FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Prototypical implementation of an animal health record (AHR) for livestock management

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medizinische Informatik

eingereicht von

Christoph Aigner
Matrikelnummer 0525400

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Ao.Univ.Prof. Dipl.-Ing. Mag.rer.nat. Dr.techn. Rudolf Freund

Wien, 28.01.2014

(Unterschrift Verfasser) (Unterschrift Betreuer)

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am ---

Name

Acknowledgments

I would like to thank first and foremost my grandparents, my parents and my friends for their continuous support without which this thesis would have never been possible.

Moreover I would like to express my sincere gratitude to my advisor, Ao.Univ.Prof. Dipl.-Ing. Mag.rer.nat. Dr.techn. Rudolf Freund for his guidance and support.

Kurzfassung

Diese Diplomarbeit beschäftigt sich mit der prototypischen Konzeptionierung und der Implementierung einer elektronischen Krankenakte für die Nutztierhaltung (abgekürzt AHR). Ein AHR besteht aus mehreren Datensätzen, welche Gesundheitsdaten für Tiere enthalten. Diese Arbeit beschäftigt sich mit Tieren, die Teil eines Tierbestandes im Sinne der Nutztierhaltung sind. Der entwickelte Prototyp ist eine Webanwendung, welche mit der Java Enterprise Edition-Technologie entwickelt wurde und von Landwirten, weiterverarbeitenden Betrieben, staatlichen Einrichtungen und Tierärzten gleichermaßen verwendbar ist. Zuerst werden die derzeitige Marktsituation, die organisatorischen bzw. technischen Standards und die Grundlagen der Java Webentwicklung beleuchtet. Danach wird die Konzeption dargelegt und Details der Implementierung diskutiert. Zum Schluss wird das Ergebnis anhand von Screenshots und textuellen Beschreibungen gezeigt. Im Rahmen der vorliegenden Arbeit wurde der Design-Prozess und die konkrete Implementierung eines Prototypen detailliert vorgestellt. Der entwickelte Prototyp ist kompatibel zu den EHR Standards, ermöglicht einen plattformunabhängigen Zugriff und ist von allen beteiligten Akteuren nutzbar.

Keywords: *veterinärmedizinische Krankenakte, AHR, hl7, Nutztierhaltung, javaee, java server faces, hibernate, mysql*

Abstract

This master thesis deals with the prototypical design and implementation of an animal health record (AHR) for livestock management. An AHR is a series of electronic datasets that contains health related information for animals. This work focuses on animals that are part of a livestock. The prototype is a web application that is implemented with Java Enterprise Edition technology and can be used by farmers, processing industries, governmental bodies and veterinarians. First the current software market situation, the organizational respectively technical standards and the fundamentals of web development with Java are being analysed. Then the software design is laid out and details of the implementation are being discussed. Finally the results are shown in the form of screenshots and textual descriptions. The goal of this thesis was to design and implement a prototype that is standard compliant, easy accessible through all computer platforms and can be used by all involved stakeholders.

Keywords: *animal health record, ahr, hl7, livestock management, javaee, java server faces, hibernate, mysql*

Table of contents

Table of figures

List of tables

1 Introduction

"*Healthcare quality improvement is an economic and moral necessity. The transformation, which is needed to improve productivity and effectiveness, will rely on computer interoperability to deliver information when and where required, support soundly-based decision-making, eliminate unnecessary repetition, reduce delays and avoid errors*." [1]

This citation originating from literature dealing with medical informatics in the domain of human health care also applies to veterinary medicine especially within livestock husbandry where governmental bodies, agricultural holdings and medical institutions must work together to fulfil various legal requirements respectively the needs of the end-consumer-market.

1.1 Problem description and motivation

Goal of this master thesis is the prototypical implementation of an animal health record (from now on abbreviated as AHR) within the domain of livestock husbandry. An AHR is a series of electronic datasets that contains health information. The main difference between an AHR for pets and an AHR for livestock husbandry is that the owner of the animal being recorded may not just be a single person, but can also be a whole agricultural holding, which has to be modelled as well. The medical field of activity differs as well. Veterinarians working with livestock are responsible for the prevention of epidemics, must be aware of legal regulations concerning food products and should assist the owner with the breeding process. Governmental respectively non-governmental entities and organizations can also be considered a part of animal health records and thus may be a part of the data model as well.

The software will be used by veterinarians, farmers respectively by governmental bodies. It is implemented as a web application and is accessible via a personal computer using standard web browser software. The system also provides interfaces for communication with other applications (e.g. Web Services). Security standards are maintained as well (e.g. SSL/TLS) as the possibility to incorporate external data (e.g. animal data from "Agrarmarkt Austria").

Health records are part of the medical documentation process, which is an essential area of medical informatics. In human medicine lifelong recording of electronic health data is an evolving concept, thus many standards for data storage and transmission have already been developed. This master thesis shows, on the basis of a prototypical implementation, which interoperability standard is best suitable for an AHR in livestock husbandry (e.g. HL7, openEHR).

Since the application is web based, suitable technology was evaluated during the development process. The premise was that the software (programming languages, APIs, IDEs, server software, tools) which is going to be applied within this application is based on open source technology, platform independent and accessible by any computer supporting modern web browsers.

1.2 Expected Results

The expected result is a functional system architecture respectively a working prototype of an animal health record for livestock management.

It will be built on the Java Platform, Enterprise Edition Version 6 (short Java EE 6) and implemented as a web application. It will include parts of the use cases defined within the AHR group to analyse the interaction between the determined user groups (e.g. farmers, veterinarians), thus aiding in creating a user-interaction-concept. In the course of the development process the following questions will be dealt with:

Standards

This work will clarify which human medicine based concept can be used to implement an AHR. It will decide if it is reasonable to use the HL7v3 RIM standard as a basis for a data model or if any other standard is more suitable for the task.

Design

The thesis will elaborate on a software design for an animal health record. It will define the project stakeholders, the requirements and a user/role concept. A database design that is based on an EHR standard will be made. The details of the underlying software architecture will be laid out.

Implementation

The software will be developed as a web application that is built on Java Enterprise Edition. Technology choices for the application logic, the database implementation, and security aspects respectively the graphical user interface will be made. The work will answer the question, if a combination of web technology (HTML, Ajax or Flash/Flex) with Java EE technology (including Spring, Hibernate) is reasonable and practical.

The thesis will discuss if external interfaces should be provided for third party applications and on which technology they should be based on (e.g. SOAP, Servlet).

It will be clarified which external data the application should integrate (e.g. animal data from AMA).

The prototype will include the whole source code and a compiled binary for deployment on the server. The system will run on any standard desktop PC and, since the application is going to be built on Java EE technology, the required software to run the application will be freely accessible.

1.3 Structure of thesis

The first step is the requirements analysis (functional and non-functional), the stakeholders and the use-cases. The next step is the design of the underlying system architecture and database. These processes will be accomplished with other colleagues working on these specific topics as a team and are depicted in chapters 3 and 4.

The Unified Modelling Language (UML) is used for the software design. The structure will be modelled with class diagrams and package diagrams. As a typical process in UI prototyping the first graphical user interfaces are going to be sketched by hand.

The next step will be the determination of the technology to build the graphical user interfaces. There are two main choices:

- JavaServer Faces (built on HTML, JavaScript and AJAX)
- Adobe Flex (built on the open-source version of Adobe Flash Technology)

Furthermore the technology for the external interface has to be determined. The two choices are SOAP or RESTful web services.

Before the implementation is going underway the right Integrated Development Environment (short IDE) has to be chosen.

The application logic will be developed with Java EE technology. The development will rely on the use of software engineering design patterns (e.g. MVC Pattern, DAO, and Dependency Injection). It will make use of the Spring Framework and Hibernate.

The database will be implemented with MySQL technology using the SQL 99 Standard for definition and manipulation of data. These processes are depicted in chapter 5.

Chapter 6 outlines the result by means of screenshots and textual descriptions.

1.4 State-of-the-art

There are already commercial AHR software systems available on the market. The products “easyVET” developed by the German based software company “IFS Informationssysteme GmbH” and “Vetinf” developed by the company “Vetinf GmbH”, which is also located in Germany focus mainly on veterinary practices and hospitals (see Chapter 2.2).

The Austrian company “SEG Informationstechnik GmbH” developed the product “ANIMAL office” which is a practice management system that also includes special modules for livestock management [2].

The School of Veterinary Medicine, University of Milan developed a veterinary electronic patient record named the O3-Vet project, that is compliant with the IT standard HL7, DICOM and IHE [3].

Viktoria Willner wrote a master thesis that evaluates the requirements for an AHR especially for small animals (e.g. pets) [4]. It discusses the EHR standards (like HL7, openEHR) and medical nomenclatures (ICD 10, SNOMED and LOINC) in connection with animal health records. Furthermore she conducted an empirical study with veterinarians to determine their needs.

Andrea Füresz wrote a master thesis about a software architecture design for an AHR [5]. After analysing the current market situation and the legal constraints she defined the stakeholders, a role concept, the requirements and the software architecture. Her work will be a basis for the prototypical implementation.

2 Fundamentals of electronic health records

This chapter will give an overview of electronic health records respectively animal health records. It will provide a basic definition of health records, a fundamental overview of existing AHR-related software products and services (especially in Austria) and an in-depth view of various technical standards used in conjunction with animal health records.

2.1 Definition of an EHR/AHR

According to [6] an electronic health record can be understood as an electronic record of a patient's health information generated by one or more encounters in any care delivery setting. It includes patient demographics, progress notes, problems, medication, vital signs, past medical history, immunizations, laboratory data and radiology reports [6]. The electronic health record is part of the medical documentation process and thus is indented to be a mechanism for integrating health care information currently collected in both paper and electronic medical records (EMR) for the purpose of improving quality of care [7].

There is no consistent definition for an animal health record (AHR) in literature [4]. According to [4] an animal health record can be defined as a lifelong cross-institutional record that contains all relevant medical data and associated documents of an animal. This work expands that definition by focusing on animals that are part of a livestock.

An AHR in the context of this work can be understood as a series of electronic records containing lifelong medical respectively organizational data (e.g. disease prevention, slaughter protocols) for animals that are part of a livestock. This animals can be:

- Cattle
- Horses
- Pigs
- Sheep
- Goats
- Poultry

2.2 Existing software products

The survey Viktoria Willner [4] conducted with Austrian veterinarians concluded that the following three software products were the most commonly used:

- "ANIMAL-office"
- "easyVET"
- "Vetinf"

"ANIMAL-office" was the most frequently installed program followed by "easyVET" and “Vetinf”.

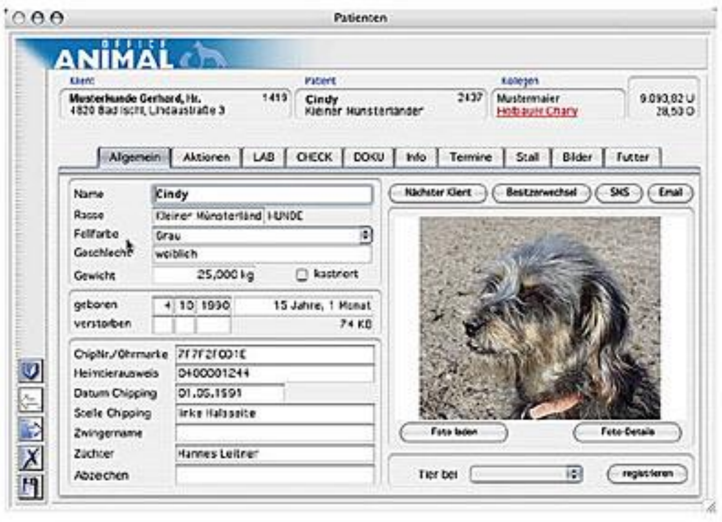
All of the highlighted software products include modules for accounting and organization as well. “ANIMAL-office” includes modules for livestock management in contrast to its competitors.

2.2.1 “ANIMAL-office”

"ANIMAL-office" is a practice management software developed by the Austria company "SEG Informationstechnik GmbH" [2]. There are versions available for small (“ANIMAL-office KLEINTIER”) and for large (“ANIMAL-office GROSSTIER”) animal vet practices. The software can be deployed as a client-server or as a standalone application.

“ANIMAL-office KLEINTIER”

The software for small animal vet practices includes modules for the management of animal standing data, treatment data, accounting and an ordering system. The use of a product code scanner simplifies animal and product (e.g. medicine, fodder) identification. The product is connected to various animal databases (including animaldata.com) [2].



Klient: Musterhunde Gerhard, Hr. 1419
4820 Bad Ischl, Lindaustraße 3

Patient: Cindy 2137
Kleiner Munsterländer

Kollegen: Mustermaier
Holbauer Chary

9.093,82 U
28,50 O

Name	Cindy
Rasse	Kleiner Münsterländ HUNDE
Fellfarbe	Grau
Geschlecht	weiblich
Gewicht	25,000 kg ☐ kastriert
geboren	4 10 1990 15 Jahre, 1 Monat
verstorben	74 KG
ChipNr./Ohrmarke	7F7F2F0D1E
Heimtierausweis	0400001244
Datum Chipping	01.05.1991
Stelle Chipping	linke Halsseite
Zwingername	
Züchter	Hannes Leitner
Abzeichen	
Tier bei	

Figure 1: Interface of ANIMAL-office for animal standing data [2]

"ANIMAL-office GROSSTIER"

The version for large animal vet practices includes modules for livestock management and breeding guidance. It is able to produce various statistics of given breeding data. It also includes solutions for pairing and insemination.

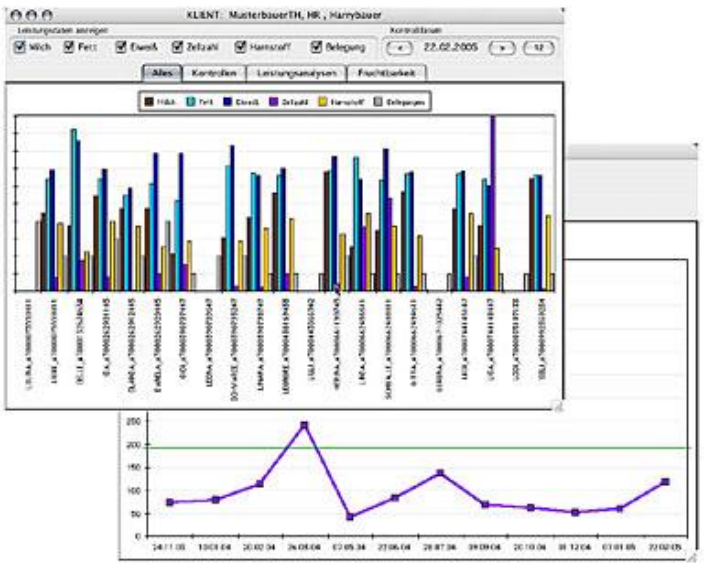
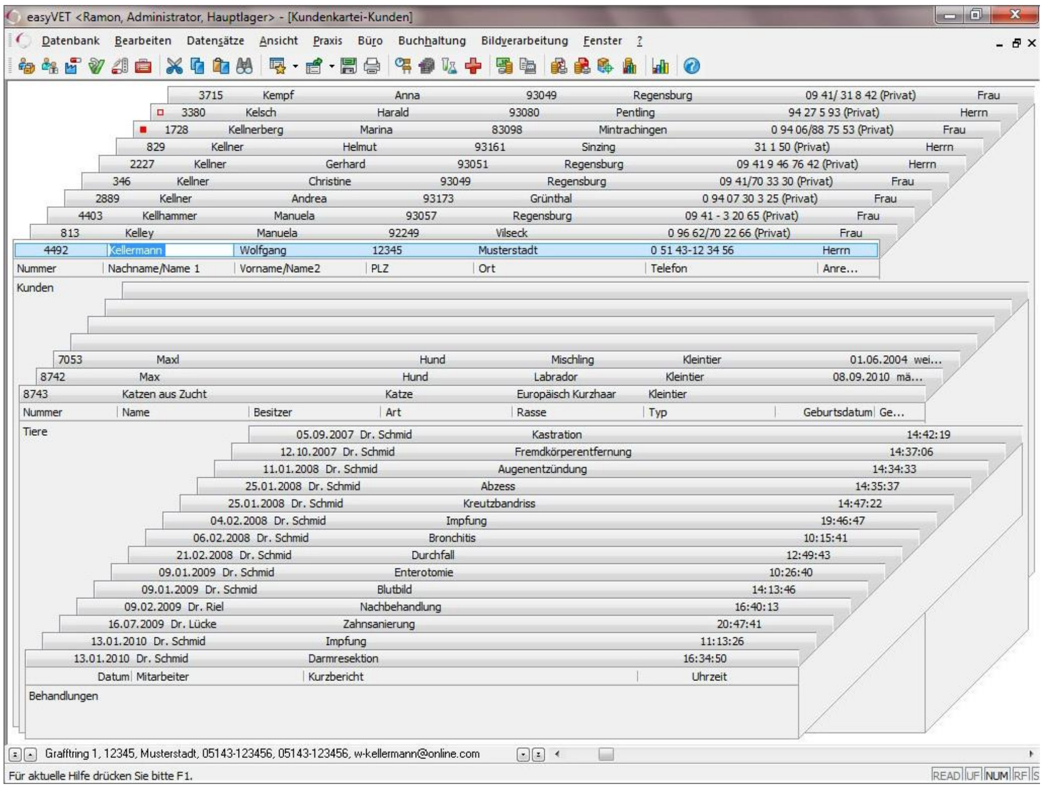


Figure 2: Statistics module of ANIMAL-office GROSSTIER [2]

2.2.2 "easyVET"

"easyVET" is a praxis management software developed by the company "IFS Informationssysteme GmbH" from Hannover, Germany [8]. Its key concept is based on filing boxes. The main screen of the program consists of three filing boxes. The first one contains the pet/livestock owners. The second one contains the livestock. The third one is a representation of the clinical records [8].

“easyVET” also includes modules for accounting, statistics and clinic management. It has a built-in calendar function and a to-do-list. It can also be connected to various laboratory systems [8].



Nummer	Nachname/Name 1	Vorname/Name2	PLZ	Ort	Telefon	Anre...
3715	Kempf	Anna	93049	Regensburg	09 41/ 31 8 42 (Privat)	Frau
3380	Kelsch	Harald	93080	Pentling	94 27 5 93 (Privat)	Herrn
1728	Kellnerberg	Marina	83098	Mintrachingen	0 94 06/88 75 53 (Privat)	Frau
829	Kellner	Helmut	93161	Sinzing	31 1 50 (Privat)	Herrn
2227	Kellner	Gerhard	93051	Regensburg	09 41 9 46 76 42 (Privat)	Herrn
346	Kellner	Christine	93049	Regensburg	09 41/70 33 30 (Privat)	Frau
2889	Kellner	Andrea	93173	Grünthal	0 94 07 30 3 25 (Privat)	Frau
4403	Kellhammer	Manuela	93057	Regensburg	09 41 - 3 20 65 (Privat)	Frau
813	Kelley	Manuela	92249	Vilseck	0 96 62/70 22 66 (Privat)	Frau
4492	Kellermann	Wolfgang	12345	Musterstadt	0 51 43-12 34 56	Herrn

Kunden

Nummer	Name	Besitzer	Art	Rasse	Typ	Geburtsdatum	Ge...
7053	Maxl		Hund	Mischling	Kleintier	01.06.2004	wei...
8742	Max		Hund	Labrador	Kleintier	08.09.2010	mä...
8743	Katzen aus Zucht		Katze	Europäisch Kurzhaar	Kleintier		

Tiere

Datum	Mitarbeiter	Kurzbericht	Uhrzeit
05.09.2007	Dr. Schmid	Kastration	14:42:19
12.10.2007	Dr. Schmid	Fremdkörperentfernung	14:37:06
11.01.2008	Dr. Schmid	Augenentzündung	14:34:33
25.01.2008	Dr. Schmid	Abzess	14:35:37
25.01.2008	Dr. Schmid	Kreutzbandriss	14:47:22
04.02.2008	Dr. Schmid	Impfung	19:46:47
06.02.2008	Dr. Schmid	Bronchitis	10:15:41
21.02.2008	Dr. Schmid	Durchfall	12:49:43
09.01.2009	Dr. Schmid	Enterotomie	10:26:40
09.01.2009	Dr. Schmid	Blutbild	14:13:46
09.02.2009	Dr. Riel	Nachbehandlung	16:40:13
16.07.2009	Dr. Lücke	Zahnsanierung	20:47:41
13.01.2010	Dr. Schmid	Impfung	11:13:26
13.01.2010	Dr. Schmid	Darmresektion	16:34:50

Behandlungen

Grafftring 1, 12345, Musterstadt, 05143-123456, 05143-123456, w-kellermann@online.com

Für aktuelle Hilfe drücken Sie bitte F1.

Figure 3: Main screen of “easyVET” displaying the card box paradigm [8]

2.2.3 „Vetinf“

“Vetinf” is developed by the German company “Vetinf Gmbh”. The product is available in two versions “Vetinf Basis” and “Vetinf Standard”.

“Vetinf Basis” includes software modules for the management of patient data, medical records and accounting.

“Vetinf Standard” contains extra modules for clinical management, statistics, pharmacy management and the integration of laboratory systems.

Additionally modules for financial accounting and mobile use are available.

2.2.4 O3-Vet

O3-Vet is an electronic patient record that is compliant with the IT standard HL7, DICOM and IHE.

The software was developed as an open-source web application. The PHP scripting language was used to implement the dynamic portion of the web interfaces. Database access was developed with the help of the PEAR framework and distribution system for reusable PHP components. MySQL was used as the database backend. According to [3] the database contains two different sections: the first one stores all data structures to satisfy the requirements of the IHE model. The second one stores all necessary structures to implement the veterinary electronic patient record.

According to [3] the system was tested from May to October 2006 and results show that the majority of the veterinarians involved in the test agreed on the advantages obtained by the use of application.

2.2.5 Rinderdatenbank AMA

According to [9] the BSE incidents of the 1990s has caused a massive decline in the consumption respectively export of beef. As a countermeasure the European Union decided to establish a new edict for the identification of cattle within a central database. The "Agrarmarkt Austria" (short AMA) was assigned the task to build such a database in Austria. The "Rinderdatenbank" was officially recognised by the EU on the 1st of October 1999.

Every cattle owner must report any incident (birth, acquisition, sale, slaughter, death) within a period of seven days to the AMA database.

The database can be accessed online via the "eAMA" web application [10]. There are no technical interfaces for third party applications available.

2.2.6 RDV4M

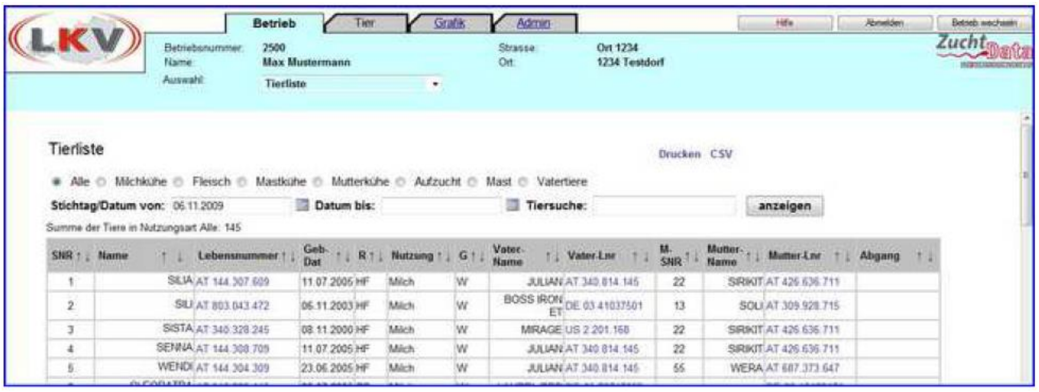
RDV4M (Rinderdatenverbund) is a software implemented by the company "Zucht Data EDV-Dienstleistungen GmbH". It is available as a web application for livestock owners that are members of a "Landeskontrollverband LKV". "eAMA" user credentials can be used to gain access to the system [11]. The system supports data export via PDF or comma-separated values.

The software consists of four modules:

- Module “BETRIEB”
- Module “TIER”
- Module “GRAFIK”
- Module “ADMIN”

Module “BETRIEB“

This module maintains data about the livestock itself, performed work, departed animals and breeding values. Inseminations can be saved and subsequently printed out as as well.



Betriebsnummer: 2500
Name: Max Mustermann
Auswahl: Tierliste
Strasse: Ort 1234
Ort: 1234 Testdorf

Tierliste

● Alle ○ Milchkühe ○ Fleisch ○ Mastkühe ○ Mutterkühe ○ Aufzucht ○ Mast ○ Vatertiere

Stichtag/Datum von: 06.11.2009 Datum bis: Tiersuche:

Summe der Tiere in Nutzungsart Alle: 145

SNR	Name	Lebensnummer	Geb-Dat	R	Nutzung	G	Vater-Name	Vater-Lnr	M-SNR	Mutter-Name	Mutter-Lnr	Abgang
1	SILIA	AT 144.307.609	11.07.2005	HF	Milch	W	JULIAN	AT 340.814.145	22	SIRIKIT	AT 426.636.711	
2	SILI	AT 803.043.472	06.11.2003	HF	Milch	W	BOSS IRON ET	DE 03 41037501	13	SOLI	AT 309.928.715	
3	SISTA	AT 340.328.245	08.11.2000	HF	Milch	W	MIRAGE	US 2.201.168	22	SIRIKIT	AT 426.636.711	
4	SENNA	AT 144.308.709	11.07.2005	HF	Milch	W	JULIAN	AT 340.814.145	22	SIRIKIT	AT 426.636.711	
5	WENDI	AT 144.304.309	23.06.2005	HF	Milch	W	JULIAN	AT 340.814.145	55	WERA	AT 687.373.647	

Figure 4: RDV4M livestock summary („Tierliste“) [11]

Module “TIER”

The module “TIER” can be accessed by selecting an animal using its identification number in one of the lists provided by the module “BETRIEB”. It contains information about the heritage, calving, lactation and meat production of the animal. Activities can be stored chronologically.

Module “GRAFIK”

This module displays interactive charts of results gathered from sample milkings.

Module “ADMIN”

This module lets the user configure various parameters of the application.

2.2.7 Veterinärinformationssystem (VIS)

The “Veterinärinformationssystem” (VIS) is a combination of the central pig database “Zentrale Schweinedatenbank” (ZSDB), which was created in 2002, and the database for sheep and goat livestock that was introduced in 2005 [12]. The legal basis is the “Tierkennzeichnungs- und Registrierungsverordnung 2009 idgF” (TKZVO 2009).

The database is helpful in the prevention of contagions and also provides a consistent animal identification for sheep and goats [12].

VIS is implemented as a web application that is accessible via the Statistik.at web portal. A username/password combination is required to access the system.

2.3 Technical Standards

Technical standards have been or are being developed to first and foremost support health care interoperability. According to [13] the development of standards is essential for sharing patient health information between health professionals respectively supporting interoperability between organisations and software from different vendors. This chapter describes the following standards:

- Health Level 7 (HL7 v2, HL7v3 RIM, CDA)
- CEN/ISO EN13606
- openEHR

HL7 and CEN/ISO EN13606 are both standards for information exchange in contrast to openEHR which claims to be a standard that defines a complete electronic health record.

The following standards were all developed for usage in the domain of humane medicine but can also be adapted to be used within veterinary medicine.

2.3.1 HL7 (Health Level 7)

The name “Health Level 7” abbr. HL7 has two meanings. First, it is the name of an international voluntary organization with affiliates in 31 countries that is accredited with the American National Standards Institute (ANSI) and also collaborates with international (ISO TC215) and European (CEN TC251) standards development organizations. It creates standards for the exchange, management and integration of electronic healthcare information for clinical and administrative purposes [1]. Three times a year, a week-long working group consisting of HL7 volunteers is working on the development of the HL7 standards [1].

The second meaning is the name of the standards themselves. It is derived from the ISO's Open System Interconnect (OSI) network model [1]. According to [14] the OSI model defines a seven-layer-stack, with each layer providing a specific network function. Level one to four provides the network services and communicates data across the network (interconnection). Level five to seven form the end-user layers and assist application management (interworking) [14]. HL7 resides on the seventh layer of the ISO/OSI reference model, the application layer and is thus called "Health Level 7".

Layer	Name	Purpose
1	Physical Layer	Data transport media
2	Data Link Layer	Local network routing and addressing
3	Network Layer	Remote network routing and addressing
4	Transport Layer	Data collection and aggregation
5	Session Layer	Extended duration connection management
6	Presentation Layer	Data conversion
7	Application Layer	End-user applications

Table 1: ISO/OSI network model [14]

HL7 in Austria

The Austrian authority for HL7 is the "HL7 Anwendergruppe Österreich". It is a non-profit organization that assists data-communication within the public health-care sector through the international standards of Health Level 7 [15]. Members have the ability to download HL7 related papers and are also able to participate in discussion forums. An annual individual membership costs 75 Euros.

HL7 Version 2

HL7 v2.0 was published in 1988, one year after the initial release of HL7 v1.0. It has since been under continuous development. The latest version HL7 v2.7 was published in 2011. According to [16] it is the workhorse of electronic data exchange in the clinical domain and arguably the most widely implemented standard for healthcare in the world.

HL7v2 messages are encoded in ASCII format and are therefore human readable [17].

An integral part of HLv2 is its backward compatibility. According to [1] this is very important because older versions are still widely used due to the technical respectively financial risks involved in upgrading to a newer version of the standard. HL7v2 achieves its backward compatibility by marking old methods as deprecated instead of removing them completely.

HL7v2 messages are sent in response to trigger events [1]. An HL7 message consists of one or more segments which are separated by the carriage return character \r or 0x0D in hexadecimal [17]. The first segment is the message header. Each segment contains one specific category of information, such as patient information or lab results [17]. The name of each segment is specified in the first field which is always three characters long[17]. "MSH" for example is the code for the message header. Segments (resp. segment groups) of the same type can be repeated.

Information that is not defined by the HL7 standards can be encoded with custom segments. Those segments begin with the letter Z and are thus called Z-segments [17]. Usually such segments are being ignored by applications who don't know how to handle them [17].

A segment is further divided into fields (also called composites) which are separated by the pipe (|) character. Composites may contain one or more sub-composites which are delimited by the caret (^) character. If those sub-composites also contains composites the ampersand (&) character is used to divide them. Sub-sub-composites must be primitive data types [17].

