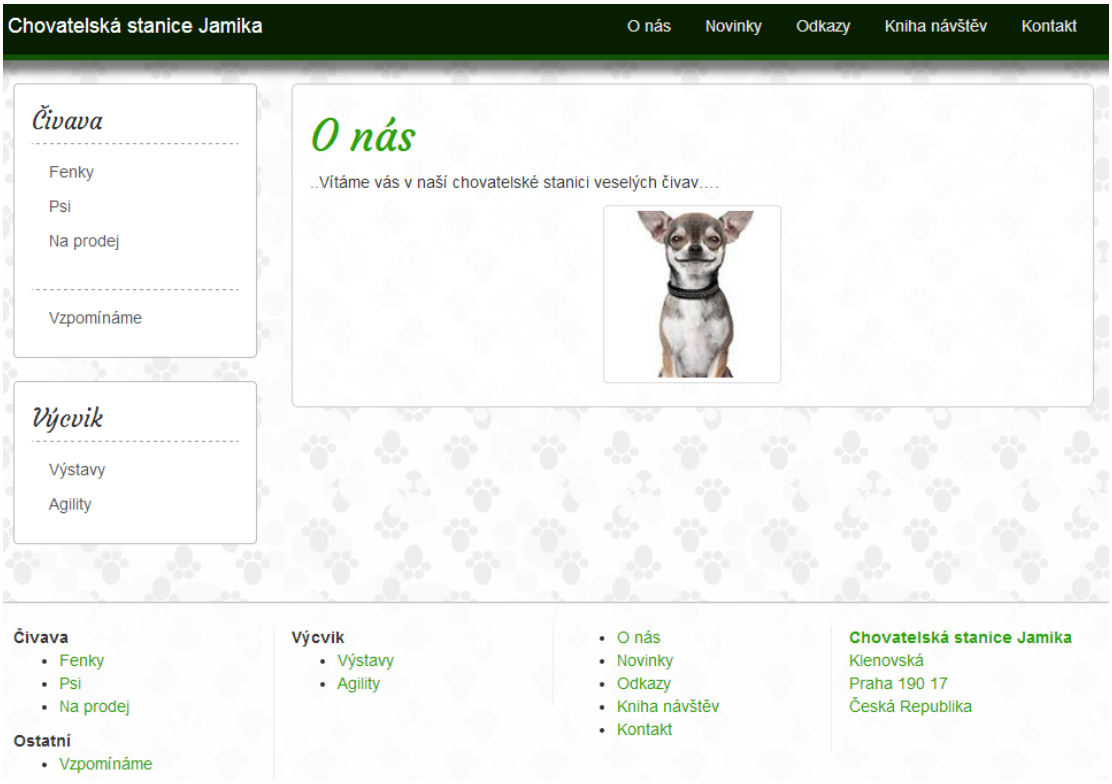
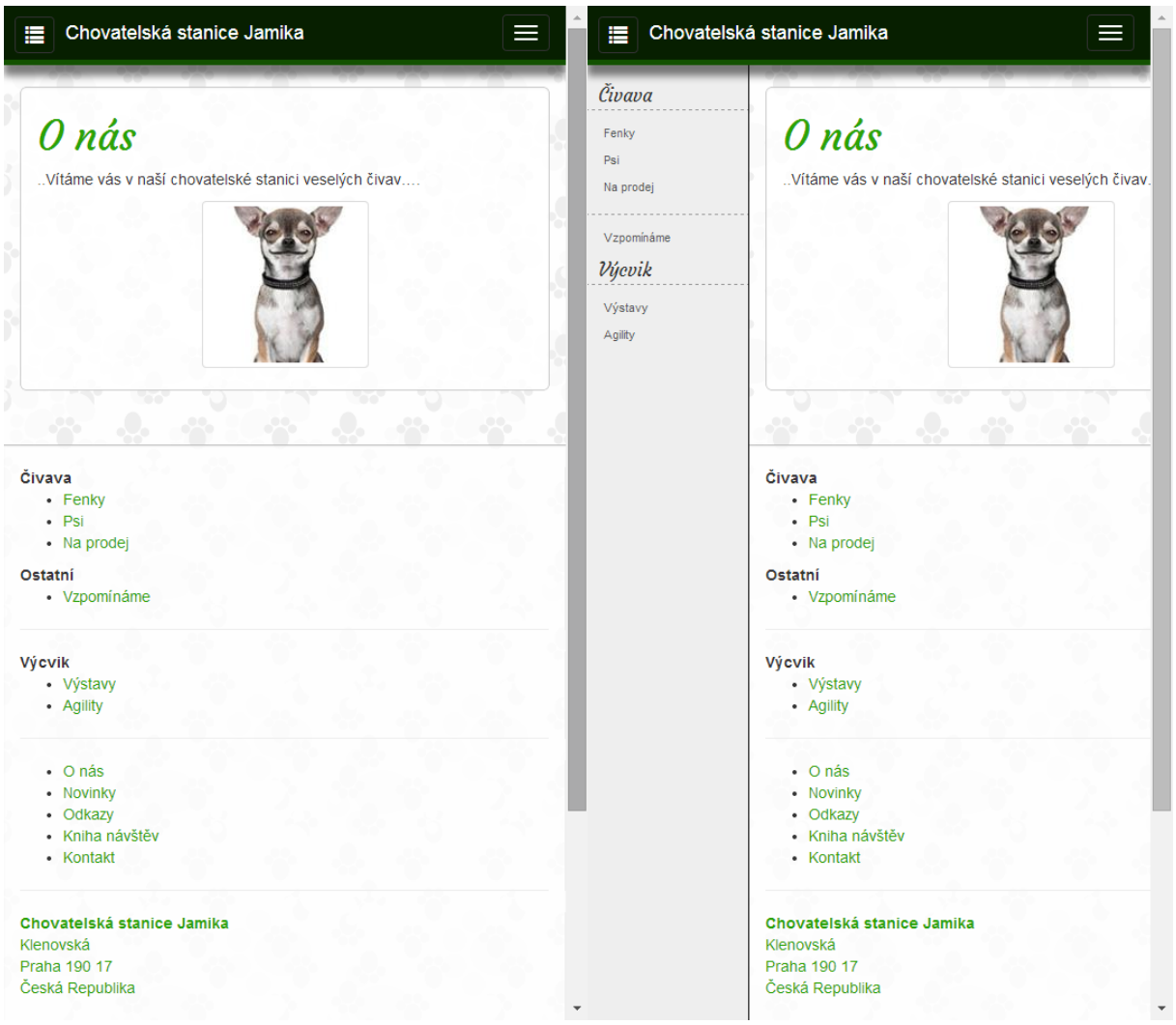


Figure 4.23. Desktop `About Us` page screenshot

The left bar menu and top right menu are shrinked to a button in the mobile version as you can see in the figure 4.24. In the `b)` picture there is a left bar menu animated to the screen upon clicking the top left button in the header. However, the left bar menu contains the same items as the footer does so the user can actually choose what navigation flow he or she prefers.

The picture 4.24 nicely shows the mobile first approach being utilized. The user does not get distracted while browsing. When he or she reaches the end of the page there is a convenient footer section which gives him or her plenty of options to continue browsing so that the user should not get lost.

Lastly, it is very important to show the contact information on every page because it should serve for the user as a main mean of getting in touch with the breeder.



a) b)

Figure 4.24. Mobile `About Us` page screenshot

The figure 4.25 shows the top right menu being displayed upon clicking on the top right menu button. It adapts to all small and medium sized devices.