Symbol	Usage
|	Field separator
^	Component separator
~	Repetition separator
\	Escape character
&	Subcomponent separator
<CR>	Segment terminator

Table 2: HL7v2 delimiters [1]

The following example shows a message returning from a laboratory containing a test result for serum glucose with a value of 182 mg/dL authored by Howard H. Hippocrates [18]. The test was ordered by Patricia Primary for Patient Eve E. Everywoman [18]. The use case takes place in the US Realm [18]. The format is HL7 V2.4.

```
MSH|^~\&|GHH LAB|ELAB-3|GHH OE|BLDG4|200202150930||ORU^R01|CNTRL-
3456|P|2.4<cr>
```

```
PID|||555-44-4444||EVERYWOMAN^EVE^E^^^^L|JONES|19620320|F|||153
FERNWOOD DR.^
^STATESVILLE^OH^35292||(206)3345232|(206)752-121||||AC555444444||67-
A4335^OH^20030520<cr>
OBR|1|845439^GHH OE|1045813^GHH
LAB|15545^GLUCOSE|||200202150730||||||||
555-55-5555^PRIMARY^PATRICIA P^^^^MD^^|||||||||F||||||444-44-
4444^HIPPOCRATES^HOWARD H^^^^MD<cr>
OBX|1|SN|1554-5^GLUCOSE^POST 12H
CFST:MCNC:PT:SER/PLAS:QN||^182|mg/dl|70_105|H|||F<cr>
```

The first segment contains the message header. Its ninth field determines the message type. In this case it is an "ORU^R01" message. ORU is described as an "Observation result - unsolicited" [1]. R01 is the trigger event. It is being describes as "ORU/ACK - Unsolicited transmission of an observation" [19]. The sending applications is the GHH Lab in ELAB-3 [18]. The receiving application is the GHH OE system located in BLDG4 [18]. The message was sent on 2002-02-15 at 09:30 [18].

The second segment contains the patient identification (PID). It includes the date of birth (1962-03-20), the place of residence (Statesville, OH) and the patient ID number (555-44-4444).

The third segment contains the observation request (OBR). It encodes the fact that Patricia Primary MD requested the observation 15545^GLUCOSE which in turn was performed by Howard Hippocrates MD.

The final segment contains the observation result (OBX). The result was 182 mg/dL.

HL7 Version 3

The design process on the HL7 Version 3 standard began in the year 1992 with the establishment of a task force [1].

One of the main characteristics of HL7v2 was its flexibility [1]. It allowed programmer to customize the protocol to their needs by using optional data elements and segments. The drawback of this flexibility is of course rising complexity. Most vendors made use of this flexibility resulting in lots of different implementations.

Version 3 addresses these and other problems by introducing an object-oriented methodology and a Reference Information Model (RIM) to create messages [1]. The primary goal for HL7v3 was to offer a definite and testable standard for health care interoperability, covering the entire healthcare domain, with the ability to certify vendors' conformance [1].

The RIM consists of three main classes (Act, Role and Entity) which are linked together using three association classes (Act-Relationship, Participation and Role-Link) [1].

Every event is modelled as an Act [1]. Acts may have numerous Participations, which are Roles being played by Entities [1]. Acts may also be related to other Acts, via Act-Relationships [1].

Each of the three main classes are modelled as "is-a" relations (generalization). According to [20] generalization is achieved by concentrating the properties, attributes respectively relations, of similar entities into one super-entity. Attributes that cannot be synthesized remain in the sub-entity [20]. Sub-entities can act as super-entities as well. A sub-entity can also be called a specialization of a super-entity. For example, Veterinarian is a sub-entity of Person. Veterinarian inherits all attributes from its super-entity Person.

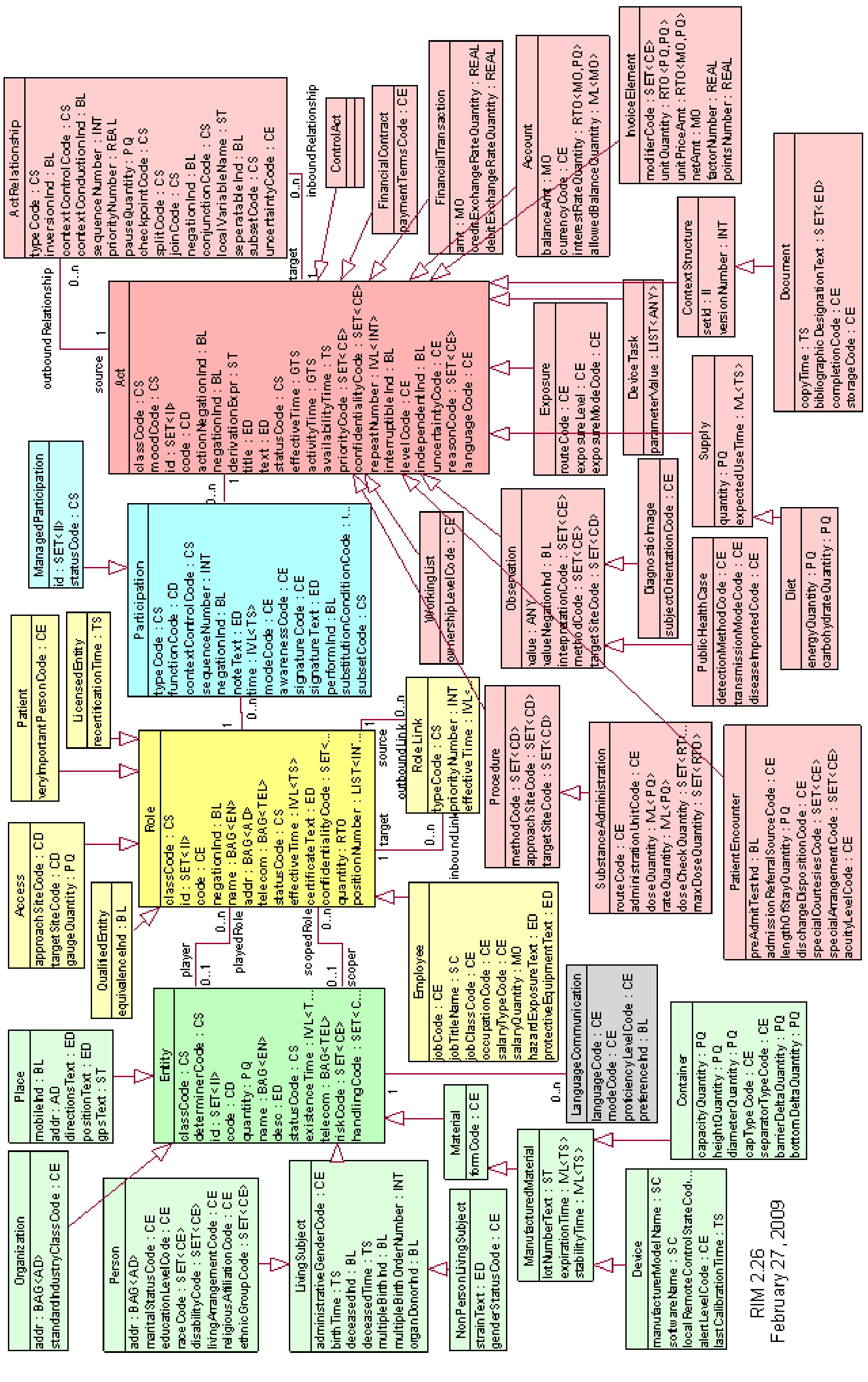


Figure 5: HL7 Reference Information Model [21]

HL7 CDA

HL7 CDA Release 2.0 is based on XML and specifies the structure and semantics of clinical documents [22]. It is part of the HL7v3 specification and was approved as an ANSI standard on May of 2005 [22]. CDA documents derive their machine processable meaning from the HL7v3 RIM and use the HL7v3 data types [23].

A basic document is wrapped by the `<ClinicalDocument>` element which contains a header and a body [23]. The header identifies and classifies the document and provides information on authentication, the encounter, the patient, and the involved providers [23]. The content of the body is the clinical report itself, which can be either an unstructured blob (binary large object) or can be comprised of structured markup [23]. Figure 2 shows a structured body (`<structuredBody>` element) that is divided into multiple sections. The narrative block (`<text>` element) is a critical component and must contain the human readable content to be rendered [23].

```
<ClinicalDocument>
  ... CDA Header ...
  <structuredBody>
    <section>
      <text>(a.k.a. "narrative block")</text>
      <observation>...</observation>
      <substanceAdministration>
        <supply>...</supply>
      </substanceAdministration>
      <observation>
        <externalObservation>...
        </externalObservation>
      </observation>
    </section>
    <section>
        <section>...</section>
    </section>
  </structuredBody>
</ClinicalDocument>
```

Figure 6: Major components of a CDA document [23]

Some implementations are using the HL7 CDA standard for the exchange of messages (e. g. laboratory results, prescriptions), making it hard to differentiate HL7 documents and HL7 messages [23].

CDA documents can be exchanged in HL7 messages or other transport solutions (e-mail/MIME, etc.) [23].

2.3.2 CEN/ISO EN13606

According to [24] the overall goal of EN13606 is to define a rigorous and stable information architecture for communicating part or all of the EHR of a single subject of care, to support the interoperability of systems and components that need to communicate EHR data via electronic messages. It was not intended to specify the internal architecture or database design of EHR systems or components [24]. It was developed by the European Committee for Standardisation 251 (CEN/TC 251) using the openEHR archetype methodology [25].

EN13606 follows a dual model architecture that defines a clear separation between information and knowledge [26]. The reference model contains the basic entities for representing any information of the EHR [26]. The archetypes, which are formal definitions of clinical concepts, are structured and constrained combinations of the entities of a reference model [26].

Reference model

The reference model consists of seven components. The top-level container is called EHR_EXTRACT. It contains EHR data as compositions, optionally organized by folders [24]. Compositions contain entries, optionally contained within sections (which may be nested) [24]. Entries contain elements, optionally contained within clusters (which may be nested) [24].

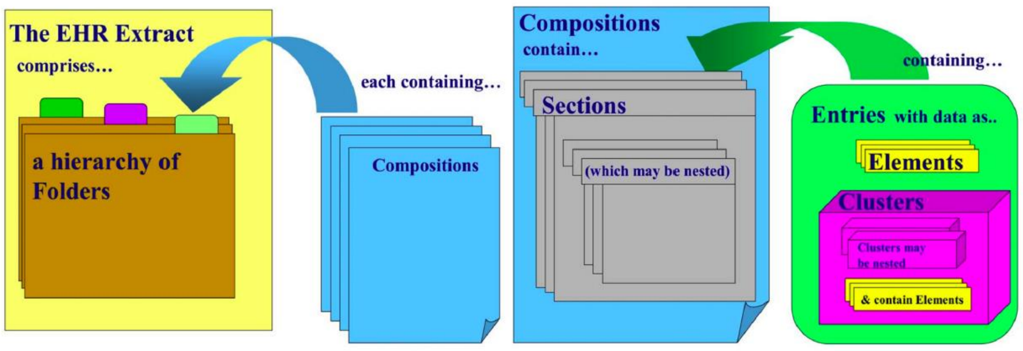


Figure 7: EN 13606 reference model [24]

Archetype model

Archetypes are comprised of three main sections [26]:

- Header

- Definition
- Ontology

The header contains metadata about the archetype (e. g. identifier, author) [26]. The definition section contains the description of the clinical concept which represents the archetype in terms of reference model entities [26]. The ontology part defines all linguistic entities [27].

2.3.3 openEHR

The openEHR standard is maintained by the openEHR Foundation which is an international, not-for-profit company whose mission is 'to promote and facilitate progress towards electronic health records of high quality, to support the needs of patients and clinicians everywhere' [25].

A minimal EHR system that follows the openEHR standard consists of an EHR repository, an archetype repository, terminology (if available), and demographic/identity information [28].

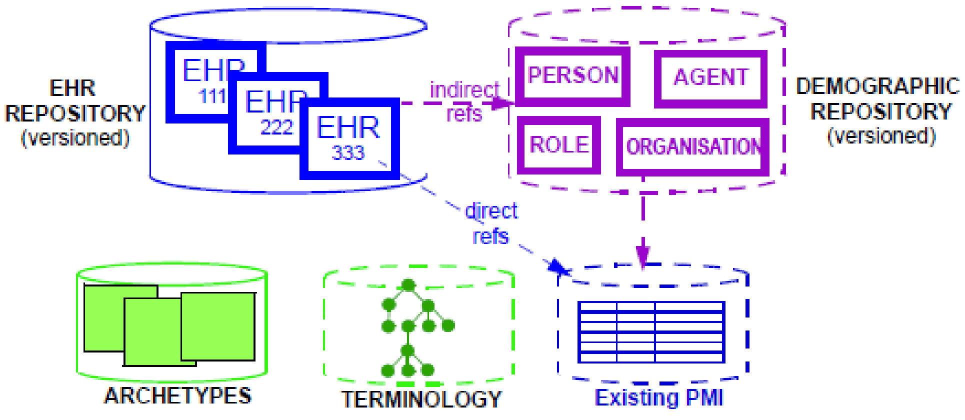


Figure 8: Minimal openEHR EHR System [28]

2.3.4 DICOM

The Digital Imaging and Communications in Medicine (short DICOM) Standard is maintained by the multi-specialty DICOM Standards Committee and is being developed by 26 different workgroups [29]. It specifies a non-proprietary data interchange protocol, a digital image format and a file structure for biomedical images including image-related information [29]. DICOM interfaces can connect image acquisition equipment (e.g. CT, MRI, and ultrasonography) with image archives and image processing respectively display devices.

The standard was first conceived in 1985 by a joint committee consisting of the ACR (American College of Radiology) and the NEMA (National Electrical Manufacturers Association). It was followed up by Version 2.0 published in 1988 and Version 3.0 published in 1993. The standard has since been revised many times, mostly on yearly bases [29].

A DICOM image contains, besides the image date itself, a header consisting of various Information Object Definitions (IODs) such as patient data or modality information. Used data compression standards are: JPEG, JPEG Lossless, JPEG 2000, or MPEG-2 for multi-image (video) sequences [29]. Images can be viewed with stand-alone applications (e.g. Adobe Photoshop, IrfanView) or complete Picture archiving and communication systems (PACS) [29].

A robust open-source implementation of the DICOM standard is "dcm4che". It is written in Java and can be deployed on JDK 1.4 and up [30].

2.4 Classification systems

2.4.1 SNOMED CT

Systematised Nomenclature of Medicine (SNOMED) maintains a controlled vocabulary with comprehensive coverage of diseases, clinical findings, etiologies, therapies, procedures and outcomes [31].

It was realized in 1975 as a successor to the Systematized Nomenclature of Pathology (SNOP) Standard, which was published in 1965 [1].

In 2002 a merger of SNOMED and NHS Clinical Terms Version 3 led to the release of SNOMED CT [1]. As of January 2009, it contained over 310,000 active concepts, 990,000 English descriptions, and 1.38 million relationships [1].

SNOMED CT is composed of components that are identified by a SNOMED CT Identifier (short SCTID) and have a validity status. These components can be concepts, relationships, descriptions, subsets and cross maps.

Concepts represent distinct clinical meanings. They are identified by a SCTID and are associated with a set of relationships and two or more descriptions [32]. A description links a human-readable term with a concept [1]. There are three description types [32]:

- **Preferred Term** is the most common word or phrase to name a concept.
- **Fully Specified Name** is an unambiguous way to name a concept
- **Synonyms** are the rest of the names that may be used for a concept.

Relations are the connections between concepts. Every SNOMED CT concept is at least related to one other concept.

Subsets are used to specify picking lists for specific data entry fields to increase usability [1].

Cross Maps link SNOMED CT terms to other terminologies [32].

Example SNOMED CT Concept [32]:

```
Fully Specified Name: Myocardial infarction (disorder)
DescriptionID 751689013
Preferred term: Myocardial infarction
DescriptionID 37436014
Synonym: Cardiac infarction
DescriptionID 37442013
Synonym: Heart attack
DescriptionID 37443015
Synonym: Infarction of heart
DescriptionID 37441018
```

2.4.2 LOINC

According to [33] Logical Observation Identifiers Names and Codes (LOINC) provides a set of universal names and ID codes for identifying laboratory and clinical test results. Its current scope includes all observations reported by clinical laboratories, including the specialty areas of chemistry, toxicology, haematology, serology, blood bank, microbiology, cytology, surgical pathology and fertility. A large number of veterinary medicine terms have also been included [33].

The official LOINC manual [33] states that the fully specified name of a test result or clinical observation has five or six main parts including the name of the component or analyte measured, the property observed, the timing of the measurement, the type of the sample, the scale of the measurement and - where relevant - the method of the measurement.

Formal syntax description of the fully specified name [33]:

```
<Analyte/component>:<kind of property of observation or
measurement>:<time aspect>:<system (sample)>:<scale>:<method>
```

Example for Gamma-GT (liver enzyme):

```
Gamma glutamyl transferase:CCnc:Pt:Ser/Plas:Qn
```

The components full name is "Gamma glutamyl transferase". "CCnc" stands for "Catalytic Concentration". The time aspect "Pt" refers to a point in time. "Ser/Plas" indicates that the sample came from blood plasma or serum. "Qn" refers to a quantitative scale, which means that the result of the test is a numeric value that relates to a continuous numeric scale [33].

2.4.3 Nomina Anatomica Veterinaria

The Nomina Anatomica Veterinaria (N.A.V.) is a compilation of anatomical terms for use in veterinary science. The scientific text is prepared by the International Committee of Veterinary Gross Anatomical Nomenclature [34].

The first edition was published in October of 1968. Before that, many publications used terms of direction related to the human standing position, with the forearms supinated in a posture that is impossible in most animals [34]. The 5th edition, which is the current edition, was initially published in 2005. A revised version, containing merely corrections of typographic errata was made available in 2012. The 5th edition was published primarily on the Internet to increase distribution and at the same time reduce costs [34].

A set of principles, which agree to a large extent with those of the Nomina Anatomica, have served as guides in the work of the Committee: Each anatomical concept should be designated by a single term. Each term should be in Latin. Terms should be as short and simple as possible. Terms should be easy to remember and should have instructive respectively descriptive value. Topographically close structures should have similar names. Differentiating adjectives should usually be opposites. Eponyms should not be used.

3 Fundamentals of web development with Java EE

This chapter explains basic methods of web development with the Java Platform, Enterprise Edition (short Java EE) on which the prototype is built on.

3.1 Basic Web technologies

Every web application, whether it is developed with Java EE, .NET or any other programming framework relies on the basic web technologies to either interact with users via web browsers or other systems via interfaces.

Web standards can be described in terms of layers. There are three basic layers that are built on each other, resembling the ISO/OSI reference model [35]:

- Structural Layer
- Presentation Layer
- Behavioural Layer

The structural layer is formed by the marked up document. It builds the foundation on which other layers may be applied [35]. The presentation layers provided instructions on how the document should look on the screen, sound when it is read aloud, or be formatted when it is printed [35]. These instructions are specified with Cascading Style Sheets (CSS). The behavioural layer on top adds interactivity and dynamic effects to a site [35]. This layer is specifically important for modern web applications, because it increases and simplifies user interaction.

3.1.1 Structural Layer

HTML

HTML (HyperText Markup Language) is a mark-up language that describes the appearance and content of a webpage. It was created in 1991 by Tim Berners-Lee as a simple way to indicate the meaning and structure of hypertext documents [35]. Since then, browser developers augmented the HTML specification with their own subset of tags. To overcome this issue, Berners-Lee founded the World Wide Web Consortium (W3C) in 1994 [35]. The W3C continues to release updated and standardized versions of HTML in publications knows as “Recommendations” [35]. The latest version is HTML 4.01 [36] respectively HTML 5.1 which is still a draft [37].

The reformulation of HTML 4 is called XHTML 1.0 which uses the same vocabulary but the syntactical rules are derived from XML (eXtensible Markup Language) [38]. For example, an empty element must either have an end tag or the start tag must end with “`/>`”. The line break in HTML 4 looks like this: “`<br>`”. In XHTML 1.0 it looks like “`<br />`” or “`<br></br>`”.

3.1.2 Presentation Layer

CSS

Cascading Style Sheets (CSS), a W3C standard, define the presentation of web documents [35]. The term presentation refers to the way a document is displayed or delivered to the user, whether it is on a computer monitor, a mobile phone display or read aloud by a screen reader [35].

Despite never intended to be a presentational language, HTML still supported tags for the look of a webpage, e.g. the `<font>` tag. With the introduction of CSS Level 1 in 1996 those tags became obsolete.

CSS 1 contained all the basics for attaching font, colour and spacing instructions to elements on a web page [35] Internet Explorer 3 was the first browser to implement CSS 1.

In 1998 CSS Level 2 was released. It added properties for positioning elements on the page, introduced media type, table layout properties, aural style sheets and additional sophisticated methods for selecting elements. CSS Level 2 Revision 1 (CSS 2.1) was released in 2011. It fixed errors, deleted properties and moved some unsupported features to the CSS 3 specification [35].

CSS 3, unlike CSS 2 is divided into several separate documents called modules. Some of them are still working drafts and some have already been published as formal recommendations. CSS 3 adds support for vertical flowing text, improved table handling, international language and better integration with other XML technologies such as SVG (Scalable Vector Graphics), MathML, and SMIL (Sychronized Multimedia Interchange Language) [35].

3.1.3 Behavioural Layer

Object models

DOM (Document Object Model) provides a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents [39]. The Document Object Model Level 1 Specification was released in 1998 and covers core HTML and XML documents as well as document navigation and manipulation [35]. DOM Level 2 was published in 2000 and includes a style sheet object model. DOM Level 3 was introduced in 2004. It enhances DOM 2 by completing the mapping between DOM and the XML Information Set, including the support for XML Base, adding the ability to attach user information to DOM Nodes or to bootstrap a DOM implementation. It also provides mechanisms to resolve namespace prefixes or to manipulate "ID" attributes, giving to type information [40]. DOM4 is still a working draft.

JavaScript

JavaScript is a web scripting language that was introduced with Netscape Navigator 2.0 [35]. It was standardized in 1998 by the W3C in coordination with the ECMA International, an international industry association dedicated to the standardization of information and communication systems [35]. JavaScript is a superset of the ECMAScript standard scripting language [35].

AJAX

Asynchronous JavaScript + XML (short AJAX) is an approach to web interaction that involves transmitting only a small amount of information to and from the server in order to give the user the most responsive experience possible [41].

Instead of the web browser, an intermediate layer called the "Ajax Engine" is responsible for initiating requests to, and processing requests from the web server [41]. Requests are done asynchronously, meaning that code execution does not wait for a response before continuing [41].

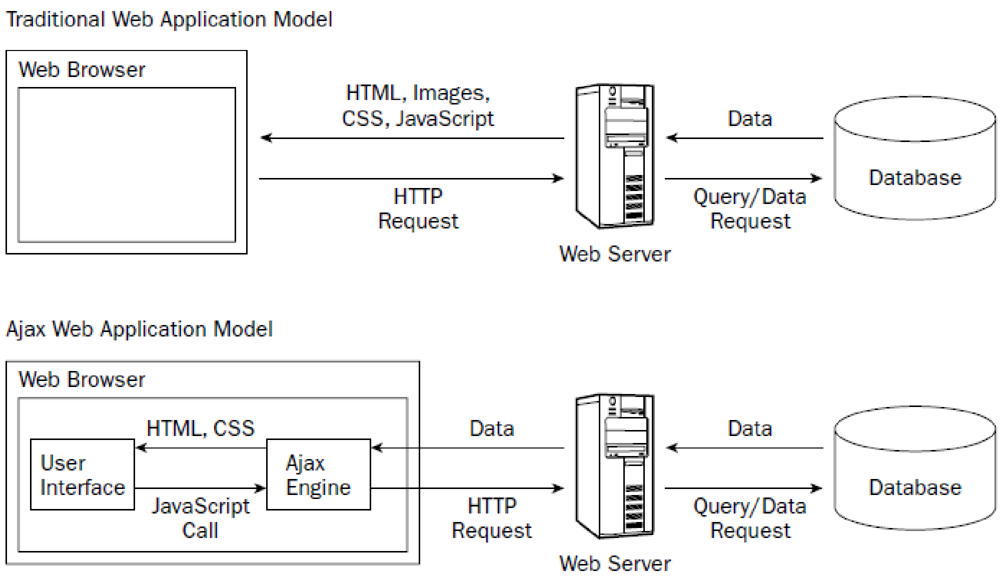


Figure 9: Comparison of the traditional and Ajax web application model [41]

The Ajax engine goes into action after receiving the server response, often parsing the data and making several changes to the user interface based on the information that was provided [41].

3.2 Overview of the Java Platform, Enterprise Edition

Java EE provide a standard-based platform for developing multi-tiered web and enterprise applications [42]. It is a superset of Java SE (Java Standard Edition). Typically these applications contain a frontend tier consisting of web frameworks, a middle tier providing security and transactions, and a backend tier providing connectivity to a database or a legacy system [42]. Java EE provides APIs for the development of transactional, interoperable and distributed applications, just like Java SE provides low-level APIs for common problems (e.g. Collections) [43]. Java EE defines four types of components [43]:

- Applets
- Applications
- Web applications
- Enterprise applications

Applets are applications that run in web-browsers using the Java-plugin often providing a GUI. Applications in this context are understood as client-side applications which are typically GUIs or batch-processing programs. Web applications are executed in a web container and respond to HTTP requests from web clients. Enterprise applications are executed in container-managed components for processing transactional business logic (Enterprise Java Beans) [43].

The initial J2EE 1.2 specification was released in December 1999. J2EE 1.3 followed in September 2001. J2EE 1.4 was released in November 2003 and introduced web services. Java EE 5 was made available in May 2006.

Edition 6 of the Java EE Platform was published in 2009. It focuses mainly on productivity improvements. XML configurations are replaced by annotations. A "web profile" was introduced which aims towards a lightweight web development process. Java EE 7 was released in May 2013 [44].

The following figure depicts the different components that are working together for the provision of an integrated stack [42]

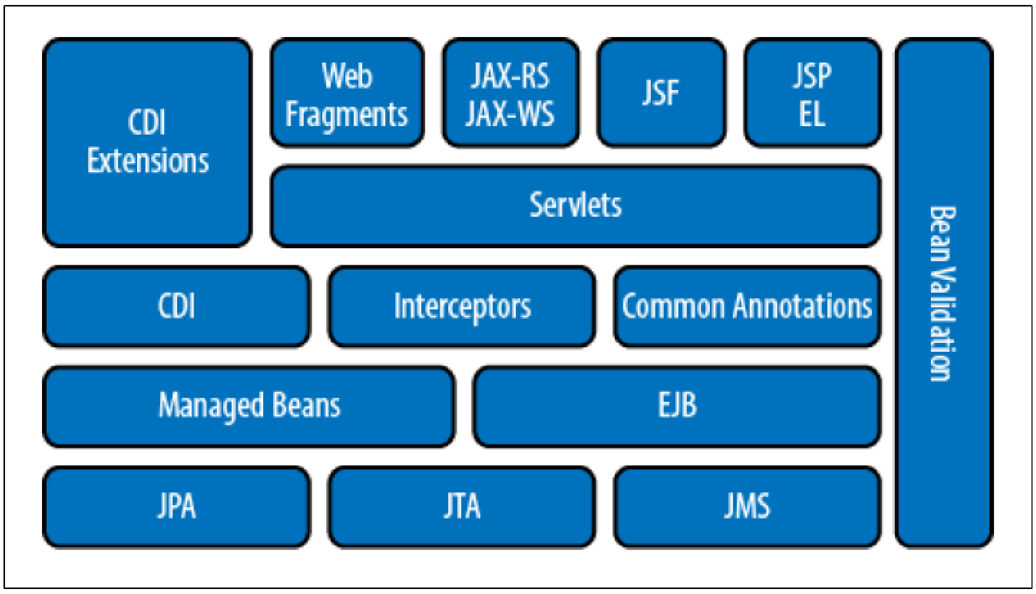


Figure 10: Java EE 6 architecture [42]

Java Persistence API (JPA), Java Transactional API (JTA) and Java Message Service (JMS) provide the basic services such as database access, transactions and messaging [42]. Managed Beans and Java Enterprise Beans (EJB) provide a simplified programming model using POJOs [42]. According to [42] Context and Dependency

Injection (CDI), Interceptors and Common Annotations provide concepts that are applicable to a wide variety of components, such as type-safe dependency injection, addressing cross-cutting concerns using interceptors and a common set of annotations. Web services using the technologies JAX-RS and JAX-WS, Java Server Faces (JSF), Java Server Pages (JSP) and Expression Language (EL) define the programming model for web applications. Third-party web frameworks can be automatically registered with Web Fragments. According to [42] CDI Extensions allow you to extend the platform beyond its existing capabilities in a standard way. Bean Validations provides a standard means to declare and validate constraints [42].

The reference implementation (RI) of Java EE 6 is the GlassFish application server. Sun Microsystems created the GlassFish project in 2005 after donating the Tomcat technology to the Apache Foundation [43]. Its latest version is 4.0 which was introduced with Java EE 7.

3.2.1 Enterprise Java Beans

According to [43] Enterprise Java Beans (EJBs) are server-side components that encapsulate business logic and take care of transactions and security. They are used to build the business layer located between the persistence and the presentation layer.

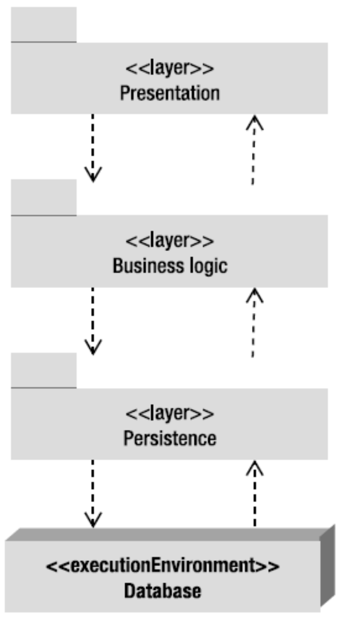


Figure 11: Architecture layering [43]

A Plain Old Java Object (POJO) can be deployed into an EJB container by using annotations [43]. These EJB containers are runtime environments that provides services, such as transaction management, concurrency control, pooling and security authorization [43].

The Java EE platform defines several types of EJBs [43]:

- Session Beans
- EJB timer service
- Message-driven Beans (MDBs)

Session Beans

Session beans are the most important part of the EJB technology, because they are used to encapsulate high-level business logic [43]. There are three different types of Session beans:

- Stateless
- Stateful
- Singleton

Stateless session beans do not maintain any conversational state on behalf of a client application. They are independent, self-contained and the most popular bean component [43]. Stateless session beans are defined by using the `@Stateless` annotation on a POJO.

Stateful beans preserve conversational state. They are used for more complex tasks that cannot be done in one single step. A popular example is the shopping cart, where various items can be added to the cart before the checkout occurs. Stateful session beans are defined by using the `@Stateful` annotation on a POJO. To eliminate the bean from memory the `@Remove` annotation can be used on a method, or alternatively the class can be annotated with the `@StatefulTimeout(value,unit)` annotation [43].

Singleton beans are session beans that are instantiated only once per application [43]. They are defined by using the `@Singleton` annotation.

Timer service

The timer service is a scheduling facility that allows EJBs to be registered for callback invocation [43]. They are intended for long-lived business processes and are by default persistent [43]. Timers can be created automatically if the bean has methods annotated with the `@Schedule` annotation or can also be created programmatically and must provide one callback method annotated with the `@Timeout` annotation.

Message driven beans (MDBs)

Message driven beans are asynchronous message consumers. They cannot be accessed by client applications directly but rather be triggered by sending messages to the destinations that the MDBs are listening to [43].

3.2.2 Java Persistence API (JPA)

The Java Persistence API (JPA) was part of the Java EE 5 specification to solve the problem of data persistence. JPA 2.0 was introduced with Java EE 6. It is an abstraction above JDBC that makes it possible to be independent of SQL [43]. The JPA consists of the following main components [43]:

- Object-relational Mapping (ORM)
- Entity Manager API
- Java Persistence Query Language (JPQL)
- Java Transaction API (JTA)
- Callbacks

ORM is the mechanism to map objects to data stored in a RDBMS [43]. The Entity Manager API performs database-related operations, such as Create, Read, Update, Delete operations [43]. JPQL retrieves data by using an object-oriented query language [43]. JTA provides transactions and locking mechanisms when accessing data concurrently.

Example of a simple Entity:

```
@Entity
public class Person {
        @Id @GeneratedValue
        private int persnr;
        @Column(length = 200)
        private String name;

        // Constructors, getters, setters
}
```

For a POJO to be recognized as an entity, it must be annotated with the `@Entity` annotation. The `@Id` annotation is used to indicate the primary key. `@GeneratedValue` tells the persistence provider to automatically generated the value of the identifier [43].

Hibernate

Hibernate is an object/relational persistence and query service that implements the JPA 2.0 (JSR 317) and the Enterprise JavaBeans 3.0 specification (JSR 220) [45]. It is licensed under the open source GNU Lesser General Public License (LGPL) [45].

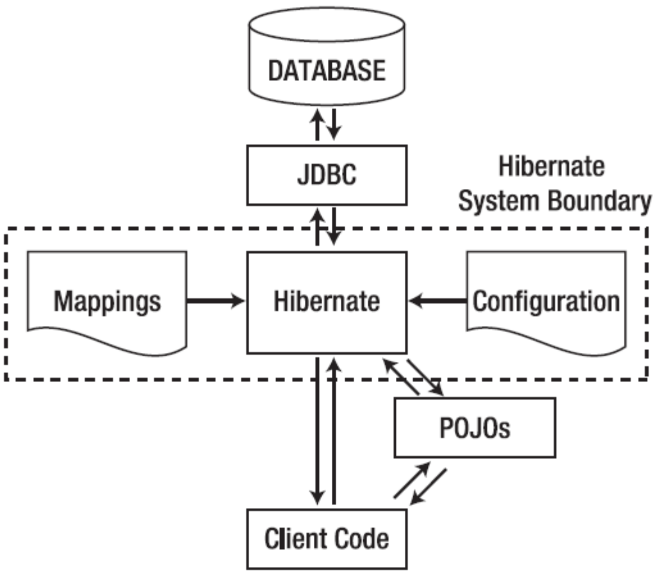


Figure 12: Role of Hibernate in a Java Application [46]

Hibernate uses Java Database Connectivity (JDBC) to communicate with the RDBMS. The client communicates directly with the Hibernate system by using queries, or indirectly via the POJO.

3.2.3 Servlets

A servlet is a web component that generates dynamic contest and is hosted in a servlet container [42]. Clients interact with servlets using a request/response pattern [42]. A POJO must be annotated with the `@WebServlet` annotation and extended with the `javax.servlet.http.HttpServlet` class, to be defined as a servlet. The servlet interface provides the `doX()` methods to handle the requests - e.g. `doGet()` for handling the (HTTP)GET-request and `doPost()` for the (HTTP)POST-request [42].

3.2.4 SOAP

A SOAP (Simple Object Access Protocol) web service constitutes a kind of business logic exposed via a service interface to a client application [43]. The interface uses XML for the communication and is therefore loosely coupled.

The web service interface is described with the use of the WSDL (Web Services Description Language). It contains the message type, port, communication protocol, supported operations, location and what the client should expect in return [43]. Messages are exchanged mostly with HTTP, but SMTP or JMS can also be used. Web services can be located using the UDDI (Universal Description Discovery, and Integration) mechanism. The UDDI registry points to public WSDL files that can be used to invoke the web services [43].

SOAP messages consist of envelopes containing an optional header and a required body [47]. The following example shows a simple SOAP response with an empty header. The payload data contains the name of a person.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:name xmlns:m="http://www.example.org/name">
      <m:firstname>John</m:firstname>
      <m:lastname>Doe</m:lastname>
    </m:name>
  </soap:Body>
</soap:Envelope>
```

The Java EE standard for the realization of SOAP web services is JAX-WS 2.2. It defines a set of APIs and annotations for building and consuming web services with Java [43]. Other standards that JAX-WS 2.2 relies on are Web Services 1.2 (JSR 109), JAXB 2.2, WS-Metadata 2.0 and JAXR 1.0 [43]. The reference implementation is Metro. The Metro stack is produced by the GlassFish community [43].

3.2.5 RESTful web service

The REST (Representational State Transfer) architecture centers on resources that can be addressed using Uniform Resource Identifiers (URIs), typically links on the web

[43]. URIs should be descriptive and target a unique resource [43]. For example a list of persons should be accessible via `http://www.example.com/info/people`.

Unlike SOAP, which relies on W3C standard, REST has no standard and is just a style of architecture with design criteria [43].

Similar to the WSDL used with SOAP web services, WADL (Web Application Description Language) can be used to describe a RESTful web service [43]. Through the use of HTTP methods all CRUD operations are available [43]:

- HTTP POST to create a resource(in XML, JSON, or text format)
- HTTP GET to read a resource
- HTTP PUT to update a resource
- HTTP DELETE to delete a resource

The Java EE standard for the implementation of RESTful web services is JAX-RS 2.0. The reference implementation is Jersey, which is open source (dual licensed under CDDL and GPL) [43].

3.2.6 JavaServer Faces

JavaServer Faces (JSF) is a server-side user interface framework for web applications [42]. It was created in response to some limitations of Java Server Pages (JSP) and the underlying servlet component [43]. The JavaServer Faces API allows developers to think in terms of components and events instead of requests and responses [43]. JSF 2.0 introduced AJAX which aids in the process of creating rich internet applications [43].

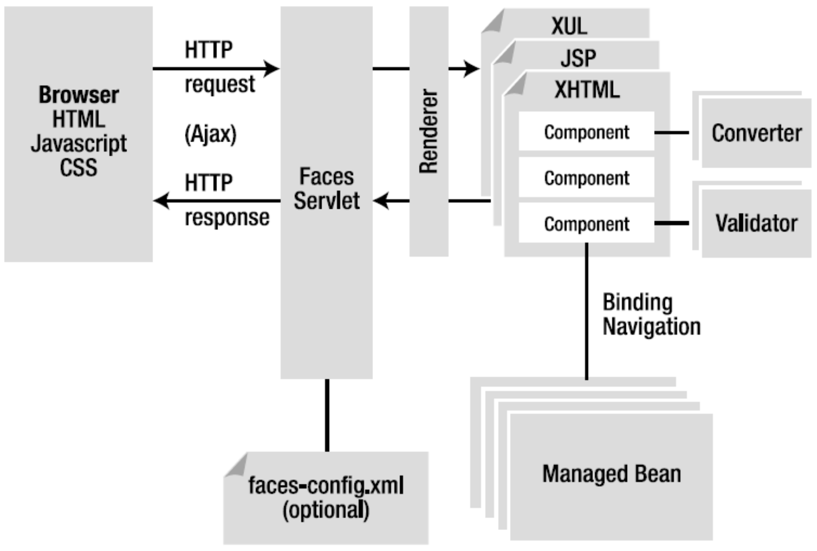


Figure 13: JSF architecture [43]

The FacesServlet is the main servlet that can optionally be configured by a `faces-config.xml` file [43]. Renderers are responsible for displaying components and translating user inputs into component property values [43]. Converters are useful for the conversion of data types. Validators are used for the validation of user input. ManagedBeans are used for the business logic and page navigation. JSF supports multiple page description languages (PDLs) including JSP and Facelets (which is the preferred PDL of JSF 2.0 and above).

Managed Beans

Managed Beans are POJOs that are treated as managed components by a Java EE container [42]. It is responsible for the synchronization of values with corresponding components, processing business logic (e.g. calling EJBs), and the navigation between pages [43]. The Expression Language (EL) is used for the association of components with specific managed Beans properties or actions.