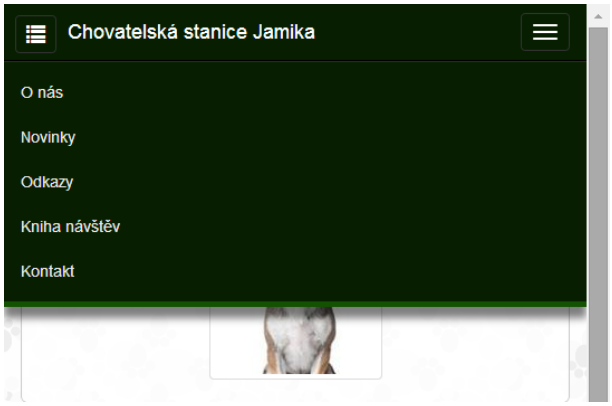


Figure 4.25. Responsive menu screenshot

The top bar is sticky as you can see in the figure 4.26. That is convenient for two reasons. Firstly, the header shows the name of the breeding station. Having the name visible all the time should assure the user of his or her whereabouts. Secondly, since the header is glued to the top edge of the screen the user can click on any of the menu buttons at any time which makes it easier for him or her to browse the pages and possibly find the information he or she is looking for faster.

Figure 4.26. Sticky top bar screenshot

Chapter 5
Deployment

The model in the figure 5.1 shows the basic overview of the system structure.

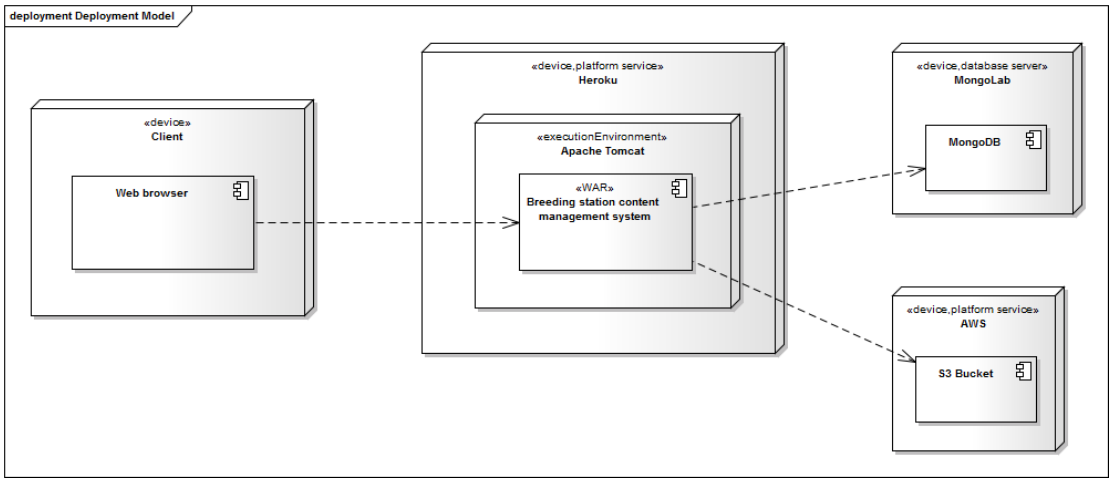


Figure 5.1. Deployment model

Deployment steps:

1. After a feature is completed the unit tests run in the development environment.
2. If the unit tests pass the feature is deployed to the staging environment.
3. Selenium tests run in the staging environment.
4. If the selenium tests pass the feature is deployed to the production environment.

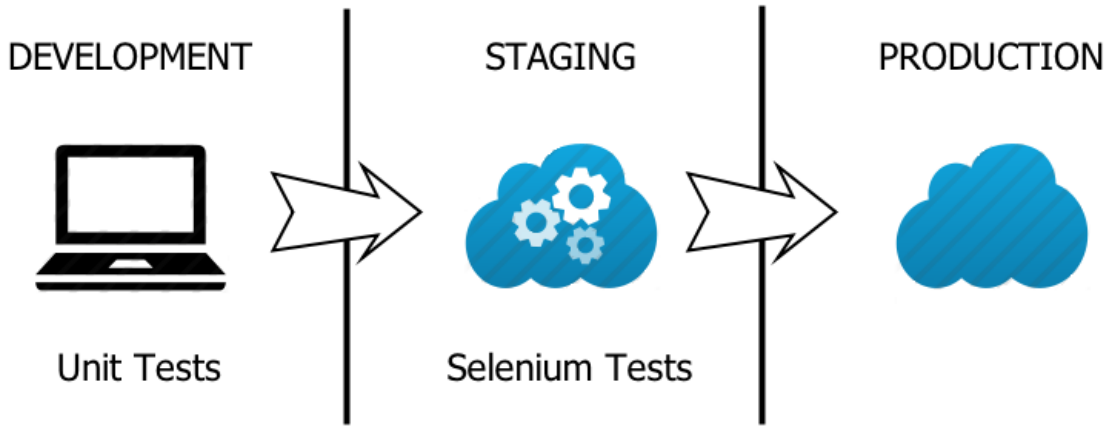


Figure 5.2. Deployment Schema

These steps are performed manually as of now simply because it is sufficient enough but for the future the plan is to use some sort of continuous integration system which would automatically trigger next step if the tests in the current step run successfully.

The development environment is a computer which uses a local database.

The staging environment is an application hosted by Heroku cloud platform. It has got it's own database and Amazon S3 bucket for uploaded pictures. This environment has to be identical to the production one in order to achieve proper simulation of testing conditions.

Chapter 6
Testing

The sooner the bug is identified, the cheaper it will be to fix the problem. The cost of fixing a bug can be represented by something around a logarithmic function, where the cost can increase by more than 10 times as the project progresses through the phases of the software development life cycle. That is the reason why it is so important to validate and verify the outcome of each phase of the life cycle.

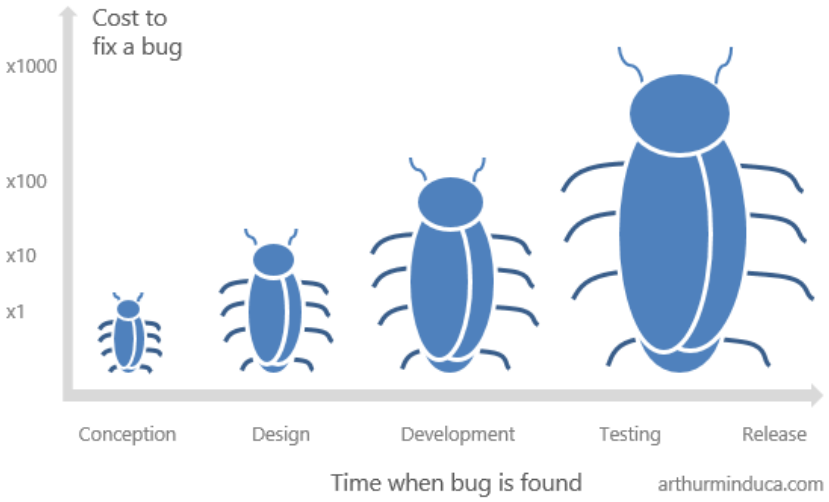


Figure 6.1. Cost to fix a bug [10]

The two terms `validation` and `verification` are often confused but understanding the difference is substantial. Validation asks the question "Are we building the right product" thus ensures that the product actually meets the user's needs, and that the specifications were correct in the first place. Whereas verification on the other hand asks "Are we building the product right?" thus ensuring that the product has been built according to the requirements and design specifications.

Another aspect of testing is proper test suites arrangement. The Y axis in the figure 6.2 shows the amount of the execution time each test suite should take as well as the ratio of automatization. The unit tests should be fully automated while UI tests are more or less considered as manual labour.

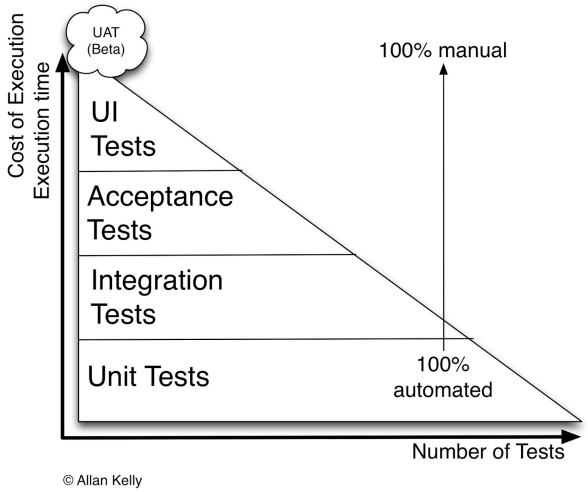


Figure 6.2. Testing Triangle [11]

There is a strong correlation between the number of the tests and the cost of the execution. The unit tests are very cheap to execute which makes them suitable for running as much as possible.

Each test suite is executed at different time of the software development life cycle which is described in the Deployment 5 section.

6.1 Unit Testing

A unit is usually a method or a class. A unit test serves as a mean of verification that the particular unit is bug free. The goal is to discover the bugs as early as possible reducing the cost of repair to minimum.

The main emphasis has been put to test the service classes since they basically are the moving parts of the system. The coverage can be seen in the table 6.1.

Package	Class, %	Method, %	Line, %
beans	33%(11/33)	27%(101/370)	39%(523/1348)
domain	72%(16/22)	40%(123/306)	67%(610/910)
orm	100%(3/3)	100%(7/7)	96%(23/24)
security	25%(1/4)	7%(1/14)	4%(1/23)
service	92%(12/13)	84%(76/90)	80%(259/320)
spring	0%(0/1)	0%(0/6)	0%(0/15)
utils	44%(4/9)	35%(13/37)	50%(73/145)
validator	0%(0/1)	0%(0/2)	0%(0/7)

Table 6.1. Unit tests coverage.

The figure 6.3 shows the DRY principle in practise. We need to test the function that takes care of creating URLs from names. Instead of writing test case for each input and expected output we code only one testing method and pass the array of inputs / outputs as a parameter.

```java
@RunWith(Parameterized.class)
public class UrlUtilTest {

    private String expected;
    private String actual;

    public UrlUtilTest(String expected, String actual) {
        this.expected = expected;
        this.actual = actual;
    }

    @Parameterized.Parameters
    public static Collection<String[]> params() {
        return Arrays.asList(new String[][]{
                {"", null},
                {"", "-"},
                {"", "-"},
                {"", "--"},
                {"a", "-a-"},
                {"b", "--b--"},
                {"c", "#$@--c-!$-!$"},
                {"d", "#$@--d-!$-!$"},
                {"b-a-c", "b#$@--a-!$-!$C"},
                {"b-a-d", "$b#$@--a-!$-!$D@#@@"},
                {"b-c", "$b#$@---!$-!$C@#@@"},
        });

    }

    @Test
    public void shouldUrlify() { assertEquals(expected, UrlUtil.urlify(actual)); }
}
```

Figure 6.3. Parameterized unit test

In the figure 6.4 is displayed how the result of the above test looks like when run in IntelliJ IDEA Integrated Development Environment (IDE).

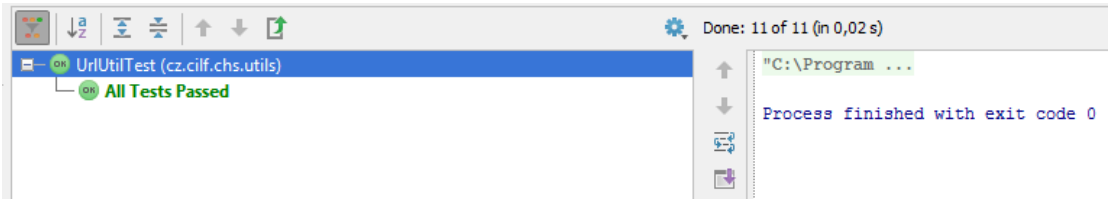


Figure 6.4. Result of a unit test

6.1.1 Test Driven Development

Test driven development is a software development process that relies on the repetition of a very short development cycle consisted of first writing the test and then implementing the only relevant code to make the test pass. It lets the developer to test the yet to be implemented unit's API helping him or her discover the possible design problems even before writing a single line of code implementation. All of that should afterwards lead to better code structure.

After the first paragraph a utilization of the test driven development may seem as a must do while programming but there is a big downside to it as well. It is time consuming as any other coding and for that reason TDD was not employed while developing the system.

Due to the nature of a rapid prototyping method the developer needs to focus on fast delivering features. But while implementing a feature the developer might discover that the architecture he or she has chosen is inappropriate thus being forced to throw away some of the code he or she has already coded and that becomes extremely painful when there is also unit tests code included.

6.2 Integration Testing

Selenium tests run in the staging environment. Their goal is to verify that the system behaves correctly. They do so by following a predefined list of steps simulating a user interaction in the browser.

The test suite has been composed following the same scenario (appended in G) that has been used during the usability testing. The steps taken should verify all of the crucial functional requirements.

There is a result of selenium test run in the figure 6.5 executed in Firefox extension Selenium IDE[1]). There is one noticeable thing. The HTML elements could not be targeted by the `ID` attributes because JSF by the specification generates the ID attributes according to the position in the component tree. Adding a single HTML element before the targeted control element later in the development would result into the need of reworking the selenium test case. That is why relative XPath selecting has been utilized.

The command `waitForElementPresent` is necessary because the form for setting the birth of date is loaded via AJAX. If the command was omitted the test case would fail because the Selenium IDE would not be able to immediately `type` the date `1.1.14` into the `//span/input`.

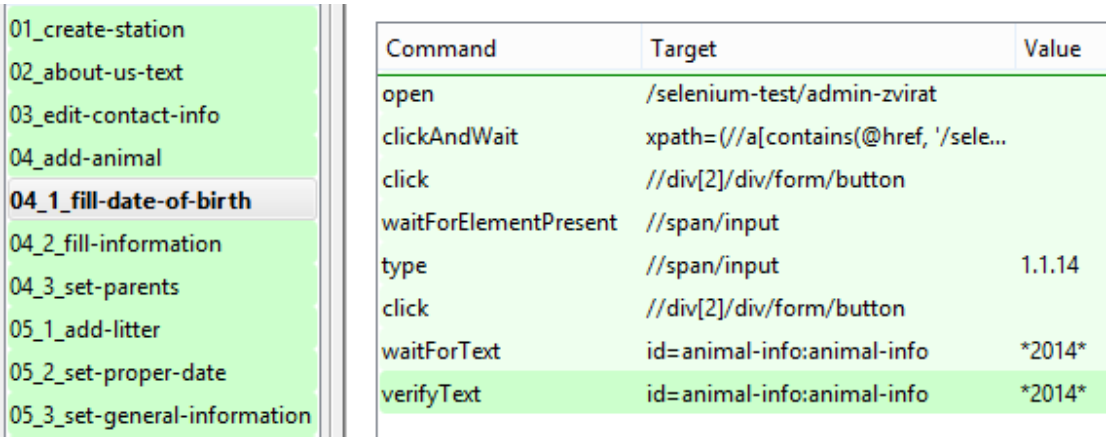


Figure 6.5. Selenium test run 1st part

[1]) `http://docs.seleniumhq.org/projects/ide/`

In the last test case in the figure 6.6 there is a verification that the system redirected the user to the index page upon station deletion. It also checks that the user has been logged out by verifing the link for the login page being present.