```
<h:inputText value="{#PersonController.Person.name}" />
```

In this example, the input text's value is directly hooked up with the property `Person.name` of a managed bean called `PersonController` [43]

PrimeFaces

The implementation of the prototype uses PrimeFaces for additional JSF components. PrimeFaces is an open source JSF component suite with built-in AJAX [48].

3.2.7 Adobe Flex

Another user-interface technology that can be used for web applications is Adobe Flex. It is an open source application framework developed by Adobe Systems. Flex consists of three languages: MXML, ActionScript and FXG [49]. MXML is a markup language and typically used for designing the user interface. ActionScript is a scripting language that controls the behaviour of the application. FXG (Flash XML Graphics) is a XML-based standard for sharing graphics [49]. Flex applications require Adobe Flash Player or Adobe AIR to run [49].

The connection between Flex and Java EE can be done with web services (SOAP or RESTful) or with BlazeDS - an open source component from Adobe Systems. It is a collection of Java components that allows the use of Java Messaging Service for communicating. BlazeDS also supports object remoting [50].

3.2.8 Security

Security in IT infrastructures can be divided into three major layers. These are the network security layer, the operating system layer and the application layer [51].

The authentication and authorization of users in web applications are part of the application security layer. A user must present valid credentials to be successful authenticated. He also needs permissions to access certain secured resources. Permissions can be managed with access control lists (ACLs). An ACL is a collection of mappings between resources, users, and permissions [51].

Java EE provides the Java Authentication and Authorization Service (JAAS) for application security. This API provides a pluggable system where authentication mechanisms can be plugged independently to applications [51]. Other solutions include Spring Security and Apache Shiro.

Spring Security

Spring Security can be used as an authentication and authorization appliance. Spring Security is a framework that adheres to the Spring Framework and provides security services to Java applications [51]. It provides layered security services and implements many authentication models such as LDAP, form authentication, certificate X.509 authentication, database authentication, Jasypt cryptography [51].

Apache Shiro

Shiro is an open-source security framework, developed by the Apache Software Foundation that handles authentication, authorization, enterprise session management and cryptography [52].It's usage is very similar to Spring Security and Java EE Security.

3.3 Software Design Patterns

The foundation of many Java EE concepts respectively mechanisms are software design patterns. Patterns describe repeated problems and their reusable solutions [53].

3.3.1 Data Access Objects

A data access object (short DAO) encapsulates and abstracts all access to a known data source [54]. It manages the connection with the data source to retrieve and write data [54] A data source could be a relational database, an external service like a B2B exchange, a LDAP database, or a business service accessed via a remote procedure call [54]. The business component uses the simpler interfaces provided by the DAO to receive the data in form of a POJO.

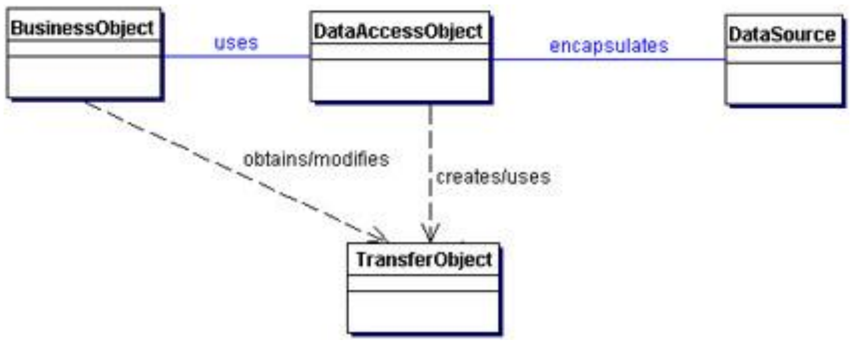


Figure 14: DAO pattern [54]

3.3.2 Dependency Injection

Dependency Injection is a form of "Inversion of control" where the control of the application is handled by the underlying framework rather than the application itself [55].

There are three main styles of dependency injection:

- Interface Injection (Type 1)
- Setter Injection (Type 2)

- Constructor Injection (Type 3)

Interface injection means that the reference to an external module is provided by an interface the user must implement.

Setter injection is defined by a setter method that the framework uses to inject the dependency.

Constructor injection means that the dependencies are provided through the class constructor.

3.3.3 Model-View-Controller pattern

The model-view-controller (short MCV) pattern is a very important design concept for graphical user interfaces that originated with Smalltalk-80 [56].

MVC requires the use of three types of objects:

Models contain the data and the methods to access respectively manipulate the data (especially getter and setter methods). A model often serves as a software approximation of a real-world process [57].

Views render the presentation of the data to the user (e. g. graphical user interface). The view must update its presentation when the model data changes. This can be achieved by two different approaches: The push model dictates that the view must register itself with the model for change notifications. The pull model on the other hand provides that the view is responsible for calling the model when it needs to retrieve the most current data [57].

Controllers process the data and update the model and the view [56]. A controller may also select a new view, if the context demands it [57].

JavaServer Faces uses the MCV pattern for its core features.

4 Design of the web application

The construction of the architecture for the animal health record for livestock management was mainly a collaborative process within the AHR research group. First the stakeholders were identified. After that, the requirements were evaluated and the database design was laid down. After that, the technology for the implementation had to be chosen. Finally the object oriented architecture of the application was defined and basic UI mock-ups were created.

The architecture was documented with the help of the Unified Modelling Language (UML) which is the standard modelling language for software and systems development [58].

4.1 Project Stakeholder

According to [59] the term project stakeholder describes an organization or individual that is actively involved in a project, or whose interest may be positively or negatively affected as a result of project execution or successful project completion. These groups can be further divided into internal or external respectively primary or secondary stakeholders.

Internal stakeholders are usually directly affiliated with the project (e. g. project managers, analysts, programmers, etc.). Customers and vendors can also be seen as internal stakeholders because they may provide direct input to the execution of the project.

External stakeholders can be depicted as every other individual or organization that is affected by the outcome of the project.

Further fragmentation of the stakeholders results in the breakdown of primary and secondary stakeholders. These terms identify the importance of a stakeholder.

The AHR research group identified the following external respectively primary stakeholders for an AHR in livestock management. Those are also the acknowledged users of the prototypical implementation:

Veterinarians

Veterinarians are responsible for farms, animals and livestock in terms of medical support. They make diagnoses, prescribe drugs and perform therapies [5].

Farmers

Farmers manage their livestock and produce food products such as milk, eggs or meat [5].

Processing industries

Processing industries handle and process animal products before they can reach the consumer, e. g. slaughterhouses, dairies, fisheries [5].

Consumers

Consumers can be customers or trading companies (e.g. supermarkets) [5].

"Agrarmarkt Austria – AMA"

The "Agrarmarkt Austria Marketing GmbH" short AMA is responsible for carrying out all agricultural marketing across Austria. This includes quality improvement and classic advertising and promotion [60]. It is also responsible for the registration of animals [5].

Organizations and associations

Austria accommodates various organizations and associations for livestock management and animal health, e.g. "Landeskontrollverbände – LKV" [5].

Governmental departments

The Austrian government is the leading power behind the efforts for the implementation animal health records. The department of agriculture, environment and water economy ("Bundesministerium für Land- und Forstwirtschaft, Umwelt und Wasserwirtschaft") is mainly concerned with food quality. The department of health ("Bundesministerium für Gesundheit") deals with animal health, drug administration and epidemic control [5].

The Austrian Veterinary Health Service ("Tiergesundheitsdienst") is divided into sub entities to represent eight of the nine Austrian provinces with the exception of Vienna. It also houses the Poultry Health Service (QGV) that supervises all sectors of the poultry industry on a national basis [61]. Its objectives are the improvement of animal health, the prevention of infectious diseases, quality assurance, continuing development, advice of participants and productivity increase of agricultural facilities [61]. Its legal basis is the Austria Veterinary Medicines Control Act that came into force in April 2002 [61].

4.2 Requirements analysis

The requirements analysis was a collaborative process that was conducted by the AHR research group. Some of the identified stakeholders were personally interviewed to get a more precise perspective of their needs and goals. The gained information was then discussed within the group and formally written down. The existing AHR software products depicted in chapter 2.2 were also taken into account.

4.2.1 Functional requirements

A functional requirement specifies a function that a system or a system component must be able to perform [62]. It can be represented by written descriptions or use-cases which in turn can be textual enumeration lists as well as diagrams, describing user actions [62].

The following functional requirements are a subset of the ones that were evaluated and laid out within the AHR research group [5], that the prototype implements:

Login/Logout

Users must be authenticated to be able to use the system. They must also be able to quit their session.

Logging

Every action the system or a user performs must be logged into a file.

Add/edit farm-data

New farms can be added to the system. Existing farm-data can be modified.

Add/edit animal-data

New animals can be added to the system. Existing animal-data can be modified.

Search

Users can search for farms, companies, animals, veterinarians and drugs.

List of farms

A list of all registered farms can be browsed by the user.

Farm detail view

After choosing a farm from the list, the user can view and edit all its metadata.

Choosing an animal by species

Many farms house multiple animal species. Therefore users must be able to choose which animal species they want to see.

List of animals

The user must be able to browse a list of animals pre-sorted by species. This list can be additional sorted and filtered by certain criteria.

Animal detail view

After choosing an animal from the list, the user can view and edit all its metadata.

Add/edit animal health data

Users, that have the specified access rights, must be able to add and edit all relevant animal health data.

Upload documents

Users can upload documents and add them to a health record.

List of veterinarians

The system maintains a list of registered veterinarians. The list can be sorted and filtered by certain criteria.

Add/edit veterinarian-data

New veterinarians can be added to the system. Existing veterinarian-data can be modified.

List of processing industries

The system maintains a list of all processing industries. The list can be sorted and filtered by certain criteria.

Add/edit processing-industry-data

New processing industries can be added to the system. Existing processing-industry-data can be modified.

Allocate a veterinarian to a farm/company

Veterinarians can be allocated to one or multiple farms. This grants them the right to see all relevant data including animal health data of that farm or company.

List of drugs

The system maintains a list of administrable drugs. The list can be sorted and filtered by certain criteria.

View log-files

Administrators must be able to manage and review all log-files.

Add/edit users

Administrators must be able to add and edit users.

Interfaces to existing AHR-software

The application includes web services for the exchange of data with existing AHR software.

4.2.2 Non-functional requirements

Every requirement that does not fit the term of a functional requirement is called a non-functional requirement. They can be categorized into three types [62]:

- Data requirements
- Constraints
- Quality requirements

Data requirements describe how functional requirements should be reflected in the system [62]. Constraints explicitly restrict the system or process. They include limitations of the engineering process, systems or system components' functionality, or its life cycle [62]. Quality requirements can be described as wanted qualities of the product that are not directly related to functionality [62].

The following non-functional requirements were evaluated [5]:

Data requirements

- The web application must be usable with all common web browsers including "Internet Explorer", "Mozilla Firefox" and "Google Chrome".

- The system must be easily expandable, e.g. mobile application.

Constraints

- All communications must be secured with SSL or TLS.
- The system should be free of charge for veterinarians.

Quality requirements

- The system must be available 24x7.
- The system must be able to handle peak periods.
- The system must be scalable.
- The system must be customizable.
- The system must be easily to maintain. This can be achieved through code documentation, automated tests or through the use of software design patterns.
- The acquisition and retrieval of data must not involve increased operating expense.
- The handling of the software must be easy and time-saving.
- Search queries must respond within 5 seconds. The response time for retrieving documents should be less than 5 seconds. For retrieving images it should be less than 1 minute.
- The server components should be deployed redundantly. This ensures reliability and load distribution.
- The user should be notified about changes or errors in the system in a consistent manner.

4.3 Roles

The following roles are a subset of the ones that were evaluated and laid out within the AHR research group [5], that the prototype implements:

Administrators

Administrators of the system are able to query, add, modify and delete all data. They can also grant rights to users and add additional roles.

Farmers

Farmers can modify their own company data. They can add treatments and diagnoses made by veterinarians. They can associate veterinarian to modify their animal data.

Veterinarians

Veterinarians can add, modify, and delete treatment data of all farms they are responsible for.

4.4 Database Design

The database design is loosely based on the HL7v3 reference information model. The first draft was more oriented towards the constraints defined in the requirement analysis. However, the decision was made to base the database model on the HL7 reference information model because it is a well-known international standard and may also be a good foundation for the data export respectively import of HL7 data.

A disadvantage of the HL7 approach is that constraints which are normally handled by entity relations must be defined programmatically as part of the business logic. For example, by defining a relation “Animal gets a treatment” it is clear that only an entity of type animal can be associated with an entity treatment. The generic HL7 approach makes it theoretical possible that an individual person can adopt the role “patient” and subsequently receive a treatment.

Thus the model was simplified to better estimate the needs of the application. The generic “Entity” and “Role” objects defined in HL7v3 RIM were omitted. The Participation relation was split into two separate relations connecting persons and animals to acts.

Despite the fact that most users are actors within the animal health record, their user account data is stored in separate tables due to security concerns.

The database design was conveyed into an Entity-Relationship (ER) diagram using the Chen diagrammatic technique including the “min, max” notation [63]. In this notation an Entity is represented by a rectangular box and each relationship set is represented by a diamond-shaped box [63]. The boxes must be connected to each other with lines. These lines also describe the min, max occurrences of those relations. Attributes are depicted as ovals but were omitted from this design to make the diagram more readable.

4.4.1 Entity-Relationship diagram

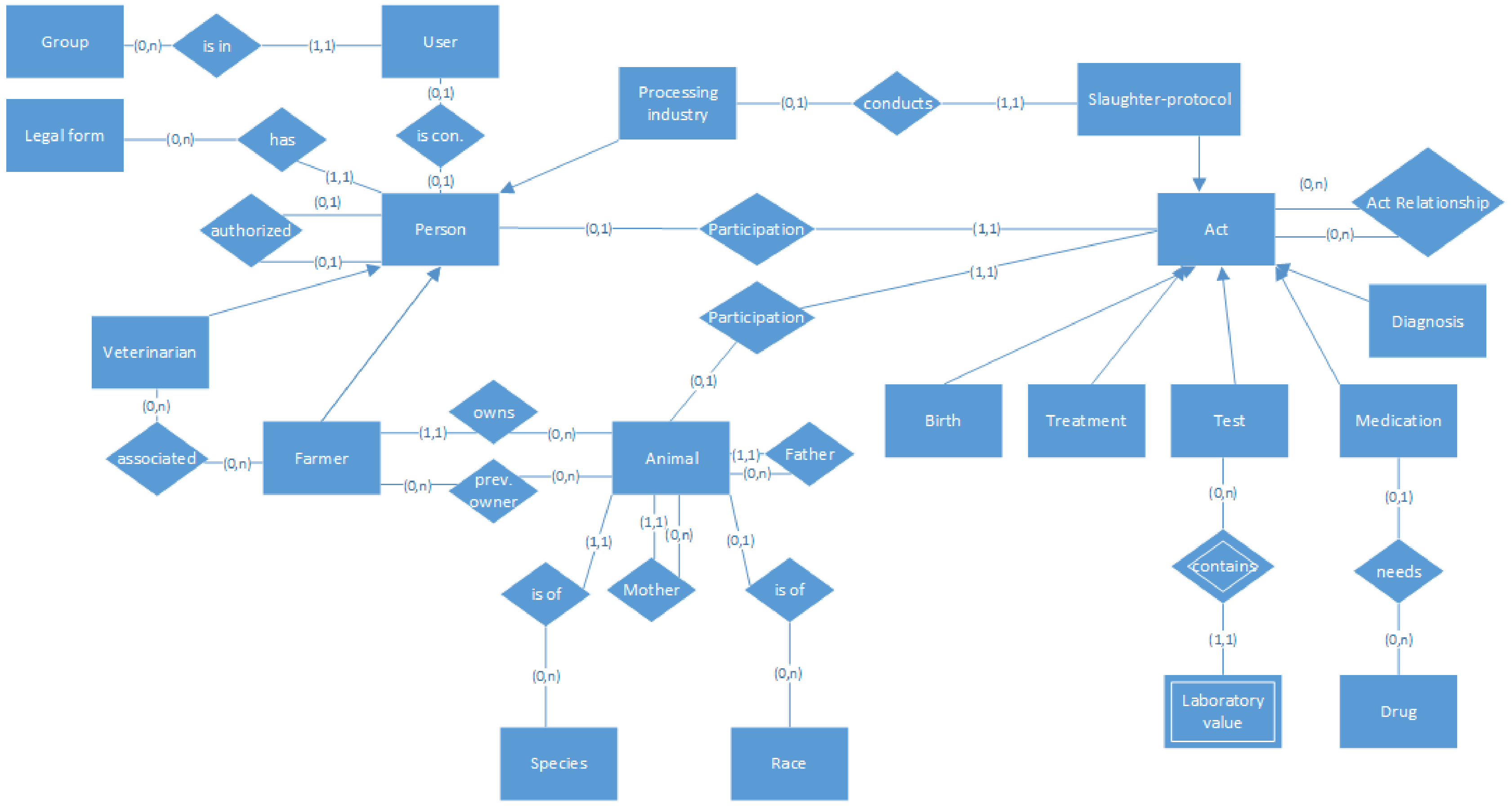


Figure 15: Entity Relationship diagram, C. Aigner

Person is the base entity for veterinarians, farmers and processing industries. Persons may have a legal form. They can also have a substitute authority. Farmers own animals. Previous owners are being stored as well. An animal is derived from a certain species. It can have a father respectively a mother. Races can be stored as well.

Every person respectively animal can participate with an Act. Acts can be related to each other. An Act can represent a medication, a treatment, a diagnosis, a test, the birth or a slaughter protocol. A medication includes the drugs that were administered. Tests include laboratory values. Processing industries are related to slaughter protocols.

Users are associated to a certain role and can be related to a Person entity.

4.4.2 Relations

The following table contains the relations derived from the Entity-Relationship diagram together with all the attributes.

Name	Description	Data type	Constraint
		Act	
CaseNo	Case number	Character(10)	primary key
PNo	Person number	Integer	foreign key
ANo	Animal number	Character(12)	foreign key
ActDate	Date	Date	
Report	Report	Text	
RecoverDate	Date of recovery	Date	
		Act_relationship	
CaseNo1	Case number #1	Character(10)	primary key, foreign key
CaseNo2	Case number #2	Character(10)	primary key, foreign key
		Treatment	
CaseNo	Case number	Character(10)	primary key, foreign key
Label	Label	Character(50)	
Description	Description	Text	
NextTreatment	Date of next treatment	Datetime	
		Diagnosis	
CaseNo	Case number	Character(10)	primary key, foreign key
DiagName	Name	Character(50)	
Description	Description	Text	
Notifiable	Notifiable sickness	Boolean	
		Birth	
CaseNo	Case number	Character(10)	primary key, foreign key

BirthDate	Date of birth	Date	
Stillbirth	Stillbirth	Boolean	
InseminationDate	Date of insemination	Date	
Natfert	Natural fertilization	Boolean	
PregnancyDuration	Duration of pregnancy in weeks	Integer	
Lactation	Lactation	Boolean	
LabValue			
Id	Identification of the value	Integer	primary key
ValueName	Name	Character(30)	
CaseNo	Case number	Character(10)	primary key, foreign key
TestResult	Test result	Double	
LowerBound	Lower bound	Double	
UpperBound	Upper bound	Double	
Farmer			
IDno	Identification no.	Character(10)	primary key, foreign key
FarmID	Identification no. of the farm	Character(30)	
FarmerName	Name of the Farm	Character(255)	
FarmerVetAssoc			
IDFarmer	ID of the farmer	Integer	primary key, foreign key
IDVet	ID of the veterinarian	Integer	primary key, foreign key
Drug			
Code	Code	Character(10)	primary key
Name	Name of the drug	Character(50)	
Patentee	Owner of the patent	Character(30)	
Approved	Approved for animals	Boolean	
Medication			
CaseNo	Case number	Character(10)	primary key, foreign key
Code	Code of the drug	Character(10)	
Dosage	Dosage	Double	
Repeat	Repeat every n day	Integer	
UntilDate	Repeat until	Date	
DosagePerDay	Dosage per day	Integer	
Patient			
IDNo	Identification no.	Character(10)	primary key, foreign key
Person			
IDNo	Identification no.	Character(10)	primary key
Authority	ID of stand-in authori-	Character(10)	foreign key

	ty		
FirstName	First name	Character(30)	
Surname	Surname	Character(30)	
Title	Title	Character(15)	
BirthDate	Date of birth	Date	
PostalCode	Postal code	Character(5)	
City	City	Character(30)	
Street	Street	Character(50)	
HouseNo	House number	Character(10)	
TelephoneNo	Telephone number	Character(20)	
Email	E-Mail address	Character(30)	
Fax	Fax number	Character(30)	
CompanyName	Name of the company	Character(200)	
LegalForm	Reference to legal form	Integer	foreign key
Race			
Id	Identification of the race	Integer	primary key
Description	Description	Character(100)	
LegalForm			
Id	Identification of the form	Integer	primary key
Description	Description	Character(50)	
SlaughterProtocol			
CaseNo	Case number	Character(10)	primary key, foreign key
SlaughterNo	Slaughter number	Integer	
SlaughterDate	Date of slaughter	Date	
ClassificationDate	Date of classification	Date	
SlaughterWeight	Slaughtering weight (“Warm-schlachtgewicht”)	Double	
MeatFatClass	Meat and fat tissue class (“Fleischigkeits- und Fettgewebeklasse)	Character(3)	
Category	Category	Character(3)	
ClassifierBadge	Badge of classifier	Character(10)	
MeatInspectionDate	Date of meat and slaughter inspection	Date	
ResultBefore	Examination result before slaughter	Character(200)	

ResultAfter	Examination result after slaughter	Character(200)	
Remarks	Remarks	Text	
IDno	ID of the processing industry	Integer	foreign key
TestAHR			
CaseNo	Case number	Character(10)	primary key, foreign key
TestType	Type of test	Character(30)	
SicknessDate	Date of sickness	Date	
Document	Document	Binary	
Result	Result	Text	
Remarks	Remarks	Text	
Animal			
IDNo	Identification no.	Character(12)	primary key, foreign key
Species	Reference to species	Integer	foreign key
DeathDate	Date of death	date	
Name	Name	Character(20)	
IDMother	ID of the mother	Character(10)	foreign key
IDFather	ID of the father	Character(10)	foreign key
Gender	Gender	Character(1)	
BreedRegNo	Breed registry number	Character(20)	
HairColour	Colour of hair cover-ing	Character(30)	
Castrate	Castrated animal	Boolean	
Race	Reference to race	Integer	foreign key
Owner	Owner	Character(10)	foreign key
PrevOwner	Previous Owner	Character(10)	foreign key
ForeignEarmarkNo	Number of foreign earmark	Character(30)	
Weight	Weight	Double	
Remarks	Remarks	Text	
Species			
Id	Identification of the race	Integer	primary key
Description	Description	Character(100)	
Veterinarian			
IDNo	Identification no.	Character(10)	primary key, foreign key
Expertise	Area of expertise	Character(50)	
OfficialVet	Official Veterinary	Boolean	
ProcessingIndustry			

IDno	Identification no.	Character(10)	primary key, foreign key
Sector	Industrial sector	Character(30)	
UserAHR			
Id	Identification no. of the user	Integer	primary key
Username	Username	Character(20)	
FirstName	First Name of the user	Character(50)	
Surname	Last Name of the user	Character(50)	
EMail	E-Mail address of the user	Character(50)	
Password	User Password (MD5)	Character(100)	
Person	Reference to person	Integer	foreign key
GroupAHR	Reference to group	Integer	foreign key
GroupAHR			
Id	Identification no. of the group	Integer	primary key
GroupName	Name of the group	Character(20)	
Description	Short description	Character(200)	

Table 3: Relations including data types and constraints, C. Aigner

4.5 Technology selection

The evaluation of the underlying technology was a very important step in the design process. The choice of using Java EE technology as a foundation was determined in the beginning of this project, but certain technology choices were made at a later point in the design process.

4.5.1 JSF 2.0

Two technologies were evaluated for the development of a rich web client - Adobe Flex (Adobe Flash) and JavaServer Faces 2.0. Both have the ability to provide the user with a rich user interface experience. Adobe Flex provides more interactivity but requires a browser plugin to run. This plugin may not be available on all operating systems respectively browser platforms, especially on mobile devices like smartphones and tablets.

JSF 2.0, enhanced with additional component suites like PrimeFaces component suite provides a decent level of interactivity while still remaining platform independent, because the API relies on JavaScript and AJAX. Web applications using JavaServer Faces can be executed on any modern personal or tablet computer without any extra software. Even though Adobe provides its own open-source API for communication with Java EE technology (BlazeDS), JSF 2.0 is a core technology of the Java EE framework.

The decision was made to develop the web application with JSF 2.0 and the PrimeFaces component suite because it is better integrated into Java EE and is cross-platform.

4.5.2 RESTful web service

SOAP and RESTful web services are very different in their technical approach. SOAP originated from the XML-RPC protocol that was designed to exchange remote procedure calls with XML encoded messages [64]. SOAP tends to be more computationally intensive because of the time-consuming parsing process of the messages. XML messages also produce larger amounts of overhead. One benefit of SOAP is, that it includes directory services which are especially useful for enterprise scaled distributed systems [64].

REST is built on HTTP and uses the basic HTTP request methods for communication. It has not been standardized, in contrary to SOAP which is a W3C standard, but can be seen as an architectural style [64].

Since RESTful web services are considered a defacto standard and are more light-weight and easier to develop because no special interfaces and description files have to be created, it was chosen as the underlying technology for the external interface of the AHR web application.

4.5.3 Spring Security

Spring security is a security API that is part of the spring framework. It has more features than the built-in Java EE security API and is more wider used [65]. Its configuration is XML-based, unlike Apache Shiro which stores its configuration into a text file, thus providing a more tight integration into the Java EE GlassFish environment.

Apache Shiro seems to be more targeted towards non-web use and cryptography. The main limitation is the lack of documentation [66].

Since Spring Security is wider used and its configuration integrates better with Java EE, it was chosen as the security API for the application.

4.6 Software architecture

The design of the software architecture was made to get a big picture of the system before the actual implementation began. It includes a basic overview of the framework components used, the class diagram, an overview of the packages and a deployment diagram.

4.6.1 Framework overview

The application will be developed with Java EE. The presentation layer was implemented with JSF 2.0 and PrimeFaces. The underlying management of the user interface components was implemented with JSF ManagedBeans. The presentation layer interacts with the application layer to execute the user requests and the corresponding business logic.

The application layer includes Enterprise Java Beans and DAOs for the business logic and Spring Security for security and authentication. The DAOS communicate with the underlying persistence tier to execute the CRUD operations.

The persistence layer consists of Hibernate as the object relational mapper that communicates with the database.

The interoperability layer consists of a RESTful web service that communicates with the application layer.

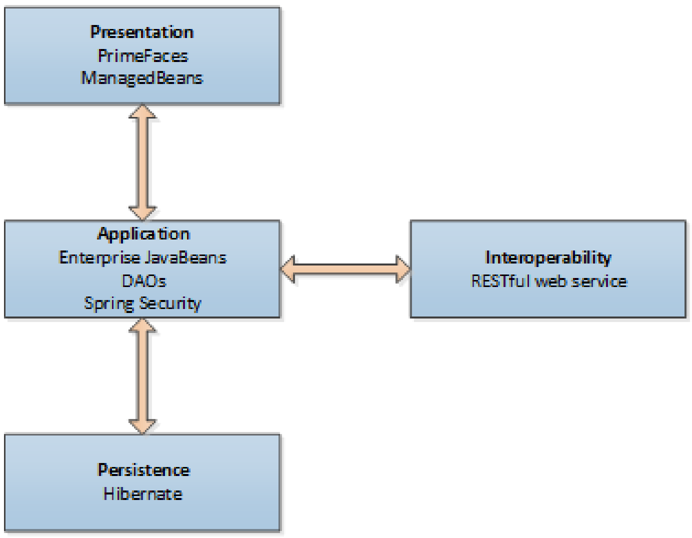


Figure 16: Overview of the Java EE framework components, C. Aigner

4.6.2 Class diagram

The attributes and their corresponding getter/setter methods were omitted from the diagram to enhance clarity.

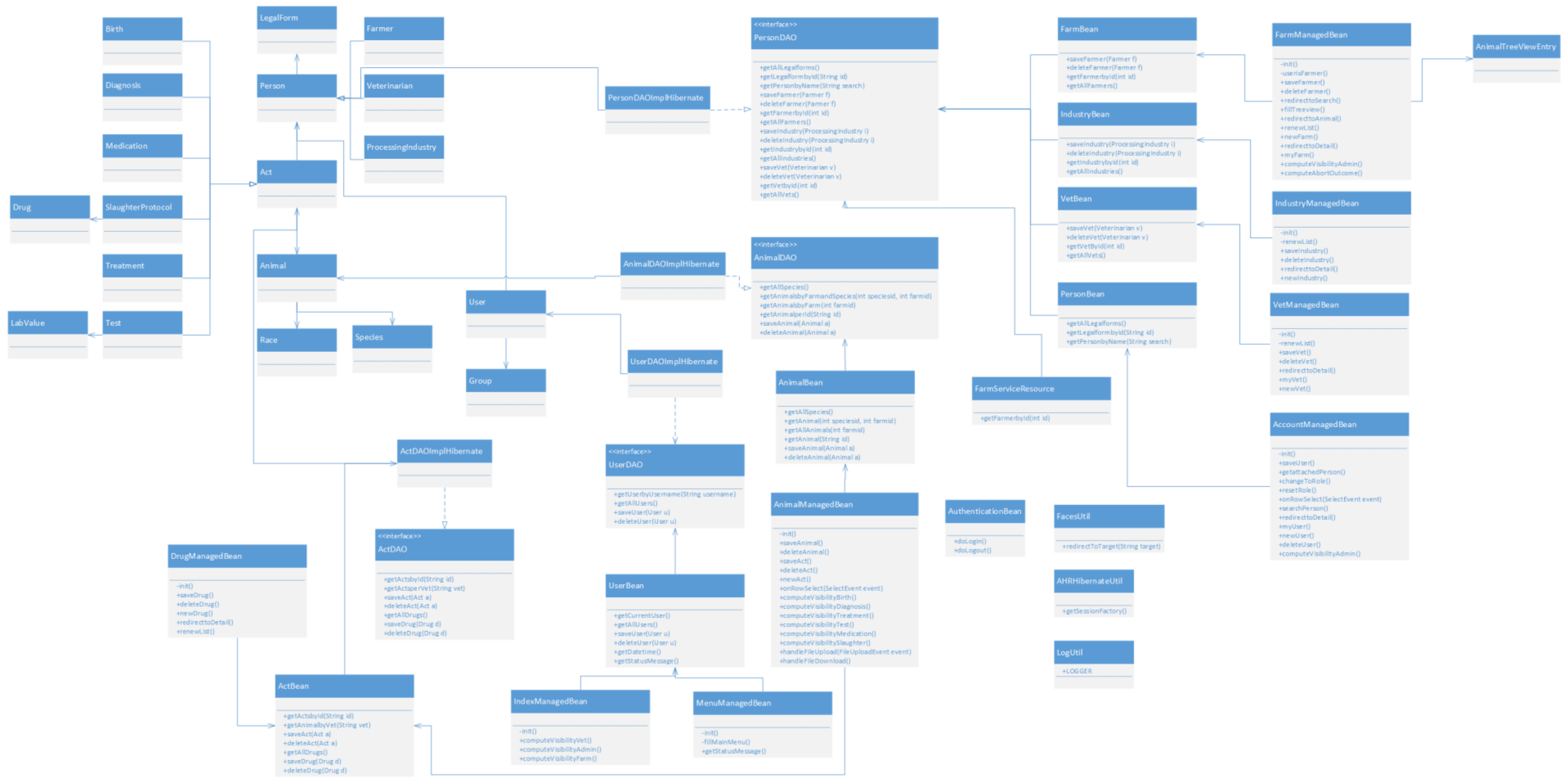


Figure 17: class diagram, C. Aigner

4.6.3 Package diagram

The package diagram shows the java packet structure and their relations.

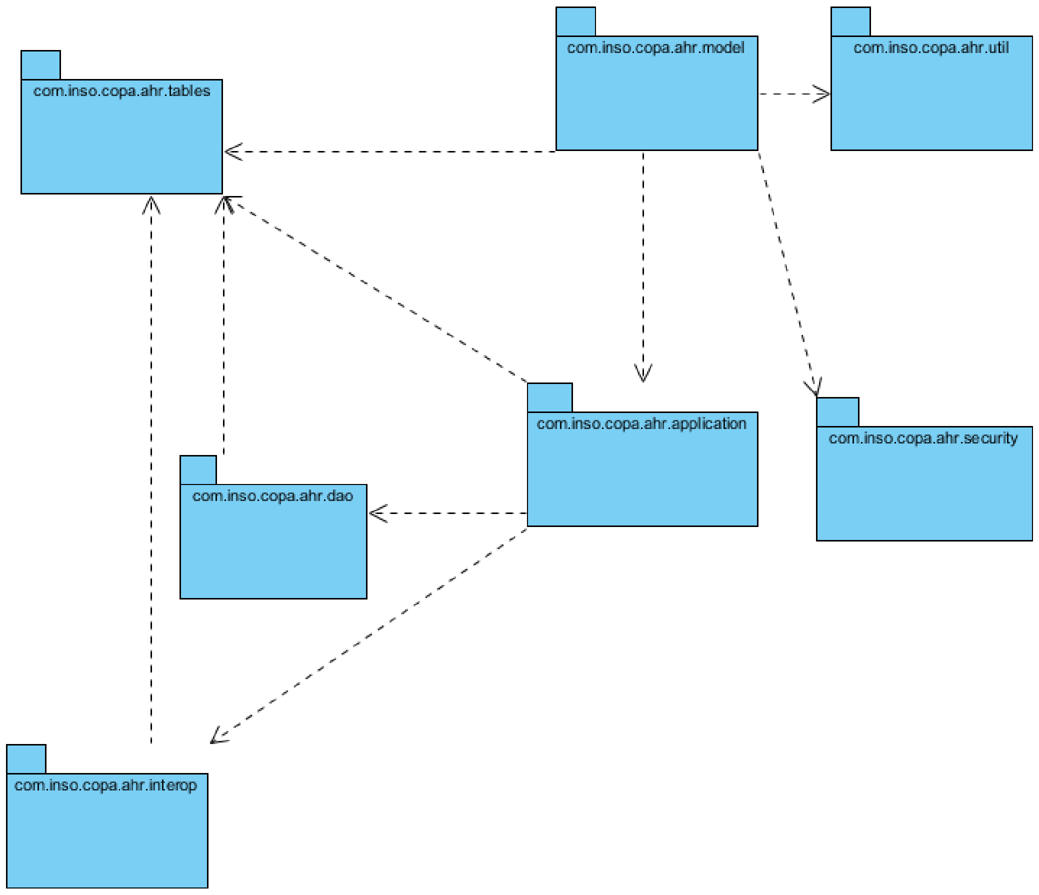


Figure 18: Package diagram, C. Aigner

4.6.4 Deployment diagram

The deployment diagram shows the connection between the systems and the user.

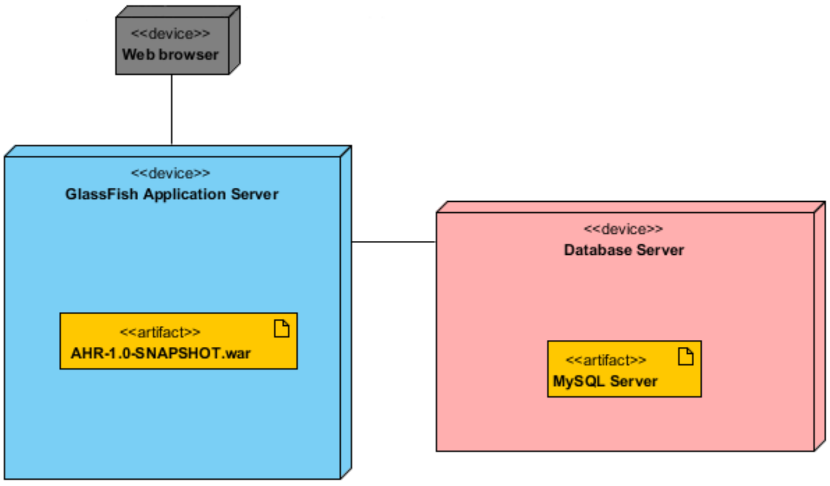


Figure 19: Deployment diagram, C. Aigner

4.7 GUI-mock-ups

The final step in the design process was to draft the graphical user interface. This was done by sketching mock-ups that were then discussed and refined within the AHR research group.

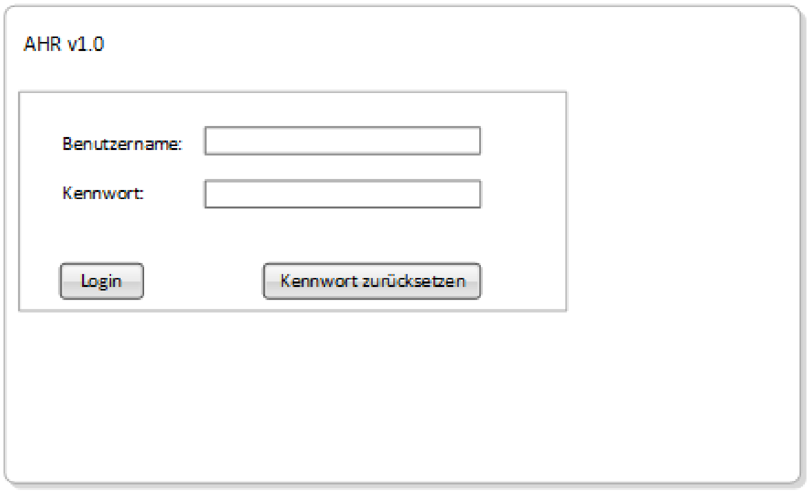
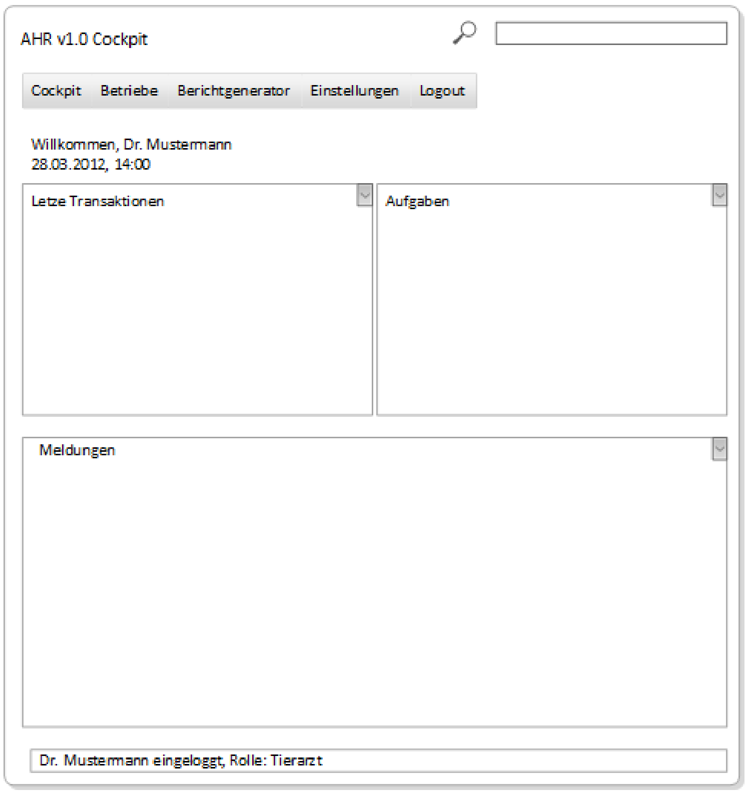


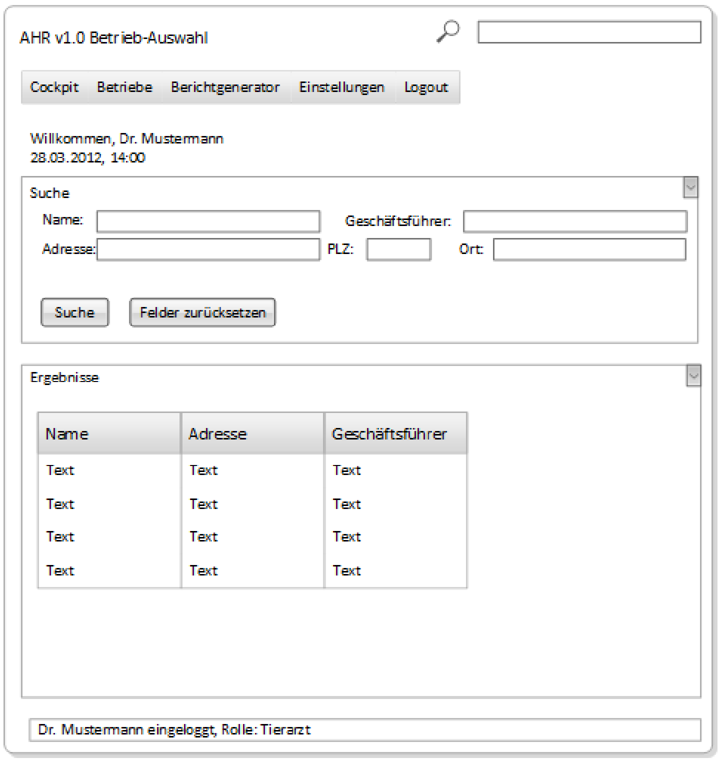
Figure 20: Login-page, C. Aigner



Willkommen, Dr. Mustermann
28.03.2012, 14:00

Dr. Mustermann eingeloggt, Rolle: Tierarzt

Figure 21: Entry-page (Cockpit), C. Aigner



Willkommen, Dr. Mustermann
28.03.2012, 14:00

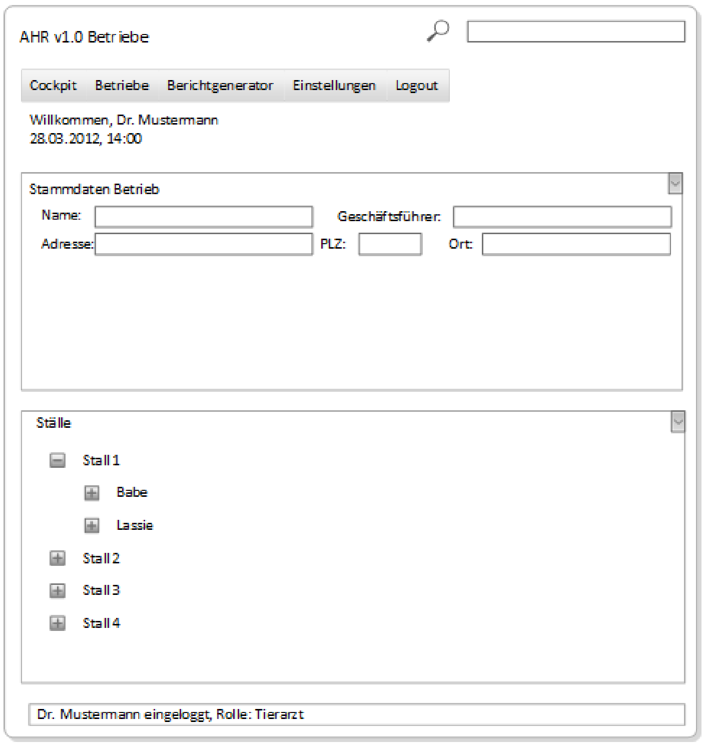
Suche

Name: Geschäftsführer:
Adresse: PLZ: Ort:

Name	Adresse	Geschäftsführer
Text	Text	Text
Text	Text	Text
Text	Text	Text
Text	Text	Text

Dr. Mustermann eingeloggt, Rolle: Tierarzt

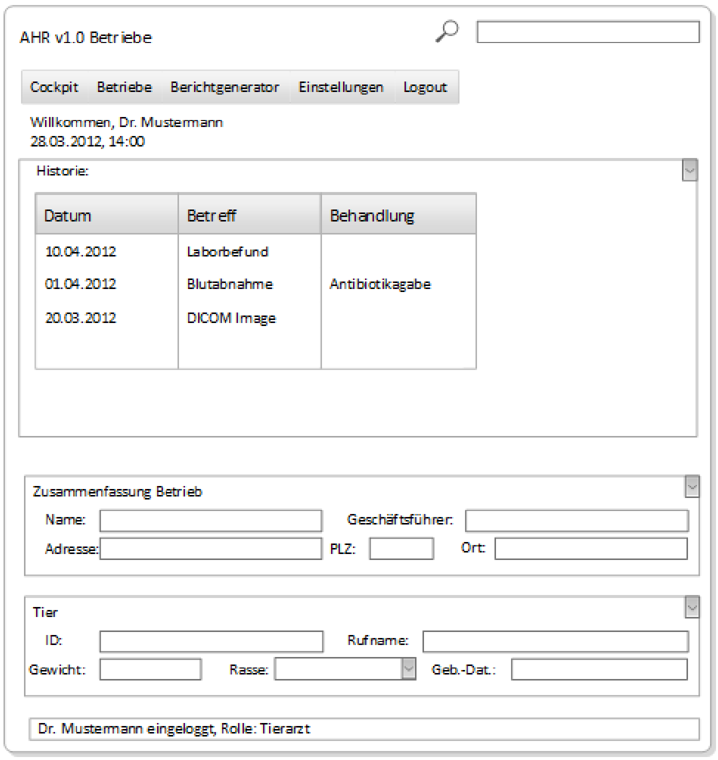
Figure 22: Search mask for farms/industries, C. Aigner



Willkommen, Dr. Mustermann
28.03.2012, 14:00

Dr. Mustermann eingeloggt, Rolle: Tierarzt

Figure 23: Mask for farm/industry standing data, C. Aigner



Willkommen, Dr. Mustermann
28.03.2012, 14:00

Datum	Betreff	Behandlung
10.04.2012	Laborbefund	
01.04.2012	Blutabnahme	Antibiotikagabe
20.03.2012	DICOM Image	

Dr. Mustermann eingeloggt, Rolle: Tierarzt

Figure 24: Animal standing and treatment data, C. Aigner

5 Implementation of the web application

This chapter gives a detailed description of the development process and its results. First the development environment is introduced and the implementation of the GUI is described. After that the implementation of the Java Persistence API as a bridge between MySQL and the web application and the security concept with Spring Security is explained. Finally the implementation of the business logic and the interaction between the various Java EE modules is described.

5.1 Development environment

The prototype was developed on an IBM-compatible Personal Computer running Windows 7 x64. The actual programming was done with the integrated development environment NetBeans version 7.3.

The prototype was deployed and tested on the GlassFish application server version 3.1.2.2. The configuration of GlassFish was stored in the `WEB-INF/web.xml` file.

MySQL version 5.6 was used as the underlying database server. MySQL Workbench 6.0 was used for the management and configuration of the MySQL database.

Apache Maven was used in conjunction with NetBeans 7.3 for build automation and library management. All dependencies and settings were stored in the `pom.xml` file.

5.2 Graphical user interface

The GUI was developed with the JavaServer Faces 2.0 server-side user interface framework. For additional controls and widgets the PrimeFaces component suite version 4.0 was used.

To use these components the following dependencies have to be added to the `pom.xml`:

```
<dependency>
        <groupId>javax.faces</groupId>
         <artifactId>javax.faces-api</artifactId>
         <version>2.1</version>
         <scope>provided</scope>
</dependency>
<dependency>
        <groupId>org.primefaces</groupId>
         <artifactId>primefaces</artifactId>
         <version>4.0</version>
</dependency>
```

JSF implements the Model-View-Controller (MVC) design pattern (see chapter 3.3.3) [67]. In this case, a ManagedBean adopts the role of the model. The Faces servlet is responsible for the control flow and can thus be seen as the controller. Facelets, basically XHTML files are responsible for the view.

For the JSF engine to work, the following lines have to be added to the `web.xml` file:

```
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.xhtml</url-pattern>
</servlet-mapping>
```

5.2.1 Layout

The design of the web application was defined in a facelet template. It served as a model for all subpages. Other facelets embedded it through the `<ui:composition>` tag.

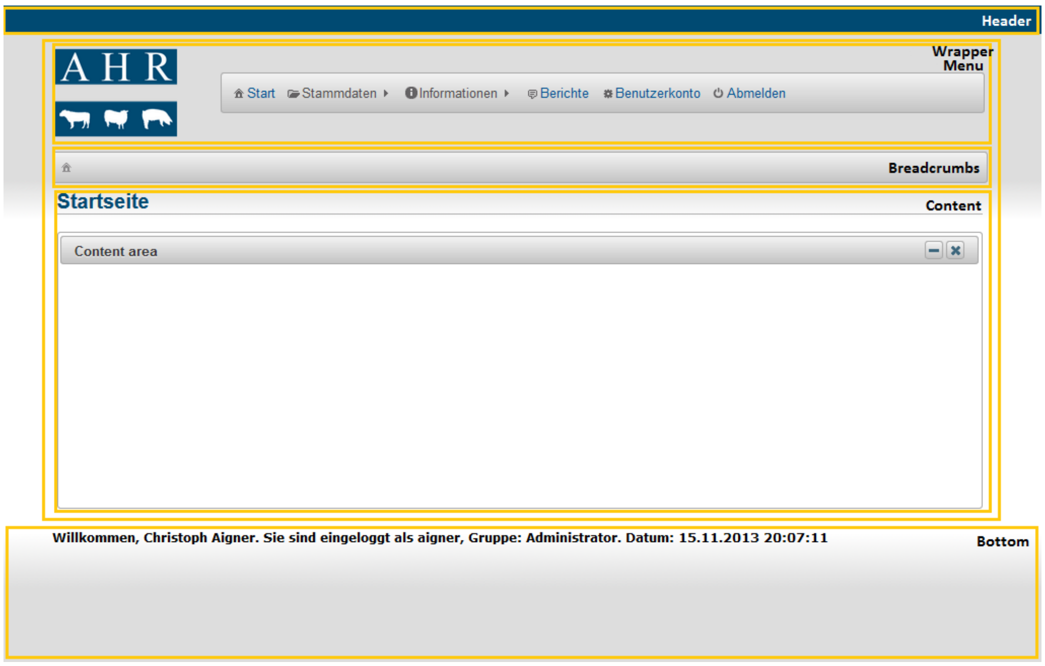
The layout was made with HTML divisions, instead of tables to improve web accessibility. The only exception was the alignment of the logo and the menu bar which was made with a HTML table for visual reasons.

The layout consists of a header, a wrapper that contains the logo, the menu, the breadcrumbs and the actual content, respectively a footer.

The visual style was implemented with cascading style sheets. The main colours used were blue `#004A72`, grey `#DDDDDD` and white `#FFFFFF`. Texts were left black.

The header is a 20 pixel high blue coloured bar that serves only a visual purpose.

The wrapper division was centred – using the CSS command `margin: 0 auto` - and set to a fixed width of 922 pixels. Two colour gradients from white to grey were used to highlight it. The content area always begins with a Heading 1 tag.



Willkommen, Christoph Aigner. Sie sind eingeloggt als aigner, Gruppe: Administrator. Datum: 15.11.2013 20:07:11

Figure 25: Layout of the application, C. Aigner

The yellow rectangles indicate the boundaries of the divisions used for the layout.

5.2.2 Logo

The logo was designed with the open-source image composition software GIMP (GNU Image Manipulation Program) [68]. It is two-coloured (white and blue). The shade of blue matches the one used for the headings and the title bar. The logo consists of two bars. The top bar is made up of the three letters AHR. The bottom bar shows the silhouettes of a cow a sheep and a pig.

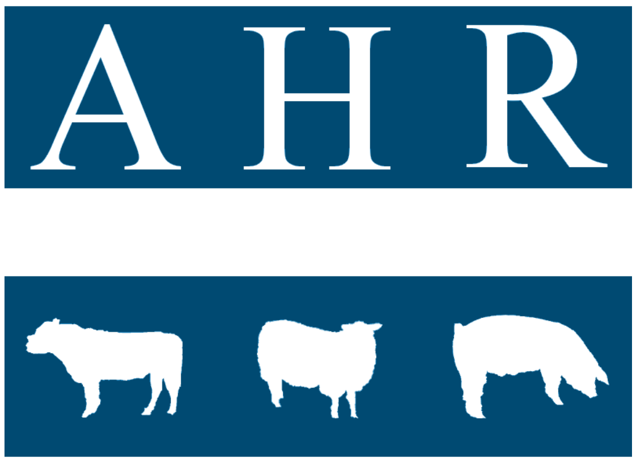


Figure 26: Logo of the web application, C. Aigner

5.2.3 Web forms and input fields

Every JSF page needs a `<h:form>` element to perform its actions, even though JSF 2.0 is heavily AJAX-driven and the default behaviour of the PrimeFaces `<p:commandButton>` widget is a partial submit. It was defined globally in the `template.xhtml` facelet.

The following PrimeFaces components were used for the web forms:

OutputLabel

The `<p:outputLabel>` component was used for the labelling of the input fields.

```
<h:outputLabel value="ID:"/>
```

InputText

The component `<p:inputText>` was used for the input of textual data. The value was bound to an object attribute of the underlying ManagedBean.

```
<p:inputText id="firstname" size="30" value="#{aMB.user.firstname}"/>
```

Figure 27: <p:inputText> component, C. Aigner

Calendar

The `<p:calendar>` component was used for the selection of dates.

```
<p:calendar locale="de" pattern="dd.mm.yyyy" size="10" id="gebdat"
         value="#{farmManagedBean.farmer.birthdate}" required="true">
```

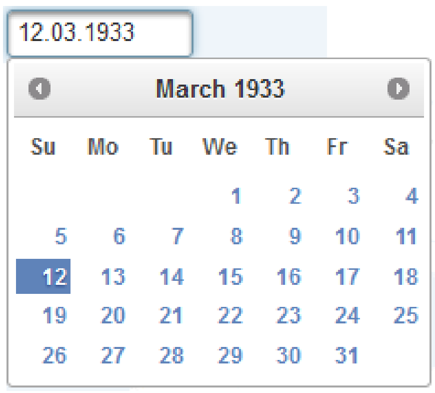

12.03.1933

Figure 28: <p:calendar> component, C. Aigner

SelectOneMenu

The component `<p:selectOneMenu>` was used for the selection of pre-loaded entries. The entries were loaded with the an `java.util.ArrayList` object and defined using the `<f:selectItems>` tag.