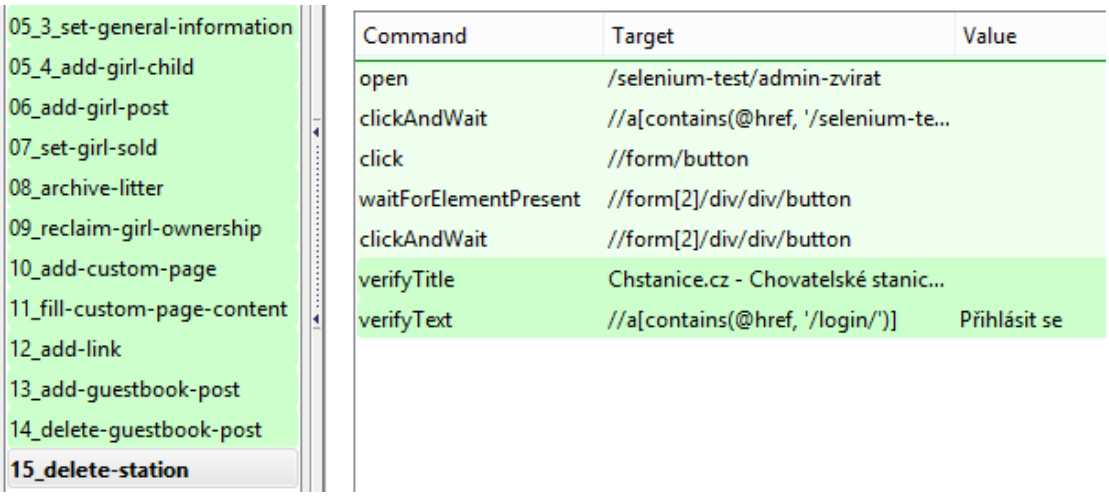


Figure 6.6. Selenium test run 2nd part

The system is ready to be deployed to the production environment upon successful selenium test suite run.

6.3 Stress Testing

A five minute stress test was carried out by Load Impact[1]) service. The figure 6.7 shows the results. The green line shows how active clients were added throughout the test. The heroku server does good job scaling up the performance under stress.

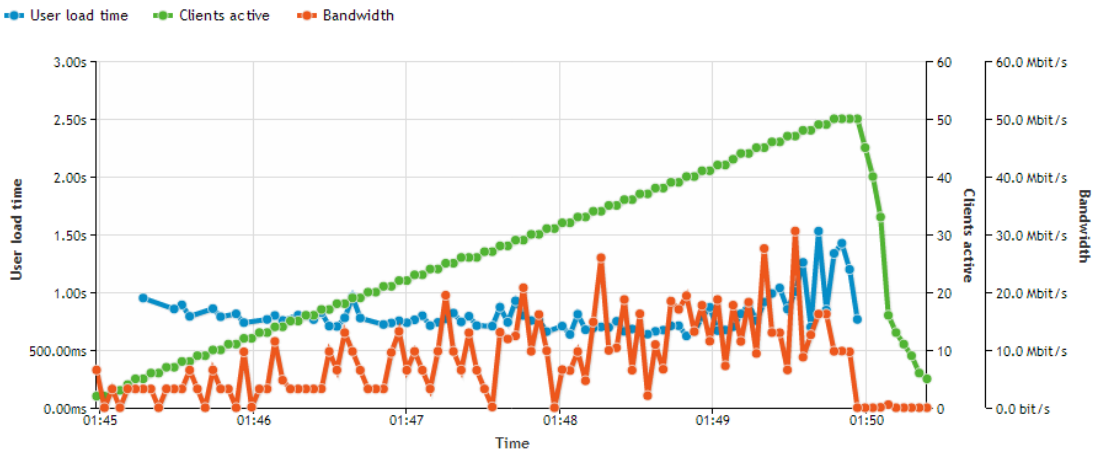


Figure 6.7. Stress Testing Chart

The `For Sale` page has been selected for the test for two reasons. Firstly, it is the most important page on the website. Secondly, it represented well an average page of the website since it is consisted of both animals and litters hence the database was put to test as well as Amazon S3 bucket which served the uploaded animals' pictures.

[1]) `https://loadimpact.com/`

Heroku cloud platform and it's free plan has been proven to satisfy the stress load needs which the proof of concept system will experience in the future.

Moreover, there always will be the possibility to easily scale up the performance if ever the load needs significantly increase.

6.4 Usability Testing

A usability testing is a powerful technique of verification that the members of the target audience are able to make the system work for them in the way it is supposed to based on the identified functional requirements.

6.4.1 Principles

Will Dayble explains in his talk [12] some of the most fundamental principles that should be followed when delivering flawless user experience. He makes a good point when making the analogy of a drunk user. He points out that a drunk person requires a guidance, gets distracted easily and needs to be reminded more times of the same thing. All of that also applies to the Internet users.

Lukáš Marvan gave a great presentation [13] about User Experience (UX) testing. He sums up the most important aspects of usability testing:

- The respondent needs to feel comfortable.
- Refreshments should be available.
- The scenario should ideally form into a story.
- The respondent should always share his or her thoughts.
- The test is about the product, not the respondent.

6.4.2 Hypothesis

The respondents should be able to follow the testing scenario and complete all of the steps.

6.4.3 Testing

The developed system has been usability tested by three members fitting the target audience 16-50 age range both male and female animal breeders. Both persons have had some experience with creating the Internet content regarding their breeding station. During the testing the appended scenario G was followed.

Only three respondents were asked to participate in the testing because similar testing outcomes were observed in both cases. Doing more testing of the exact UI would not be cost effective.

The testing scenario was created so it would verify the most important functional requirements. Firstly, the testers were asked to browse through breeding station presentation simulating a potential buyer behaviour. That involved finding the `for sale` section, locating the profile of an animal of their interest and finally finding the contact information. The detailed list of steps is described in the Public Pages Usability Testing appendix G.1.

Secondly, the testers were told they liked the platform and fancied creating their own presentation. There is a list of detailed steps in the appendix G.2. Before each step they were told the motivation behind the action they were about to take. They started by creating the breeding station followed by filling in some text in the `About Us` page. Contact information was also demanded to insert.

They went on by creating their first animal, setting a planned litter and adding some children to it. They also experienced the process of changing the state of a child in the litter as well as a final sale of all the children followed by archiving the litter.

Lastly, the testers were told to create some content using the `custom pages` feature and also to put some links in the `Links` page.

6.4.4 Results

All three respondents were able to successfully finish the presented scenario.

In the first part examining the public profile they found the relevant information pretty quickly and without any hesitation. The section `For Sale` is logically placed in the upper part of the left menu. The contact information is displayed on every page at the bottom in the footer area where the visitor acknowledges it even before he or she needs it. Hence the assumptions made during the Design Proposal phase were confirmed.

The admin interface seemed to be more difficult to handle. Parts like breeding station creation or filling in the contact information were proven to be fine. The assumption is that the respondents have not struggled because they have already used similar forms in their lifetime.

However, the animal and litter creation became more difficult to execute. The respondents were confronted with UI components that they have never seen anywhere else before. It is due to the fact that the components were designed to solve the particular use case in the most sophisticated manner creating a room for possible confusion as a result of unfamiliarity.

Step #	times occured	Problem	Severity
11	2	Was unable to create a custom page.	**Critical**
12	2	Was unable to create a link.	**Critical**
4-2	2	Closed the uploading pictures dialog before the pictures got uploaded.	Serious
5-2	2	Did not use the selection menu for existing animals.	Serious
5-4	2	Did not use the diary feature when creating content for a child.	Serious
3	1	Did not follow the watermarked pattern for mobile phone number.	Minor
11	1	Deleted chars “=” acting as a underline for the markdown heading.	Minor
3	1	Filled in the street name without the house number.	Minor
12	1	Filled in a text with diacritics in the URL field.	Minor

Table 6.2. Severity levels of usability problems report

Another misconception has been observed when the respondents were asked to create a custom page. The system requires the user first to create a group for the page so that the link to the page could be properly categorized in the left menu. The testing showed that the respondents were confused about the group necessity which in the end forbade them to create the custom page at all.

The same behaviour was observed in the `Links` page. A better approach is proposed which would automatically create a general group for the first couple custom pages and let the breeder reorganize the custom pages later if he or she feels like it.

Overall, the respondents liked the system and showed interest in using it when it gets released.

Chapter 7
Static Code Analysis

In this chapter we talk about source code statistics. Source code line stats were in combination by IntelliJ IDEA Statistic plugin[1]) and SonarQube platform[2]).

The number of comment lines in table 7.1 might seem very low (below 1%), but as Robert C. Martin says [14] "The proper use of comments is to compensate for our failure to express ourself in code."

Language	Total	Blank Lines	Comments
Java	8051	1591	57
HTML (JSF)	3539	755	6

Table 7.1. Source code statistics

The numbers in table 7.2 were gathered using the Sonar tool. The complexity of Java code is computed following these rules:

- Keywords and operators increasing the complexity: `if`, `for`, `while`, `case`, `catch`, `return` (not being the last command), `&&`, `||`, `?`.
- Keywords that do not increase complexity: `else`, `default`, `finally`.
- Getters and setters do not count as methods hence they do not increase the complexity.

Code stats	Complexity	Duplications
101 classes	1.7 / method	0.7%
9 packages	9.0 / class	57 lines
542 methods	9.4 / file	4 blocks

Table 7.2. Java code statistics

[1]) `http://plugins.jetbrains.com/plugin/4509`
[2]) `http://www.sonarqube.org/`

Static code analysis available in IntelliJ IDEA IDE[1]) has been utilized. It offers over 600 automated code inspections [15] such as

- Finding probable bugs
- Locating the "dead" code
- Detecting performance issues
- Improving code structure and maintainability
- Conforming to coding guidelines and standards
- Conforming to specifications

All of these mentioned above were used during the development which resulted in higher code quality.

Chapter 8
Future

The proposed system is a proof of concept a.k.a prototype or pilot. It serves the purpose of finding out whether the service provided by the system would be useful to the target audience.

8.1 Scope Extension

The system is available only in the Czech Republic meaning that there is only one `.cz` internet domain running and only czech mutation was prepared. On the other hand the service is ready to expand and it is only matter of resources to translate texts and prepare additional set of internet domains. Since the system is just a proof of concept the resources available for the project have been spent on proving the concept, not scaling it to different areas.

The prototype is only considering dog breeders but the inner parts of the system are ready for handling more creatures. The plan is to add the creatures one by one because there needs to be some customization done for each individual creature.

8.2 Additional features

The use case is fairly simple. Let's say an average family decides to buy a puppy. Nowadays they can either browse through advertisements or use a search engine looking for "chihuahua puppies for sale in prague". But that is exactly all the information our system has to offer.

There could be a search component created. A visitor can filter puppies by breed and location of the breeding station having the results shown in a nice friendly way either on a map or in a list sorted by distance of the breeding station or for example age of the puppy. This feature becomes extremely powerful if the service spreads globally since a user (client) would be able to easily find the animal just across the borders.

8.3 Monetization

The service aims to be profitable in the future. The eventual profit might come in a long run since more functionality might be needed providing the prototype gets successful.

One way of making money could be via advertisements since the visitors will be mostly interested in a particular animal breed which can lead to great targeting options for potential advertisers.

Freemium Strategy might also be introduced which would offer more functionality for paying breeders.

8.4 No Success Prediction

There is a book about good decisions in life and work[16]. In the chapter "Prepare To Be Wrong" Chip Heath and Dan Heath introduce a study which talks about the `prospective hindsight` approach. It shows that when one tries to come up with reasons for an event that might happen in the future it is better to take the prospective hindsight approach and ask the other way around. Instead of thinking why the system would not succeed in the future the question is "It is one year after releasing the service to the public and the system is a total failure. Why hasn't it succeeded?". It forces us to fill in the blanks between today and a certain future event which feels a bit more concrete, offering firmer cognitive footholds.

Taking `prospective hindsight` approach here is a list of couple reasons why the service may not succeed:

- A similar service will emerge. One that will be more sophisticated and possibly a duplication of the system.
- Breeders will find the system hard to use due to it's complexity.
- The system will fail to satisfy one or more non-functional requirements which will result in driving away breeders and visitors.

Having such a list of possible reasons of failure will help to plan for the future.

Chapter 9
Conclusion

The analysis showed that there is a market opportunity for the Animal Breeders Content Management System service. The implemented system has yet to prove the concept.

The system has been designed and implemented in a fast prototyping way. On the other hand, the stress has been put to develop reusable code which would lead to the possibility of faster delivering the additional features.

The service is thoroughly tested embracing unit tests and integration tests which will result in significantly minimizing the downtime of the service.

As of now the system works flawlessly in the staging environment. The project will be released to the public later this year when all of the necessary business aspects are taken care of.

References

[1] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Prentice Hall, 2004, 2000. ISBN 9780131489066.

[2] Serdar Yegulalp. *Surprise! Java is fastest for server-side Web apps*, 2013. http://www.infoworld.com/t/java-programming/surprise-java-fastest-server-side-web-apps-230565.

[3] Ing. Jiří Mlejnek. *Návrh - návrhové třídy a vzory*, 2011. Katedra softwarového inženýrství, FIT, ČVUT. https://edux.fit.cvut.cz/oppa/BI-SI1/prednasky/BI-SI1-P06m.pdf.

[4] Luke Wroblewski. *Mobile First*. A Book Apart, 2011. ISBN 978-1-937557-02-7.

[5] Brad Frost. *Mobile-First Responsive Web Design*, 2011. http://bradfrostweb.com/blog/web/mobile-first-responsive-web-design/.

[6] Nitin Hayaran. *Vertically Responsive Design : Keeping Things Above The Fold*, 2013. 23Spaces. http://www.nitinh.com/2013/03/vertically-responsive-design-keeping-things-above-the-fold/.

[7] Danyl Bosomworth. *Mobile Marketing Statistics 2014*, 2014. http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/.

[8] James Bach. *The Challenge of "Good Enough" Software*, 1995, 2003. http://www.satisfice.com/articles/gooden2.pdf.

[9] Scott W. Ambler. *Just Barely Good Enough Models and Documents: An Agile Best Practice*, 2005-2014. http://www.agilemodeling.com/essays/barelyGoodEnough.html.