```
<p:selectOneMenu value="#{farmManagedBean.lfid}" >
   <f:selectItems value="#{farmManagedBean.legalforms}" var="lf"
                  itemValue="#{lf.id}" itemLabel="#{lf.rechtsform}" />
</p:selectOneMenu>
```

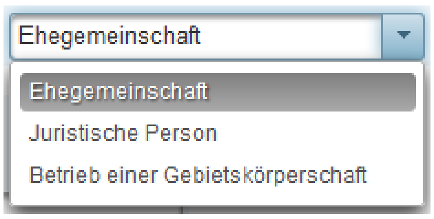


Figure 29: <p:selectOneMenu> component, C. Aigner

SelectOneRadio

The <p:selectOneRadio> component was used for the selection of three or less items, otherwise <p:selectOneMenu> was used.

```
<p:selectOneRadio id="gender" value="#{aMB.animal.gender}"
                  required="true" >
   <f:selectItem itemLabel="männlich" itemValue="m" />
   <f:selectItem itemLabel="weiblich" itemValue="f" />
</p:selectOneRadio>
```

Figure 30: <p:selectOneRadio> component, C. Aigner

SelectBooleanCheckbox

The component `<p:selectBooleanCheckbox>` was used for Boolean attributes.

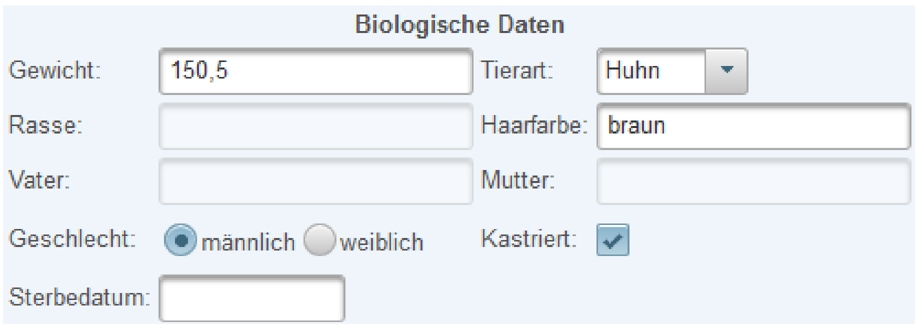
```
<p:selectBooleanCheckbox id="cast" value="#{aMB.animal.castrate}" />
```

Figure 31: <p:selectBooleanCheckbox> component, C. Aigner

Panel

The input fields were grouped into panels and further subdivided into panel grids. The following example shows the definition of a panel and a grid containing four columns. The heading for the panel-grid was inserted with the `<f:facet>` tag:

```
<p:panel id="personaldata" header="Benutzerdaten" toggleable="true"
closable="false" toggleSpeed="500" closeSpeed="500" widgetVar="panel">
   <h:panelGrid columns="2" bgcolor="#eff5fa" >
      <f:facet name="header">Biologische Daten</f:facet>
...
   </h:panelGrid>
</p:panel>
```


Biologische Daten

Gewicht:	150,5	Tierart:	Huhn
Rasse:		Haarfarbe:	braun
Vater:		Mutter:	
Geschlecht:	● männlich ○ weiblich	Kastriert:	✓
Sterbedatum:			

Figure 32: <p:Panel> component with a <h:panelGrid> layout, C. Aigner

5.2.4 Navigation

Navigation was handled programmatically by the ManagedBeans, since JSF and PrimeFaces do not support a forward-based navigation within an AJAX request [69]. The `faces-redirect=true` attribute must be appended to the URL to let JSF 2.0 send the HTTP redirect 3xx code to the client, implementing the Post/Redirect/Get(PRG) design pattern [69]. The code for handling the redirects were put into a static method of the `FacesUtil()` class:

```
public static void redirectToTarget(String target) {
   FacesContext ctx = FacesContext.getCurrentInstance();
   ExternalContext ext = ctx.getExternalContext();
   try {
      ext.redirect(target + "?faces-redirect=true");
   } catch (IOException ex) {
      Logger.getLogger(FacesUtil.class.getName()).log(Level.SEVERE,
                                                    null, ex);
   }
}
```

5.2.5 Validators and converters

Before the model on the managed bean gets updated, after the user hit the submit button the data had to be converted from string to target objects [43]. The reserve action took place when the data was sent back to the client. Primitive data types are converted automatically. For complex types JSF 2.0 provides converters for common types like dates and numbers [43].

For converting date objects the `javax.faces.convert.DateTime` converter was used:

```
<p:calendar locale="de" pattern="dd.mm.yyyy" size="10" id="gebdat">
   <f:convertDateTime type="date"/>
</p:calendar>
```

JSF 2.0 provides server-side validators for verifying input data [43]. For the validation of e-mail addresses the javax.faces.validator.RegexValidator was used:

```
<p:inputText id="mailadresse">
   <f:validateRegex pattern="(^([a-zA-Z0-9_\-\.]+)@((\[[0-9]{1,3}\.
                             [0-9]{1,3}\.[0-9]{1,3}\.)|(([a-zA-Z0-9\-
                             ]+\.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(\]?)
                             $)?"
    />
</p:inputText>
```

5.2.6 Notifications

Part of the non-functional requirements (see chapter 4.2.2) was the consistent approach of user notifications. These can be error messages (e. g. validator messages) or informational messages (e. g. save confirmation).

Notifications were implemented with the PrimeFaces component `<p:growl>`. Growl is based on the Apple Macintosh growl notification widget [48]. It displays a rectangular message in the upper right corner of the browser screen. The component was declared globally in the `template.xhtml` file, so that every view can use it:

```
<p:growl id="growl" showDetail="false" globalOnly="false" />
```

After a command button is pressed, the underlying ManagedBean method that generates the message is executed and on completion the growl component is updated.

```
<p:commandButton value="Speichern" update="growl" id="submit"
action="#accountManagedBean.saveUser"/>
```

To be displayed the message must be added to the current Faces context:

```
FacesContext.getCurrentInstance().addMessage(null,
                new FacesMessage(FacesMessage.SEVERITY_INFO,
                "Änderungen gespeichert", ""));
```

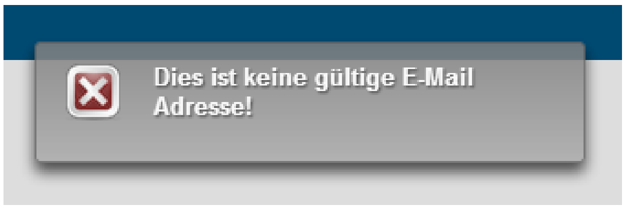

Dies ist keine gültige E-Mail Adresse!

Figure 33: Error message displayed with a p:growl component, C. Aigner

5.2.7 Menu

The menu was realized with the PrimeFaces component `<p:menubar>`. The content was defined within the corresponding ManagedBean because the composition depends on the logged-in user's role. For this purpose a "DefaultMenuModel" object was created and attached to the menu bar by using the attribute model.

```
<p:menubar model="{#menuManagedBean.mainMenuModel}" />
```

At creation time the ManagedBean calls the method `fillMainMenu()` which populates the menu model according to the user role. The following code demonstrates the creation of an ordinary menu item and a submenu containing one child.

```
DefaultMenuItem menuItem;
DefaultSubMenu submenuItem;

menuItem = new DefaultMenuItem();
menuItem.setValue("Start");
menuItem.setUrl("/ahrapp/index.xhtml");
menuItem.setIcon("ui-icon-home");
mainMenuModel.addElement(menuItem);

submenuItem = new DefaultSubMenu();
submenuItem.setLabel("Stammdaten");
submenuItem.setIcon("ui-icon-folder-open");

menuItem = new DefaultMenuItem();
menuItem.setValue("Landwirte suchen");
menuItem.setUrl("/ahrapp/searchfarm.xhtml");
menuItem.setIcon("ui-icon-search");
submenuItem.getElements().add(menuItem);

mainMenuModel.addElement(submenuItem);
```

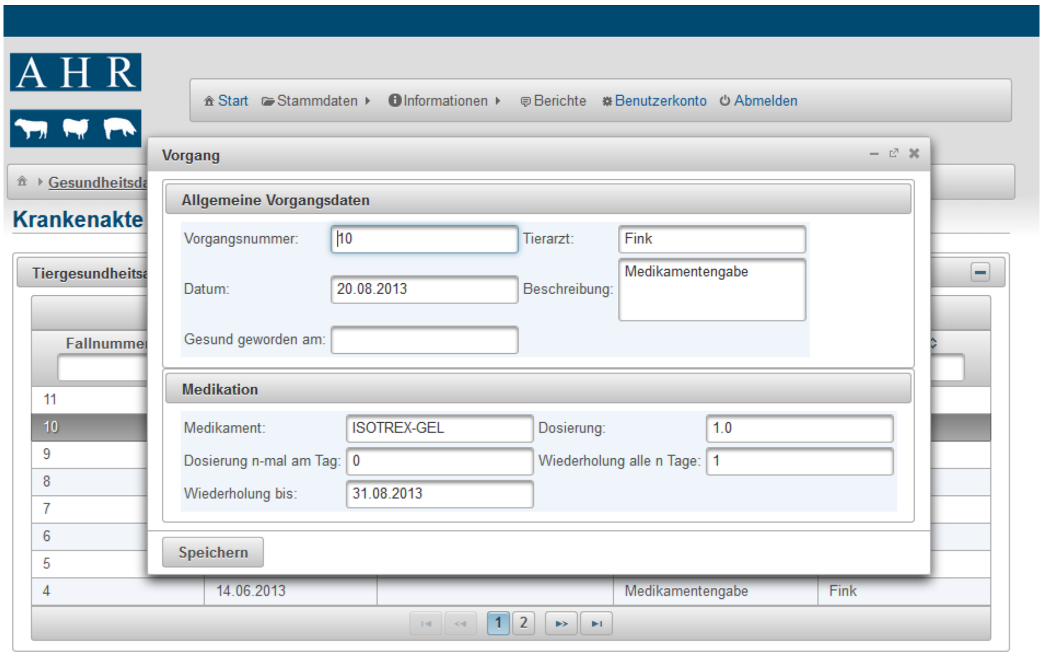
5.2.8 Dialogs

Dialogs are PrimeFaces components that can be defined within the facelet or can be referenced from another file.

The dialogs used in the applications are non-modal dialogs that are resizable, max- and minimizable. They are completely AJAX driven.

```
<p:dialog header="Vorgang" widgetVar="ahrDialog" id="ahrdialogid"
resizable="false" showEffect="puff" hideEffect="puff"
maximizable="true" minimizable="true">
```

They were made visible by the AJAX call `widgetVar.show()`.



Vorgang

Allgemeine Vorgangsdaten

Vorgangsnummer:	10	Tierarzt:	Fink
Datum:	20.08.2013	Beschreibung:	Medikamentengabe
Gesund geworden am:			

Medikation

Medikament:	ISOTREX-GEL	Dosierung:	1.0
Dosierung n-mal am Tag:	0	Wiederholung alle n Tage:	1
Wiederholung bis:	31.08.2013		

11				
10				
9				
8				
7				
6				
5				
4	14.06.2013		Medikamentengabe	Fink

1 2

Figure 34: <p:Dialog> widget containing two panels and a button, C. Aigner

5.3 Database

5.3.1 MySQL

After defining the database schema with the CREATE SCHEMA statement, the relations defined in chapter 4.4.2 were transcribed into a MySQL compatible SQL batch and loaded into the schema.

5.3.2 Hibernate

The following dependencies have to be added to the pom.xml for Hibernate integration:

```
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-entitymanager</artifactId>
   <version>3.3.2.GA</version>
</dependency>
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>ejb3-persistence</artifactId>
   <version>1.0.1.GA</version>
</dependency>
<dependency>
   <groupId>javax.sql</groupId>
   <artifactId>jdbc-stdext</artifactId>
   <version>2.0</version>
</dependency>
<dependency>
   <groupId>javax.transaction</groupId>
   <artifactId>jta</artifactId>
   <version>1.0.1B</version>
</dependency>
```

Configuration

The `hibernate-entitymanager` package represents the O/RM implementation of the JPA specification. The `ejb3-persistence` package contains the Hibernate Entity Manager EJB3 persistence. Additionally the JDBC standard extension (`jdbc-stdext`) and Java Transaction API (`jta`) packages are required.

The Hibernate configuration is stored in the file `hibernate.cfg.xml`. It contains the connection to the database and the class mappings. The database connection was made with JDBC. The property `hibernate.current_session_context_class` defines the scope of the session that is given by the method `SessionFactory.getCurrentSession()`. The option `thread` indicates that current sessions are tracked by thread of execution rather than JTA transactions [70].

```
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">
        org.hibernate.dialect.MySQLDialect
    </property>
    <property name="hibernate.connection.driver_class">
        com.mysql.jdbc.Driver
    </property>
    <property name="hibernate.connection.url">
       jdbc:mysql://localhost:3306/ahr
    </property>
    <property name="hibernate.connection.username">user</property>
    <property name="hibernate.connection.password">xxx</property>
    <property name="hibernate.current_session_context_class">
       Thread
    </property>

    <mapping class="com.healthresearch.ahr.tables.Person" />
    ...

  </session-factory>
</hibernate-configuration>
```

Object relational mapping

The configuration of the mapped Java classes is denoted with annotations. `@Entity` declares the class as being mapped with a database table. It is the only Hibernate annotation that is not optional. The `@Table` annotation can be used to define the name of the table if it not identical to the unqualified class name [70]. The class attributes can be annotated as well. `@Id` marks the column containing or being part of the primary key constraint. The `@Column` annotation can be used to define the name of the database column the attribute is mapped to. For date columns the `@Temporal` annotation can be used. Ids that are declared with the auto increment constraint, meaning that their value will be generated by the MySQL database engine, can be annotated with `@GeneratedValue`, so that they are left out in the insert statement Hibernate creates. The following example shows the mapping of the table `Person` with the Java class `Person`. The attribute id is the primary key and was declared as a generated value. The column `Firstname` is mapped to the attribute `firstname` of type `java.lang.String`. Birthdate has the SQL data type `DATE` and is mapped to a `java.util.Date` attribute named `birthdate` that is marked with the `TemporalType.DATE`.

```
@Entity
@Table(name="person")
@Inheritance(strategy = InheritanceType.JOINED)
public class Person implements Serializable {

    @Id
    @Column(name = "IDno")
    @GeneratedValue
    private int id;

    @Column(name = "Firstname")
    private String firstname;

    @Column(name = "Birthdate")
    @Temporal(javax.persistence.TemporalType.DATE)
    private Date birthdate;
...
}
```

For the modelling of the “is-a” relations defined in the ER-diagram (see chapter 4.4.1) java class inheritance was used. Hibernate knows three inheritance types [46]:

- One Table per class hierarchy (`InheritanceType.SINGLE_TABLE`)
- One Table per concrete Class (`InheritanceType.TABLE_PER_CLASS`)
- One Table per subclass (`InheritanceType.JOINED`)

The `SINGLE_TABLE` approach stores the entire class hierarchy into one single database table. There are columns for each mapped field of the superclass and the distinct fields of all derived classes within the hierarchy. Hibernate determines the appropriate type with the use of a discriminator column that distinguishes between each of the types used. The root class must contain the `@DiscriminatorColumn` annotation [46].

The `TABLE_PER_CLASS` approach stores all of the fields of each type in the inheritance hierarchy in distinct tables [46].

The `JOINED` approach stores the fields of the various derived types in distinct tables and connects them with a one-to-one primary key relation [46]. Hibernate does not require a discriminator column for this approach [70]. This method was used for the Person and Act relations. In the example above the Person class was annotated with `@Inheritance(strategy = InheritanceType.JOINED)`. All sub classes were then annotated with `@PrimaryKeyJoinColumn(name="IDno").`

The `@OneToMany` annotation was used to map 1:n entity relations. The following example shows the mapping of laboratory values to a given Test. The fetched data is stored in a `java.util.ArrayList` collection. The `FetchType` can be `LAZY` which means that the SQL query is executed when the field is accessed or `EAGER` which means that the SQL query is executed on creation of the class instance. The `@JoinColumn` annotation marks the foreign key constraint within the table.

```
@OneToMany(fetch = FetchType.EAGER)
@JoinColumn(name = "Caseno")
private List<LabValue> labvalues = new ArrayList<LabValue>();
```

The opposite side of the 1:n relation can be modelled as well with the `@ManyToOne` annotation, because in some cases it makes more sense to store the 1 side of the relation instead of the n one. For example, it was more reasonable to store the legal form in the class `Person` instead of storing a collection of all Persons associated with the form in the class `LegalForm`.

```
@ManyToOne
@JoinColumn(name="Legalform")
private LegalForm legalform;
```

The `@OneToOne` annotation was used to map 1:1 entity relations. The following example shows the mapping of the 1:1 relation Authority which is a self-reference to the table Person.

```
@OneToOne
@JoinColumn(name="Authority")
private Person vertretungsbefugter;
```

M:n relations where modelled with the `@ManyToMany` annotation. `@JoinTable` defines the join table and the corresponding columns. The following example shows the implementation of the m:n relation between farmers and veterinarians realized as a `java.util.HashSet` within the `Farmer` class. The name of the join table is `farmervetassoc`. The foreign key that references the table Farmer is called `Idfarmer`. The column for the reference to the Veterinarian table is called `Idvet`.

```
@ManyToMany
@JoinTable(name = "farmervetassoc", catalog = "ahr",
           joinColumns = { @JoinColumn(name = "Idfarmer",
                                       nullable = false,
                                       updatable = false) },
           inverseJoinColumns = { @JoinColumn(name = "Idvet",
                                              nullable = false,
                                              updatable = false) })
private Set<Veterinarian> associatedvets = new
HashSet<Veterinarian>();
```

Data Manipulation

The actual CRUD operations were implemented with DAOs (see chapter 3.3.1). The package `com.healthresearch.ahr.dao` contains all the DAO interfaces and the implementations. The `com.healthresearch.ahr.util.AHRHibernateUtil` provides the DAOs with the current Hibernate session. The following example shows the interface and the implementation of the `UserDAO`. It contains a method for saving a user and for searching a user by its username:

```
public interface UserDAO {
   public User getUserbyUsername(String username);
   public void saveUser(User u);
}

public class UserDAOImplHibernate implements UserDAO {

   @Override
   public User getUserbyUsername(String username) {
      User user = new User();
      Session s =
             AHRHibernateUtil.getSessionFactory().getCurrentSession();
      Transaction t = s.beginTransaction();
      Query q =
            s.createQuery("from User where username LIKE(:username)");
      q.setString("username", username);
      List<User> results = q.list();
      t.commit();
      if(!results.isEmpty())
         user = results.get(0);
      return user;
    }

   @Override
   public void saveUser(User u) {
      Session s =
             AHRHibernateUtil.getSessionFactory().getCurrentSession();
      Transaction t = s.beginTransaction();
      s.saveOrUpdate(u);
      t.commit();
    }
}
```

The method `getUserbyUsername(String username)` uses the Hibernate Query Language (HQL) to fetch the user from the table `User`. The addressed table in the `from` clause is actually the java class name not the name of the database table. The result is fetched into a `java.util.List` of User objects. The operation has to take place in a transaction.

The second method `saveUser(User u)` persists the User object to the database. If it's a new user Hibernate will generate an `INSERT` statement. For existing users it will generate a corresponding `UPDATE` statement. These operations must be performed in a transaction as well.

5.4 Security

The role concept described in chapter 4.3 was realized with the component Spring Security version 3.1.3. Spring Security is an authentication and access-control framework that is part of the Spring Component framework [65]. The first step was to add the following dependencies to the `pom.xml`:

```
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-core</artifactId>
   <version>3.1.3.RELEASE</version>
</dependency>
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-crypto</artifactId>
   <version>3.1.3.RELEASE</version>
</dependency>
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-web</artifactId>
   <version>3.1.3.RELEASE</version>
</dependency>
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-config</artifactId>
   <version>3.1.3.RELEASE</version>
</dependency>
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-taglibs</artifactId>
   <version>3.1.3.RELEASE</version>
</dependency>
```

Maven automatically resolves the missing Spring core libraries that Spring Security needs in order to work properly.

The next step was to create the configuration, which was stored in `WEB-INF/security-app-context.xml`. The http tag defines which files are allowed to which roles.

```
<http use-expressions="true">
   <intercept-url pattern="/ahrapp/index.xhtml"
                  access="isAuthenticated()" />
   <intercept-url pattern="/ahrapp/account.xhtml"
                  access="isAuthenticated()" />
   <intercept-url pattern="/ahrapp/**"
                  access="hasRole('Landwirt') or hasRole('Tierarzt')
                       or hasRole('Administrator')" />
   <intercept-url pattern="/**" access="permitAll" />
   <form-login login-page="/login.xhtml"
               authentication-failure-url="/loginfailed.xhtml"
               default-target-url="/ahrapp/index.xhtml" />
   <remember-me />
   <logout logout-success-url="/logout.xhtml"
           invalidate-session="true" />
</http>
```

The `<authentication-manager>` tag defines where the users and groups are stored. The simple form is to define the users in the XML file itself with the `<user-service>` tag. For this application the users and groups were stored in the database and were fetched by Spring Security with a data source defined in `WEB-INF/applicationContext.xml`. Spring Security demands a field in the query where it checks if the user is enabled. Since this field did not exist in the database it always evaluated to `true` in the SQL query.

```
<authentication-manager>
   <authentication-provider>
      <jdbc-user-service data-source-ref="dataSource"
          users-by-username-query=
             "select username,passwort, 'true' as enabled from userahr
              where username=?"
          authorities-by-username-query=
             "select u.username,g.groupname from userahr u, groupahr g
              where u.groupahr = g.id and u.username = ?"
    />
    </authentication-provider>
</authentication-manager>
```

The final step was to modify the `WEB-INF/web.xml` file. The `org.spring.web.context.ContextLoaderListener` is in charge of starting and stopping the Spring root `ApplicationContext` interface and determines which configurations are to be used, by looking at the `<context-param>` tag for `contextConfigLocation` [65]. The before mentioned `applicationContext.xml` and `security-app-context.xml` were added to this parameter.

The `springSecurityFilterChain` intercepts all requests. The filter mappings are considered in the order that they are declared [65].

```
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>
      /WEB-INF/applicationContext.xml
      /WEB-INF/security-app-context.xml
   </param-value>
</context-param>
<filter>
   <filter-name>springSecurityFilterChain</filter-name>
   <filter-class>
      org.springframework.web.filter.DelegatingFilterProxy
   </filter-class>
</filter>
<filter-mapping>
   <filter-name>springSecurityFilterChain</filter-name>
   <url-pattern>/*</url-pattern>
   <dispatcher>FORWARD</dispatcher>
   <dispatcher>REQUEST</dispatcher>
</filter-mapping>
<listener>
   <listener-class>
      org.springframework.web.context.ContextLoaderListener
   </listener-class>
</listener>
```

5.5 Business Logic

The actual business logic was developed with ManagedBeans – the model component of JavaServer Faces - and Enterprise Java Beans. The ManagedBeans were mainly used to control the data and information flow whereas the EJBs were used for computing the business logic and communicating with the DAOs to perform the CRUD operations.

The ManagedBeans were annotated with `@SessionScoped` to last the whole user session. Access to other ManagedBeans were made with managed properties. Java EE performs a setter injection (see chapter 3.3.2) for the attribute that is annotated with `@ManagedProperty`. EJBs can be injected with the `@EJB` annotation. The following example shows an excerpt of the class `FarmManagedBean`:

```
@ManagedBean
@SessionScoped
public class FarmManagedBean {

   @EJB
   private AnimalBean animalBean;

   @EJB
   private ActBean actBean;

   @ManagedProperty(value = "#{animalManagedBean}")
   private AnimalManagedBean animalmb;

...

   @PostConstruct
   Private void init() {
      doInit();
   }

   public void setAnimalmb(AnimalManagedBean animalmb) {
      this.animalmb = animalmb;
   }

   public void redirecttoAnimal() {
      //Set Animal object
      animalmb.setAnimal(animalBean.getAnimal(((Animal)
                         selectedNode.getData()).getId()));
      //Set corresponding Farmer object
      animalmb.setFarmer(((Animal)
                          selectedNode.getData()).getOwner());
      //Set Acts
      animalmb.setActlist((ArrayList<Act>)
       actBean.getActsbyId(((Animal)selectedNode.getData()).getId()));
      //Redirect to Animal detail page
      FacesUtil.redirectToTarget("animal.xhtml");
    }
...
```

The `AnimalManagedBean` bean was injected to set the Animal object the user had chosen from the data table before the redirection to `animal.xhtml` occurred. The static method `FacesUtil.redirectToTarget(String target)` loaded the external Faces context and set a redirect to the given target. Since EJBs are injected after the instantiation of the ManagedBean all initialisation code concerning EJBs must be done after the constructor. This behaviour can be achieved by annotation a method with the `@PostConstruct` annotation.

The EJBs were declared with the `@Stateful` annotation. `@Stateful` EJBs maintain conversational state with the client. Every request that is coming from the ManagedBean is passed to the same instance. This behaviour can be seen as a 1-to-1 relation between EJB instance and ManagedBean instance [43]. The following example shows parts of the `UserBean` that was used by the `AccountManagedBean` for the `account.xhtml` Facelet that displays and manipulates the current user details.

```
@Stateful
public class UserBean {

   UserDAO db = new UserDAOImplHibernate();

   public UserBean() {
   }
...
   public void saveUser(User u) {
      try {
         db.saveUser(u);
      } catch (Exception e) {
         FacesContext.getCurrentInstance().addMessage(null,
                    new FacesMessage(FacesMessage.SEVERITY_ERROR,
                    "Fehler beim Speichern!", ""));
      }
      FacesContext.getCurrentInstance().addMessage(null,
                new FacesMessage(FacesMessage.SEVERITY_INFO,
                "Änderungen gespeichert", ""));
    }
}
```

The method `saveUser(User u)` calls the `UserDAO` to persist the given user. The user gets notified with a success message or if an exception occurred with an error message.

5.6 Logging

Logging was done with the Apache log4j logging utility version 1.2.17. To use it, the following dependency had to be added to the `pom.xml`:

```
<dependency>
   <groupId>log4j</groupId>
   <artifactId>log4j</artifactId>
   <version>1.2.17</version>
   <type>jar</type>
</dependency>
```

Log4j supports various log levels that are ordered [71]. These are `TRACE < DEBUG < INFO < WARN < ERROR < FATAL`, `TRACE` being the most verbose, only to written to log files only and `FATAL` being a severe error. [71].

The configuration was stored in the `log4j.properties`:

```
log4j.rootLogger = WARN, stdout
log4j.logger.AHR = DEBUG, FILE, stdout

# Direct log messages to a log file
log4j.appender.FILE=org.apache.log4j.RollingFileAppender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.ConversionPattern=
                            %d{dd.MM.yyyy HH:mm:ss} %-5p %c[26] - %m%n
log4j.appender.FILE.File=C:\\temp\\ahr.log

# Direct log messages to stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=
                            %d{dd.MM.yyyy HH:mm:ss} %-5p %c[26] - %m%n
log4j.appender.stdout.Target=System.out
```

The `rootLogger` definition set the utility to print `WARN`, `ERROR` and `FATAL` messages of any components respectively APIs to the console. The AHR logger was used to write all CRUD actions, according to the functional requirements to a log file and additionally to the console. The entries were marked as `INFO` messages. The logger was called via the final static attribute `com.healthresearch.ahr.LogUtil.LOGGER`. The entries were formatted with the `org.apache.log4j.PatternLayout` and written to a predefined folder to the local hard disk. The following example shows an extraction of the `ahr.log` log file:

```
...
26.11.2013 19:01:45 INFO  AHR - User: aigner has been saved
26.11.2013 19:01:53 INFO  AHR - Veterinarian: Steinke has been saved
26.11.2013 19:01:58 INFO  AHR - Veterinarian: Jan has been saved
26.11.2013 19:05:20 INFO  AHR - User: groth has been saved
...
```

5.7 RESTful web service

The interface for external applications was realized with a RESTful web service. The Java reference implementation of RESTful web services Jersey is already a part of the Glassfish application server.

To initialize it, the following content has to be added to the `web.xml`:

```
<servlet>
        <servlet-name>jersey-serlvet</servlet-name>
        <servlet-class>
            com.sun.jersey.spi.container.servlet.ServletContainer
        </servlet-class>
        <init-param>
            <param-name>
                com.sun.jersey.config.property.packages
             </param-name>
            <param-value>com.healthresearch.ahr.interop</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>jersey-serlvet</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
...
</servlet>
```

The Jersey-servlet must be declared with the initialization parameter `com.sun.jersey.config.property.packages` pointing to the package the web service classes reside. The servlet mapping declares the basic URI to call the services.

The following source code demonstrates the implementation of the web service that retrieves farmers by their ids. The result is a XML file with all attributes of the farmer-object:

```
@Path("farmer")
```

```
@Produces(MediaType.APPLICATION_XML)
@Stateless
public class FarmServiceResource {

    PersonDAO db = new PersonDAOImplHibernate();