[10] Arthur Minduca. *Quality assurance in software development: When should you start the testing process?*, 2014. http://arthurminduca.com/2014/03/07/quality-assurance-in-software-development-when-should-you-start-the-testing-process/.

[11] Allan Kelly. *Testing triangles, pyramids and circles, and UAT*, 2013. http://allankelly.blogspot.cz/2013/05/testing-triangles-pyramids-and-circles.html.

[12] Will Dayble. *The User Is Drunk*, 2013. Squareweave. https://www.youtube.com/watch?v=r2CbbBLVaPk.

[13] Lukáš Marvan. *Testování použitelnosti prakticky*, 2011. AVG Technologies. http://pt.slideshare.net/BoBMarvan/testovn-pouitelnosti-prakticky.

[14] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, 2008. ISBN 9780132350884.

[15] IntelliJ IDEA Team. *IntelliJ IDEA How-To: Static Code Analysis*, 2013. JetBrains. http://www.jetbrains.com/idea/documentation/static_code_analysis.html.

[16] Dan Heath Chip Heath. *Decisive: How to Make Better Choices in Life and Work*. Crown Business, 2013. ISBN 9780307956392.

Appendix A
Abbreviations

AJAX **Asynchronous JavaScript and XML** is a group of interrelated Web development techniques used on the client-side to create asynchronous Web applications.

API **Application Programming Interface** specifies how some software components should interact with each other.

CSS **Cascading Style Sheets** is a style sheet language used for describing the look and formatting of a document written in a markup language.

DRY **Don't Repeat Yourself** is a principle of software development aimed at reducing repetition of information of all kinds.

GRASP **General Responsibility Assignment Software Patterns** consists of guidelines for assigning responsibility to classes and objects in object-oriented design.

HTTP The **Hypertext Transfer Protocol** is an application protocol for distributed, collaborative, hypermedia information systems.

IDE An **Integrated Development Environment** is a software application that provides comprehensive facilities to computer programmers for software development.

JSF **JavaServer Faces** is a Java specification for building component-based user interfaces for web applications.

JSON **JavaScript Object Notation** is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs.

PaaS **Platform as a Service** is a category of cloud computing services that provides a computing platform and a solution stack as a service.

SEO **Search Engine Optimization** is the process of affecting the visibility of a website or a web page in a search engine's `natural` or un-paid (`organic`) search results.

UI The **User Interface** is the space where interaction between humans and machines occurs.

URL A **Uniform Resource Locator** is a specific character string that constitutes a reference to an Internet resource.

UX **User Experience** involves a person's behaviors, attitudes, and emotions about using a particular product, system or service.

XML **Extensible Markup Language** is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

Appendix B
Functional Requirements

B.1 Implemented

1. System must let user manage his account.
    1. System must let user create a station.
    2. System must let user delete the station.
    3. System must let Individual user register using the link provided in a notification email received when buying an animal.
    4. System must let user sign in.
2. System must let user fill in About Us page content.
3. System must let user manage entries in the News page.
    1. System must let user fill in date, title, text and pictures.
4. System must let user manage links in the Links page.
    1. System must let user fill in URL and title.
5. System must let anonymous users leave a post in Guestbook.
    1. System must let anonymous user fill in name, email, link and text; fields name and text are mandatory.
    2. System must let admin of a station leave a reply to a post.
    3. System must let admin of a station modify or delete a post.
6. System must let user fill in contact information about the Station.
    1. System must let user fill in: Station name, FCI organization number, street, zip code, city, country, GPS, owner's name, phone number, email.
    2. System must provide Contact Page with a map of Station's location.
    3. System must let user insert HTML Content after the contact info.
7. System must let user manage animals.
    1. System must let user create an animal which s/he owns.
        1. System must let user select a breed.
        2. System must let user fill in animal's name.
    2. System must let user create and modify an animal which s/he doesn't own only for a litter parenthood purposes.
        1. System must let user only insert or modify name, profile picture and pedigree related information.
    3. System must let user set animal's state to Retired(not for breeding anymore).
    4. System must let user set date of death of an animal.

5. System must let user delete an animal only if there's no relationship in the system to it.
6. System must let user upload animal's profile picture.

8. System must let user manage litters.
   1. System must let user create a litter.
      1. System must let user choose dom and dad (create if non existing).
      2. System must let user fill in (expected) date of birth.
   2. System must let user manage children in a litter.
      1. System must let user change the state of an Animal (just born, for sale, reserved, sold, not for sale).
      2. System must let user transfer the ownership of a child.
         1. to another station.
         2. to individual person who will create an account if s/he has got none.
   3. System must let user delete a litter.
9. System must let user manage custom pages content.
10. System must provide a double footer.
    1. Upper part holds info about station.
    2. Bottom part holds info about the system.
11. System must provide appropriate error pages (404, etc.)

B.2 Not implemented

1. System must let user insert infinite number of warnings.
   1. Each warning can point to any user created page.
   2. Warnings are shown one at a time in random order at the top of the page.
   3. System must provide automatic warnings for litters for sale.
2. System must provide every page with an appropriate call to action element.
3. System must provide RSS feed with recent changes.
   1. System must provide updates for custom page content changes, new litter, new animal, new page, contact information changes.
4. System must let user change his password.
5. System must provide a wizard to help a user get acquainted with the system functions.
6. System must give user to the option of remembering him or her on the computer.
7. System must send an email notification to the anonymous user about the reply in the guestbook with a link to the reply.

Appendix C

Non-functional Requirements

1. To each Hypertext Transfer Protocol (HTTP) frontend page request shall system respond with fully generated html (SEO bots friendly).
2. System shall deliver all public frontend pages under 500 ms.
3. System frontend shall keep 60 fps.
4. System shall keep 99.9% uptime.
5. Webpage design shall be responsive.
6. Each page has to be represented with meaningful URL.

Appendix D
Use Case Scenarios

D.1 Station

Create station

Actor: Guest

1. UC starts with the user navigating himself to the sign up page.
2. He or she fills in `station name`, `email` and `password`.
3. The system generates automatically the URL based on the station name.
4. IF station with generated URL exists.
   - The system shows appropriate error message.
   - The system enables the user to change the URL to an available one.
5. The system creates the station.
6. The system sends an email that verifies the email address.
7. The user clicks on the verification link in the email body.
8. The system sets the user email and the station as confirmed.
9. UC ends.

Edit contact information

Actor: Admin

1. UC starts with the user being on the contact page.
2. The user clicks on the `edit contact information` link.
3. The user fills in the information.
4. The system automatically updates the map according to the information.
5. The user clicks on the `save` button.
6. The system saves the contact information.
7. UC ends.

Edit the about us section

Actor: Admin

1. UC starts with the user located on the `About Us` page.
2. The user clicks on the `Edit Text` link.
3. The system shows form.
4. The user edits the text.
5. IF the users selects `edit photos` option.

- The user deletes / uploads new pictures.
- The user saves the pictures.

6. The user clicks on the `save` button.
7. The system saves the modifications.
8. IF the user does now want to save the modifications.
   - The user cancels the modifications.
9. UC ends.

Delete a station

Actor: Admin

1. UC starts with the user being on the contact page.
2. The user clicks on the `edit contact information` link.
3. The user clicks on the `delete station` link.
4. The system deletes the station and all associated information.
5. The user is redirected to index page.
6. UC ends.