    @Context
    private UriInfo context;

    public FarmServiceResource() {
    }

    @GET
    @Path("{id}")
    public Farmer getFarmerbyId(@PathParam("id") int id) {
        Farmer f = db.getFarmerbyId(id);
        return f;
    }
}
```

The java class must be annotated with `@Path(URI)` to be declared a RESTful web service. The `@Produces` annotation defines the MIME type of the output of GET requests. The opposite would be the @Consumes annotation, which defines the MIME type of the input for POST requests. These two annotations can also be attached to a method. The `@Stateless` annotations gives the web service EJB capabilities, e.g. transactional access to a persistent layer [43].

The `@GET` annotation connects the method to a HTTP GET request. `@Path("{id}")` makes the parameter, the farmer id, part of the URI, to which the `@PathParam("id")` annotation refers too. The method itself calls the DAO method to retrieve the farmer by its id from the database using Hibernate and returns it. Since the class `com.healthresearch.ahr.tables.Farmer` class is annotated with `@XmlRootElement` and the `@Produces` annotation was set to produce XML files, the return value for the GET request will be the farmer object with the given id formatted as a XML file.

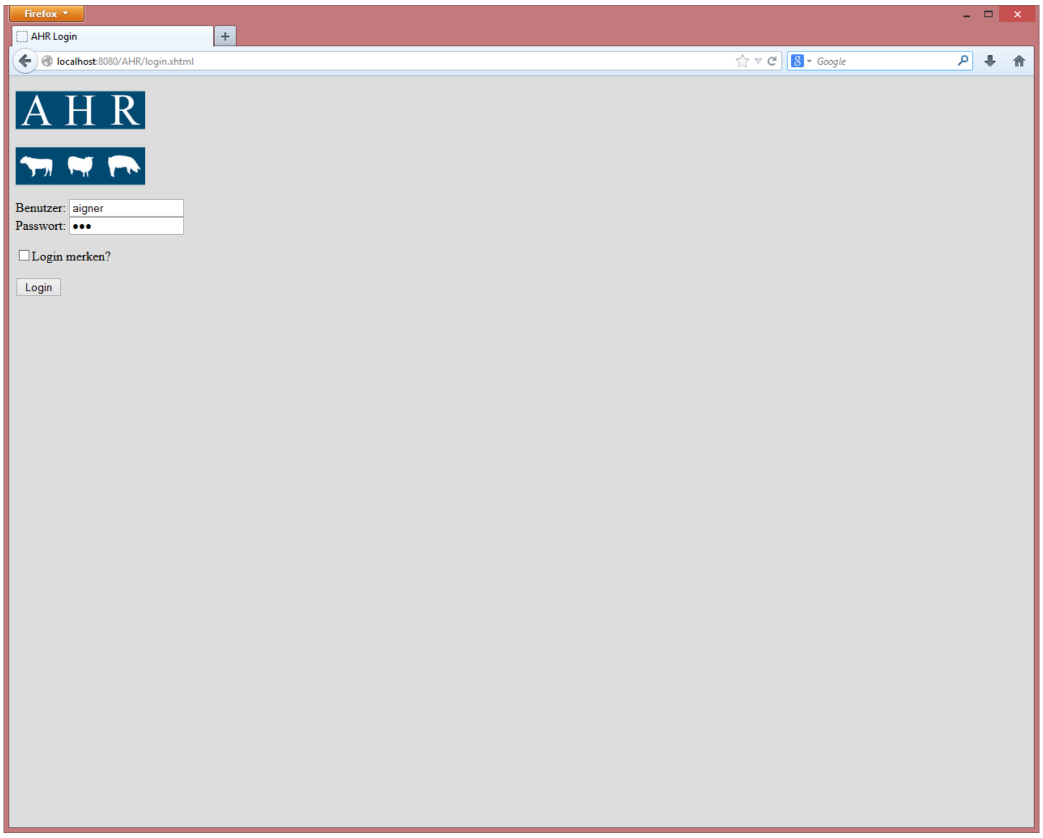
To retrieve the farmer with id 1, the following URL must be called: `http://localhost:8080/AHR/rest/farmer/1`

6 Results

This chapter describes the functional prototype – being the result of this thesis – in the form of screenshots and descriptions based on the functional requirements. The screenshots were made at a resolution of 1280x1024 pixels.

6.1 Login/Logout

When the user requests any link under the URI `/ahrapp` without prior authorization, the `login.xhtml` page is sent back to him instead. It contains two input fields and a checkbox. The first input field is the username. The second input field is the password. The entered password is being shown as dots for security reasons. The checkbox "Login merken?" – "Remember login?" sets a cookie, so that when the user closes the web browser and opens it again, he is still authenticated within the system.



Benutzer: aigner
Passwort: •••
☐ Login merken?

Figure 35: Login page, C. Aigner

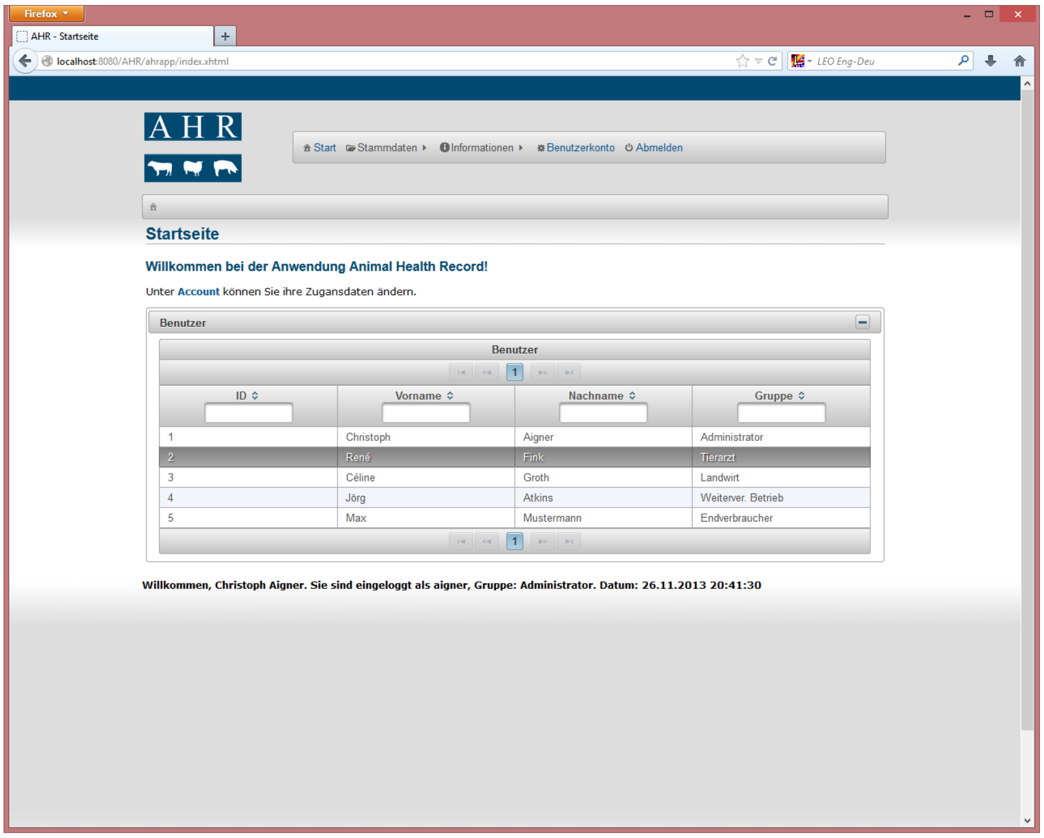
6.2 Start page

After a successful login the user is presented with the start page. Its content depends on the role the user has.

6.2.1 Administrators

Administrators get a list of registered users on their start page. Their main menu contains the following items:

```
Start
Stammdaten
> Landwirte
>> Landwirte suchen
>> Neuer Landwirt
> Tierärzte
>> Tierärzte suchen
>> Neuer Tierarzt
> Weiterver. Betriebe
>> Betriebe suchen
>> Neuer Betrieb
> Benutzer
>> Benutzer suchen
>> Neuer Benutzer
> Arzneimittelkatalog
>> Neues Medikament
Informationen
> Arzneimittelkatalog
Benutzerkonto
Abmelden
```



Startseite

Willkommen bei der Anwendung Animal Health Record!

Unter **Account** können Sie ihre Zugansdaten ändern.

Benutzer

ID	Vorname	Nachname	Gruppe
1	Christoph	Aigner	Administrator
2	René	Fink	Tierarzt
3	Céline	Groth	Landwirt
4	Jörg	Atkins	Weiterver. Betrieb
5	Max	Mustermann	Endverbraucher

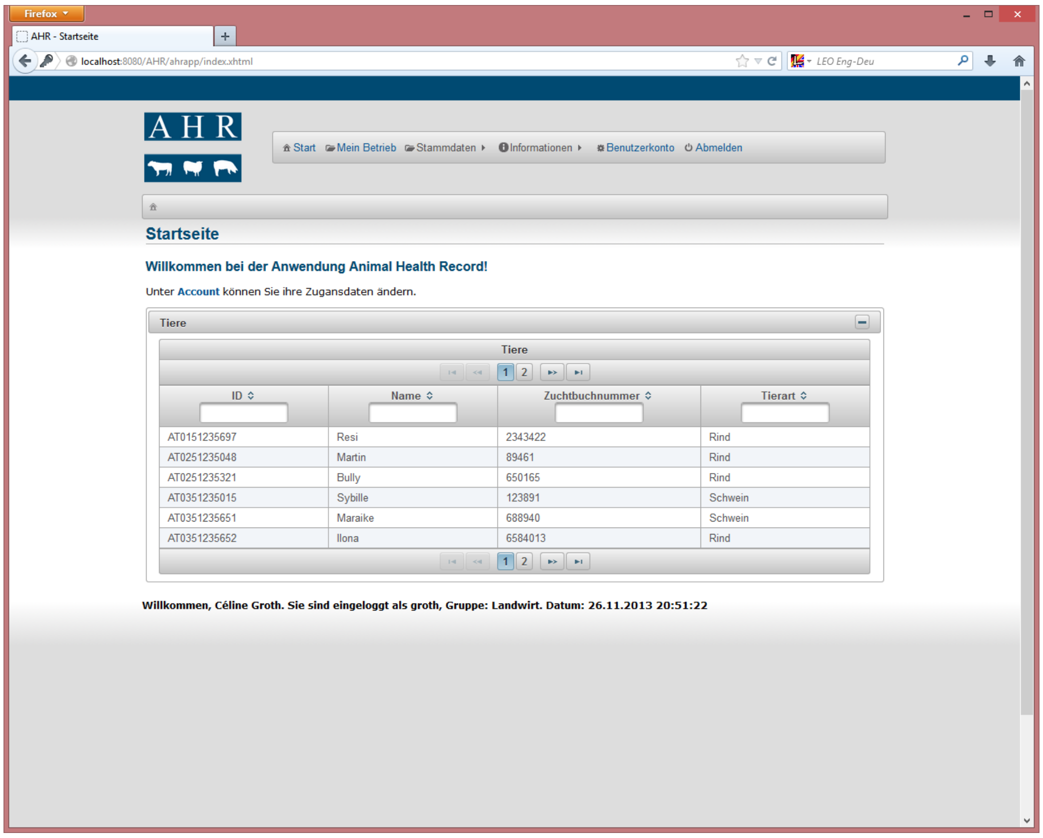
Willkommen, Christoph Aigner. Sie sind eingeloggt als aigner, Gruppe: Administrator. Datum: 26.11.2013 20:41:30

Figure 36: Start page for admins, C. Aigner

6.2.2 Farmers

Farmers get a searchable list of their animals. Their main menu contains the following items:

```
Start
Mein Betrieb
Stammdaten
> Tierärzte
>> Tierärzte suchen
> Weiterver. Betriebe
>> Betriebe suchen
Informationen
> Arzneimittelkatalog
Benutzerkonto
Abmelden
```



Startseite

Willkommen bei der Anwendung Animal Health Record!

Unter Account können Sie ihre Zugansdaten ändern.

Tiere

ID	Name	Zuchtbuchnummer	Tierart
AT0151235697	Resi	2343422	Rind
AT0251235048	Martin	89461	Rind
AT0251235321	Bully	650165	Rind
AT0351235015	Sybille	123891	Schwein
AT0351235651	Maraike	688940	Schwein
AT0351235652	Ilona	6584013	Rind

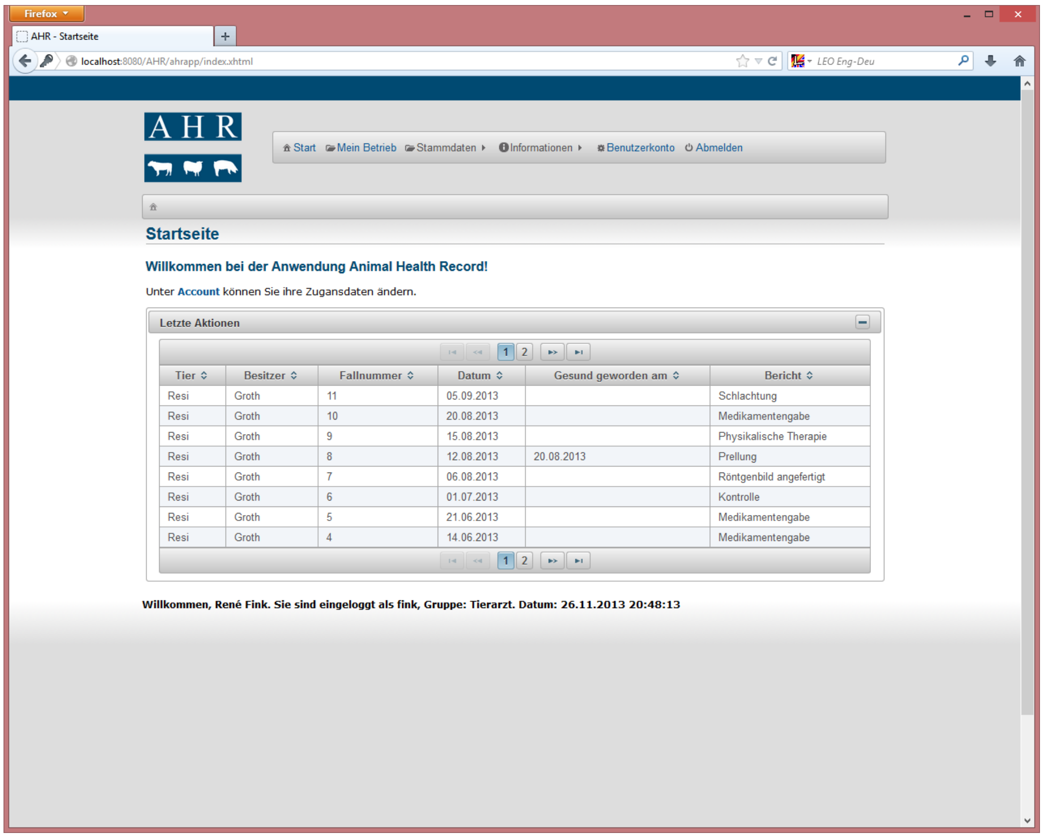
Willkommen, Céline Groth. Sie sind eingeloggt als groth, Gruppe: Landwirt. Datum: 26.11.2013 20:51:22

Figure 37: start page for farmers, C. Aigner

6.2.3 Veterinarians

Veterinarians get a list of the last treatments they conducted. Their main menu contains the following items:

```
Start
Mein Betrieb
Stammdaten
> Landwirte
>> Landwirte suchen
> Tierärzte
>> Tierärzte suchen
> Weiterver. Betriebe
>> Betriebe suchen
Informationen
> Arzneimittelkatalog
Benutzerkonto
Abmelden
```



Startseite

Willkommen bei der Anwendung Animal Health Record!

Unter Account können Sie ihre Zugansdaten ändern.

Letzte Aktionen

Tier	Besitzer	Fallnummer	Datum	Gesund geworden am	Bericht
Resi	Groth	11	05.09.2013		Schlachtung
Resi	Groth	10	20.08.2013		Medikamentengabe
Resi	Groth	9	15.08.2013		Physikalische Therapie
Resi	Groth	8	12.08.2013	20.08.2013	Prellung
Resi	Groth	7	06.08.2013		Röntgenbild angefertigt
Resi	Groth	6	01.07.2013		Kontrolle
Resi	Groth	5	21.06.2013		Medikamentengabe
Resi	Groth	4	14.06.2013		Medikamentengabe

Willkommen, René Fink. Sie sind eingeloggt als fink, Gruppe: Tierarzt. Datum: 26.11.2013 20:48:13

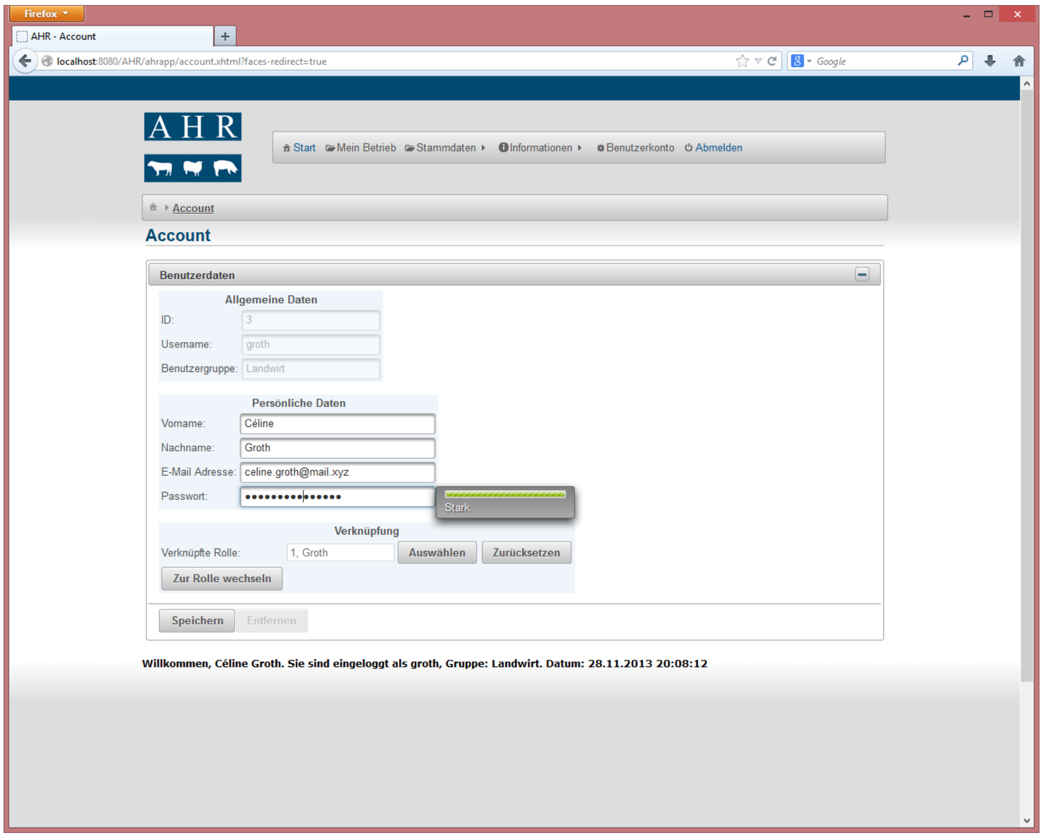
Figure 38: start page for veterinarians, C. Aigner

6.3 Account page

With the menu button "Benutzerkonto – Account" the user gets the form to edit his user details. The user can edit his first name, surname, e-mail address, password and his associated role. The password field is attached with an indicator to inform the user of the password strength he entered. The associated role input field has two command buttons on its right side. The first one opens a dialog where the user can search for a person (farmer, veterinarian or processing industry). After the search has been conducted, he can click on a result in the list to connect it to his user object. On success the associated role input field adopts the name of the associated role. The second button deletes this reference.

The save button on the bottom of the page saves all changes and notifies the user of the result of this operation with the help of the growl component.

The delete button is only enabled for users with administrative privileges and deletes the current user (chosen before from the user-search form).



Account

Benutzerdaten

Allgemeine Daten

ID:	3
Username:	groth
Benutzergruppe:	Landwirt

Persönliche Daten

Vorname:	Céline
Nachname:	Groth
E-Mail Adresse:	celine.groth@mail.xyz
Passwort:	•••••••••••••• Stark

Verknüpfung

Verknüpfte Rolle:	1, Groth

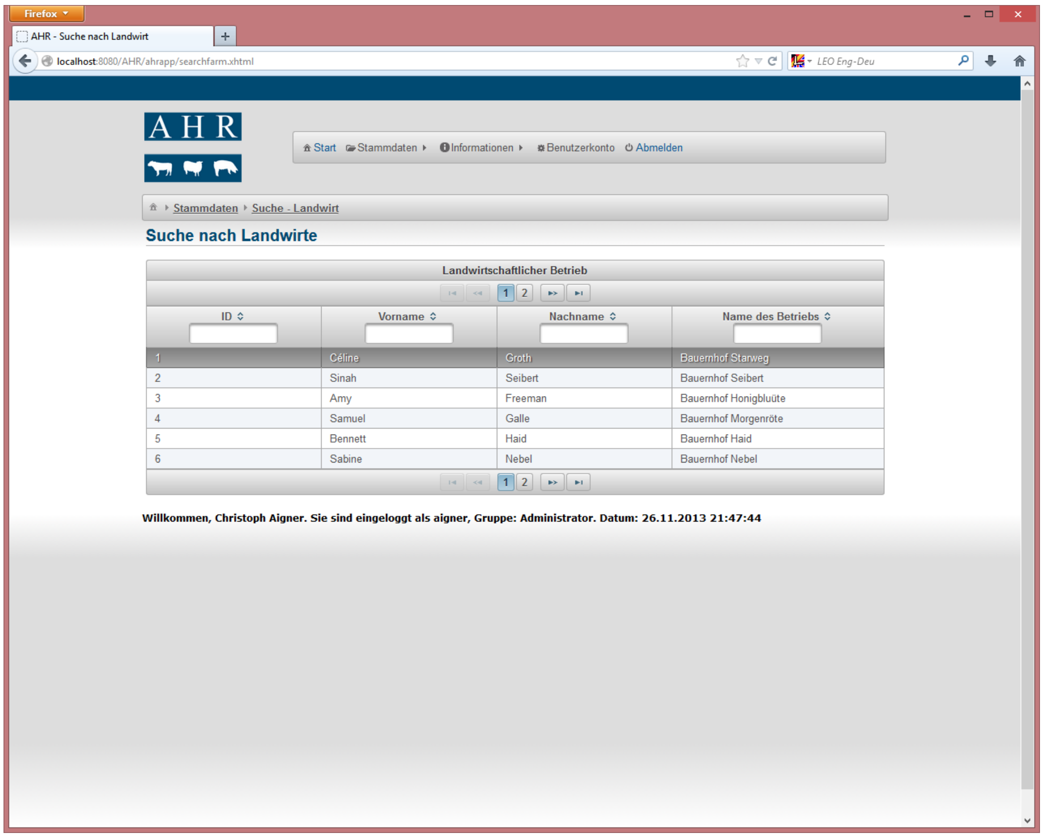
Willkommen, Céline Groth. Sie sind eingeloggt als groth, Gruppe: Landwirt. Datum: 28.11.2013 20:08:12

Figure 39: account page, C. Aigner

6.4 Search for farms

Administrators and veterinarians can search for farms by clicking on the menu item "Stammdaten – Landwirte – Landwirte suchen". The view consists of a paginated data table with input fields on every column. When the user enters a search criteria in one of the input fields the system immediately begins the search and narrows down the results. The order of the columns can be manipulated by dragging them to a desired position. The list can also be sorted by any column.

By clicking on an entry in the data table, the user gets directed to the detail-view of the chosen farm.



Suche nach Landwirte

Landwirtschaftlicher Betrieb

ID	Vorname	Nachname	Name des Betriebs
1	Céline	Groth	Bauernhof Starweg
2	Sinah	Seibert	Bauernhof Seibert
3	Amy	Freeman	Bauernhof Honigbluüte
4	Samuel	Galle	Bauernhof Morgenröte
5	Bennett	Haid	Bauernhof Haid
6	Sabine	Nebel	Bauernhof Nebel

Willkommen, Christoph Aigner. Sie sind eingeloggt als aigner, Gruppe: Administrator. Datum: 26.11.2013 21:47:44

Figure 40: farm search view, C. Aigner

6.5 Add/edit farm-data

Administrators can add farms via the menu button "Stammdaten – Landwirt – Neuer Landwirt". Existing farm data can be edited by searching a farm and clicking on the result or by clicking on "Mein Betrieb" if the current user is associated with a farmer object.

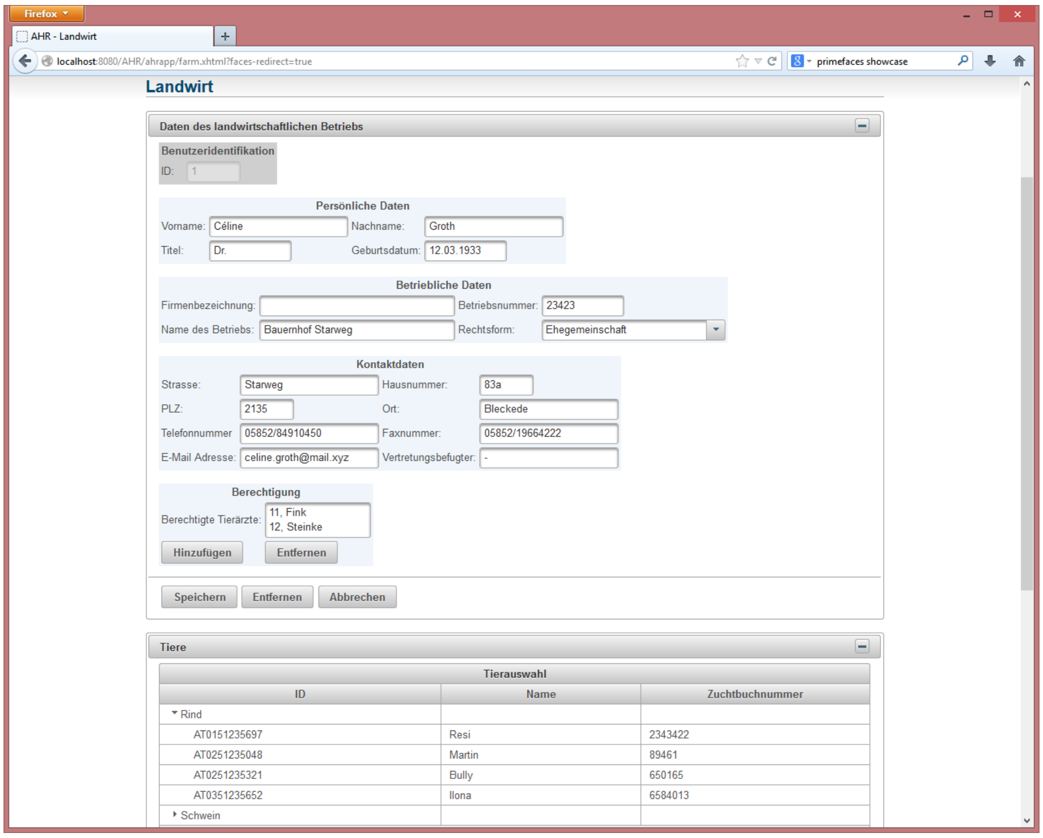
The farm-detail view consists of two collapsible panels. The first panel displays the standing data of the farm object. The second panel displays the managed animals, with a tree view component.

The user can edit his personal data, company data and contact data. The birthdate gets chosen with the help of an overlay calendar that appears when the user clicks on the input field. Legal forms can be chosen from a selection list. An associate can be searched and added by clicking on the matching input field.

The associated veterinarians are managed with a list box and two buttons. After clicking the “Hinzufügen” button, a dialog opens and lets the user chose a veterinarian. With the “Entfernen” button a selected veterinarian can be deleted.

There are three buttons at the bottom of the first panel. The first button saves the entered data and notifies the user of the outcome. The second button deletes the farm (only administrators can perform this operation). After deleting the object the user is redirected to the start page. The “Abbrechen” button redirects the user back to the start page. The administrator gets redirected to the farm-search view.

The second panel contains all animals that are managed by the farm. They are pre-sorted by species and displayed as a tree. The user can click on the “+” sign left to the species label to expand the list. By clicking on an animal he gets redirected to the animal detail view.



Landwirt

Daten des landwirtschaftlichen Betriebs

Benutzeridentifikation
ID: 1

Persönliche Daten

Vorname:	Céline	Nachname:	Groth
Titel:	Dr.	Geburtsdatum:	12.03.1933

Betriebliche Daten

Firmenbezeichnung:		Betriebsnummer:	23423
Name des Betriebs:	Bauernhof Starweg	Rechtsform:	Ehegemeinschaft

Kontaktdaten

Strasse:	Starweg	Hausnummer:	83a
PLZ:	2135	Ort:	Bleckede
Telefonnummer	05852/84910450	Faxnummer:	05852/19664222
E-Mail Adresse:	celine.groth@mail.xyz	Vertretungsbefugter:	-

Berechtigung

Berechtigte Tierärzte: 11, Fink; 12, Steinke

Tiere

Tierauswahl

ID	Name	Zuchtbuchnummer
Rind		
AT0151235697	Resi	2343422
AT0251235048	Martin	89461
AT0251235321	Bully	650165
AT0351235652	Ilona	6584013
Schwein		

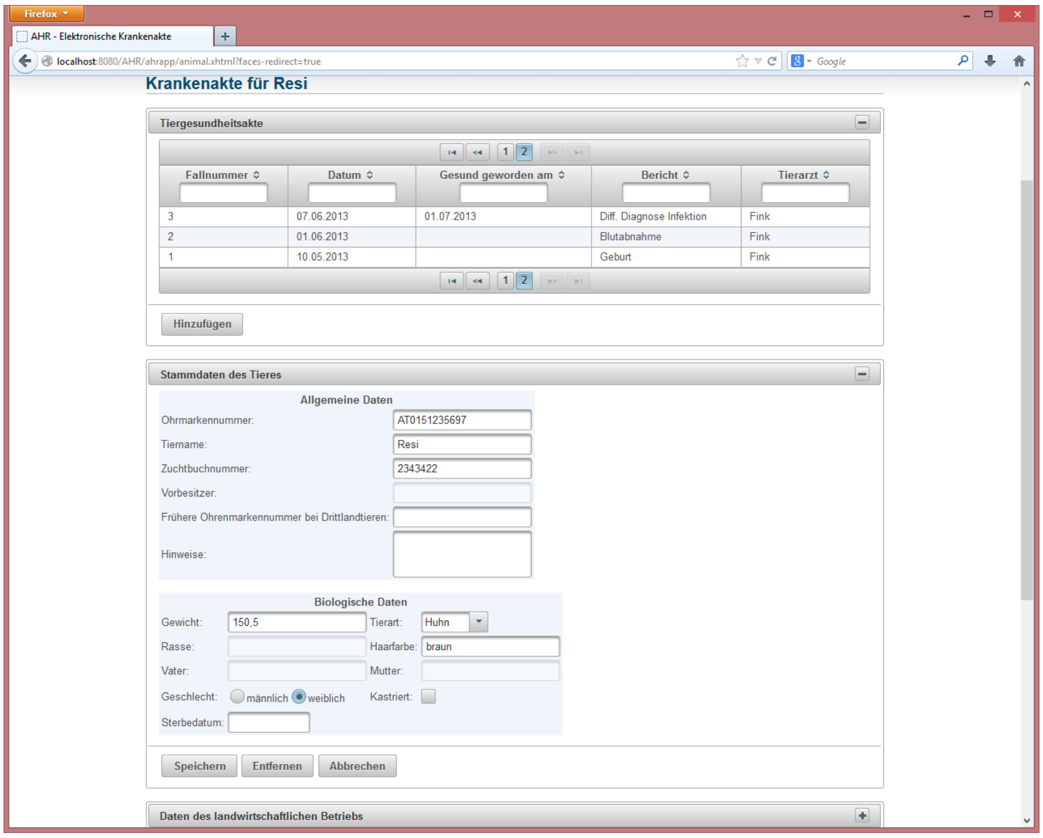
Figure 41: farm detail view, C. Aigner

6.6 Add/edit animal-data

The animal detail view consists of three panels. The first panel contains the actual animal health record displayed as a data table. The columns can be sorted, reordered and contain input fields for the search. When the user selects a row, the corresponding dialog opens with the detail view of the particular act. The table footer contains the command button “Hinzufügen”, that adds a new act.

The second panel contains the animal standing data. The user can modify general and biological data of the animal. The input fields for “Rasse”, “Vater” and “Mutter” are clickable and will open a search dialog for the association of the particular entities “race”, “father animal” and “mother animal”. The panel footer contains three command buttons. The first one “Speichern” saves the changes. The second one “Entfernen”, only accessible to the farmer and administrators, deletes the animal. The third button “Abbrechen” return the user to the farm view.

The third panel, which is collapsed by default contains personal and operating data of the farm.


Firefox
AHR - Elektronische Krankenakte
localhost:8080/AHR/ahrapp/animal.xhtml?faces-redirect=true
Google

Krankenakte für Resi

Tiergesundheitsakte

Fallnummer	Datum	Gesund geworden am	Bericht	Tierarzt
3	07.06.2013	01.07.2013	Diff. Diagnose Infektion	Fink
2	01.06.2013		Blutabnahme	Fink
1	10.05.2013		Geburt	Fink

Hinzufügen

Stammdaten des Tieres

Allgemeine Daten

Ohrmarkennummer:	AT0151235697
Tiername:	Resi
Zuchtbuchnummer:	2343422
Vorbesitzer:	
Frühere Ohrenmarkennummer bei Drittlandtieren:	
Hinweise:	

Biologische Daten

Gewicht:	150,5	Tierart:	Huhn
Rasse:		Haarfarbe:	braun
Vater:		Mutter:	
Geschlecht:	männlich [✓ weiblich]	Kastriert:	☐
Sterbedatum:			

Speichern Entfernen Abbrechen

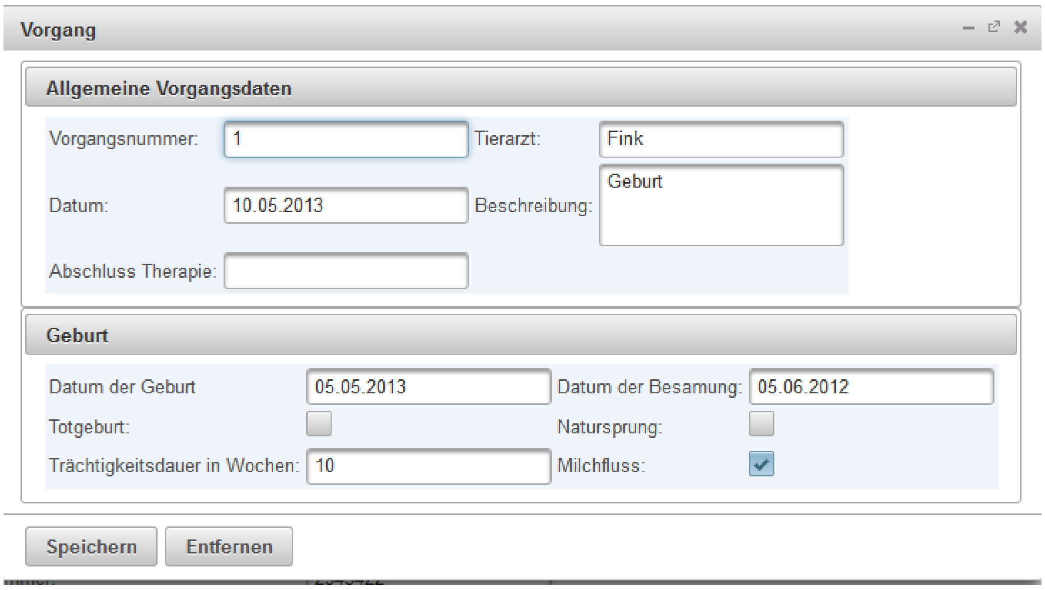
Daten des landwirtschaftlichen Betriebs

Figure 42: animal detail view, C. Aigner

Dialogs are movable within the browser window, can be minimized respectively maximized and be closed with three button located on the right side of the title bar. They contain panels for the basic act data and the special act-type data. The footer contains a button to save the changes and a button to delete the act from the database.

6.6.1 Birth dialog

The birth dialog contains all relevant data for the act “Birth”.



Vorgang

Allgemeine Vorgangsdaten

Vorgangsnummer:	1	Tierarzt:	Fink
Datum:	10.05.2013	Beschreibung:	Geburt
Abschluss Therapie:			

Geburt

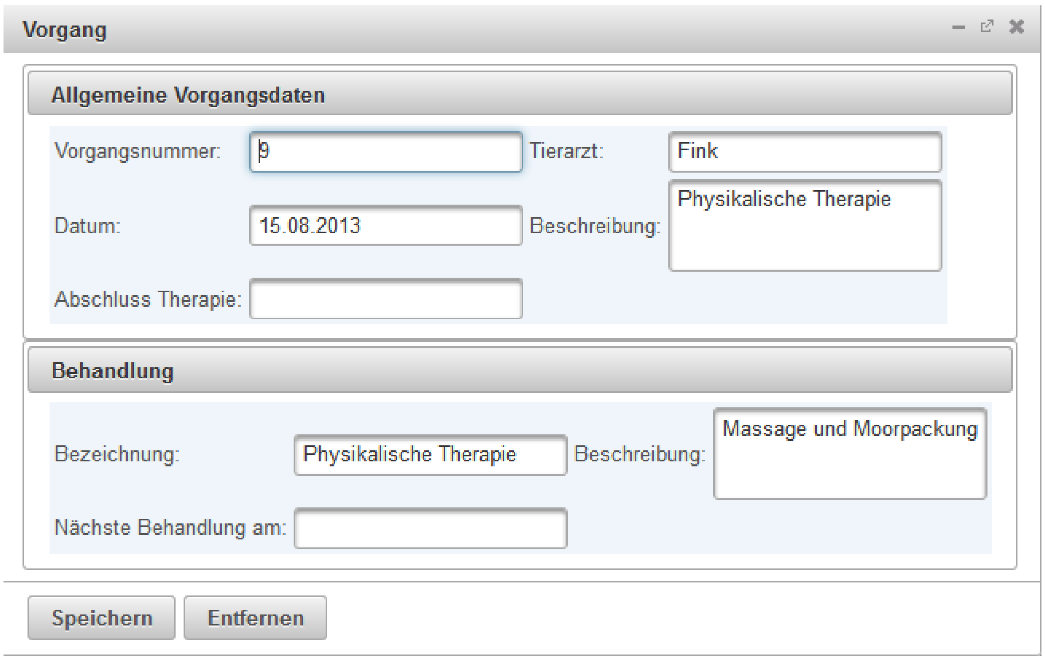
Datum der Geburt	05.05.2013	Datum der Besamung:	05.06.2012
Totgeburt:	☐	Natursprung:	☐
Trächtigkeitsdauer in Wochen:	10	Milchfluss:	✓

Speichern Entfernen

Figure 43: birth dialog, C. Aigner

6.6.2 Treatment dialog

The treatment dialog contains all relevant treatment data.



Vorgang

Allgemeine Vorgangsdaten

Vorgangsnummer:	9	Tierarzt:	Fink
Datum:	15.08.2013	Beschreibung:	Physikalische Therapie
Abschluss Therapie:			

Behandlung

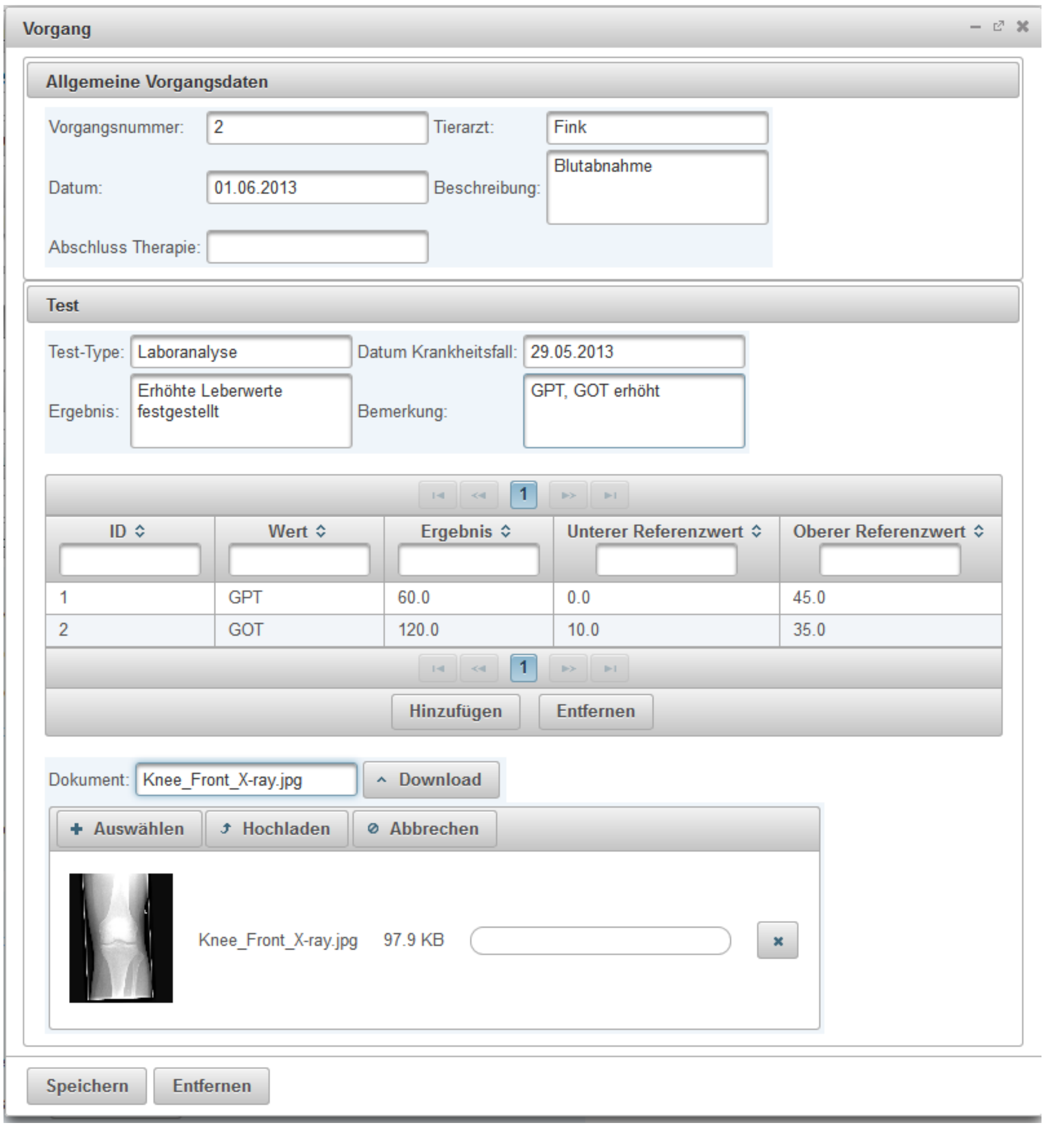
Bezeichnung:	Physikalische Therapie	Beschreibung:	Massage und Moorpackung
Nächste Behandlung am:			

Speichern Entfernen

Figure 44: treatment dialog, C. Aigner

6.6.3 Test dialog

The test dialog contains all relevant test data. It includes a sortable, searchable data table for the analysed lab values. The button “Hinzufügen” adds a new lab value to the act. “Entfernen” deletes the selected lab value from the database. A file can be uploaded by first clicking on “Auswählen”. This button opens the browsers native file open dialog where the user can choose a file that needs to be uploaded. If the file passes validation it can be uploaded with the “Hochladen” button. The actual upload happens automatically with an AJAX call. The download button retrieves the saved document from the database.


Vorgang

Allgemeine Vorgangsdaten

Vorgangsnummer: 2
Tierarzt: Fink
Datum: 01.06.2013
Beschreibung: Blutabnahme
Abschluss Therapie:

Test

Test-Type: Laboranalyse
Datum Krankheitsfall: 29.05.2013
Ergebnis: Erhöhte Leberwerte festgestellt
Bemerkung: GPT, GOT erhöht

ID	Wert	Ergebnis	Unterer Referenzwert	Oberer Referenzwert
1	GPT	60.0	0.0	45.0
2	GOT	120.0	10.0	35.0

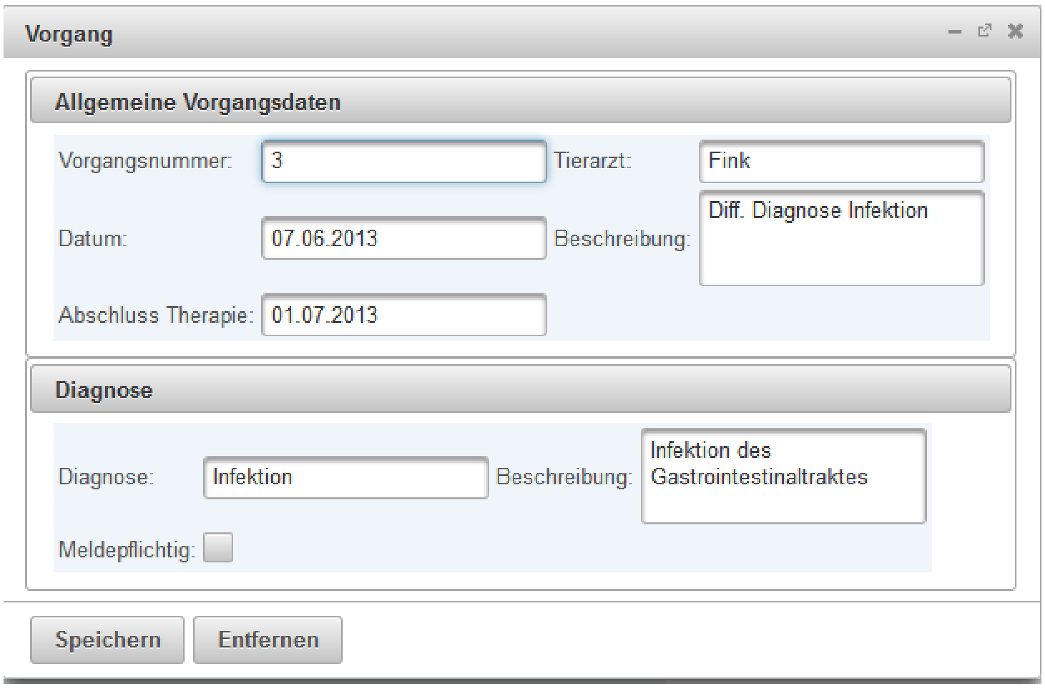
Dokument: Knee_Front_X-ray.jpg

Knee_Front_X-ray.jpg 97.9 KB

Figure 45: test dialog, C. Aigner

6.6.4 Diagnosis dialog

The diagnosis dialog contains all relevant diagnosis data.


Vorgang

Allgemeine Vorgangsdaten

Vorgangsnummer:	3	Tierarzt:	Fink
Datum:	07.06.2013	Beschreibung:	Diff. Diagnose Infektion
Abschluss Therapie:	01.07.2013		

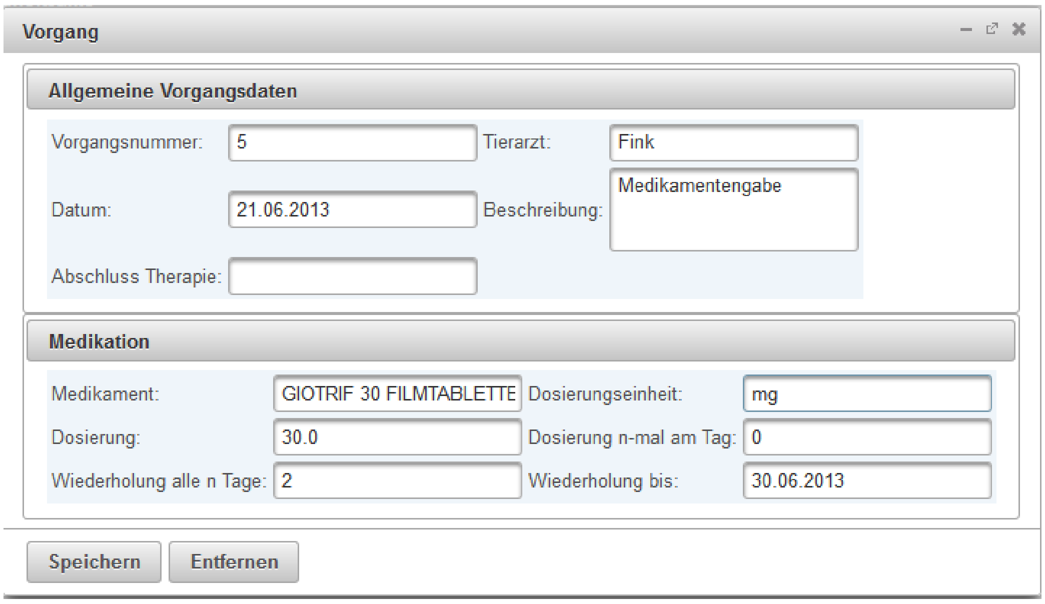
Diagnose

Diagnose:	Infektion	Beschreibung:	Infektion des Gastrointestinaltraktes
Meldepflichtig:	☐		

Figure 46: diagnosis dialog, C. Aigner

6.6.5 Medication dialog

The medication dialog contains all relevant data for the act "Medication". The drug can be selected by clicking on the medication input field, which opens a search dialog.


Vorgang

Allgemeine Vorgangsdaten

Vorgangsnummer:	5	Tierarzt:	Fink
Datum:	21.06.2013	Beschreibung:	Medikamentengabe
Abschluss Therapie:			

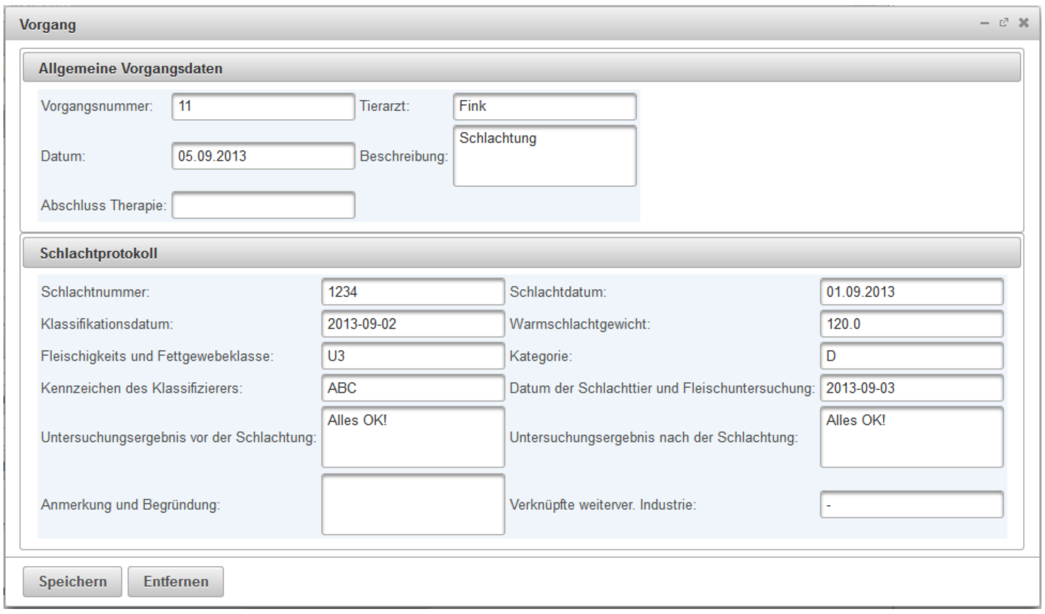
Medikation

Medikament:	GIOTRIF 30 FILMTABLETTE	Dosierungseinheit:	mg
Dosierung:	30.0	Dosierung n-mal am Tag:	0
Wiederholung alle n Tage:	2	Wiederholung bis:	30.06.2013

Figure 47: medication dialog, C. Aigner

6.6.6 Slaughter protocol dialog

The slaughter protocol dialog contains all relevant data for the act “Slaughter protocol”. The user can associate a processing industry by clicking on the corresponding input field which opens a search dialog.



Vorgang

Allgemeine Vorgangsdaten

Vorgangsnummer:	11	Tierarzt:	Fink
Datum:	05.09.2013	Beschreibung:	Schlachtung
Abschluss Therapie:			

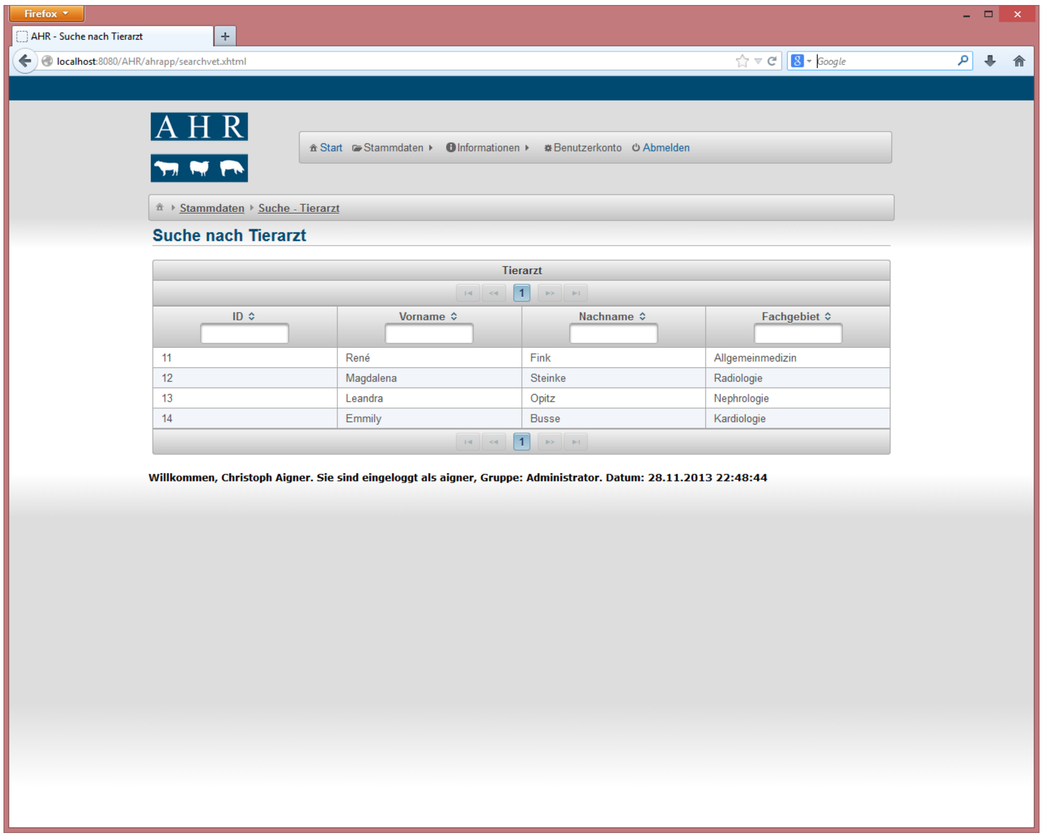
Schlachtprotokoll

Schlachtnummer:	1234	Schlachtdatum:	01.09.2013
Klassifikationsdatum:	2013-09-02	Warmschlachtgewicht:	120.0
Fleischigkeits und Fettgewebeklasse:	U3	Kategorie:	D
Kennzeichen des Klassifizierers:	ABC	Datum der Schlachttier und Fleischuntersuchung:	2013-09-03
Untersuchungsergebnis vor der Schlachtung:	Alles OK!	Untersuchungsergebnis nach der Schlachtung:	Alles OK!
Anmerkung und Begründung:		Verknüpfte weiterver. Industrie:	-

Figure 48: slaughter protocol dialog, C. Aigner

6.7 Search for veterinarians

Users can search for veterinarians by clicking on the menu button “Stammdaten – Tierärzte – Tierärzte suchen”. The view contains a data table that is paginated, sortable and searchable by column. Selecting a row redirects the user to the veterinarian detail view.



Suche nach Tierarzt

Tierarzt

ID	Vorname	Nachname	Fachgebiet
11	René	Fink	Allgemeinmedizin
12	Magdalena	Steinke	Radiologie
13	Leandra	Opitz	Nephrologie
14	Emmily	Busse	Kardiologie

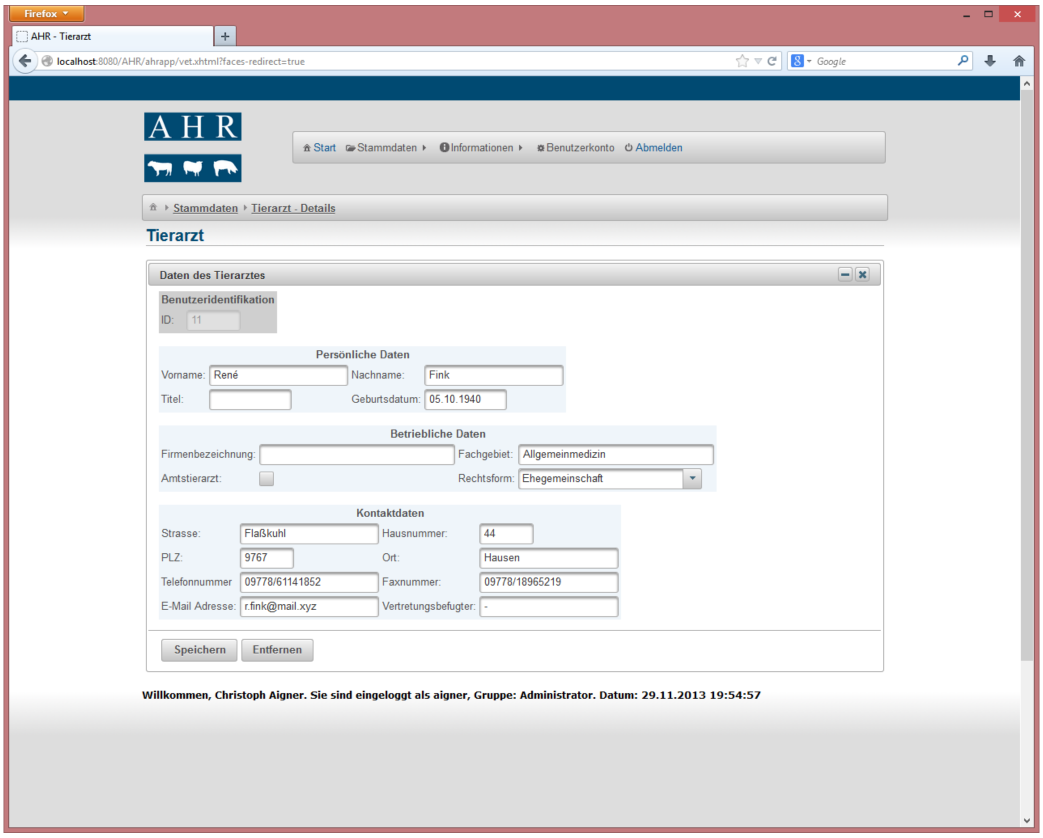
Willkommen, Christoph Aigner. Sie sind eingeloggt als aigner, Gruppe: Administrator. Datum: 28.11.2013 22:48:44

Figure 49: search veterinarian view, C. Aigner

6.8 Add/edit veterinarian-data

Administrators can add veterinarians via the menu button “Stammdaten – Tierärzte – Neuer Tierarzt”. Existing veterinarian data can be accessed via the farm search form or by clicking on “Mein Betrieb” if the current user is associated with a veterinarian object.

The panel contains all relevant veterinarian data. The footer contains the button “Speichern” that saves the veterinarian and the button “Entfernen”, which is only enabled for administrators, to delete the veterinarian.



Tierarzt

Daten des Tierarztes

Benutzeridentifikation

ID: 11

Persönliche Daten

Vorname:	René	Nachname:	Fink
Titel:		Geburtsdatum:	05.10.1940

Betriebliche Daten

Firmenbezeichnung:		Fachgebiet:	Allgemeinmedizin
Amtstierarzt:	☐	Rechtsform:	Ehegemeinschaft

Kontaktdaten

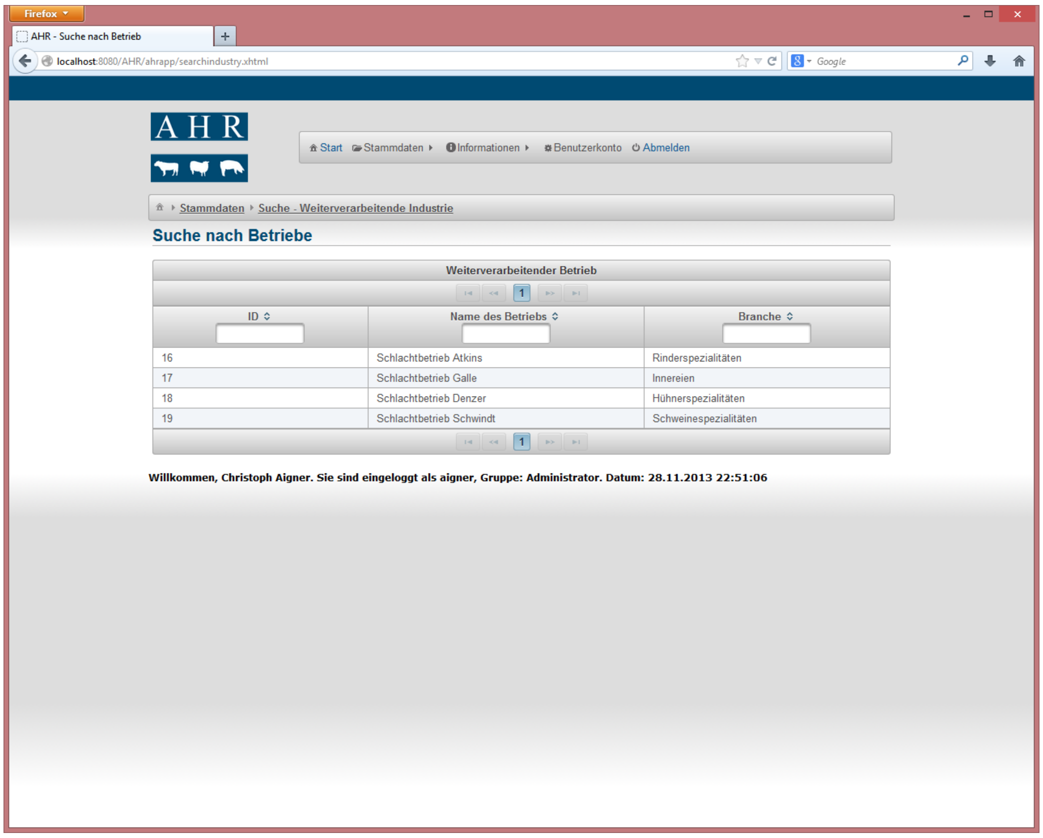
Strasse:	Flaßkuhl	Hausnummer:	44
PLZ:	9767	Ort:	Hausen
Telefonnummer	09778/61141852	Faxnummer:	09778/18965219
E-Mail Adresse:	r.fink@mail.xyz	Vertretungsbefugter:	-

Willkommen, Christoph Aigner. Sie sind eingeloggt als aigner, Gruppe: Administrator. Datum: 29.11.2013 19:54:57

Figure 50: add/edit veterinarian data, C. Aigner

6.9 Search for processing industries

The search form for processing industries is accessible via the menu entry “Stammdaten – Weiterver. Betriebe – Betriebe suchen”. The view contains a data table that is paginated, sortable and searchable by every column criteria. Authorized users may select a row and then be redirected to the detail page of the industry.



Suche nach Betriebe

Weiterverarbeitender Betrieb

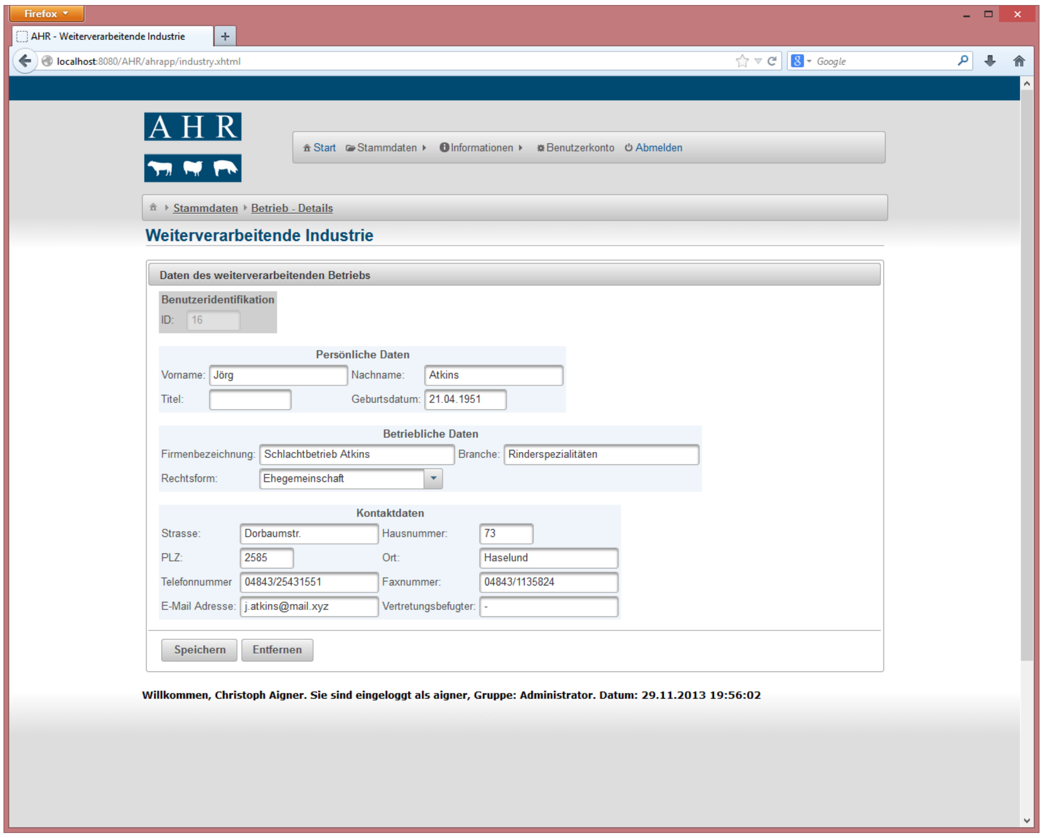
ID	Name des Betriebs	Branche
16	Schlachtbetrieb Atkins	Rinderspezialitäten
17	Schlachtbetrieb Galle	Innereien
18	Schlachtbetrieb Denzer	Hühnerspezialitäten
19	Schlachtbetrieb Schwindt	Schweinespezialitäten

Willkommen, Christoph Aigner. Sie sind eingeloggt als aigner, Gruppe: Administrator. Datum: 28.11.2013 22:51:06

Figure 51: search processing industry view, C. Aigner

6.10 Add/edit processing-industry-data

Administrators can add processing industries via the menu button "Stammdaten – Weiterver. Betriebe – Neuer Betrieb". Existing industries can be edited by selecting them in the processing industry search form.



Weiterverarbeitende Industrie

Daten des weiterverarbeitenden Betriebs

Benutzeridentifikation

ID: 16

Persönliche Daten

Vorname:	Jörg	Nachname:	Atkins
Titel:		Geburtsdatum:	21.04.1951

Betriebliche Daten

Firmenbezeichnung:	Schlachtbetrieb Atkins	Branche:	Rinderspezialitäten
Rechtsform:	Ehegemeinschaft		

Kontaktdaten

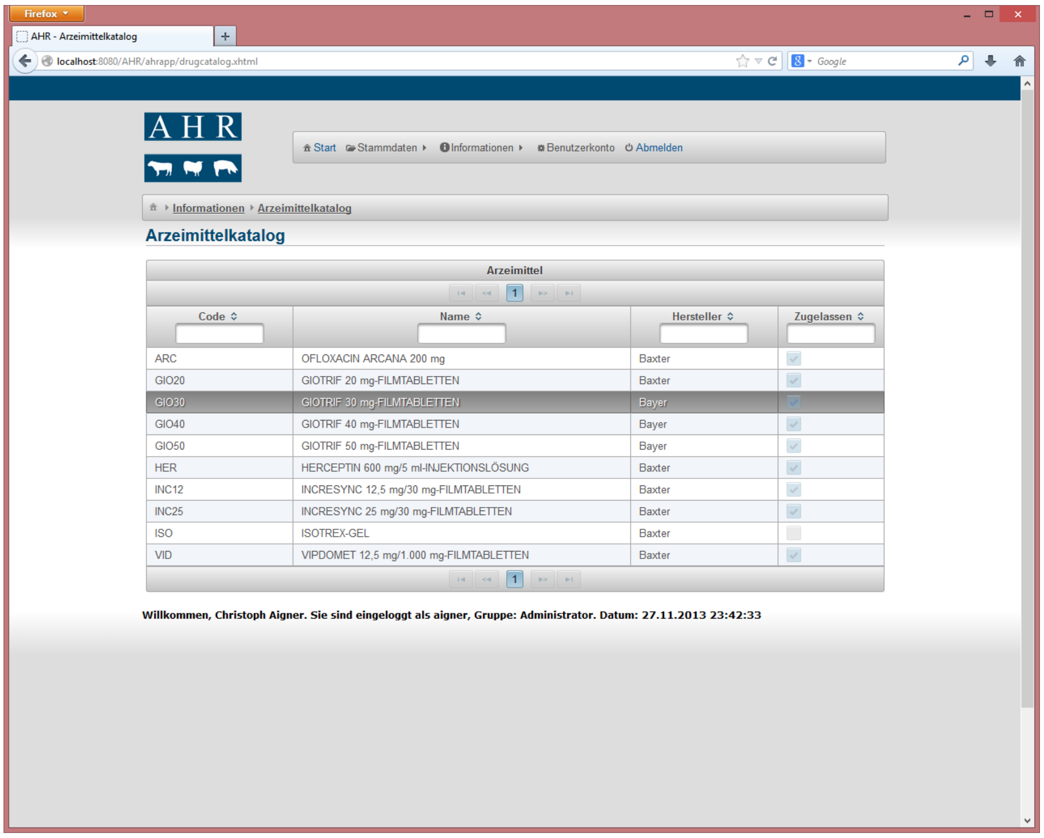
Strasse:	Dorbaumstr.	Hausnummer:	73
PLZ:	2585	Ort:	Haselund
Telefonnummer	04843/25431551	Faxnummer:	04843/1135824
E-Mail Adresse:	j.atkins@mail.xyz	Vertretungsbefugter:	-

Willkommen, Christoph Aigner. Sie sind eingeloggt als aigner, Gruppe: Administrator. Datum: 29.11.2013 19:56:02

Figure 52: add/edit processing industry data, C. Aigner

6.11 List of drugs

The list of drugs is available via the menu button “Informationen – Arzneimittelkatalog”. The view contains a data table that is paginated, sortable and searchable by every column. Selecting a row redirects the user to the drug detail view.



Informationen › Arzeimittelkatalog

Arzeimittelkatalog

Arzeimittel

Code	Name	Hersteller	Zugelassen
ARC	OFLOXACIN ARCANA 200 mg	Baxter	✓
GIO20	GIOTRIF 20 mg-FILMTABLETTEN	Baxter	✓
GIO30	GIOTRIF 30 mg-FILMTABLETTEN	Bayer	✓
GIO40	GIOTRIF 40 mg-FILMTABLETTEN	Bayer	✓
GIO50	GIOTRIF 50 mg-FILMTABLETTEN	Bayer	✓
HER	HERCEPTIN 600 mg/5 ml-INJEKTIONSLÖSUNG	Baxter	✓
INC12	INCRESYNC 12,5 mg/30 mg-FILMTABLETTEN	Baxter	✓
INC25	INCRESYNC 25 mg/30 mg-FILMTABLETTEN	Baxter	✓
ISO	ISOTREX-GEL	Baxter	
VID	VIPDOMET 12,5 mg/1.000 mg-FILMTABLETTEN	Baxter	✓

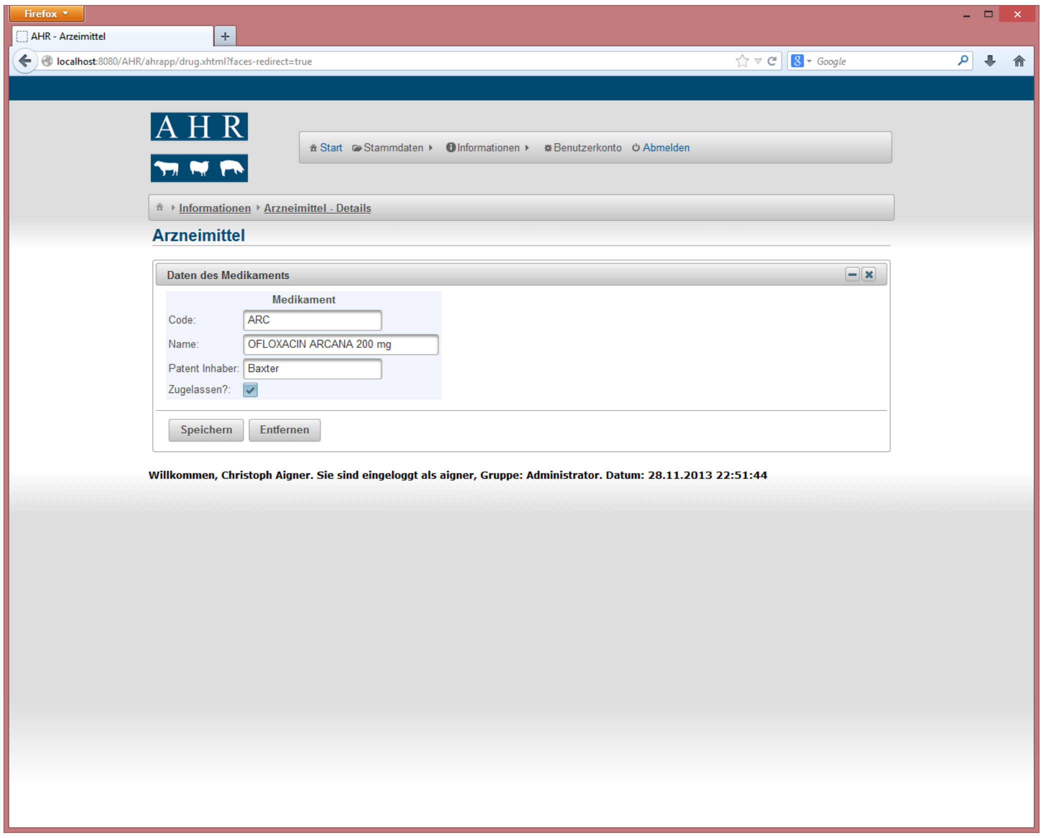
Willkommen, Christoph Aigner. Sie sind eingeloggt als aigner, Gruppe: Administrator. Datum: 27.11.2013 23:42:33

Figure 53: List of drugs, C. Aigner

6.12 Add/edit drug-data

To edit a drug, one must choose it from the data table. To add a new drug, administrators must click the menu item “Stammdaten – Arzeimittelkatalog – Neues Medikament”.

The form contains all relevant drug data. The panel footer contains two command buttons. The first button “Speichern” saves the drug. The second one “Entfernen” is only enabled for administrators and deletes the drug.



Arzneimittel

Daten des Medikaments

Medikament

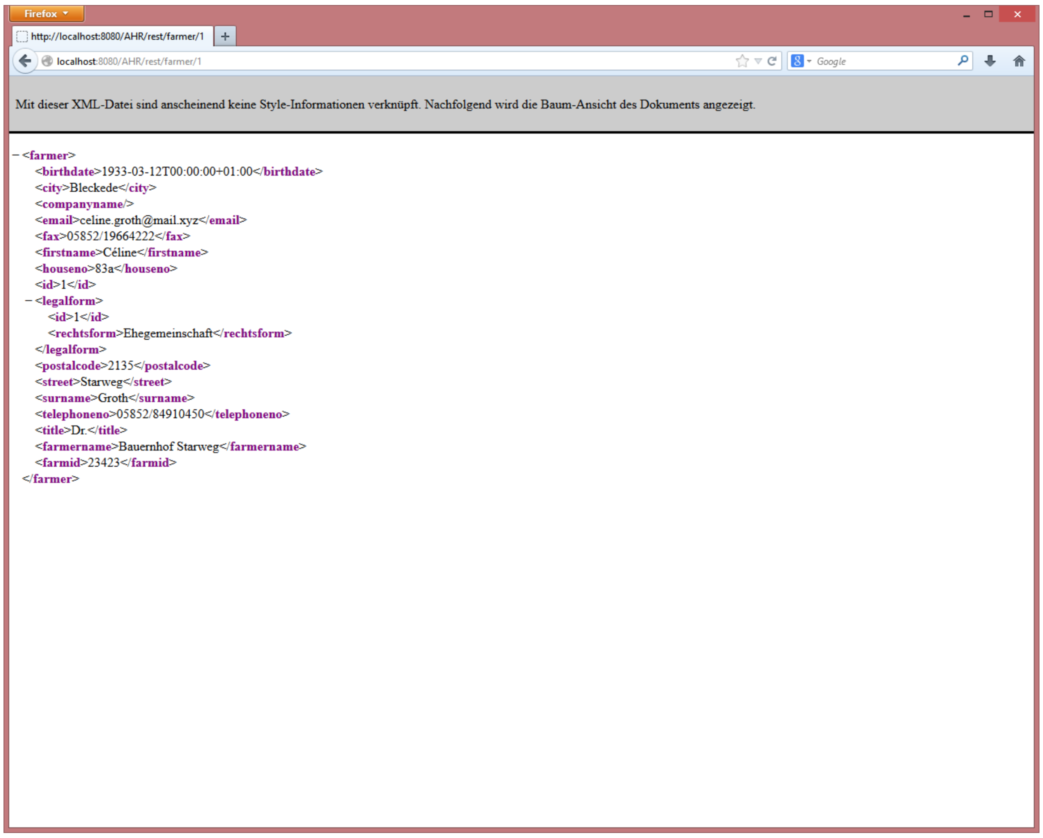
Code:	ARC
Name:	OFLOXACIN ARCANA 200 mg
Patent Inhaber:	Baxter
Zugelassen?:	✓

Willkommen, Christoph Aigner. Sie sind eingeloggt als aigner, Gruppe: Administrator. Datum: 28.11.2013 22:51:44

Figure 54: add/edit drug data, C. Aigner

6.13 Web service

To access the web service to retrieve a farmer by his identification number, the user or the external program must use the URI `http://localhost:8080/AHR/rest/farmer/{userid}`. The following screenshot shows the retrieval of the XML file via a web browser:



Firefox
http://localhost:8080/AHR/rest/farmer/1
localhost:8080/AHR/rest/farmer/1
Google

Mit dieser XML-Datei sind anscheinend keine Style-Informationen verknüpft. Nachfolgend wird die Baum-Ansicht des Dokuments angezeigt.