D.2 News Section

Add post

Actor: Admin

1. UC starts with the user being on the news page.
2. The user clicks on the `add post` link.
3. The user inserts text / uploads images.
4. The user clicks on the `save` button.
5. IF the user does now want to save the post.
   - The user cancels the creation of the post.
   - UC ends.
6. The system saves the post.
7. UC ends.

Edit post

Actor: Admin

1. UC starts with the user being on the news page.
2. The user clicks on the `edit post` link at the particular post.
3. The user modifies text / images.
4. The user clicks on the `save` button.
5. IF the user does now want to save the modifications.
   - The user cancels the modifications.
   - UC ends.
6. The system saves the modifications.
7. UC ends.

Delete post

Actor: Admin

1. UC starts with the user being on the news page.
2. The user clicks on the `delete post` link at the particular post.
3. The system deletes the post.
4. UC ends.

D.3 Links Section

Add link

Actor: Admin

1. UC starts with the user being on the links page.
2. The user clicks on the `add link group` link.
3. The user fills in the title for the group.
4. The user clicks on the `save` button.
5. The system saves the group.
6. The user clicks on the `add link` button.
7. The user inserts the title and the URL.
8. The user clicks on the `save` button.
9. IF the user does now want to save the link.
   - The user cancels the creation of the link.
   - UC ends.
10. The system saves the link.
11. UC ends.

Edit link

Actor: Admin

1. UC starts with the user being on the links page.
2. The user clicks on the `edit link` button at the particular post.
3. The user modifies the title or the URL.
4. The user clicks on the `save` button.
5. IF the user does now want to save the modifications.
   - The user cancels the modifications.
   - UC ends.
6. The system saves the modifications.
7. UC ends.

Delete link

Actor: Admin

1. UC starts with the user being on the links page.
2. The user clicks on the `delete link` button at the particular link.

3. The system deletes the link.
4. UC ends.

D.4 Guestbook

Leave post

Actors: Guest, Admin, System Emailer

1. UC starts with the user being on the guestbook page.
2. The user fills in `name` and `text`, optionally `email` and `web`.
3. The user clicks on the `save` button.
4. The system saves the entry.
5. The system sends a notification email to the breeder about the new post.
6. The system shows the entry at the top of the list of entries.
7. UC ends.

Edit post

Actor: Admin

1. UC starts with the user being on the guestbook page.
2. The user clicks on the `edit` link at the particular post.
3. The user modifies the post.
4. The user clicks on the `save` button.
5. The system saves the modifications.
6. UC ends.

Delete post

Actor: Admin

1. UC starts with the user being on the guestbook page.
2. The user clicks on the `delete` link at the particular post.
3. The system deletes the particular post.
4. The system updates the list of posts.
5. UC ends.

Leave reply

Actors: Guest, Admin, System Emailer

1. UC starts with the user being on the guestbook page.
2. The user clicks on the `reply` link at the particular post.
3. The system shows a form with prefilled name and a text field.
4. The user fills in `name` and `text`.
5. The user clicks on the `save` button.
6. The system saves the reply.
7. IF the user who left the original post had filled the email address.
   - The system sends a notification email to the creator of the original post.
8. UC ends.

D.5 Animal

Add animal

Actor: Admin

1. UC starts with the user being on the animal administration page.
2. The user clicks on the `add animal` link.
3. The system shows the option to select the sex of the animal.
4. The user selects the sex.
5. The system shows the option to select the breed of the animal.
6. The user selects the breed.
7. The system shows the list of animals yet without an owner.
8. The user selects an animal which he owns
9. IF the user cannot find the animal in the list of animals without an owner.
   - The user clicks on the `create animal` button.
   - The system shows a text field for the animal's name.
   - The user fills in the animal's name.
   - The user clicks on the `save` button.
   - The system creates the animal and adds it to the user's breeding station.
   - UC ends.
10. The system adds the animal to the breeding station.
11. UC ends.

Delete animal

Actor: Admin

1. UC starts with the user being on the animal administration page.
2. The user clicks on the `delete animal` button at the particular animal.
3. The system deletes the animal.
4. IF the animal is assigned as a parent to a litter or another animal.
   - The system removes the animal only from the user's breeding station.
5. The system updates the list of animals.
6. UC ends.

Edit animal's birthdate

Actor: Admin

1. UC starts with the user being on the animal profile page.
2. The user clicks on the `edit birthdate` button.
3. The system shows a text field with a calendar popup.
4. The user types or selects the date of birth.
5. The user clicks on the `save` button.
6. The system saves the birthdate.
7. UC ends.

Edit animal's general information

Actor: Admin

1. UC starts with the user being on the animal profile page.
2. The user clicks on the `edit general information` button.
3. The system shows a text field.
4. The user fills in the text / uploads pictures.
5. The user clicks on the `save` button.
6. The system saves the general information.
7. UC ends.

Set animal's parents

Actor: Admin

1. UC starts with the user being on the animal profile page.
2. The user clicks on the `add mom` / `add dad` button.
3. IF the animal already has a mom / dad.
   - The user clicks on the `remove mom` / `remove dad` button.
   - The system removes mom / dad from the animal.
   - UC continues with 2nd step.
4. The system shows the option to select an animal currently in the system having the same breed and the particular sex.
5. The user selects an existing animal.
6. IF the animal does not exist within the system.
   - The user clicks on the `create animal` button.
   - The system shows a text field for animal's name.
   - The user fills in the name of the animal.
   - The user clicks on the `save` button.
   - The system creates the animal.
7. The system assigns the animal as a mom / dad.
8. UC ends.

Add post to animal's diary

Actor: Admin

1. UC starts with the user being on the animal profile page.
2. The user clicks on the `add diary post` button.
3. The system shows form.
4. The user fills in text, date and uploads pictures.
5. The user clicks on the `save` button.
6. The system saves the post.
7. IF the owner is not the breeder.
   - The system sends an email notification to the breeder about the post.
8. The system updates list of posts in diary.
9. UC ends.

Change animal's state

Actor: Admin

1. UC starts with the user being on the animal profile page.
2. The user clicks on the `for sale` / `retired` / `in memory` button.
3. The system saves the animal state.
4. IF there is no `for sale` section for the breed.
   - The system creates `for sale` section in the left menu.
   - UC ends.
5. IF there is no `retired` / `in memory` section in the left menu.
   - The system creates `retired` / `in memory` section in the left menu.
6. UC ends.

D.6 Litter

Add litter

Actor: Admin

1. UC starts with the user being on the litter administration page.
2. The user fills in the breed and a letter in the add litter section.
3. The user clicks on the `save` button.
4. The system saves the litter.
5. The system updates the list of litters.
6. UC ends.

Set the date

Actor: Admin

1. UC starts with the user being on the litter administration page.
2. The user clicks on the date of the particular litter in the litters list.
3. The system shows a text field with calendar popup.
4. The user fills in the date.
5. The user blurs the text field.
6. The system saves the date.
7. UC ends.

Edit general information

Actor: Admin

1. UC starts with the user being on the litter profile page.
2. The user clicks on the `edit general information` button.
3. The system shows a form.
4. The user fills in the text / uploads pictures.
5. The user clicks on the `save` button.
6. The system saves the general information.

7. UC ends.

Set parents

Actor: Admin

1. UC starts with the user being on the litter profile page.
2. The user clicks on the `add mom` / `add dad` button.
3. The system shows the option to select an animal currently in the system having the same breed and the particular sex.
4. The user selects an existing animal.
5. IF the dad does not exist within the system.
   - The user clicks on the `create animal` button.
   - The system shows a text field for animal's name.
   - The user fills in the name of the animal.
   - The user clicks on the `save` button.
   - The system creates the animal.
6. IF the mom does not exist within the breeding station.
   - The system shows a message that the user needs to add the mom in the animal administration first.
   - UC ends.
7. The system assigns the animal as a mom / dad.
8. UC ends.

Add child

Actor: Admin
Precondition: The litter has to have both parents assigned.

1. UC starts with the user being on the litter profile page.
2. The user clicks on the `add child` button.
3. The system shows the form.
4. The user fills in the name and the sex.
5. The user clicks on the `save` button.
6. The system saves the child.
7. The system updates the list of children.
8. UC ends.

Add post to child's diary

Actor: Admin

1. UC starts with the user being on the litter profile page.
2. The user clicks on the `add post` button at the particular child.
3. The system shows the form.
4. The user fills in the text / uploads pictures.
5. The user clicks on the `save` button.
6. The system saves the post.
7. The system sets the first uploaded picture as a new profile picture for the child.

8. UC ends.

Change child's state

Actor: Admin

1. UC starts with the user being on the litter profile page.
2. The user click on the `not for~sale` / `for sale` / `reserved` / `sold` button at the particular child.
3. The system saves the state.
4. UC ends.

Archive litter

Actor: Admin
Precondition: All of the litter's children have to have the state `sold` or `not for~sale`.

1. UC starts with the user being on the litter profile page.
2. The user clicks on the `archive the~litter` button at the particular litter.
3. The system saves the `archived` litter state.
4. IF there is no `past litters` section for the particular breed in the left menu.
   - The system creates the `past litters` section for the particular breed in the left menu.
5. UC ends.

D.7 Custom Page

Add custom page

Actor: Admin

1. UC starts with the user being on the custom pages administration page.
2. The user clicks on the `add custom page group` link.
3. The user fills in the title of the group.
4. The user saves the group.
5. The user clicks on the `add custom page` link.
6. The user fills in the name of the custom page.
7. The user clicks on the `save` link.
8. The system saves the custom page.
9. The system shows the custom page in the left menu.
10. UC ends.

Edit custom page content

Actor: Admin

1. UC starts with the user being on the particular custom page.
2. The user clicks on the `edit content` link.
3. The user modifies the text / uploads / deletes the pictures.
4. The user confirms the modifications.

5. IF the user does now want to save the modifications.
   - The user cancels the modifications.
   - UC ends.
6. The system saves the modifications.
7. UC ends.

Delete custom page

Actor: Admin

1. UC starts with the user being on the custom pages administration page.
2. The user clicks on the `delete` link at the particular custom page.
3. The system deletes the page along with it's content.
4. The system updates the list of custom pages in the left menu.
5. UC ends.

Appendix E
Public Pages Content

Full list of public pages along with it's URL and content.

E.1 About Us

`/<station-name>`

- Introduction [content]
- Latest piece of news [dated content]

E.2 News

`/<station-name>/news`

- List of news [dated content]
  - Date
  - Title (optional)
  - Text [content]

E.3 Links

`/<station-name>/links`

- List of links groups
  - List of links
    - Url
    - Title

E.4 Guestbook

`/<station-name>/guestbook`

- Post form
- List of posts
  - Contributor's Name
  - Date

- Email
- Website
- Text
- Breeder's reply

E.5 Contact

```
/<station-name>/contact
```

- Map
- Contact Info
  - Breeding station name
  - Street
  - City
  - Zip
  - Country
  - GPS (optional)
  - Owner's name
  - Phone
  - Email
- More information (optional) [content]

E.6 Animal Profile

```
   station - /<station-name>/<animal-name-urlfied>
no station - /animal/<breed>/<animal-name-urlfied>
```

- Animal Info
  - Profile picture
  - Name
  - Date of birth
  - Date of death (optional)
  - Additional info (examinations, show winnings) [content]
- Parents
- Pedigree
- Siblings
- Diary (only for Animal with station) [dated content]
- Past Litters
- Descendants

E.7 Breed — (Fe)Males

```
/<station-name>/<breed>/(fe)males
```

- List of animals

E.8 Breed For Sale

```
/<station-name>/<breed>/for-sale
```

- Individual animals for sale
- Current litters for sale
  - Letter
  - Date
  - Parents
  - Children
    - Name
    - State
    - Latest post in diary
- Planned litters for sale
  - Letter
  - Date
  - Parents

E.9 Past Litters

```
/<station-name>/<breed>/past-litters
```

- List of past litters
  - Letter
  - Date
  - Parents
  - Children

E.10 Litter Profile

```
/<station-name>/<breed>/litter/<letter>/<litter-id>
```

- Letter
- Date
- Parents
- Information (optional) [content]
- Children
  - Name
  - Latest post in diary

E.11 Retired

```
/<station-name>/retired
```

- List of retired animals

E.12 In Memory

```
/<station-name>/in-memory
```

- List of dead animals

E.13 Custom Page

```
/<station-name>/s/<page-group-name-urlfied>/<page-name-urlfied>
```

- Content [content]

Appendix F
Admin Pages Content

Full list of admin pages along with it's URL and content.

F.1 Animals Admin

```
/<station-name>/animal-admin
```

- List of animals
- Add an animal form

F.2 Litters Admin

```
/<station-name>/litter-admin
```

- List of litters
- Add a litter form

F.3 Custom Pages Admin

```
/<station-name>/custom-pages-admin
```

- List of custom page groups
  - List of custom pages

Appendix G
User Experience Test Scenario

User experience test scenario presented to the respondents.

G.1 Public Pages

This section describes the most common case when a potential buyer is looking for a puppy.

1. You would like to buy a puppy. Find the animals for sale.
2. Choose a puppy that you fancy the most and look for more pictures of it.
3. You really like the puppy and want to get in touch with the breeder. Find the contact information.

G.2 Admin Pages

A breeder likes the service and wants to create his or her own presentation.

1. Sign up your breeding station.
2. Write one paragraph in the "About Us" section describing the essence of your station.
3. Fill in the contact information.
4. Add a bitch.
    1. Fill in her birthdate.
    2. Upload her profile picture.
    3. Fill in the important information such as her nature, awards won, etc.
    4. Add her parents.
5. There is a litter expected to be born in two months. Create it.
    1. Fill in the expected date.
    2. Add the parents of the litter.
    3. The bitch just gave birth to three vital puppies. Add a general information about the litter along with a picture of the mum and her children.
    4. One week passed and you decide to add the children to the litter. Add the first puppy girl publish her fresh pictures along with her weight measurement.
    5. Add a boy.
    6. Add the last girl.
6. Mark that you want keep the first girl thus it is not going to be for sale.
7. You just sold the boy to another breeding station also registered with the service. Transfer the ownership of the animal.

8. The last girl was just sold to a guy without a breeding station. Transfer the ownership using his email address.
9. All of the puppies are sold. Archive the litter moving it to the "Past Litters" section.
10. The guy who bought the last girl does not want to pay for it so you decide to take the puppy back. Reclaim your ownership.
11. What is it that you like besides dogs? Oh you have a horse? Ok, create a custom page about the horse.
12. Publish two links in the links section pointing to your favourite breeding stations.
13. You just figured that you want to sell the first bitch that you own. Mark it for sale.
14. Unfortunately it died before you managed to sell it. Mark it dead moving it to "In Memory" section.

Appendix H
CD Contents

CD Contents

- `/src/` - diploma thesis source files in plainTEX format
- `/polcama1_2014dipl.pdf` - diploma thesis in PDF format