```
– <farmer>
    <birthdate>1933-03-12T00:00:00+01:00</birthdate>
    <city>Bleckede</city>
    <companyname/>
    <email>celine.groth@mail.xyz</email>
    <fax>05852/19664222</fax>
    <firstname>Céline</firstname>
    <houseno>83a</houseno>
    <id>1</id>
  – <legalform>
      <id>1</id>
      <rechtsform>Ehegemeinschaft</rechtsform>
    </legalform>
    <postalcode>2135</postalcode>
    <street>Starweg</street>
    <surname>Groth</surname>
    <telephoneno>05852/84910450</telephoneno>
    <title>Dr.</title>
    <farmername>Bauernhof Starweg</farmername>
    <farmid>23423</farmid>
  </farmer>
```

Figure 55: RESTful web service call, C. Aigner

7 Discussion and Conclusion

This master thesis describes the design process and implementation of a prototype for an animal health record in livestock management. It defines the AHR as a series of electronic records of medical and organizational data for an animal that is part of a livestock. Compared to existing commercial or academic software products, this prototype places a special emphasis on the management of livestock owners. Most of the commercial software products tend to focus more on the domestic animal sector. The O3-Vet project, developed at the University of Milan, focuses on veterinary hospitals. This prototype illustrates how software can not only be used by the veterinarian that oversees the livestock but by the owner himself. Since the prototype is a web application, all the user needs is an internet connection and a capable device. Conventional AHR-software must be installed and deployed in a hospital or private practise environment. This prototype can be hosted on a central server for all potential users.

The technological foundation of the prototype is the industry standard Java Enterprise Edition, in contrast to the O3-Vet project which was implemented using the PHP scripting language. PHP is a pure web technology whereas Java can be used in various scenarios. This makes it easier to deploy it in production environments where Java is already a prevalent technology. It gives the user a more state-of-the-art user experience without the use of special browser plugins which makes the application more platform-independent. The database was loosely based on the EHR standard HL7v3 RIM and shows that an animal health record can incorporate standards and methods from the field of human medicine.

The general conditions for this work were laid down in the AHR research group. Especially the evaluation of the functional and non-functional requirements, the stakeholders and roles were collaboratively done within the research group. These findings were the basis for the software and database design of the prototype.

The database was loosely based on the HL7v3 RIM standard with the integration of all the attributes and relations that were gathered by the research group.

Certain technologies had to be selected during the development process. It was decided to use JSF 2.0 and the PrimeFaces component suite for the GUI because it is better integrated into Java EE and is cross-platform. Since RESTful web services are considered a defacto standard and are more light-weight they were chosen as the underlying technology for the external interface of the AHR web application. Spring Security was chosen as the security API because it is wider used and its configuration integrates better with Java.

The prototype was developed and tested with the integrated development environment NetBeans version 7.3 and was deployed on the GlassFish application server version 3.1.2.2. MySQL version 5.6 was used as the underlying database server.

8 Perspective

This work can be used as a source for implementing enhancements and additional use cases that can improve the user experience and productivity.

A big step forward would be the support of mobile devices, especially smartphones. The PrimeFaces framework includes components for mobile devices powered by jQuery Mobile. It supports various platforms such as iPhone, Android, Palm, Blackberry and Windows Mobile [72]. These components could be used to implement new JSF pages that could utilize the existing business logic. The following example demonstrates the look and feel of a standard web form developed with the PrimeFaces Mobile component suite:

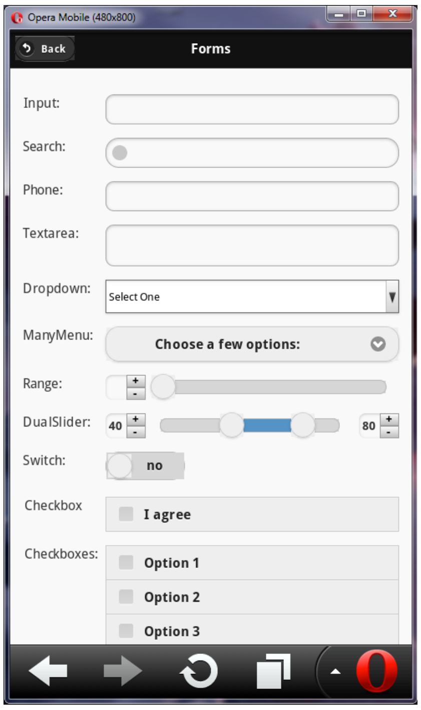


Figure 56: PrimeFaces Mobile [72]

The prototype does not implement all use cases the AHR group defined. Connecting the application to existing governmental or non-government systems for the exchange of data would be the next logical step. The development of a comprehensive report generator would be a valuable feature as well.

The use of more advanced JSF components, like the `<p:autoComplete>` component, that executes queries the moment the user starts typing would be an interesting enrichment for the application.

The drug database used for treatment data in the prototype is managed by an administrative user. A replication of or remote connection to the official drug catalogue provided by the ministry of health is mandatory for veterinarians managing their own medical supplies. The Austrian federal law "BGBl. II Nr. 65/2005 Apothekenbetriebsordnung" paragraph 58 states that amongst others it is required to possess a copy of the official drug catalogue, called the "Austria Codex-Fachinformation".

Furthermore, an individualization of the software could be beneficial. This could involve theming (including a theme switcher) and distinct start pages for every user.

A requirement for a regular operation would be a comprehensive test phase with farmers and veterinarians. The expected result of such a field test would be a refinement of the uses cases and the role concept. Important questions the stakeholders could answer are: Who should be able to access what data? Who wants to use the system besides the roles already defined within the scope of this thesis? Are all the data fields, especially for the various act datasets plausible?

Glossary

AHR	Animal health record
AJAX	Asynchronous JavaScript and XML
API	Application programming interface
CDDL	Common Development and Distribution License
CRUD	Create, read, update and delete
CT	computed tomography
DAO	Data access object
DICOM	Digital Imaging and Communications in Medicine
EHR	Electronic Health Record
EJB	Enterprise Java Beans
GPL	GNU General Public License
HL7	Health Level 7
HL7v3 RIM	HL7 version 3 Reference Information Model
HQL	Hibernate Query Language
HTML	HyperText Markup Language
IHE	Integrating the Healthcare Enterprise
IDE	Integrated development environment
JDK	Java Development Kit
JMS	Java Message Service
JPEG	Joint Photographic Experts Group
JPQL	Java Persistence Query Language
JSON	JavaScript Object Notation
JTA	Java Transaction API
MRI	Magnetic resonance tomography

MPEG	Moving Picture Experts Group
MVC	Model view controller
ORM	Object-relational Mapping
PACS	Picture archiving and communication system
PEAR	PHP Extension and Application Repository
REST	Representational State Transfer
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSL	Secure socket layer
TLS	Transport layer security
UDDI	Universal Description Discovery and Integration
UI	User interface
UML	Unified Modelling Language
WADL	Web Application Description Language
WSDL	Web Services Description Language
XML	Extensible Mark-up Language

Bibliography

[1] T. Benson, *Principles of Health Interoperability HL7 and SNOMED*. London: Springer-Verlag London Limited, 2010.

[2] S. I. GmbH. (2012, 01.10.2012). *Animal Office*. Available: http://www.animal-office.at

[3] F. M. T. S. C. M. Zaninelli, A. Ferrara, E. Ferro, P.G. Brambilla, S. Faverzani, S. Chinosi, P. Scarpa, M. Di Giancamillo, D. Zani, A. Zepponi, C. Saccavini, "The O3-Vet project: A veterinary electronic patient record based on the web technology and the ADT-IHE actor for veterinary hospitals," *Computer methods and programs in biomedicine,* pp. 68-77, 2007.

[4] V. Willner, "Erhebung und Analyse der Anfordedrungen an einen Animal Health Record (AHR) für Kleintiere," Fakultät für Informatik, Technische Universität Wien, Wien, 2011.

[5] A. Füresz, "Analyse, Systemdesign und Architekturentwurf einer elektronischen Gesundheitsakte für Nutztiere," Fakultät für Informatik, Technische Universität Wien, Wien, 2012.

[6] Y. Yamada, "The electronic health record as a primary source of clinical phenotype for genetic epidemiological studies," *Genomic Medicine,* vol. 2, p. 5, 2008.

[7] N. P. T. Tracy D Gunter, "The Emergence of National Electronic Health Record Architectures in the United States and Australia: Models, Costs, and Questions," *Journal of Medical Internet Research,* vol. 7, 2005.

[8] I. I. GmbH. (2012, 01.10.2012). *easyVET*. Available: http://www.easyvet.eu

[9] A. Austria. (2013, 27.03.2013). *Lebendrinderkennzeichnung und -registrierung*. Available: http://www.ama.at/Portal.Node/ama/public?gentics.am=PCP&p.contentid=10007.19455

[10] A. Austria. (2013, 27.03.2013). *eAMA*. Available: https://services.ama.at/servlet/

[11] Z. D. E.-D. GmbH. 27.03.2013). RDV4M Rinderdatenverbund für Mitglieder Benutzerhandbuch. Available: http://cgi.zar.at/download/Newsletter/RDV4M-Doku-neu.pdf

[12] S. Austria. (2012). *Veterinärinformationssystem*. Available: http://www.statistik.at/ovis/start/index.html

[13] T. B. Peter Schloeffel, George Hayworth, Sam Heard, Heather Leslie, "The relationship between CEN 13606, HL7, and openEHR," presented at the HIC 2006, Sydney, Australia, 2006.

[14] N. Krawetz, *Introduction to Network Security*: Cengage Charles River Media, 2007.

[15] H. A. Österreich. (2012, 06.12.2012). *HL7 Anwendergruppe Österreich*. Available: http://www.hl7.at

[16] H. L. S. International. (2012, 02.10.2012). *HL7 Messaging Standard Version 2.7*. Available: http://www.hl7.org/implement/standards/product_brief.cfm?product_id=146

[17] Interfaceware. (2012, 11.12.2012). *Understanding HL7 Messages*. Available: http://www.interfaceware.com/understanding_hl7_messages.html

[18] R. Spronk. (2007, 11.12.2012). *HL7 Message examples: version 2 and version 3*. Available: http://www.ringholm.de/docs/04300_en.htm

[19] jwenet.net. (2005, 11.12.2012). *HL7 Event Summary*. Available: http://jwenet.net/notebook/1777/1800.html

[20] A. E. A. Kemper, *Datenbanksysteme*, 2004.

[21] R. Brull. (2011, 14.01.2014). *HL7 v3 RIM: Is It Really that Intimidating?* Available: http://www.hl7standards.com/blog/2011/05/31/hl7-v3-rim-is-it-really-that-intimidating/

[22] iEHR. (2013, 25.01.2013). *HL7 CDA*. Available: http://iehr.eu/standards/hl7-cda/

[23] L. A. Robert H Dolin, Sandy Boyer, Calvin Beebe, Fred M Behlen, Paul V Biron, Amnon Shabo Shvo, "HL7 Clinical Document Architecture, Release 2," 2005.

[24] ISO, "13606-1 Health informatics - Electronic health record communication," in *Reference model*, ed, 2008.

[25] H. Leslie, "International developments in openEHR archetypes and templates," *Health Information Management Journal,* vol. 37, 2008.

[26] E. Association. (2013, 25.01.2013). *The CEN/ISO EN13606 standard*. Available: http://www.en13606.org

[27] ISO, "13606-2 Health informatics - Electronic health record communication," in *Archetype interchange specification*, ed, 2008.

[28] o. Foundation, "openEHR Architecture Release 1.0.2," in *Architecture Overview*, ed, 2008.

[29] J. W. Dean Bidgood, MD, MS, Steven C. Horii, MD, Fred W. Prior, PhD, Donald E. Van Syckle, "Understanding and Using DICOM, the Data Interchange Standard for Biomedical Imaging," *Journal of the American Medical Informatics Association,* vol. 4, 1997.

[30] dcm4che.org. (2013, 05.07.2013). *Open Source Clinical Image and Object Management*. Available: http://www.dcm4che.org

[31] J. Wilcke, "SNOMED, What's in it? Who's Involved? What's it for?," presented at the AVMA Vendors Meeting, New Orleans, Louisiana, 1999.

[32] M. F. Anne Casey RN, "SNOMED Clinical Terms - Introduction," ed: Royal College of Nursing.

[33] M. Clem McDonald, Stan Huff, MD, Jamalynne Deckard, Kathy Mercer, Jacqueline Phillips, and P. Daniel J. Vreeman, DPT. (2012). *Logical Observation Identifiers Names and Codes Users' Guide*. Available: http://loinc.org/downloads/files/LOINCManual.pdf

[34] I. C. on and V. G. A. N. (I.C.V.G.A.N.). (2012). *Nomina Anatomica Veterinaria Fifth Edition*. Available: http://www.wava-amav.org/Downloads/nav_2012.pdf

[35] J. N. Robbins, *Web Design in a Nutshell, Third Edition*: O'Reilly, 2006.

[36] W3C, "HTML 4.01 Specification," ed, 1999.

[37] W3C, "HTML 5.1 Nightly," 2013.

[38] W3C, "XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)," ed: W3C, 2000.

[39] W3C, "Document Object Model (DOM) Level 1 Specification," ed, 1998.

[40] W3C, "Document Object Model (DOM) Level 3 Core Specification," 2004.

[41] J. M. Nicholas C. Zakas, Joe Fawcett, *Professional Ajax, 2nd Edition*: Wiley Publishing, Inc., 2007.

[42] A. Gupta, *Java EE 6 Pocket Guide*: O'Reilly, 2012.

[43] A. Goncalves, *Beginning Java EE 6 with Glassfish 3: From Novice to Professional, Second Edition*: Apress, 2010.

[44] C. D. o. J. T. Specifications. (2013, 26.08.2013). *JSR 342: Java Platform, Enterprise Edition 7 (Java EE 7) Specification*.

[45] Hibernate. (2013, 02.09.2013). *About Hibernate*. Available: http://www.hibernate.org/about.html

[46] J. L. a. D. Minter, *Beginning Hibernate, Second Edition*, 2010.

[47] J. S. Doug Tidwell, Pavel Kulchenko, *Programming Web Services with SOAP*, 2001.

[48] Ç. Çivici, *Prime Faces Users' Guide 3.5*, 2013.

[49] E. R. Alaric Cole, *Learning Flex 4*, 2011.

[50] D. Marx. (2009, 22.11.2013). *Java EE and Flex, Part 1: A compelling combination*.

[51] C. Scarioni, *Pro Spring Security*, 2013.

[52] A. S. Foundation. (2013, 22.11.2013). *Apache Shiro Documentation*.

[53] R. H. Erich Gamma, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1994.

[54] S. M. Inc. (2002). *Core J2EE Patterns - Data Access Object*. Available: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html

[55] M. Fowler. (2004). *Inversion of Control Containers and the Dependency Injection pattern*. Available: http://martinfowler.com/articles/injection.html

[56] T. V. F. Stuart Hansen, "Refactoring model-view-controller," *Journal of Computing Sciences in Colleges . JCSC,* 2005.

[57] R. Eckstein. (2007, 17.11.2013). *Java SE Application Design With MVC*. Available: http://www.oracle.com/technetwork/articles/javase/index-142890.html

[58] R. M. Kim Hamilton, *Learning UML 2.0*, 2006.

[59] L. W. Smith, "Project Clarity Through Stakeholder Analysis," *CrossTalk,* vol. 13, pp. 4-9, 2000.

[60] A. A. M. G. (Ltd.). (2013, 04.09.2013). *AMA Marketing*. Available: http://www.amaexport.at/en/ama-marketing.html

[61] F. I. f. R. E. a. Training, "Quality assurance in agricultural animal husbandry," ed, 2010.

[62] P. L. H. Eide. (2005, Quantification and Traceability of Requirements.

[63] P. P.-S. Chen, "The Entity-Relationship Model - Toward a Unified View of Data," *ACM Transactions on Database Systems,* vol. 1, 1976.

[64] B. Zwattendorfer. (2013, Analyse SOAP vs. REST.

[65] P. M. Robert Winch, *Spring Security 3.1*, 2012.

[66] M. Raible. (2011, 27.11.2013). *Web Application Security - Part III: Apache Shiro*. Available: http://raibledesigns.com/rd/entry/java_web_application_security_part2

[67] M. K. Martin Marinschek, Gerald Müllan, Gerhard Petracek, Marcus Kröger, Andrea Schnabl. (2013, 14.11.2013). *JSF@Work*.

[68] T. G. Team. (2013, 14.05.2013). *GNU Image Manipulation Program*. Available: http://www.gimp.org/

[69] O. V. Mert Caliskan, *PrimeFaces Cookbook*: PACKT, 2013.

[70] I. Red Hat. (2013, 18.11.2013). *Hibernate Reference Documentation*. Available: http://docs.jboss.org/hibernate/orm/4.2/manual/en-US/html/

[71] A. S. Foundation. (2012, 26.11.2013). *Short introduction to log4j*. Available: http://logging.apache.org/log4j/1.2/manual.html

[72] PrimeFaces. (2013, 29.11.2013). *PrimeFaces Mobile*. Available: http://www.primefaces.org/showcase/mobile/index.jsf